Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Tesis de Licenciatura

Algoritmos Genéticos y la Estimación de Parámetros en Medios Acústicos

por

María Alejandra Arriola
María Teresa Arriola
Analía Fridman
Edgardo Sergio Safranchik

Directora
Lic. María Cristina Parpaglione

- Marzo de 1998 -

Abstract

Este trabajo estudia Algoritmos Genéticos en profundidad, haciendo hincapié en el basamento teórico desarrollado por Holland y en contribuciones posteriores que analizan detalladamente el rol de los operadores de cruza y mutación en el proceso de búsqueda. Se propone un método de diseño de Algoritmos Genéticos que permite enriquecerlos con conocimiento específico del problema a resolver y se lo aplica a la resolución de un problema complejo como es la inversión de la ecuación de onda acústica. Especial consideración merecen en este trabajo la representación del cromosoma y la utilización de operadores con heurística como factores de fundamental importancia para mejorar el desempeño del Algoritmo Genético.

Palabras Claves: algoritmos genéticos, inversión, onda acústica

Abstract

This work covers Genetic Algorithms in depth. It stresses on the theoretical background developed by Holland and further contributions on the role of crossover and mutation in the search process. A design method is given, in order to enhance Genetic Algorithms with problem oriented knowledge. This method is applied to the resolution of a complex problem as is the inversion of the acoustic wave equation. Chromosome representation and heuristic operators as important factors for improving the performance of Genetic Algorithms deserve special consideration.

Keywords: *genetic algorithms, inversion, acoustic waves*

Tabla de Contenidos

Indice de Tablas

Indice de Figuras

Introducción

Algoritmos Genéticos es un método especialmente apto para la optimización de funciones en espacios de búsqueda amplios y complejos. Inspirado en mecanismos de selección natural, opera sobre una población de individuos, cada uno de los cuales es una solución posible. La población evoluciona a través del tiempo mediante recombinación de sus integrantes y selección de los más aptos.

El objetivo del presente trabajo es el estudio de Algoritmos Genéticos y su aplicación a un caso práctico de difícil resolución para los métodos tradicionales.

La primera parte de esta tesis presenta un estudio de Algoritmos Genéticos, sus características, propiedades y variantes. En el Capítulo 1 se desarrolla una introducción al concepto de Algoritmos Genéticos, mostrando con un ejemplo sencillo el algoritmo básico. Las variantes más conocidas para cada operador se resumen en el Capítulo 2. En este se describen diversos métodos de selección, cruza y mutación resaltando ventajas y desventajas de cada uno. El Capítulo 3 consta de dos partes, en la primera se presenta el marco teórico desarrollado por Holland [HOL/75] para el Algoritmo Genético clásico: el concepto de esquemas y el *Teorema Fundamental de Algoritmos Genéticos*. La segunda parte del Capítulo 3 extiende esta teoría a otras variantes de operadores basando el análisis en dos conceptos: *disrupción* y *construcción* de esquemas. El Capítulo 4 completa el análisis de un Algoritmo Genético planteando el problema de la codificación de parámetros y el diseño de la función de aptitud. Este capítulo culmina con la propuesta de un método de diseño para un Algoritmo Genético capaz de resolver un problema dado. El Capítulo 5 ubica el concepto de Algoritmo Genético en la familia de los *Algoritmos Evolucionarios*, presentando otros algoritmos pertenecientes a esta categoría.

La segunda parte de esta tesis plantea un problema de optimización dificultoso para la mayoría de los métodos de búsqueda tradicionales y describe el proceso de diseño de un Algoritmo Genético capaz de resolverlo. El Capítulo 6 presenta el problema de optimización, el cual consiste en invertir la ecuación de onda acústica para la estimación del perfil sísmico vertical en las primeras capas de la corteza terrestre. Este problema presenta la dificultad de buscar un mínimo en una superficie multimodal y altamente dimensional. El desconocimiento de la función a minimizar, la abundancia de mínimos locales y la gran cantidad de dimensiones convierten al problema en un desafío para Algoritmos Genéticos. El Capítulo 7 plantea sucesivamente tres diseños de Algoritmo Genético para resolver el problema presentado en el Capítulo 6. Cada uno de estos diseños surge a partir del conocimiento aportado por el anterior y como resultado de aplicar el método de diseño de Algoritmo Genético presentado en el Capítulo 4. El Capítulo 8 presenta un resumen de los resultados obtenidos en la resolución del problema descripto con los tres Algoritmos Genéticos propuestos. En éste se muestran los avances logrados en cuanto a eficiencia y eficacia para hallar un perfil sísmico que satisfaga las condiciones del problema. En el Anexo A se presenta el manual del usuario del sistema *Algen*, que implementa las tres propuestas de Algoritmos Genéticos planteadas en el Capítulo 7. Por último, en el Anexo B se describen los detalles de implementación de este sistema.

Antecedentes

Algoritmos Genéticos ha sido desarrollado por John Holland a principios de la década del 70 en la Universidad de Michigan, EE.UU. El trabajo fundamental para la teoría de Algoritmos Genéticos es *"Adaptation in Natural and Artificial Systems"* [HOL/75]. Otros investigadores como David E. Goldberg y Kenneth De Jong, han realizado importantes aportes y expandido ampliamente el campo de investigación.

El problema de la estimación del perfil sísmico de la corteza terrestre se basa en el trabajo realizado por E. Fernández, P. Gausellino, J. Santos y D. Sheen *"Parameter Estimation in Multidimensional Acoustic Media"*, Center for Applied Mathematics, Purdue University,1993 [FER/93a].

Motivación

Existen diversos métodos de búsqueda de mínimos. El método del gradiente descendente es propenso a caer en mínimos locales. Los métodos de búsqueda al azar exploran el espacio total del problema, pero desechan el conocimiento de las soluciones parciales que van obteniendo, lo cual puede resultar muy costoso en tiempo computacional. Algoritmos Genéticos usa la información de las soluciones intermedias para orientar la búsqueda y generar nuevas posibles soluciones. Además al explorar el espacio de búsqueda en varios puntos a la vez se reduce la posibilidad de caer en mínimos locales. Algoritmos Genéticos no requiere que la función a optimizar cumpla con características especiales como derivabilidad o restricciones sobre la cantidad de óptimos.

El problema de inversión de la Ecuación de Onda Acústica presenta características que lo hacen adecuado para ser resuelto por Algoritmos Genéticos. La gran amplitud y alta dimensionalidad del espacio de búsqueda, el desconocimiento de la función a optimizar y la existencia de mínimos locales son obstáculos que otros métodos de búsqueda hallan dificultuoso sortear.

Este trabajo plantea la utilización de Algoritmos Genéticos como método de resolución de problemas de optimización. En particular pretende brindar una solución razonable al problema de Inversión de la Ecuación de Onda Acústica para estimar el perfil sísmico vertical en las primeras capas de la corteza terrestre.

Capítulo 1

Algoritmos Genéticos

Un sistema biológico eficiente desarrolla estrategias exitosas de adaptación para lograr su supervivencia y propagación. La Teoría de la Evolución postula que las mejoras en una especie se logran por la evolución de la misma a lo largo de cientos y miles de generaciones, donde los individuos que sobreviven son los más aptos y los menos aptos desaparecen [GOL/89].

Un Algoritmo Genético, en adelante AG, simula la evolución de una ***población*** de individuos, mediante un proceso iterativo aplicado sobre un conjunto de estructuras. Cada estructura esta compuesta de ***características*** que definen la ***aptitud*** del individuo en un entorno. La población de estructuras evoluciona de generación en generación mediante la ***recombinación*** de sus integrantes, la ***mutación*** de algunas características elegidas al azar, y la ***selección*** de aquellas estructuras que se mostraron más aptas. El material genético sobrevive a cada generación, combinándose y ampliando su presencia en la población, en la medida en que las estructuras que lo contienen manifiesten aptitud relativamente buena.

Desde el punto de vista de la resolución de problemas, un individuo representa una solución posible a un problema dado. La aptitud del mismo es una medida de cuán buena es la solución para dicho problema. El objetivo del AG es entonces buscar una "buena" solución al problema.

1.1 Algoritmo Simple

Previo a la aplicación de un AG para resolver un problema, es necesario definir dos elementos: la representación de las soluciones candidatas en individuos de una población, y la construcción de la función que medirá la aptitud de cada uno de los individuos.

La generación de la población inicial se realiza habitualmente creando individuos en forma aleatoria. Luego, cada generación se crea a partir de la generación anterior tras la aplicación de tres operadores básicos: selección, cruza y mutación. El proceso iterativo prosigue hasta que el algoritmo cumple con la condición de parada. Esta condición puede ser: algún individuo presenta una aptitud suficientemente buena, se alcanza una cota máxima de evaluaciones o generaciones, toda la población está dominada por el mismo individuo, o se cumple alguna condición específica del problema.

Los operadores considerados fundamentales para un AG son la selección y la cruza. La mutación habitualmente es considerada un operador secundario, si bien cumple un rol necesario para la diversificación de la población.

La selección es el proceso por el cual sobreviven los mejores individuos de acuerdo a su aptitud. Esta aptitud esta representada por una función *f* que, intuitivamente, se puede pensar como alguna medida de utilidad o beneficio que se pretende maximizar. Este operador se implementa asignando a cada individuo una cantidad de copias de sí mismo acorde a su aptitud de modo que un individuo con buena aptitud tendrá más probabilidad de participar de la cruza que uno poco apto.

La ***selección*** provee la abstracción del mecanismo de selección natural.

Algunos de los individuos seleccionados sobreviven directamente en la próxima generación, y otros participan de la cruza contribuyendo en forma indirecta con el aporte de su material genético. La proporción de la población que es reemplazada en cada generación se denomina ***salto generacional***. La cruza toma dos individuos sobrevivientes, recombina algunas de sus características - material genético - y produce dos nuevos individuos que pertenecen a la próxima generación [1].

La ***cruza*** abstrae la reproducción sexual en los sistemas naturales.

La mezcla de código genético provista por la cruza brinda un método de búsqueda altamente eficiente en el espacio de estructuras. No se trata de un proceso totalmente estocástico ya que la selección impone un sesgo en la búsqueda que realiza la cruza. Esto ocurre porque los mejores individuos tienden a dominar la población. Para garantizar que ninguna característica se pierda irremediablemente de la población se utiliza el operador de mutación. La mutación toma unos pocos individuos generados por la cruza y altera algunas características tomadas al azar.

La ***mutación*** altera aleatoriamente una o más características de un individuo existente.

La mutación no sólo evita que se pierdan alelos, sino que también contribuye a la creación de nuevos individuos.

El ciclo de ejecución básico de un AG es simple [Figura 1-1]. Este comienza con la generación aleatoria de una población inicial de estructuras. Luego se evalúa la aptitud de cada una aplicando la función *f* definida a tal fin. Si alguna estructura, o la evolución misma, cumple con la condición de parada, el algoritmo termina y el resultado es la mejor estructura presente en la población. De lo contrario, se seleccionan las estructuras más aptas asignando a estas una cantidad de copias proporcional a su aptitud. Parte de esta población intermedia se cruza para generar nuevas estructuras que heredan el material genético de sus padres. Por último, algunas características de la población son elegidas para mutar, alterando así las estructuras que los contienen. La población recién creada es evaluada y el ciclo vuelve a comenzar.

[1] La cruza tradicional produce dos hijos, sin embargo no es la única alternativa. Otra posibilidad utilizada habitualmente consiste en generar un único hijo, o inclusive más.

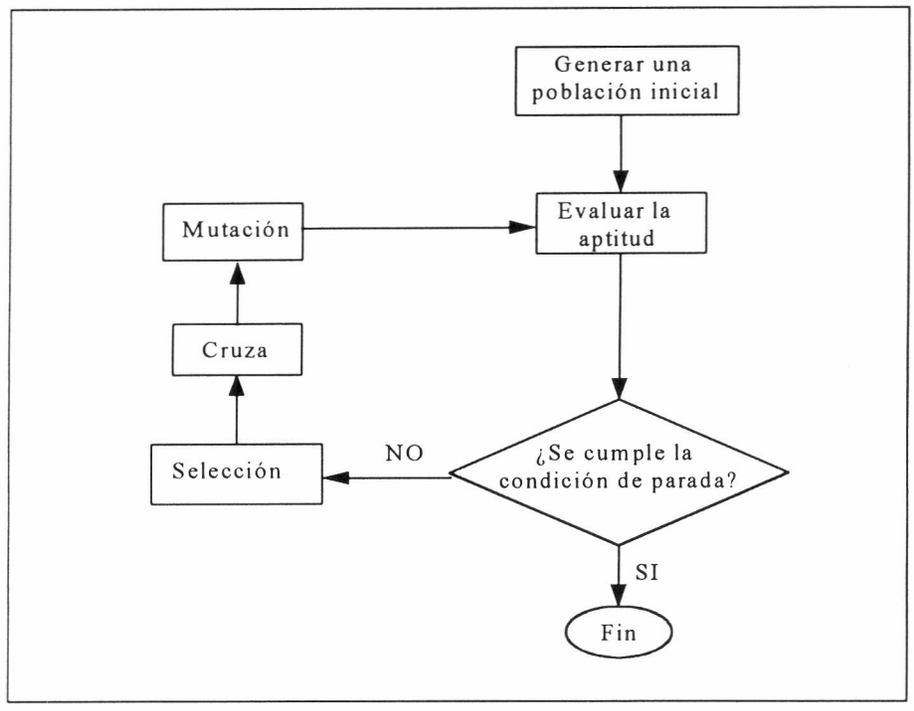

Figura 1-1. Algoritmo Genético Simple

1.2 Ejemplo Sencillo

Sea una población inicial de cuatro individuos de longitud l=5, donde cada característica (***gen***) puede tomar los valores (***alelos***) X o Y:

XYYXY
YYXXX
XYXXX
YXXYY

La selección puede ser implementada de varias formas. Una de ellas es simular una ruleta donde cada individuo de la población tenga asignada una ranura. El tamaño de esta ranura será proporcional a la aptitud del individuo respectivo. El método consiste en lanzar una "bolilla imaginaria" tantas veces como individuos se pretenda seleccionar. Cada individuo obtiene una copia por cada vez que la bolilla cae en su ranura. Obviamente, aquellos individuos con una ranura más grande tendrán mayor oportunidad de ser elegidos. Este método es conocido como ***selección por ruleta*** [GOL/89].

En la [Tabla 1-1] se muestra una posible asignación de aptitudes para los individuos, junto con la cantidad de copias esperadas en base a esta aptitud, y la cantidad de copias asignadas realmente.

Nº	*Individuo*	*Aptitud*	*Porcentaje*	*Copias Esperadas*	*Copias Asignadas*
1	*XYYXY*	*169*	*14.4*	*0*	*1*
2	*YYXXX*	*576*	*49.2*	*2*	*2*
3	*XYXXX*	*64*	*5.5*	*0*	*0*
4	*YXXYY*	*361*	*30.9*	*2*	*1*
Total		***1170***	***100.0***	***4***	***4***

Tabla 1-1. Selección por Ruleta

Luego de la selección se aplica el operador de cruza. En este ejemplo se utilizará la ***cruza en un punto*** [GOL/89] que se implementa de la siguiente forma. Se toma un par de individuos previamente seleccionados y se elige en forma aleatoria una posición k entre 1 y la longitud del individuo menos 1 [1, l -1]. Dos nuevos individuos son creados intercambiando los genes entre la posición $k + 1$ y l inclusive. Por ejemplo, eligiendo las estructuras 1 y 2, con $k = 3$ se tiene:

$$\begin{array}{ccc} \text{XYY|XY} & & \text{XYY|}\mathbf{XX} \\ & \rightarrow & \\ \mathbf{\text{YYX|XX}} & & \mathbf{\text{YYX|}}\text{XY} \end{array}$$

Luego se aplica el operador de mutación, que consiste en alterar el valor de algunos genes dentro de los individuos de la población. Por ejemplo:

$$\text{XYY}\mathbf{\text{XX}} \rightarrow \text{XYY}\mathbf{\text{Y}}\text{X}$$

La población resultante es nuevamente evaluada y si la condición de parada no se cumple, se continúa aplicando los operadores a la generación obtenida.

1.3 Sistemas Naturales y Sistemas Artificiales

Existe una correspondencia entre la terminología usada en la genética natural y la artificial [GOL/89]. En este punto es necesario aclarar los términos para evitar confusiones.

En la naturaleza, las características de un organismo están determinadas por uno o más ***cromosomas***. En los sistemas genéticos artificiales, cada ***individuo*** se representa con una única ***estructura***, y ambos términos se consideran equivalentes al de cromosoma. Un cromosoma esta compuesto por ***genes***, cada uno de los cuales puede tomar un valor llamado ***alelo***. La posición de un gen dentro del cromosoma se denomina ***locus***. Por ejemplo, en el gen que representa el color de ojos de un animal, su locus podría ser la posición 10, y el valor de su alelo, ojos marrones. En la genética artificial los estructuras estan compuestas de ***características*** que pueden tomar diferentes ***valores***. Las características pueden estar ubicadas en diferentes ***posiciones*** de un individuo.

Es importante destacar los conceptos de ***genotipo*** y ***fenotipo***. El conjunto de genes de un individuo se llama genotipo. El fenotipo es el resultado de la interacción

del genotipo con su entorno. Desde el punto de vista de la genética artificial, el fenotipo se corresponde con la ***aptitud*** del individuo.

Sistemas Naturales	*Algoritmos Genéticos*
cromosoma	*individuo, o estructura*
gen	*característica o atributo*
alelo	*valor*
locus	*posición en la estructura*
genotipo	*conjunto de genes o estructura*
fenotipo	*aptitud del individuo*

Tabla 1-2. Sistemas Naturales y Artificiales

A lo largo del presente trabajo se usará la terminología natural y la artificial en forma indistinta.

1.4 Aplicaciones

Los Algoritmos Genéticos son atractivos por varias razones:

1. Pueden resolver en forma rápida y eficiente problemas complejos con características como:

   - Alta cardinalidad del espacio de búsqueda
   - Alta dimensionalidad de la función de aptitud
   - Funciones de aptitud no lineales

2. Utilizan muy poca información específica del problema, dado que sólo requieren la posibilidad de proponer una solución y asignar una evaluación.

3. Son extensibles. En los problemas reales es muy difícil anticipar todas las dificultades, sin embargo es muy fácil modificar o incorporar conocimiento en los operadores genéticos.

4. Pueden ser utilizados para realizar una primera búsqueda global, para luego continuar con algún método de búsqueda local.

Los Algoritmos Genéticos tienen básicamente dos campos de aplicación: los problemas de optimización y la simulación de ambientes donde el objetivo es maximizar los beneficios acumulados a través del tiempo. Los problemas de optimización son los más habituales como ser: la maximización de eficiencia o calidad, minimización de riesgos, costos o tiempo, en proyectos técnicos, económicos o científicos [BAC/97].

Algunos ejemplos de utilización de AG para problemas de búsqueda y optimización son [GOL/89]: coloreo de grafos, diseño de circuitos VLSI, optimización del tamaño de enlaces en una red de comunicaciones, problema del viajante, optimización de funciones, procesamiento de imágenes, reconocimiento de patrones,

sistemas clasificadores, y programación genética. En particular, los sistemas clasificadores utilizan AG para explorar el espacio de reglas de producción de un sistema de aprendizaje, y la programación genética realiza una búsqueda sobre un espacio de programas de computación escritos habitualmente en LISP [BAC/97].

Capítulo 2

Operadores Básicos: Variantes

El propósito de cada uno de los tres operadores básicos está bien definido: elegir los individuos más aptos (selección), recombinar algunos de ellos para producir nuevos individuos (cruza), y alterar características de algunos para garantizar diversidad (mutación). Con estos objetivos presentes, se han desarrollado gran cantidad de variantes para cada uno de los operadores desde la presentación de AG por Holland en 1975 [HOL/75]. Cada variante, o método, tiene características particulares que afectarán el comportamiento del AG. Una elección adecuada para cada operador puede influir decisivamente en la eficacia y eficiencia del proceso de búsqueda. En este capítulo se brinda una breve reseña de los métodos más conocidos de selección, cruza y mutación para un AG simple.

2.1 Métodos de Selección

Los individuos elegidos con el operador de selección aportarán sus genes a la generación siguiente, por lo tanto, es deseable que el operador de selección, elija los mejores individuos de la población actual. Existen dos tipos de métodos usados comúnmente: los ***proporcionales*** y los basados en el ***orden*** [GOL/95].

Los métodos proporcionales de selección eligen individuos teniendo en cuenta el peso de su aptitud respecto del resto de la población. En cambio, los métodos basados en el orden confeccionan una tabla ordenada de individuos en base a su aptitud, seleccionando cada uno de acuerdo a su ubicación es esta tabla de posiciones. A continuación se describen los métodos de selección más comunes [GOL/89].

2.1.1 Selección por Ruleta

Este método consiste en construir una ruleta particionada en ranuras de igual tamaño, las cuales se numeran. A cada individuo de la población se le asigna una cantidad de ranuras proporcional a su aptitud [Figura 2-1]. La "manecilla" de la ruleta se gira, y con probabilidad uniforme se elige una ranura seleccionándose aquel individuo dueño de la misma. El proceso se repite hasta completar la cantidad de individuos deseados. Este método de selección otorga mayor probabilidad de contribuir a la siguiente generación a los individuos con mayor aptitud. Como se trata de un método probabilístico, el número obtenido de copias para un individuo puede ser muy distante del esperado.

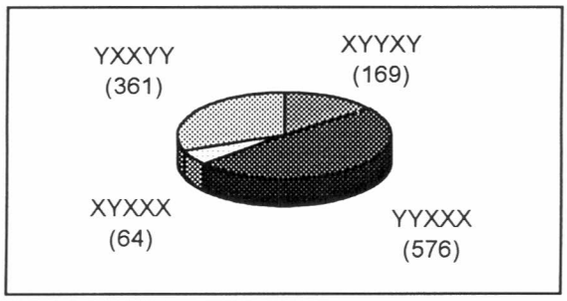

Figura 2-1. Selección por Ruleta

Este método fue utilizado para el ejemplo de la Sección 1.2. En la [Figura 2-1] se muestra la ruleta correspondiente, con las ranuras asignadas a cada estructura.

2.1.2 Selección con Control sobre el Número Esperado

La selección por ruleta intenta asignar una cantidad de copias adecuada a los mejores individuos, pero no lo garantiza. Como se mencionó anteriormente, puede haber una gran distancia entre el número esperado de copias y la cantidad asignada. Para reducir este error estocástico, De Jong [GOL/89] diseñó el modelo del valor esperado. Se define una cantidad de copias c_i para un individuo i, tal que:

$$c_i = \frac{f_i}{\overline{f}}$$

donde f_i es la aptitud del i-ésimo individuo y $\overline{f}$ es la aptitud promedio de la población. Cada individuo recibe una cantidad de copias igual a la parte entera de c_i, más una copia adicional con probabilidad igual a la parte fraccionaria de c_i. Esto garantiza que la cantidad de hijos de cualquier individuo es por lo menos la parte entera del número esperado de hijos.

N°	*Individuo*	*Aptitud*	*ci*	*Copias Esperadas*	*Copias Asignadas*
1	*XYYXY*	*169*	*0.58*	*≥0*	*0+1*
2	*YYXXX*	*576*	*1.97*	*≥1*	*1+1*
3	*XYXXX*	*64*	*0.22*	*≥0*	*0+0*
4	*YXXYY*	*361*	*1.23*	*≥1*	*1+0*
Total		***1170***	***4***		***4***

Tabla 2-1. Selección con Control sobre el Número Esperado

En la [Tabla 2-1] se muestra este método de selección aplicado al ejemplo visto en la Sección 1.2.

2.1.3 Selección Elitista

Los dos métodos anteriores no garantizan la preservación de los mejores individuos, ya que estos pueden ser reemplazados por sus hijos durante la cruza. La

selección elitista soluciona este problema, común a la mayoría de los métodos de selección.

El método elitista preserva los mejores *m* individuos de la generación actual, incluyéndolos directamente en la siguiente. Se busca mejorar la búsqueda local a expensas de la perspectiva global. También provoca que la aptitud del mejor individuo de la población mejore o a lo sumo se mantenga constante, pero nunca decrezca. La selección elitista siempre es utilizada en combinación con otras variantes de selección.

2.1.4 Selección por Ranking

Un problema habitual de los métodos proporcionales es la posibilidad de que súper-individuos dominen rápidamente la población, antes de que se haya podido realizar una exploración suficientemente amplia del espacio de búsqueda. Este problema se denomina ***convergencia prematura***. Una de las alternativas para solucionarlo consiste en utilizar un método de selección que tenga en cuenta cuáles son los mejores individuos, pero que no considere el peso de su aptitud[1].

En este método cada individuo recibe una cantidad de copias que sólo depende de su ubicación dentro de la tabla. Para esto, la población se ordena en forma descendente por la aptitud de cada individuo en una tabla de posiciones, y solamente los primeros *m* individuos reciben copias. Dentro de estas *m* primeras posiciones de la tabla, el total de copias a asignar se distribuye en forma lineal.

Una posible implementación de este método es la siguiente [GRE/90]. Sea *n* el tamaño de la población. Se define un mapeo lineal de tal forma que la peor estructura recibe un número esperado de copias $R_{\min} \in (0,1)$ y la mejor recibe un número esperado de $2 - R_{\min}$. Al individuo ubicado en la posición *i* de la tabla se le asigna la siguiente cantidad de copias:

$$R_{\min} + 2 \cdot \frac{(n-i)(1-R_{\min})}{n-1}$$

Aplicando este método al ejemplo visto en la Sección 1.2 se tiene:

N°	*Individuo*	*Aptitud*	*Copias Esperadas*	*Copias Asignadas*
2	*YYXXX*	*576*	*1,25*	*2*
4	*YXXYY*	*361*	*1,08*	*1*
1	*XYYXY*	*169*	*0,91*	*1*
3	*XYXXX*	*64*	*0,75*	*0*
Total		***1170***	***4***	***4***

Tabla 2-2. Selección por Ranking

[1] Otra alternativa, mucho mas difundida, consiste en el escalado de la función de aptitud. La utilización de escalado se describe en la Sección 3.4.2.

Este método de selección tiene dos desventajas frente a los métodos proporcionales. La primera es que la evolución de la población frecuentemente es más lenta. La segunda desventaja es que no existe una teoría sólida que lo soporte, como existe para los proporcionales.

2.2 Métodos de Cruza

Es importante el rol que juega el operador de cruza en el diseño e implementación de sistemas adaptativos robustos. En la mayoría de los AG los individuos son representados por estructuras de longitud fija y la recombinación es implementada por el operador de cruza. Este opera sobre pares de estructuras (padres) para producir nuevas estructuras (hijos) por el intercambio de segmentos entre los padres.

Cada estructura participa de la cruza con probabilidad P_c . La cantidad de hijos generados depende de la implementación. La elección más aceptada consiste en generar dos hijos, siendo uno el complemento del otro. Sin embargo, también hay variantes donde la cruza genera un único hijo o incluso más de dos. A continuación se describen los métodos más conocidos del operador de cruza [GOL/89].

2.2.1 Cruza Simple

La cruza simple, o en un punto, elige al azar uno de los l-1 posibles puntos de cruza. Luego de esta elección se intercambian los segmentos de cromosoma separados por este punto. Suponiendo las siguientes estructuras de longitud l = 8, y eligiendo 3 como el punto de cruza se tiene:

XYX|XYYYX → XYX|**YYXXY**

YYX|**YYXXY** → **YYX**|XYYYX

2.2.2 Cruza Multipunto

En este caso, el cromosoma es considerado un anillo, y se eligen n puntos de cruza en forma aleatoria. Si la cantidad de puntos de cruza es par, se intercambian las porciones de cromosomas definidas entre cada par de puntos consecutivos, si es impar se asume un punto de cruza adicional en la posición cero y se procede de igual modo.

Para ilustrar esta variante, supongamos dos estructuras de longitud l = 8, con n = 4 puntos de cruza. Intercambiando los segmentos de la posición 2 a 4 y 6 a 7, se tiene:

X|YXX|Y|YY|X → X|**YXY**|Y|**XX**|X

Y|**YXY**|**Y**|**XX**|**Y** → **Y**|YXX|**Y**|YY|**Y**

La ***cruza simple*** en realidad es un caso particular de la cruza multipunto cuando $n = 1$. Se debe mencionar, también, la ***cruza de dos puntos*** por la cantidad de estudios e implementaciones de que ha sido objeto.

2.2.3 Cruza Binomial

Para generar un cromosoma hijo por cruza binomial, se define la probabilidad P_0 como la probabilidad de que el alelo de cualquier posición del descendiente se herede del padre, y 1 - P_0 como la probabilidad de que lo herede de la madre[1]. En este caso se puede construir un único hijo por cada aplicación del operador, o bien generar un segundo hijo como complemento del primero.

Cuando existe igual probabilidad de heredar del padre como de la madre, $P_0 = 0{,}5$, la cruza se denomina ***uniforme***. Para estructuras de longitud l la cruza uniforme implica un promedio de $l/2$ puntos de cruza [DEJ/92].

2.3 Métodos de Mutación

La mutación permite mantener diversidad en la población disminuyendo el riesgo de convergencia prematura. A la vez cumple una función de exploración ya que este operador brinda la posibilidad de generar cualquier estructura válida dentro del espacio de búsqueda.

Durante la aplicación del operador de mutación, cada gen es mutado con una probabilidad P_m, generalmente baja. La probabilidad puede ser constante durante toda la búsqueda genética o bien adaptativa. En el primer caso no se comparte información, en cambio en el segundo caso, se utilizan frecuentemente estadísticas de la población para adaptar la velocidad de mutación. Se describen varias implementaciones de este operador.

2.3.1 Mutación Simple

La mutación simple [GOL/89] elige en forma aleatoria un gen, el cual se muta con cierta probabilidad, habitualmente muy baja. La probabilidad de mutación se mantiene constante durante las sucesivas generaciones. Debido a esto, es bastante difícil que la probabilidad elegida sea adecuada en todo momento, y por lo tanto, el operador de mutación no es bien explotado. Por otra parte, es difícil determinar la forma de adaptar el grado de mutación, así que generalmente se utiliza este método debido a su sencillez.

2.3.2 Mutación Adaptativa por Convergencia

En la mutación adaptativa por convergencia [SPE/92], la probabilidad de mutación varía en base a información proveniente de la búsqueda genética. Esta

[1] P_0 se puede ver, también, como la probabilidad de intercambiar alelos [SPE/92].

información está dada por el ***grado de convergencia*** de la población[1]. Esta variante de mutación tiene la ventaja de aprovechar la información histórica para orientar la búsqueda, aumentando la mutación cuando la población se hace muy homogénea y disminuyéndola cuando hay demasiada diversidad. Generalmente, se comienza con probabilidades bajas y se aumenta o disminuye en función de la evolución de la población.

2.3.3 Mutación Adaptativa por Temperatura

El concepto de mutación adaptativa por temperatura deriva del método *Simulated Annealing*. Este tipo de mutación no utiliza información genética de la población, por lo tanto, es independiente de las características de la misma.

La probabilidad de mutación, P_m , depende del tiempo o cantidad de generaciones; por lo tanto $P_m = P_m(t)$. El rango de valores que puede tomar está acotado por valores mínimo y máximo:

$$P_m^{\min} \leq P_m(t) \leq P_m^{\max}$$

Partiendo de un valor inicial P_m (0) , la actualización se realiza hasta alcanzar un valor final, de la siguiente manera:

$$P_m(t+1) = P_m(t) + \lambda$$

Dependiendo del signo de λ y de los valores inicial y final de $P_m(t)$, hay dos variantes del método de mutación adaptativa por temperatura.

2.3.3.1 Mutación Adaptativa por Temperatura Ascendente

La probabilidad de mutación aumenta con las sucesivas generaciones hasta alcanzar el valor máximo y, a partir de allí, mantenerse constante. El objetivo de esta variante es mantener la diversidad de la población, que con el transcurso del tiempo tiende a hacerse homogénea. El aumento de mutación debe tener una cota máxima, de lo contrario, se hace imposible la supervivencia de buenos individuos a lo largo de la evolución y se corre el riesgo de que la población nunca converja. En este caso hay un exceso de exploración del espacio de búsqueda y una falta de explotación del material genético histórico. Por el mismo motivo, es necesario cuidar que esta cota tampoco sea demasiado grande.

Los valores inicial y final para $P_m(t)$ son $P_m^{\min}$ y $P_m^{\max}$ respectivamente, y el parámetro λ es positivo. Tomando $P_m(0) = P_m^{\min}$, se tiene:

$$P_m(t+1) = \begin{cases} P_m(t) + |\lambda| & \text{si } P_m < P_m^{\max} \\ P_m^{\max} & \text{en caso contrario} \end{cases}$$

[1] En la Sección 3.1 se define formalmente este concepto.

2.3.3.2 Mutación Adaptativa por Temperatura Descendente

En este caso se invierte la forma de actualizar la probabilidad de mutación. Se comienza con una mutación alta, la cual va descendiendo con el paso del tiempo. El objetivo es contribuir a una exploración más alta en las primeras generaciones, evitando el problema de convergencia prematura. A medida que el tiempo avanza, la probabilidad de mutación desciende hasta una cota mínima y se estabiliza. Esta cota siempre debe ser mayor a cero, puesto que sin mutación habrá probabilidad de perder para siempre estructuras interesantes del espacio de búsqueda.

En este caso, los valores inicial y final para $P_m(t)$ son $P_m^{\max}$ y $P_m^{\min}$ respectivamente, y el parámetro λ es negativo. Entonces:

$$P_m(t+1) = \begin{cases} P_m(t) - |\lambda| & \text{si } P_m^{\min} < P_m \\ P_m^{\min} & \text{en caso contrario} \end{cases}$$

con $P_m(0) = P_m^{\max}$ como valor inicial.

Capítulo 3

Algoritmos Genéticos en Profundidad

Para que un AG tenga éxito es necesario considerar los siguientes aspectos [HOL/75]:

- La búsqueda se debe prolongar en el tiempo sólo si se continúan realizando avances significativos.
- Durante la búsqueda se deben explotar posibilidades que mejoren la aptitud, de lo contrario, se desperdicia una considerable cantidad de esfuerzo.
- Siempre se puede estar *no explotando* posibilidades que contienen la clave para lograr la aptitud óptima.

De esto, y de lo visto hasta ahora, surgen las siguientes preguntas. ¿Cuál es el principio de funcionamiento de un AG? ¿Qué es lo que procesa un AG? ¿Cómo saber si lo que procesa un AG conducirá a buenos resultados para un problema particular? ¿Cuál es la mejor forma de aprovechar los operadores para lograr el resultado buscado?

El presente capítulo presenta las herramientas teóricas para responder a estas preguntas. En primer término se plantea el análisis de Holland [HOL/75] para un AG simple, que constituye la base para todos los análisis posteriores. Luego se introducen los conceptos más importantes que profundizan este análisis para cada operador.

3.1 Esquemas

Algoritmos Genéticos prescinde de conocimiento propio del problema, entonces sólo se puede trabajar con las estructuras mismas. Si se considera que una estructura está compuesta de atributos, y que cada atributo representa una propiedad determinada de la estructura, basta entonces con comparar atributos ubicados en la misma posición de dos o más estructuras. Esto permite determinar que puntos en común tienen aquellos individuos con aptitud superior a la promedio. La herramienta utilizada para comparar estructuras es el esquema.

> Un ***esquema*** es una plantilla que describe un subconjunto de estructuras con similitudes en determinadas posiciones de atributos [GOL/89].

El concepto de esquema es el que provee una base para asociar combinaciones de atributos con potencial para mejorar la aptitud actual [HOL/75].

Cada estructura posible en el dominio de un problema es, en definitiva, una concatenación de l atributos definidos sobre un alfabeto V dado. Se supone, sin pérdida de generalidad, que este alfabeto es binario, $V = \{0, 1\}$. Extendiendo este alfabeto con un símbolo más que significa "no interesa", * , se puede ahora construir esquemas sobre el alfabeto extendido $V^* = \{0, 1, *\}$. Entonces, se dice que:

> Una estructura pertenece a un esquema dado si la posición i de la estructura y del esquema tienen el mismo valor, 0 ó 1, o bien el esquema tiene el valor *.

Nótese que esta definición implica que una estructura pertenece a más de un esquema, y todo esquema representa un conjunto compuesto por una o más estructuras. Por ejemplo, la estructura 110100 pertenece al esquema *10*00, y también al esquema 11**00, y al ***10*. Como casos particulares, esta estructura pertenece, además, a los esquemas 110100 y ******. El esquema *01* representa el conjunto { 0010, 0011, 1010, 1011 } de estructuras. Por lo tanto, un individuo es instancia de gran cantidad de esquemas. Para estructuras de longitud l se tiene que:

> Una estructura es instancia de 2^l esquemas.

Para una población de tamaño n y estructuras de tamaño l:

> Un AG procesa entre 2^l y $n2^l$ esquemas, dependiendo de la diversidad de la población.

Cuando se utilizan alfabetos de cardinalidad mayor, la longitud l de la estructura decrece, con lo cual la cantidad de esquemas representados también decrece. Debido a esto, muchas veces se prefiere la codificación binaria, porque cada estructura representa la mayor cantidad de esquemas posibles.

> Para todas las estructuras de longitud l, sobre alfabetos de cardinalidad k, hay $(k+1)^l$ esquemas.

Considerando las últimas conclusiones, y para simplificar el análisis, de ahora en más se tendrán en cuenta sólo estructuras binarias de longitud fija.

> La ***aptitud de un esquema*** es el promedio de las aptitudes de todas las instancias posibles del mismo.

La aptitud promedio del esquema se puede estimar promediando la aptitud de las instancias del esquema que pertenecen a la población en una generación dada.

El planteo básico de AG consiste en muestrear a lo largo de sucesivas generaciones distintos esquemas, hasta encontrar uno que exhiba una aptitud superior al promedio. Cuando ocurre esto, muy probablemente sus instancias no muestreadas también tengan mejor aptitud que el promedio de la población. El espacio de búsqueda se debe pensar como compuesto de esquemas y no simplemente de estructuras. La población en cada generación se considera, entonces, como una muestra del espacio de esquemas. Holland plantea el siguiente algoritmo básico [HOL/75]:

1. *Evaluar instancias de varios esquemas hasta que alguno exhiba una aptitud superior al promedio (exploración).*
2. *Generar nuevas instancias del esquema observado, retornando al paso anterior cuando la aptitud observada de dicho esquema y el promedio se acerquen demasiado (explotación).*

El hecho de considerar a cada estructura como una muestra de múltiples esquemas hace que la cantidad de información procesada por un AG sea muy grande. En cada generación se evalúan n individuos, sin embargo se procesan $O(n^3)$ esquemas [GOL/89]. Esta característica de AG se denomina: ***paralelismo implícito***.

Se trata ahora de determinar el efecto provocado por la aplicación de los operadores de selección, cruza y mutación a lo largo de sucesivas generaciones sobre el espacio de esquemas para un problema particular. Para llegar a esto son necesarias algunas definiciones.

La posición j de un esquema H se dice ***definida*** si su alelo es 0 ó 1. Por ejemplo, el esquema 10**0* tiene las posiciones 1, 2 y 5 definidas. En general:

Un esquema se dice ***definido*** sobre un conjunto de posiciones $\{i_1,...,i_h\}$ si en cada posición i_j el alelo no es *.

Entonces, hay 2^h esquemas distintos definidos en todo conjunto de $h \leq l$ posiciones. Para el caso particular de l posiciones, el individuo completo, hay 2^l esquemas definidos posibles.

El ***orden*** de un esquema H, denotado por $o(H)$, es el número de posiciones definidas del esquema.

El orden de un esquema definido sobre h posiciones es el cardinal del mismo conjunto $\{i_1,...,i_h\}$. Por ejemplo, el orden del esquema 1**01* es 3, y el orden de **1*0 es 2.

La ***longitud definida*** de un esquema H, $\delta(H)$, es la distancia entre la primera y última posición definida del individuo.

La longitud definida del esquema **11** es 1, mientras que en el caso de 0*****1 es 6.

Por último, es necesario definir formalmente el concepto de convergencia de una población. Sea $P_{eq}(d)$ la probabilidad de que dos esquemas de la población tengan el mismo alelo en una posición definida d [SPE/92]. Se define:

El ***grado de convergencia de la población***, P_{eq} , es el promedio de $P_{eq}(d)$ sobre todas las posiciones d, para todos los esquemas representados en la población.

Entonces, P_{eq} = 0,5 indica una población diversificada, donde la probabilidad de que cada alelo sea 0 ó 1 es igual, 0,5. Cuando P_{eq} está cercano a 1, la población carece de diversidad y converge. En el caso general de un alfabeto de cardinalidad k se tiene que $P_{eq} \in [\frac{1}{k},1]$.

3.2 Teorema Fundamental

El primer análisis teórico del comportamiento de AG fue desarrollado por Holland [HOL/75] que asume un algoritmo genético simple, dado por los siguientes parámetros:

- estructuras binarias de longitud fija
- selección proporcional por ruleta
- cruza simple
- mutación simple

El Teorema Fundamental es la base de todos los análisis posteriores que consideran otras elecciones de parámetros. El planteo consiste en analizar el efecto aislado de cada operador sobre un esquema, y luego combinar los resultados en uno único, el Teorema Fundamental.

3.2.1 Efecto de la Selección

La selección permite que aquellos esquemas con aptitud superior al promedio sobrevivan y aquellos con aptitud inferior desaparezcan. Sea $A(t)$ la población A en la generación t y $\bar{f}$ la aptitud promedio de $A(t)$. La cantidad de instancias de un esquema H que aparecen en la generación t es:

$$m = m(H,t)$$

Estas m instancias tienen una aptitud promedio $f(H)$, que estima la aptitud real del esquema. Dado que durante la selección una estructura es reproducida un número de veces que es proporcional a su aptitud respecto del promedio de la población, se tiene que:

$$m(H,t+1) = m(H,t)\frac{f(H)}{\bar{f}}$$

Este comportamiento involucra a cada uno de los esquemas representados en la población. Suponiendo que las instancias de un esquema particular H tienen una aptitud promedio superior al de la población y, además, esta diferencia permanece cercana a una constante c a lo largo de sucesivas generaciones [HOL/94], entonces ese esquema exhibirá un crecimiento exponencial a partir de su descubrimiento hasta que ocupe una proporción sustancial de la población, siempre que no haya otros esquemas

que crezcan rápidamente. Debe haber suficientes representantes del esquema H como para proveer una buena estimación de la aptitud del esquema.

Asumiendo que H tiene aptitud promedio observada $f(H) = \bar{f} + c\bar{f}$, con c constante, $m(H, t)$ se puede aproximar como [GOL/89]:

$$m(H,t+1) = m(H,t) \cdot \frac{\bar{f} + c\bar{f}}{\bar{f}} = m(H,t) \cdot (1+c)$$

Comenzando con $t=0$, se tiene:

$$m(H,t) = m(H,0) \cdot (1+c)^t$$

La selección, en definitiva, incrementa (decrementa) en forma exponencial el número de estructuras pertenecientes a aquellos esquemas con aptitud por encima (debajo) del promedio de la población.

La selección por sí sola no explora nuevas regiones del espacio de búsqueda. Solamente explota las similitudes existentes entre individuos de la población, permitiendo la supervivencia y reproducción de los representantes de esquemas más aptos. Con selección sola, un sistema adaptativo eventualmente alcanza un "punto fijo", una generación en la que todos los individuos son iguales, aunque considerando la componente estocástica del proceso, esto puede llevar mucho tiempo.

La selección no produce ***construcción*** de nuevos esquemas, sólo ***supervivencia*** de los mejores.

3.2.2 Efecto de la Cruza

La cruza tiene dos efectos directos sobre una población: la creación de nuevas instancias de esquemas ya presentes en la población, y la generación de nuevos esquemas. Cada intercambio de información entre estructuras afecta a un gran número de esquemas, de aquí el concepto de paralelismo implícito. Basta recordar que cada estructura es instancia de 2^l esquemas.

El efecto de la cruza simple sobre un esquema es sencillo de analizar. Un esquema sobrevive a la cruza si el punto de cruza cae fuera del segmento comprendido entre la primera y la última posición definida. Entonces el esquema sobrevive con mayor probabilidad si la longitud definida es relativamente pequeña[1]. Si este tipo de esquema, además, tiene una aptitud superior a la promedio, aparecerá con mayor frecuencia en la población.

[1] Es necesario recordar que se está analizando la cruza simple. Estudios posteriores al análisis de Holland [HOL/75] muestran que, para la cruza multipunto y uniforme, los esquemas con mayor probabilidad de supervivencia tienen otras características [DEJ/92]. Este tema se analiza en profundidad en la Sección 3.5.

Sea P_s la probabilidad de supervivencia de un esquema a la cruza. Eligiendo un punto de cruza con igual probabilidad entre los l-1 puntos posibles, es posible calcular una cota inferior a la probabilidad P_s para cualquier esquema H. El esquema es destruido sólo si el punto de cruza se elige dentro del segmento de posiciones definidas extremas, esto es $\delta(H)$ posibilidades. Entonces la probabilidad P_d de destrucción de un esquema por efecto de la cruza es:

$$P_d \leq \frac{\delta(H)}{l-1}$$

Además, si la cruza misma se realiza con cierta probabilidad P_c, se tiene que la probabilidad de supervivencia P_s es:

$$P_s \geq 1 - P_c \cdot \frac{\delta(H)}{l-1}$$

Se debe observar que la cruza de dos representantes de un mismo esquema no destruye el esquema mismo, aún cuando la longitud definida sea larga. Por ejemplo, dadas las estructuras 110001 y 011111, representantes del esquema *1***1, la cruza en la posición 4 resulta en 110011 y 011101, que también son instancias de tal esquema.

La cruza simple tiende a ***destruir*** los esquemas de longitud definida grande, y a combinar esquemas de bajo orden para ***construir*** esquemas de orden superior.

3.2.3 Efecto de la Mutación

La mutación por sí sola provee una enumeración aleatoria de estructuras. Introducir una tasa pequeña de mutación en un AG evita perder para siempre alelos que desaparecieron por efectos de la selección o nunca fueron testeados y tal vez sean interesantes.

La mutación asegura que ningún alelo se pierde irremediablemente de la población.

Dado que el operador de mutación altera el valor de cada posición con probabilidad P_m, un esquema H sobrevive si cada una de sus $o(H)$ posiciones definidas sobrevive. Si una posición dada sobrevive con probabilidad 1-P_m, y cada mutación es estadísticamente independiente, se tiene que H sobrevive a la mutación con probabilidad $(1-P_m)^{o(H)}$. Cuando P_m es suficientemente pequeño ($P_m << 1$), se puede aproximar la probabilidad de supervivencia por la expresión:

$$1 - o(H) \cdot P_m$$

3.2.4 Teorema Fundamental de los Algoritmos Genéticos

El efecto combinado de la selección, cruza y mutación sobre un esquema particular H está dado por:

$$m(H,t+1) \geq m(H,t) \cdot \frac{f(H)}{\bar{f}} \left[1 - P_c \frac{\delta(H)}{l-1} - o(H) P_m \right]$$

A este resultado, si bien no es un teorema, se lo conoce por su importancia como ***Teorema Fundamental de los Algoritmos Genéticos***. Este resalta la importancia de determinado tipo de esquemas, que permiten la construcción de estructuras cada vez más aptas, a partir de aquellas logradas en el pasado.

Los esquemas de ***bajo orden***, ***corta longitud definida*** y ***aptitud superior al promedio***, tienen un crecimiento exponencial en cuanto a la cantidad de representantes dentro de la población [HOL/75].

A partir del Teorema se plantea una alternativa distinta a la tradicional para la búsqueda de solución a un problema. En lugar de intentar encontrar una muy buena solución testeando un subconjunto de todas las posibles, lo que se hace es construir cada vez mejores soluciones parciales explotando aquellas obtenidas en el pasado y explorando nuevas. Estas soluciones parciales están representadas por los esquemas de bajo orden, corta longitud definida y aptitud superior a la promedio. Los esquemas con estas características se denominan ***bloques constituyentes***. AG consta, desde este punto de vista, de dos fases simultáneas que deben ser balanceadas. Una etapa de ***exploración*** de conocimiento, y otra de ***explotación*** de ese conocimiento adquirido.

El análisis de Holland se basa en el AG simple, sin embargo, se han planteado múltiples variantes para cada uno de los operadores. Esto ha llevado a posteriores análisis que intentan formalizar dichos métodos y justificarlos. En lo que resta de este capítulo se resumen los conceptos más importantes estudiados sobre cada operador.

3.3 Mecanismos de Selección

La selección es el operador que favorece el crecimiento de los mejores esquemas, y la desaparición de aquellos poco aptos. Para esto, un buen mecanismo de selección debe asignar un número adecuado de copias a cada miembro de la población en las sucesivas generaciones. Esto depende del método de selección y de la función de aptitud.

Para una función de aptitud dada, distintos métodos de selección pueden seleccionar con mayor o menor intensidad a los mejores individuos de la población. La ***presión de la selección*** es el grado por el cual se favorece a los mejores individuos [MIL/95]. Esta es una característica de todo método de selección. Con una alta presión, los individuos más aptos serán mucho más favorecidos, y la aptitud de la población mejorará en sucesivas generaciones. Pero si la presión es demasiado alta,

sobre todo en las primeras generaciones, se corre el riesgo de que el algoritmo genético converja prematuramente hacia óptimos locales. Por el contrario, una presión demasiado baja tiende a elegir individuos en forma casi aleatoria, con individuos pobres compitiendo casi en igualdad de condiciones con buenos individuos. En este caso los individuos poco aptos podrán sobrevivir por un largo período de tiempo. Obviamente, es necesario lograr un equilibrio entre ambos extremos.

A diferencia del método de selección, la función de aptitud depende del problema a resolver. Sin embargo, se debe tener en cuenta algunos conceptos cuando se define dicha función puesto que ejerce una influencia decisiva en la efectividad del operador de selección.

3.3.1 Función de Aptitud

En el Capítulo 1 se presentó la función de aptitud f como una métrica necesaria para la aplicación del operador de selección. Habitualmente se pretende que la función f sea positiva y que el AG evolucione maximizando dicha función. Esto se debe a que es más intuitivo hablar del mejor individuo cuando tiene la mayor aptitud posible, y no la menor.

Desde el punto de vista del problema a resolver, se dispone de una función a optimizar, g, denominada ***función objetivo***. Muchas veces se define $f = g$ por simplicidad. Sin embargo, la función objetivo no necesariamente es adecuada como función de aptitud. Esto se debe a que puede requerir ser minimizada, o toma valores negativos, o impide una adecuada presión de la selección, o simplemente no conviene dado el método de selección a utilizar.

Es necesario realizar, entonces, un mapeo de la función objetivo g a la función de aptitud f, de tal forma que f cumpla los requisitos pedidos.

3.3.1.1 Construcción de la Función de Aptitud

Para transformar el problema de minimización de la función objetivo g a uno de maximización de la función f basta con definir:

$$f(x) = g_{\max} - g(x)$$

con $g_{\max}$ constante y $g_{\max} \geq g(x)$ para toda estructura x. Esta transformación garantiza que $f(x)$ es positiva siempre que se conozca el valor $g_{\max}$ a priori, lo cual no es habitual. Hay dos alternativas para solucionar este problema. La primera consiste en definir:

$$f(x) = \begin{cases} g_{\max} - g(x) & \text{si } g_{\max} \geq g(x) \\ 0 & \text{en caso contrario} \end{cases}$$

Esta tiene la desventaja de que demasiados valores de $f(x)$ pueden hacerse cero, si no se utiliza un valor $g_{\max}$ adecuado. Por esto, otra alternativa es utilizar la siguiente definición:

$$f(x) = g_W - g(x)$$

donde g_W es el máximo valor de g logrado durante las últimas W generaciones [GRE/90]. El parámetro W se denomina ***ventana de escalado*** y permite controlar la actualización del valor base g_W.

3.3.1.2 Necesidad de Escalado

El objetivo del escalado es regular la presión de la selección. Los métodos de selección proporcional presentan, habitualmente, dos problemas. El primero consiste en la posible existencia y predominio de unos pocos individuos muy buenos en las primeras generaciones, ***convergencia prematura***. Esto ocurre cuando la aptitud máxima está muy alejada de la aptitud promedio, y la población converge con una exploración todavía demasiado pobre del espacio de búsqueda. En este caso, la presión de la selección es excesiva.

El segundo problema puede aparecer más tarde en la evolución, cuando la aptitud promedio de la población está muy cercana a la aptitud máxima. Bajo esta condición, la mayoría de los integrantes de la población compiten en igualdad de condiciones con los mejores, y la evolución se asemeja a una ***búsqueda aleatoria***. La función de aptitud no puede distinguir suficientemente los mejores individuos del resto, y la presión de la selección tiende a ser escasa.

Hay varias soluciones a ambos problemas: la utilización de métodos de selección por ranking, mutación adaptativa, o la más aceptada: ***escalado de la función de aptitud***.

Existen diversos métodos de escalado. El más usual es el escalado lineal [GOL/89], que consiste en un mapeo lineal de la función original f a una función escalada u , tal que:

$$u = af + b$$

Los valores a y b se pueden elegir de diversas maneras. En general, dada una población se calcula su aptitud máxima y promedio, y se define:

$$u_{avg} = f_{avg}$$
$$u_{\max} = c \cdot f_{\max} \qquad c > 1$$

Luego se resuelve el siguiente sistema de ecuaciones:

$$u_{avg} = a \cdot f_{avg} + b$$
$$u_{\max} = a \cdot f_{\max} + b$$

De este modo se logra que la cantidad esperada de hijos para los individuos con aptitud promedio no se altere, y se ajusta la distancia entre el mejor individuo y el promedio mediante un ***factor de escalado*** c.

3.3.1.3 Manejo de Restricciones

Muchas veces el problema a resolver está sujeto a una serie de restricciones. Estas se reflejan en subespacios de estructuras que representan soluciones inválidas. Es necesario, entonces, un mecanismo que permita a la aptitud distinguir las soluciones válidas de aquellas que no lo son. Existen diversas formas de enfrentar el problema.

- **Discriminación**

La primera alternativa consiste en asignar aptitud cero a aquellas estructuras que violan alguna restricción [GOL/89]. Este mecanismo funciona bien excepto cuando existe una cantidad excesiva de individuos inválidos. En este caso, se puede hacer tan dificultoso encontrar individuos válidos como encontrar el mejor.

- **Penalización**

La segunda alternativa consiste en incorporar a la función de aptitud f un término de penalización [GOL/89]. Este reduce la aptitud del individuo según la importancia de la restricción que violó. La aptitud penalizada f' tendría la siguiente forma:

$$f' = f(x) - r\sum_{i=1}^{n} \theta[h_i(x)]$$

donde θ es la función de penalización, r un coeficiente de penalización, y $h_i(x)$ el conjunto de restricciones a verificar.

- **Restricción del Espacio de Búsqueda**

La tercera alternativa consiste en restringir el espacio de búsqueda a aquellos individuos que cumplan con las restricciones. Esto es más dificultoso, puesto que no sólo hay que generar una población de individuos válida sino que, también, hay que evitar que la mutación y la cruza introduzcan individuos fuera del espacio restringido. Para alguna clase de restricciones esto puede ser bastante costoso.

3.4 Mecanismos de Cruza

Los individuos que sobreviven a la etapa de selección se cruzan para crear nuevos individuos. La característica principal de este operador es que preserva los alelos comunes a los individuos que intervienen en una aplicación del mismo. Esto limita el poder exploratorio de la cruza, a cambio de aumentar la explotación de los esquemas dominantes en la población que, por efecto de la selección, son los que exhiben mejor aptitud.

Para conocer las características principales de los distintos métodos de cruza, interesa estudiar la misma desde el punto de vista de la preservación de los esquemas obtenidos y de la construcción de nuevos esquemas. En esta sección se analizan la cruza multipunto[1] y la cruza binomial a la luz de dos parámetros: disrupción y construcción [DEJ/92] [SPE/92].

La ***disrupción*** se produce cuando un esquema dado, presente en uno de los individuos que participa de una cruza, no está presente en ninguno de los hijos resultantes de la misma.

La ***construcción*** es la capacidad de crear un nuevo esquema.

Se estudiará, en particular, cómo a partir de dos individuos pertenecientes a esquemas no solapados se puede construir un esquema de orden superior incluido en ambos.

3.4.1 Disrupción

Si se compara la probabilidad de disrupción de dos esquemas de igual orden pero distinta longitud definida, es fácil ver que sobrevivirá con mayor probabilidad el de menor longitud definida cuando se utiliza cruza simple. Esta es una de las conclusiones del Teorema Fundamental, e implica que este operador sufre de ***sesgo de representación***, ya que el grado de disrupción depende de las posiciones de los genes que conforman el esquema. Esta característica también afecta el caso general de cruza multipunto.

Para que un esquema se pierda en una operación de cruza es necesario que ocurran los siguientes dos eventos:

A. La distribución de los puntos de cruza en el cromosoma sea tal que no todas las posiciones definidas del esquema vayan al mismo hijo.

B. El otro padre no provea las posiciones definidas que faltan para completar el esquema.

La probabilidad de disrupción de un esquema H, P_d, se puede expresar en base a la probabilidad de ocurrencia de los eventos A y B:

$$P_d(H) = P(A) \cdot P(B)$$

Por ejemplo: sea H un esquema de orden $o(H) = 2$, con posiciones definidas i y j. Suponiendo que este esquema está presente en el cromosoma X, se quiere calcular la probabilidad de supervivencia del mismo, $P_s = 1 - P_d$, luego de la cruza con el cromosoma Y.

[1] La cruza multipunto es suficientemente genérica, dado que la mayoría de las variantes de cruza son casos particulares de esta.

En el caso de la cruza simple, el evento A ocurre cuando el punto de cruza cae entre las posiciones i y j[1]. Entonces:

$$P(A) = \frac{\delta(H)}{l}$$

El evento B ocurre cuando ninguna de las dos posiciones definidas aparece en el cromosoma Y. Entonces:

$$P(B) = P(X_i \neq Y_i) \cdot P(X_j \neq Y_j)$$

La probabilidad del evento B está dada por el grado de convergencia de la población [SPE/92]. Cuanto mayor es la convergencia, más improbable es que la cruza destruya los esquemas existentes debido a que éstos tendrán una creciente cantidad de representantes. Este fenómeno hace que aumente la probabilidad de cruzar representantes de un mismo esquema entre sí, lo cual no provoca disrupción porque no cumplen el evento B.

El gráfico de la [Figura 3-1] muestra la probabilidad de supervivencia P_s a la cruza para un esquema de orden 2, dados su longitud definida y la cantidad de puntos de cruza. Se asume cromosomas de longitud $l = 10$ y una población diversificada con $P_{eq} = 0{,}5$.

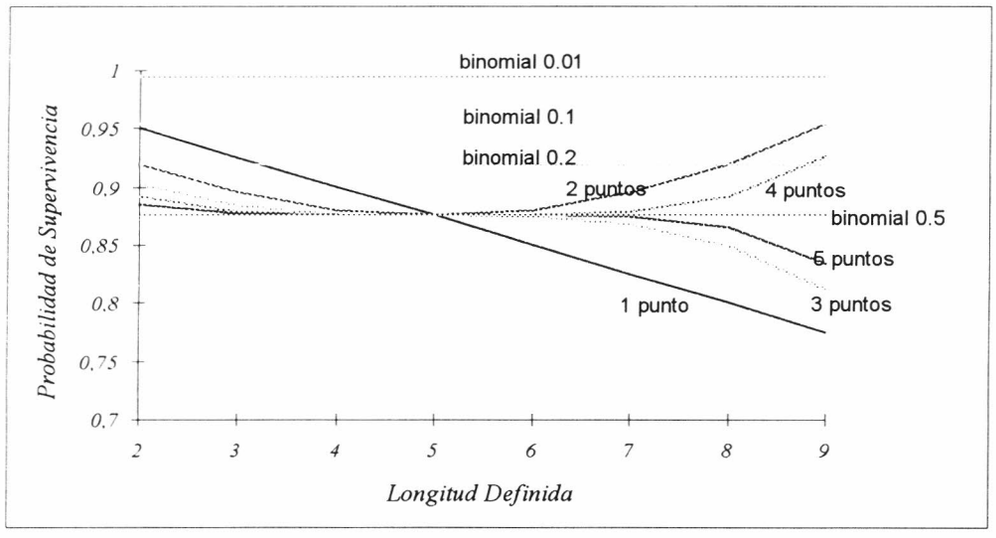

Figura 3-1. Probabilidad de supervivencia para esquemas de orden 2

La cruza multipunto es sesgada respecto a la representación de los individuos. Como se observa en la [Figura 3-1] el tipo de esquema privilegiado con una mayor probabilidad de supervivencia difiere según la cantidad de puntos de cruza.

La cruza de dos puntos no es tan sesgada como la cruza simple y no penaliza los esquemas de alta longitud definida. Similar característica presentan las variantes

[1] Se asume que la posición 0 tambien es punto de cruza.

que tienen una cantidad de puntos par, siendo un argumento válido a favor de éstos métodos de cruza. Cuando la cruza opera con una cantidad impar de puntos, la probabilidad de supervivencia decrece rápidamente a medida que crece la longitud definida del esquema.

Al contrario de la cruza multipunto, la cruza binomial no presenta sesgo de representación. Para una probabilidad P_0, de cruza de genes dada, la probabilidad de supervivencia, P_s, permanece constante, más allá de la longitud definida.

El sesgo de representación se observa claramente cuando se aumenta la longitud del cromosoma agregando genes que no tengan influencia en la función de aptitud. La probabilidad de supervivencia de los esquemas bajo la cruza multipunto aumenta pero la misma no varía bajo cruza binomial [DEJ/92].

El sesgo de representación no necesariamente es perjudicial. Es más, una codificación inteligente de los cromosomas, que tome en cuenta las correlaciones conocidas entre los genes, puede ser aprovechada por AG para acelerar la búsqueda. Generalmente se busca elegir un operador de cruza y una codificación que privilegie la supervivencia de esquemas formados por genes altamente correlacionados, de este modo se induce, con conocimiento del problema, a la formación de bloques constituyentes funcionales al mismo. El sesgo de representación puede aprovecharse incluso si se desconocen las correlaciones entre genes. Para ello se utilizan operadores auxiliares como el operador de inversión[1].

Resumiendo, los factores que influyen en la disrupción de un esquema bajo la cruza son:

- Orden del esquema
- Grado de convergencia de la población
- Cantidad de puntos de cruza (cruza multipunto)
- Longitud definida (cruza multipunto)
- Probabilidad de intercambio de genes, P_0 (cruza binomial)

3.4.2 Construcción

La cruza permite construir nuevos esquemas a partir de esquemas más simples contenidos en la generación anterior. Se estudiará, en particular, cómo a partir de dos individuos pertenecientes a esquemas no solapados de orden m y n respectivamente se puede construir un esquema H, de orden superior $k = m + n$ incluido en ambos.

Para constuir el esquema de orden k con las características mencionadas es necesario que los subesquemas de orden m y n sobrevivan en el mismo individuo. Esta situación se dá si ocurre alguno de los siguientes eventos:

[1] Ver Sección 4.1.2.

A'. Los puntos de cruza cortan el cromosoma de modo que se intercambian todas las posiciones definidas de uno de los subesquemas y ninguna del otro.
B'. Los puntos de cruza no cortan el cromosoma de ese modo (B_1'), pero todas las posiciones definidas que quedan en el individuo X después de la cruza pertenecen a H o bien todas las posiciones definidas que quedan en el individuo Y después de la cruza pertenecen a H (B_2').

Por ejemplo, se pretende construir el esquema H= 111000 a partir de los subesquemas 111*** y ***000, mediante la cruza de los cromosomas X= 111110 y Y= 001000. El evento A' ocurre cuando el punto de cruza cae en la tercera posición:

$$\begin{array}{ll} \text{X= } \mathbf{111}|\mathbf{110} & \text{X= } \mathbf{111}|000 \\ & \rightarrow \\ \text{Y= } 001|000 & \text{Y= } 001|\mathbf{110} \end{array}$$

Siguiendo con el ejemplo, si los puntos de cruza caen en las posiciones 2 y 5 se está ante el evento B':

$$\begin{array}{ll} \text{X= } \mathbf{11}|\mathbf{111}|\mathbf{0} & \text{X= } \mathbf{11}|100|\mathbf{0} \\ & \rightarrow \\ \text{Y= } 00|100|0 & \text{Y= } 00|\mathbf{111}|0 \end{array}$$

Para el caso de cruza simple, $m = n = 1$ la probabilidad de construcción de H es:

$$P_{con}(H) = P(A') + P(B')$$

$$P(A') = \frac{\delta(H)}{l}$$

$$P(B') = P(B_1') \cdot P(B_2')$$

$$P(B_1') = \frac{l - \delta(H)}{l}$$

$$P(B_2') = P(X_i = Y_i) + P(X_j = Y_j) - P(X_i = Y_i \text{ y } X_j = Y_j)$$

El gráfico de la [Figura 3-2] muestra la probabilidad de construcción P_{con} de esquemas de orden 2 a partir de dos esquemas de orden 1 a través de la cruza, con longitud definida l= 10 y una población diversificada P_{eq}= 0,5 .

Hay una relación directa entre poder disruptivo y constructivo de los operadores: aquellos que son más disruptivos también son los más constructivos. El sesgo de representación se sigue observando en la cruza multipunto y los factores que influyen en la construcción de un esquema son los mismos que influyen en la disrupción.

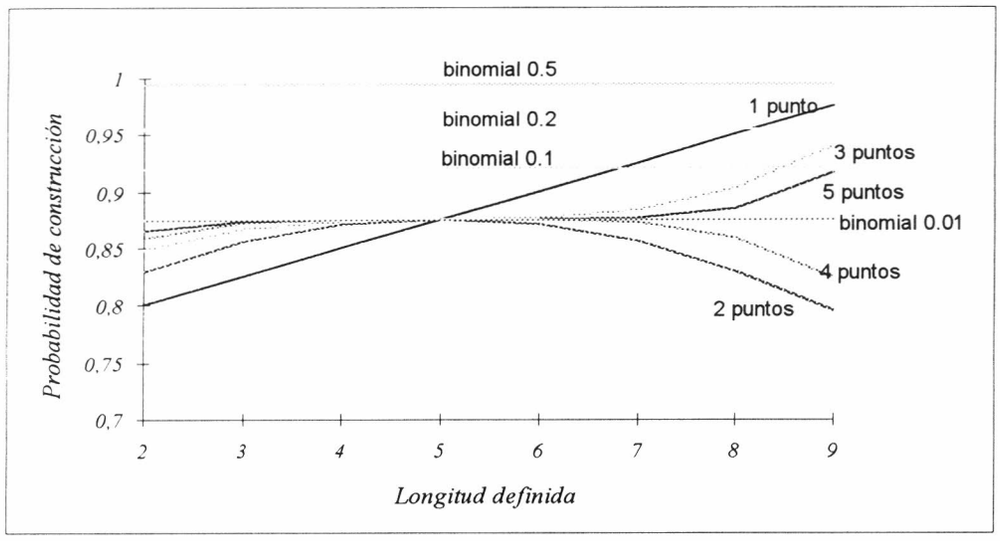

Figura 3-2. Probabilidad de construcción para esquemas de orden 2

Es importante que el operador de cruza no sea sesgado contra los bloques constituyentes naturales del problema. Es decir, que no tienda a destruirlos. Se trata de regular el poder disruptivo-constructivo de la cruza mediante la elección adecuada de un método de cruza, representación del cromosoma, y de sus parámetros. No es conveniente que la disrupción sea muy alta porque los buenos esquemas obtenidos no se mantienen, ni que sea muy baja porque se produce estancamiento (no hay suficiente construcción de nuevos esquemas). Entonces, se debe buscar un equilibrio entre exploración y explotación.

3.5 Mecanismos de Mutación

De igual forma que para el caso de la cruza, en esta sección se analiza la mutación simple observando cómo provoca disrupción y construcción de esquemas.

3.5.1 Disrupción

Del Teorema Fundamental se conoce que, suponiendo P_m la probabilidad de mutación y H un esquema de orden $o(H) = k$, la probabilidad de supervivencia de un esquema al operador de mutación es la probabilidad de que ninguna de sus posiciones definidas sea mutada:

$$P_s(H) = (1 - P_m)^k$$

Como se puede observar, el efecto disruptivo de la mutación es independiente del tiempo; no depende del grado de convergencia de la población.

La mutación puede proveer los mismos niveles de disrupción que la cruza binomial. Basta con que la tasa de mutación sea una función de P_{eq}, k y P_0. Por el contrario, la cruza no puede proveer los altos niveles de disrupción de la mutación. El poder exploratorio de la cruza es más limitado ya que preserva los alelos comunes a los individuos.

3.5.2 Construcción

Sean H_i y H_j dos esquemas de orden k, tal que ambos esquemas coinciden en c alelos y difieren en d alelos. La probabilidad P_{con} de construir un individuo perteneciente a H_i a partir de un individuo que pertenece al esquema H_j es:

$$P_{con}(H_i, H_j) = (1 - P_m)^c (P_m)^d$$

Esta es la probabilidad de que la mutación no afecte a los alelos que sí coinciden y sí afecte a los alelos en los que no coinciden.

Cuando la población es diversa, la mutación no puede alcanzar los niveles de construcción de la cruza. Esta ventaja se incrementa cuanto mayor es el orden del esquema que se desea construir. La mutación no puede alcanzar simultáneamente altos niveles de construcción y supervivencia.

3.6 Conclusión

El Teorema Fundamental describe la evolución de los mejores esquemas, cuando un AG simple converge hacia la solución buscada. Dada la gran variedad existente para cada operador, y para evitar extender este análisis a cada alternativa posible, es más conveniente describir las características comunes, deseables para cada método de selección, cruza y mutación.

Los métodos de selección se describen en función de la presión que ejercen sobre una población. Esto está fuertemente ligado a las características de la función de aptitud. La adecuada elección del método, junto con la construcción de la función de aptitud, son determinantes para evitar la convergencia prematura y favorecer la selección de los mejores individuos.

La cruza provee exploración a la vez que explotación del conocimiento obtenido, mientras que la mutación es más explorativa. Si se trata de maximizar los beneficios acumulados a lo largo de la evolución, el peso de la cruza es decisivo. Esta sirve como acelerador de los avances logrados por la población al combinar esquemas exitosos de distintos individuos. La mutación atenúa un poco la homogeneización de la población a la que conduce la cruza creando diversidad en la población. En definitiva, es necesario conocer las capacidades de exploración-explotación y disrupción-construcción de cada operador, para poder decidir, frente a un problema determinado, la mejor opción.

Capítulo 4

Resolución de problemas con AG

El objetivo de este capítulo es proponer una metodología de diseño de un AG para resolver un problema arbitrario. Para esto, se analizan dos aspectos claves para el éxito del algoritmo: lograr una representación apropiada para los cromosomas, y definir una función de aptitud con características que faciliten la búsqueda. Estos aspectos son los únicos estrechamente relacionados con el problema en sí, puesto que una vez definidos ambos parámetros, el AG es absolutamente independiente del problema.

El capítulo se divide en tres partes. La primera parte trata sobre el problema de la representación de cromosomas, la segunda sobre la elección de la función de aptitud, y la tercera plantea el método de diseño propuesto.

4.1 El problema de la Representación

Para la resolución de problemas mediante AG es crítico realizar una adecuada elección de la semántica de una estructura. Esto es, cómo representar una solución al problema en un individuo que pueda ser procesado por un AG. Los factores a tener en cuenta para la representación de los individuos son dos:

- La codificación de cada parámetro en un gen.
- La distribución de los genes dentro del cromosoma.

Ambos factores influyen en el tamaño del espacio de búsqueda, y en la construcción de bloques constituyentes funcionales al AG.

4.1.1 Codificación de Parámetros

Algoritmos Genéticos no presenta restricciones en cuanto a la forma que deben adoptar los genes. Estos pueden representarse como cadenas binarias, números enteros o reales, o estructuras más complejas. Se debe tener en cuenta que la representación debe elegirse conjuntamente con operadores genéticos adecuados para manejarla.

Una buena codificación es aquella que permite que los operadores genéticos exploten adecuadamente las similitudes entre individuos. Y estas, a su vez, deben reflejarse como similitudes en la codificación. A continuación se describen los métodos básicos de codificación.

4.1.1.1 Parámetros Binarios

La mayor parte de los fundamentos teóricos de AG se desarrolló suponiendo que las estructuras son cadenas binarias. Esto hace que la codificación binaria sea la elección por excelencia, y muchas veces se prefiera llevar los parámetros a una forma binaria por simplicidad, o para contar con herramientas adecuadas para analizar el comportamiento del algoritmo.

Para problemas que tienen parámetros de dos estados posibles: Si/No, Alto/Bajo, Grande/Chico, la codificación binaria es la elección natural. La codificación de estos parámetros no ofrece dificultad. Se codifica uno de los estados como 1 y el otro como 0.

4.1.1.2 Parámetros No Binarios

La mayoría de los problemas presentan parámetros que pueden tomar más de dos valores. En particular, los números enteros y los números reales se utilizan en muchos problemas de interés. Debido a esto, se los trata a cada uno por separado, si bien es claro que pertenecen a la categoría de parámetros no binarios.

Para este tipo de problemas se puede optar entre una codificación en un alfabeto no binario, o bien un mapeo a un alfabeto binario. Los partidarios de los alfabetos binarios sostienen que los mismos permiten explorar mayor cantidad de esquemas en cada evaluación[1]. Sin embargo, este aprovechamiento se da en el caso de una codificación binaria racional, que codifique con subcadenas binarias iguales aquellas letras del alfabeto que representan elementos con características comunes relevantes para el problema. Por ejemplo, si hubiera que codificar los elementos químicos de la Tabla Periódica de los Elementos puede plantearse dos alternativas: la primera consiste en codificar cada elemento con una letra distinta del alfabeto, y la segunda consiste en definir una codificación binaria que exprese las similitudes entre los elementos.

4.1.1.3 Números Enteros

Como se mencionó anteriormente, los números enteros son un caso particular de los parámetros no binarios. Entonces, pueden codificarse como números enteros o bien mapearse a una representación binaria. La conversión a un alfabeto binario presenta características particulares que lo distinguen. Estas son estudiadas en las secciones siguientes.

Para medir la similitud entre dos estructuras binarias se utiliza la distancia de Hamming. Esta se define de la siguiente manera:

La ***distancia Hamming*** entre dos cadenas binarias es la cantidad de posiciones en que ambas difieren.

[1] Ver Sección 3.1.

- **Codificación Binaria de Parámetros Enteros**

La búsqueda que realiza un AG se basa en la explotación de las similitudes de los individuos. En ese sentido, la codificación binaria presenta el problema de que la distancia de Hamming entre dos números no siempre es una buena medida de la distancia real entre los mismos. Esto es porque la distancia de Hamming entre dos números adyacentes no es constante para todo par de números. A este problema se lo denomina ***Hamming cliffs***. En la [Figura 4-1] se observa la distancia de Hamming entre el número *i* y el *i*-1 para los primeros 15 números enteros codificados en forma binaria.

Suponiendo parámetros enteros en el rango 0 a 255, mapeados en el rango binario [0,11111111], el número 127 se codifica como 01111111 y el 128 como 10000000. Ambos números son consecutivos, "similares", y sin embargo sus codificaciones están a distancia Hamming 8. Si 01111111 es el alelo dominante para determinado gen en la población pero el verdadero óptimo es 10000000, será muy difícil que la población logre evolucionar hacia él.

En general las discontinuidades de la representación binaria no presentan mayores problemas y sólo son relevantes en lo que hace a los bits más significativos.

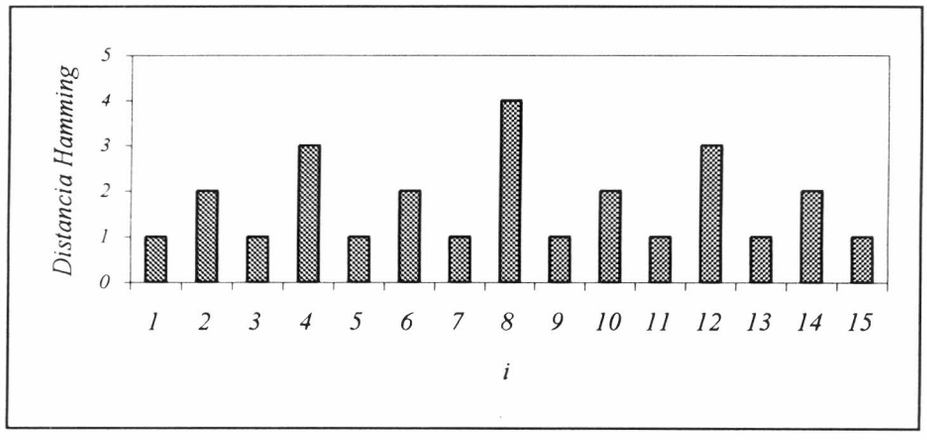

Figura 4-1. Distancia Hamming entre el número *i* y el *i*-1.

- **Codificación Gray**

La ***codificación Gray*** es una solución posible para el problema de Hamming cliffs, mencionado en el parágrafo anterior. Esta codificación tiene la propiedad de que enteros adyacentes son representados por estructuras binarias que difieren en un solo bit.

La codificación Gray de un número binario se realiza de la siguiente manera. El primer bit (el más significativo) se define idéntico al del número que se está codificando. El siguiente bit se define 0 si el correspondiente bit del número original no cambió respecto del anterior; de lo contrario se define 1. Se prosigue así hasta codificar todos los bits. En la [Tabla 4-1] se compara la codificación binaria con la Gray para los primeros 16 números naturales [GOL/89].

i	*Binario*	*Gray*
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Tabla 4-1. Codificación Gray

En la [Tabla 4-1] se observa que lo que se codifica son las diferencias con el bit anterior. Si bien esta codificación cumple con la propiedad de que enteros adyacentes son representados por estructuras que difieren en un sólo bit, la recíproca no es cierta. Es decir que estructuras Gray que difieren en un único bit, no necesariamente representan enteros adyacentes. Esto hace dudosos los beneficios de esta codificación. Por ejemplo, en la [Tabla 4-1] se observa que 1000 y 0000 tienen distancia de Hamming de uno y sin embargo representan al 15 y al 0 respectivamente.

4.1.1.4 Números Reales

A diferencia de los números enteros, los números reales se representan en una recta numérica continua. Esto es un problema, puesto que sólo un subconjunto de todos los números posibles pueden representarse computacionalmente. El mismo problema se presenta siempre que el dominio de algún parámetro es infinito o excesivamente grande. La solución consiste en acotar el intervalo de interés, muestrearlo con una cierta precisión o granularidad, y representar este conjunto discreto de puntos con una codificación entera.

Si se utilizan b bits para representar un parámetro x, tal que $x \in [c_{\min}, c_{\max}]$, entonces la ***granularidad*** está dada por la distancia h entre dos muestras consecutivas de x.

$$h = \frac{|c_{\max} - c_{\min}|}{2^b}$$

Entonces, los valores x que puede tomar el parámetro discretizado x' son:

$$x' \in \{c_{\min}, c_{\min} + h, c_{\min} + 2h, \ldots, c_{\min} + 2^b h\}$$

De la definición de h se desprende que $h \to 0$ cuando $b \to \infty$. Por lo tanto, aumentar la cantidad b de bits para representar al parámetro x permite que la precisión aumente y que x' pueda tomar un mayor número de valores.

El problema más importante que se debe resolver para muestrear números reales consiste en determinar la precisión adecuada. Utilizar una precisión demasiado baja (h demasiado grande) puede llevar al problema de ***aliasing*** [GOL/89]: se pueden perder puntos relevantes causando que el AG no pueda distinguir entre diferentes picos de la superficie de búsqueda. Esto habitualmente ocurre cuando la función a tratar tiene picos y valles muy abruptos, por lo cual requiere ser tratada con extrema precisión y se debe aumentar la cantidad b de bits utilizada.

Se debe utilizar el conocimiento del problema para encontrar el menor conjunto de puntos que incluya la solución. A menudo no es necesario que exactamente el óptimo esté en el conjunto. Basta con que haya puntos cercanos al óptimo lo suficientemente próximos como para orientar al AG en su búsqueda.

Si se presume que la función objetivo requiere máxima precisión por ser altamente variable, se deberá aumentar la precisión de la codificación. Sin embargo, no tiene sentido aumentar la precisión más allá de cierto límite, porque de todos modos sólo un subconjunto de estos puntos será efectivamente visitado. A lo sumo se visitarán $(p \cdot g)$ puntos; siendo p el tamaño de la población y g la cantidad máxima de generaciones que la población evoluciona.

Una posible solución al problema de aliasing consiste en utilizar ***Codificación Dinámica de Parámetros*** [FOG/94]. Este método consiste en reducir dinámicamente el intervalo de cada parámetro, $|c_{\max} - c_{\min}|$, y aumentar en forma simultánea su precisión. De este modo, AG realiza primero una búsqueda global de las zonas más prometedoras y luego profundiza su investigación en cada una.

4.1.2 Distribución de los genes dentro del cromosoma

El segundo factor a tener en cuenta para la representación de los cromosomas es la distribución de los genes dentro del mismo. Como la cruza multipunto con pocos puntos de cruza genera bloques consitituyentes de corta longitud definida, es conveniente que los genes que estén correlacionados entre sí se ubiquen en posiciones adyacentes dentro del cromosoma. Si no se conoce el problema lo suficiente como para determinar esta correlación, se puede utilizar el ***operador de inversión*** [HOL/75].

El operador de inversión realiza permutaciones de un mismo cromosoma. Los genes pueden cambiar de posición dentro del cromosoma, para lo cual es necesario redefinir su estructura de tal forma que ésto refleje la posición de cada gen. Para esto, cada gen es un par ordenado compuesto por el alelo y su ubicación. Por ejemplo, el cromosoma genérico de longitud 5:

$$(a_1, a_2, a_3, a_4, a_5)$$

con este nuevo concepto pasa a ser:

$$((1, a_1), (2, a_2), (3, a_3), (4, a_4), (5, a_5))$$

El símbolo "no interesa" - * - no tiene asociado una posición, ya que ocupa todas las posiciones no definidas. Por ejemplo, los siguientes esquemas son equivalentes: ((1, X), *, *, (4, Y)) y (*, *, (4, Y), (1, X)).

Para aplicar el operador de inversión se elige un cromosoma al azar, y se determinan dos puntos de corte, x_1 y x_2 tal que $x_1 < x_2$. La nueva estructura se genera invirtiendo el segmento que comienza a la derecha de la posición x_1 y termina a la izquierda de la posición x_2.

Sea el cromosoma $A = A_1 \ldots A_l$ de longitud l, donde $A_i = (i, a_i)$:

$$(A_1, \ldots, A_j, A_{j+1}, A_{j+2}, \ldots\ldots\ldots, A_{k-2}, A_{k-1}, A_k, \ldots, A_l)$$

y eligiendo los puntos de corte j y k , con $j < k$, se tiene el siguiente resultado luego de aplicar el operador de inversión:

$$(A_1, \ldots, A_j, A_{k-1}, A_{k-2}, \ldots\ldots\ldots, A_{j+2}, A_{j+1}, A_k, \ldots, A_l)$$

El operador de inversión puede acercar genes que previamente estaban muy alejados; por ejemplo: A_j y A_{k-1}, o A_{j+1} y A_k . También produce el efecto contrario, alejar genes que estaban muy cercanos entre sí. Como propiedad se puede mencionar que:

Cualquier posible permutación del cromosoma puede ser producida por una secuencia apropiada de inversiones.

Un cromosoma afectado por el operador de inversión no altera su significado, dado que cada gen es independiente de su posición y conserva su alelo. El operador preserva la posición lógica del gen dentro del cromosoma pero altera su distribución física. El efecto de la inversión consiste en generar permutaciones de los esquemas ya existentes, con distintas longitudes definidas. Como consecuencia de esto, la representación evoluciona de generación en generación hacia un estado que permite aprovechar el sesgo de la cruza.

4.2 Elección de la Función de Aptitud

En los últimos años se han investigado las características de los problemas que son de difícil resolución a través de AG. Actualmente se puede identificar con bastante certeza aquellos problemas extremadamente difíciles para AG, con lo cual es posible restringir la aplicación de éstos a problemas de dificultad acotada. Se discutirán algunas características habitualmente consideradas críticas de la superficie de aptitud que pueden influir considerablemente en la eficiencia de un AG.

Un problema es dificultoso para AG cuando la superficie de aptitud presenta alguna de las siguientes características [GOL/93]:

- ***Aislamiento***: la solución deseada está aislada y su cuenca de atracción es muy pequeña.

- ***Decepción***: el AG es atraído por subóptimos locales y alejado del óptimo global.

- ***Multimodalidad***: existencia de más de un óptimo local.

- ***Ruido***: la función objetivo posee pequeños errores respecto de la función que se quiere optimizar.

Una elección adecuada de la función de aptitud a menudo elimina la presencia de algunas de estas características. La mayor parte de las investigaciones se han centrado en los problemas que combinan aislamiento y decepción.

El problema de ***aislamiento*** puede ser crítico dependiendo de la cuenca de atracción del óptimo global. Si la cuenca es excesivamente pequeña, el problema tiende a convertirse en "hallar una aguja en un pajar". Un caso extremo de este tipo se observa en la [Figura 4-2].

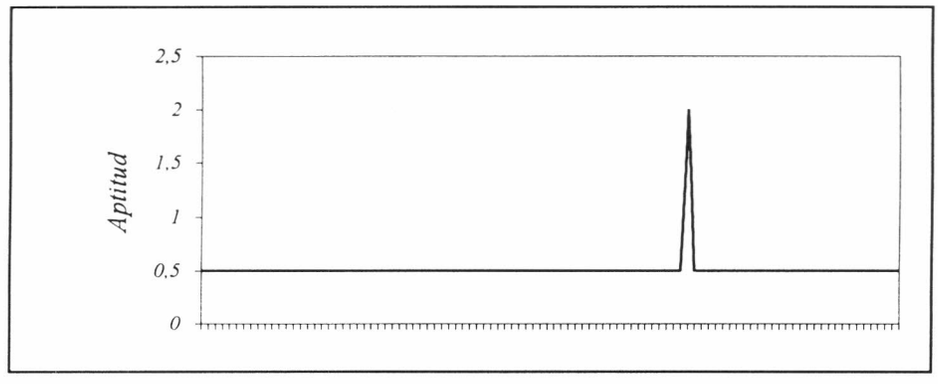

Figura 4-2. Problema de la Aguja en el Pajar

El problema de ***decepción*** es más difícil de determinar, sobre todo en funciones multimodales, multidimensionales, y desconocidas. Para observar su efecto, se analiza el siguiente ejemplo. Se trata de maximizar la cantidad de unos en estructuras binarias de longitud 5. Este problema se conoce con el nombre de ***Onemax*** [GOL/93]. En la [Figura 4-3] se observa la función de aptitud elegida para resolver este problema mediante AG. Está claro que no será fácil para AG encontrar la estructura con mayor cantidad de unos utilizando esta función de aptitud, porque la aptitud de una estructura con todos ceros es más cercana a la óptima que aquellas con cuatro unos, que es casi óptima. Tal función de aptitud presenta las características de ***decepción*** y ***aislamiento***. El sub-óptimo local tiene una cuenca de atracción muy amplia, que aleja al AG del óptimo global, que está aislado.

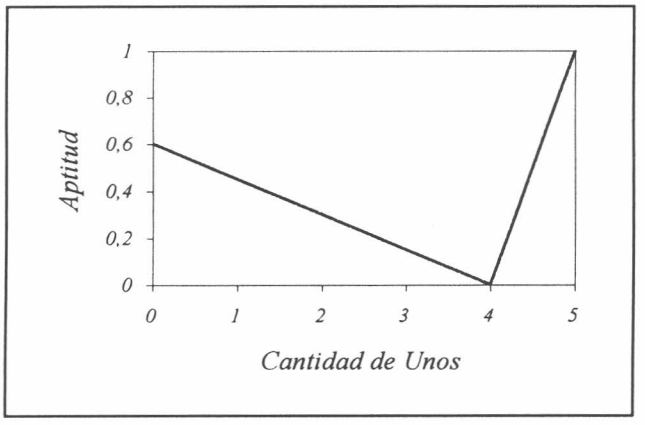

Figura 4-3. Función de Aptitud para el problema Onemax

Como se mencionó, la función de aptitud de la [Figura 4-3] no es la más apropiada para resolver el problema. Sin embargo, ejemplifica muy claramente que, para un problema arbitrario, no siempre se conoce a priori una "buena" función que mida la aptitud. Para el ejemplo dado, la función descripta en la [Figura 4-4] también resuelve el problema, pero presenta características que la hacen más adecuada para ser utilizada por un AG.

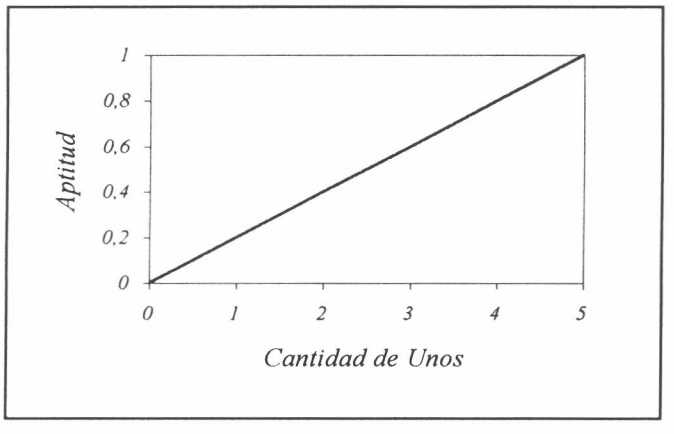

Figura 4-4. Nueva Función de Aptitud para Onemax

La ***multimodalidad*** es la característica mejor resuelta por AG, debido a la cantidad de soluciones simultáneas que maneja y que le permiten explorar varios puntos al mismo tiempo.

La presencia de ***ruido*** en la función de aptitud no será problema para un AG, en la medida en que no distorsione excesivamente la ubicación del óptimo global y su cuenca de atracción. Si el nivel de ruido es suficientemente grande, el óptimo global tenderá a no distinguirse de los óptimos locales, y el AG no podrá ubicarlo.

4.3 Metodología de Diseño de AG

Algoritmos Genéticos es un método apto para la resolución de problemas, con una característica peculiar: necesita poco conocimiento específico del problema. Esto

significa dos cosas: no exige condiciones a la función objetivo, y una vez definidos la representación del cromosoma y la función de aptitud, no es necesario ningún conocimiento adicional para la implementación del algoritmo. Sin embargo, la función de aptitud y la estructura del cromosoma, son el nexo clave del AG con el problema a resolver.

Se propone a continuación una metodología que describe los pasos a seguir para la resolución de un problema arbitrario con AG. Estos pueden separarse en tres fases: dos dependientes del problema y una independiente. Los pasos a seguir son:

- *Fase Dependiente del Problema*
  - *Análisis del Problema*
  - *Diseño del Cromosoma*
  - *Elección de la Función de Aptitud*

- *Fase Independiente del Problema*
  - *Elección de los Operadores Genéticos*
  - *Implementación del AG*

- *Fase Dependiente del Problema*
  - *Análisis de Resultados*

En alguna de las etapas se puede retroceder si se considera que la implementación de la misma no fue satisfactoria [Figura 4-5].

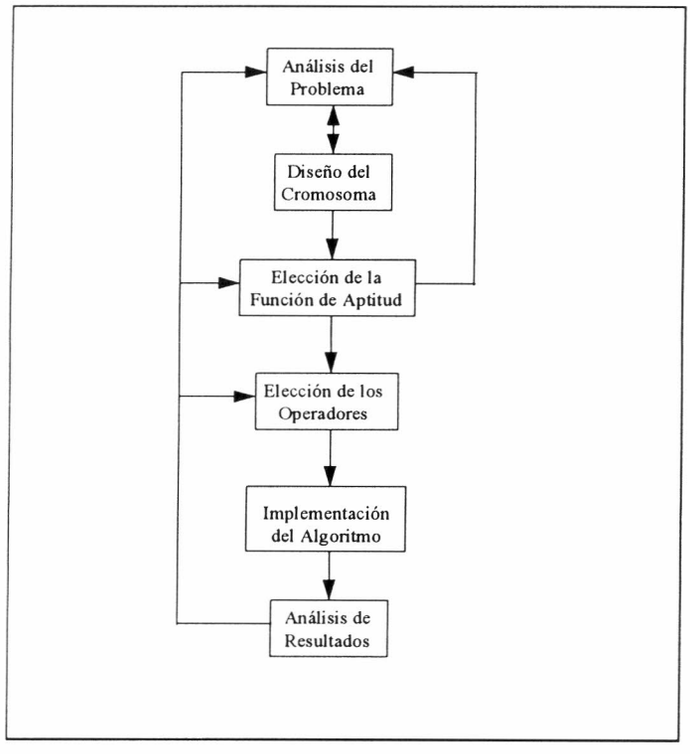

Figura 4-5. Resolución de Problemas con AG

4.3.1 Fase Dependiente del Problema

Es necesario analizar y comprender el problema para definir cada etapa del algoritmo en forma adecuada y de esta manera lograr un diseño final que permita una solución genética eficaz y eficiente. En una primera iteración en el método de diseño el conocimiento del problema probablemente sea pobre. A medida que se ensayan soluciones y analizan resultados este conocimiento crecerá y permitirá mejorar las nuevas soluciones. Las mejoras se logran rediseñando alguna etapa de acuerdo a la experiencia con otros problemas o con intentos previos de solución al problema actual.

Partiendo de un conocimiento dado del problema se define la representación del cromosoma, es decir la codificación de los parámetros y la distribución de los genes dentro del mismo. Si no se puede determinar una adecuada distribución de los genes, se debe analizar la conveniencia de utilizar el operador de inversión[1].

Dada la función objetivo a optimizar y el diseño del cromosoma, se define la función de aptitud que puede coincidir o no con la función objetivo[2]. Esta definición depende del conocimiento que se tenga del problema, por lo cual, probablemente, sea necesario revisar el análisis del mismo.

El análisis de los resultados obtenidos es la última etapa en el diseño de un AG, pero pertenece a la fase dependiente del problema. Esta etapa probablemente llevará a retroceder a alguna anterior, donde se pueda mejorar la decisión realizada y volver a avanzar en la resolución [Figura 4-5].

4.3.2 Fase Independiente del Problema

Dada la representación del cromosoma y la función de aptitud, es necesario definir los operadores genéticos a utilizar. Es decir, elegir un método de selección que provea una presión adecuada, un método de cruza que permita construir bloques constituyentes funcionales al problema, y un método de mutación que garantice un mínimo de diversidad en la población.

Si bien la elección de los operadores se puede realizar utilizando conocimiento del problema, este conocimiento previo no es una condición necesaria. Sin embargo se pueden adaptar o crear nuevos operadores de tal forma que dependan del problema. Estas son las razones por las cuales la elección de operadores se considera dentro de la fase independiente del problema.

Con los operadores genéticos definidos, queda por implementar el algoritmo. El programa resultante debe ser suficientemente flexible como para poder cambiar la representación del cromosoma, la función de aptitud, o los operadores genéticos sin grandes modificaciones.

---

[1] Ver Sección 4.1.2.
[2] Ver Sección 3.3.1.

4.4 Conclusión

En este capítulo se plantearon los aspectos prácticos a tener en cuenta para resolver problemas con AG[1]. En particular, qué representa un gen en función del problema, y los obstáculos a evitar en la definición de la función de aptitud.

La metodología de diseño presentada pretende organizar las diferentes etapas que se deben realizar para resolver un problema dado con AG. El éxito del algoritmo resultante dependerá de un estudio cuidadoso de cada etapa, y de la explotación adecuada del conocimiento adquirido en las etapas previas.

En los Capítulos 6, 7 y 8 se presenta como ejemplo de aplicación de esta metodología la resolución de un problema concreto con AG, la inversión de parámetros geofísicos.

[1] Los aspectos teóricos se plantearon en el Capítulo 3.

Capítulo 5

Algoritmos Evolucionarios

Los Algoritmos Genéticos pertenecen a una clase más genérica de algoritmos, denominados Algoritmos Evolucionarios, en adelante AE. Los AE simulan la evolución de una población de individuos mediante los procesos de selección, recombinación y mutación. Estos procesos dependen de la aptitud de cada individuo definida en base a su entorno. Los distintos tipos de AE se diferencian entre sí de acuerdo a la importancia que cada uno le asigna a estos procesos, el diseño de los mismos, y la representación de los individuos.

Los AE son un concepto general adaptable para la resolución de problemas, y no una colección de algoritmos relacionados y listos para ser usados [BAC/97]. Son especialmente adecuados para resolver problemas difíciles de optimización. Los AE, en general, datan de 1950, pero en las últimas décadas emergieron las tres variantes más importantes [SPE/93]: Programación Evolucionaria (PE), Estrategia Evolucionaria (EE) y Algoritmos Genéticos (AG). Estas variantes implementan algoritmos evolucionarios de diferente manera.

Los AG se pueden, a su vez, clasificar en dos categorías [CAN/95]: Secuenciales (AGS) y Paralelos (AGP). El AGS es el AG tradicional, estudiado ampliamente en los primeros capítulos de este trabajo. El AGP se deriva de la posibilidad de implementar el algoritmo en arquitecturas paralelas o distribuídas.

La [Figura 5-1] muestra una clasificación de las variantes mencionadas de AE.

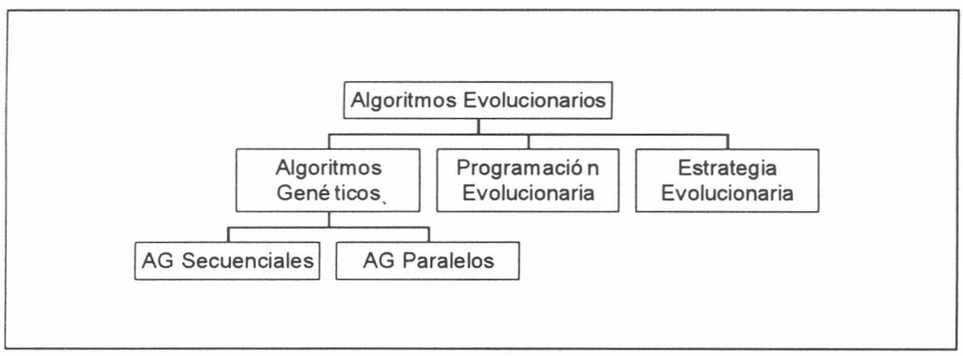

Figura 5-1. Clasificación de Algoritmos Evolucionarios

La estructura de un AE genérico se observa en la [Figura 5-2]. Este algoritmo mantiene una población de estructuras que renueva de acuerdo a las reglas de selección, recombinación y mutación. A cada individuo de la población se le asigna una medida de su aptitud en el entorno. La selección enfoca su atención en los individuos de mayor aptitud para explotar la información provista por los mismos. La

recombinación y mutación perturban estos individuos ampliando el espacio de exploración.

```
t ← 0
generar población inicial P(t)
evaluar P(t)
Hasta (condición de parada)
        t ← t + 1
        seleccionar P(t)
        recombinar P(t)
        mutar P(t)
        evaluar P(t)
        supervivencia P(t)
Fin Hasta
```

Figura 5-2. Algoritmo Evolucionario Genérico

En la [Tabla 5-1] se resumen las características distintivas de los distintos Algoritmos Evolucionarios, las cuales se ampliarán a lo largo del capítulo.

	Representación	***Selección***	***Cruza***	***Mutación***
Programación Evolucionaria	*Dependiente del problema*	*Determinística*	*No utiliza*	*Operador principal*
Estrategia Evolucionaria	*Dependiente del problema*	*Ranking*	*Operador secundario*	*Operador principal*
Algoritmos Genéticos	*Independiente del problema*	*Estocástica*	*Operador principal*	*Operador secundario*

Tabla 5-1. Variantes de Algoritmos Evolucionarios

5.1 Programación Evolucionaria

La PE fue desarrollada por Fogel en 1966. La representación de los individuos en PE está adaptada al dominio del problema. Por ejemplo: en problemas de optimización con valores reales, los individuos dentro de la población son vectores de números reales; para el problema del viajante se utilizan listas ordenadas, y grafos para aplicaciones con máquinas de estados finitos. PE es frecuentemente usado como un optimizador.

Los n individuos de la población son siempre seleccionados como padres, y luego son mutados generando n hijos. Estos hijos son evaluados para luego elegir n sobrevivientes del total de $2n$ individuos, usando una función probabilística basada en la aptitud de cada individuo. La PE no utiliza operador de recombinación.

La mutación está basada en la representación usada y, frecuentemente, es adaptativa, es decir, en lugar de usar una probabilidad de mutación global, esta se codifica como información propia de cada individuo y evoluciona con él.

5.2 Estrategia Evolucionaria

En 1973 Rechenberg desarrolló EE, utilizando una población de tamaño uno, y operadores de selección y mutación. En 1981 se amplió la metodología a poblaciones de tamaño mayor a uno, y se introdujo el operador de cruza. Debido a que inicialmente estos algoritmos se utilizaban para resolver problemas de optimización hidrodinámicas, los individuos se representan, habitualmente, con un vector de números reales.

Se utiliza selección por ranking[1], donde los n mejores individuos son seleccionados para ser padres. Estos padres producen hijos por medio del operador de cruza, incorporando perturbaciones a traves de la mutación. El algoritmo permite generar un número de hijos mayor a n.

El operador de mutación es la principal herramienta de exploración, y , al igual que PE, se utiliza frecuentemente mutación adaptativa. Se utiliza operador de cruza, pero como operador secundario que interviene en la adaptación de la mutación.

La supervivencia es determinística y puede ser implementada en una de dos formas: los n mejores hijos sobreviven y reemplazan a sus padres en la próxima generación, o se eligen los n mejores entre hijos y padres.

Es importante destacar que: tanto PE como EE usualmente tienen como objetivo optimizar. Es decir, el interés se concentra en hallar la mejor solución tan rápidamente como sea posible.

5.3 Algoritmos Genéticos Secuenciales

Como se mencionó previamente, los AGS son los AG tradicionales desarrollados por Holland en 1975. Estos se distinguen dentro de los AE por las siguientes características.

La representación utilizada es generalmente binaria, lo cual permite independizarse del dominio del problema. La selección de padres se realiza en base a una función probabilística que depende de la aptitud de cada individuo. A diferencia de PE, la cruza es considerada el operador principal de exploración mientras que la mutación cumple un rol secundario.

5.4 Algoritmos Genéticos Paralelos

Los Algoritmos Genéticos Paralelos, AGP, han sido usados para resolver problemas dificultosos que necesitan una gran población, trasladándose esto en un gran costo computacional. La motivación básica de estudiar los AGP ha sido la de reducir el tiempo de procesamiento necesario para la búsqueda de una solución aceptable.

[1] Ver Sección 2.1.4.

Actualmente, los Algoritmos Genéticos Paralelos se están estudiando ampliamente. Los AG son fáciles de paralelizar y muchas variantes del modelo básico han sido implementadas con muy buenos resultados en diferentes clases de problemas.

Con los Algoritmos Genéticos Secuenciales, se debe elegir entre obtener un buen resultado y pagar un alto costo en tiempo de procesamiento o perder en la calidad del resultado en favor de encontrarlo en menor tiempo. En contraste, los Algoritmos Genéticos Paralelos pueden obtener un resultado de alta calidad y encontrarlo rápidamente porque, al usar máquinas paralelas, pueden ser procesadas grandes poblaciones en menor tiempo.

Los AGP se pueden clasificar en cuatro categorías [CAN/95]: Global, Grano Grueso, Grano Fino e Híbrido. Este último consiste en alguna combinación de características de los tres primeros.

5.4.1 Global

En esta clase de AGP, la evaluación de los individuos y la aplicación de los operadores genéticos son explícitamente paralelizados. Cada individuo tiene probabilidad de cruzarse con el resto de los individuos de la población. Por lo tanto, la semántica de los operadores genéticos no cambia.

La evaluación puede ser paralelizada asignando un subconjunto de individuos a cada procesador disponible. La comunicación únicamente ocurre al comienzo y al final de esta fase.

En un multiprocesador de memoria compartida, los individuos son almacenados en dicha memoria. Cada procesador puede leer el individuo asignado y escribir el resultado de la evaluación.

Mientras que en un multiprocesador de memoria distribuida, la población es almacenada en un procesador para simplificar la aplicación de los operadores genéticos. Este procesador servidor será el responsable de enviar los individuos a los otros procesadores (clientes) para su evaluación, recolectar los resultados, y aplicar los operadores genéticos para producir la próxima generación. Sin embargo, podría producirse un cuello de botella en el servidor mientra los clientes permanecen ociosos.

Otro de los problemas planteados es que no se puede asegurar que la paralelización de la aplicación de los operadores resulte en una mejora de la performance. Los operadores genéticos son muy simples y el tiempo de comunicación puede ser mayor que el tiempo de cálculo. Esto es especialmente cierto en máquinas de memoria distribuida, donde el overhead de comunicación puede ser considerable.

5.4.2 Grano Grueso

La característica más importante de esta categoría es que la población se divide en unas pocas subpoblaciones, estando relativamente aisladas unas de otras, y se

introduce un nuevo operador, ***migración***, que es utilizado para intercambiar individuos entre subpoblaciones. Los AGP Grano Grueso son el modelo más popular.

La población puede ser implementada con dos modelos distintos: ***island*** o ***stepping stone***. La población, en ambos modelos, es particionada en pequeñas subpoblaciones. En el modelo island los individuos pueden migrar a cualquier otra subpoblación. Mientras que en el modelo stepping stone, la migración está restringida a subpoblaciones vecinas.

Es importante destacar que la migración es controlada por muchos parámetros; la ***topología*** que define la conexión entre las subpoblaciones, una ***tasa de migración*** que controla cuántos individuos serán migrados, y un ***intervalo de migración*** que determina cuándo se llevará a cabo la migración.

Una de las preguntas que se realiza con mayor frecuencia es: ¿cuándo se debe migrar?. Si la migración ocurre antes que los individuos a migrar hayan evolucionado lo suficiente entonces la influencia en la búsqueda hacia la dirección correcta será muy pobre y se gastará mucho tiempo en comunicación.

Por otro lado, es importante determinar la topología de interconexión entre subpoblaciones porque determina cuán rápido (o lento) una buena solución será diseminada a las otras subpoblaciones. Si la topología tiene una conectividad densa, buenas soluciones llegarán rápidamente a todas las subpoblaciones y predominarán en la población. Sin embargo, si la topología esta débilmente conectada, las soluciones serán distribuidas en las subpoblaciones lentamente, permitiendo la aparición de individuos autóctonos potencialmente mejores.

Es de esperar que si las subpoblaciones están relativamente aisladas, cada una encuentre diferentes soluciones parciales del problema. De ahí se puede concluir que sí soluciones parciales pueden ser combinadas para formar una mejor solución, entonces los AGP probablemente encuentren mejores soluciones que los AG secuenciales.

En general los AGP grano grueso son conocidos como AG Distribuidos dado que usualmente son implementados en computadoras de memoria distribuída.

5.4.3 Grano Fino

La población es particionada en un gran número de pequeñas subpoblaciones. El caso ideal es que cada subpoblación tenga un sólo individuo para todo elemento de procesamiento disponible. Para casos donde el cálculo de la función de aptitud requiera de un proceso complejo, esta alternativa puede ser adecuada. Este modelo es apto para ser implementado en computadoras de arquitectura paralela.

Tanto en el método de Grano Grueso como en el de Grano Fino la selección y cruza ocurren dentro de cada subpoblación. Dado que el tamaño de cada subpoblación es mucho menor que el usado en AG secuenciales, es de esperar que los AGP convergan más rápidamente. Esta convergencia se atenúa con las migraciones de individuos entre subpoblaciones.

Es interesante destacar que, mientras el método Global es simplemente la implementación del AG secuencial en una máquina paralela, en los métodos de Grano Grueso y Grano Fino el paralelismo está dado por la semántica del algoritmo, por lo que pueden implementarse tanto en máquinas secuenciales como paralelas.

Capítulo 6

Estimación de Parámetros en Medios Acústicos

En este capítulo se presenta el problema elegido como caso de estudio para la aplicación de AG como método de resolución: la inversión de la ecuación de onda acústica. Este problema se tomó de los siguientes trabajos: *Parameter Estimation in Multidimensional Acoustic Media* [FER/93a] y *An Algorithm for Parameter Estimation in Acoustic Media, Practical Issues* [FER/93b]. La principal dificultad que plantea es la optimización de una función multimodal y multidimensional, difícil de ser tratada con los métodos tradicionales de búsqueda de mínimos. Ambas características hacen que AG sea una alternativa prometedora como método de solución.

La resolución de este problema se hace siguiendo la metodología presentada en el Capítulo 4. Este capítulo está dedicado a la primera estapa: *Análisis del Problema*. El Capítulo 7 presenta tres propuestas de solución, la primera de las cuales surge como consecuencia inmediata del *Análisis del Problema*. La segunda y tercer propuesta se desarrollan tomando como base la primera, y a partir de variantes en las etapas intermedias: *Diseño del Cromosoma*, *Elección de la Función de Aptitud* y *Elección de los Operadores*. La última etapa, *Análisis de Resultados*, se presenta en el Capítulo 8. La *Implementación del Algoritmo* se detalla en los Capítulos 9 y 10.

6.1 Planteo del Problema

El problema a resolver consiste en determinar la composición - o perfil sísmico vertical - de las primeras capas de la corteza terrestre, a partir de un área delimitada en la superficie de la misma. Es decir, trazar un mapa con los materiales que componen el terreno en el área de observación, desde la superficie hasta una profundidad determinada. Para esto, un método utilizado habitualmente consiste en generar una explosión en la superficie, registrar la señal reflejada, e inferir las características del terreno a partir de esta señal [Figura 6-1a]. El objetivo es estimar parámetros físicos que caractericen las capas de interés, y así construir el perfil.

Si bien el problema real consiste en la exploración de una superficie de terreno, el trabajo original [FER/93] asume que ésta se circunscribe a un punto para simplificar el análisis. En el presente trabajo se continúa con el mismo razonamiento. Entonces, el experimento se analiza en una dimensión, sobre el segmento $\Omega = (0,1)$ [Figura 6-1b].

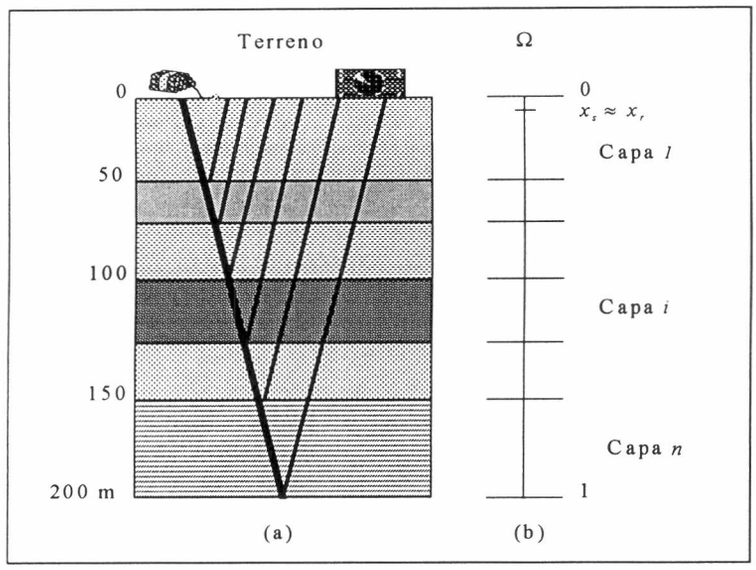

Figura 6-1. Modelo del Terreno

La simulación del experimento, generación de la onda y observación de la presión generada, se realiza de la siguiente manera. Durante el intervalo de tiempo $I = (0, T)$ se excita el medio aplicando una función fuente $S(x, t)$, $x \in \Omega$ y $t \in I$ en el punto $x = x_s$, y se registran los valores de presión $p^{obs}(x, t)$ en el punto $x = x_r$, [Figura 6-1b]. Los valores x_s y x_r , que representan al emisor y al receptor respectivamente, se encuentran ligeramente por debajo de la superficie. La secuencia de presiones observadas $p^{obs}(x, t)$, que se genera como consecuencia de la explosión, depende de la composición del terreno.

El material que compone cada capa está caracterizado por dos parámetros físicos: el ***módulo bulk*** $K(x)$ o ***elasticidad***, y la ***densidad*** $\rho(x)$. Asumiendo que $\rho(x)$ es conocido para cada capa, queda una única incógnita, $K(x)$. Luego, la presión p que genera la onda $S(x, t)$ al atravesar el medio, depende de K, x y t. Entonces $p = p(K, x, t)$.

Por último, la cantidad de capas también se supone dato conocido. Esto es bastante realista dado que, contando los picos que tiene la función $p^{obs}(x, t)$, se puede inferir y sobre-estimar este valor[1]. Por ejemplo, en la [Figura 6-3] se observa que el perfil que representa debe tener entre trece y veinte capas.

El algoritmo básico [Figura 6-2] se puede plantear de la siguiente manera: estimar un K inicial, K^0, y calcular $p(K^i, x, t)$ realizando una simulación del experimento. Medir el error cometido en la estimación de K^i comparando $p(K^i, x_r, t)$ y $p^{obs}(x_r, t)$. Si el error es suficientemente pequeño, entonces K^i es la solución. Sino, estimar mediante algún método de optimización el nuevo K^{i+1} y comenzar una nueva iteración. La medida de error utilizada es el error cuadrático medio entre la presión observada y la presión calculada.

[1] La sobre-estimación de capas no presenta problemas ya que, a lo sumo, se obtendrán capas consecutivas con el mismo material.

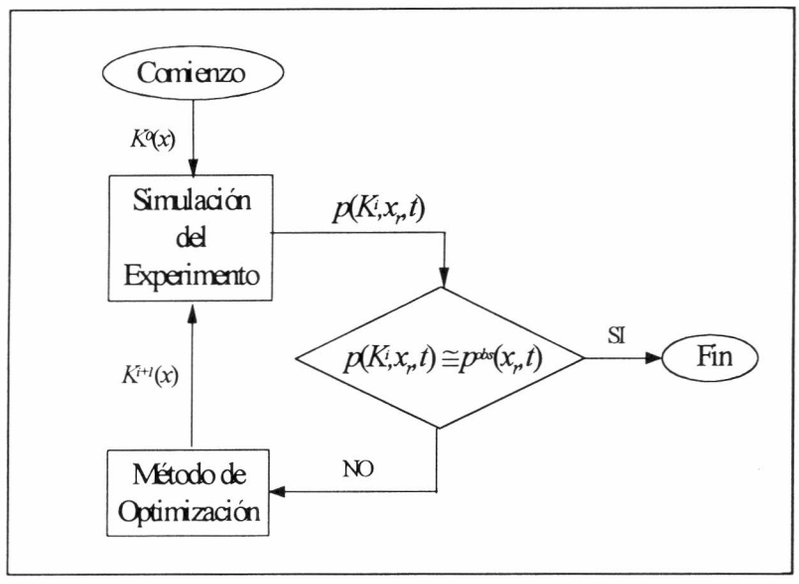

Figura 6-2. Algoritmo Básico de Inversión

De la descripción del algoritmo se observa que es necesario resolver dos sub-problemas.

El primero, llamado ***problema directo***, consiste en simular la generación de la onda $S(x, t)$ en el punto x_s y calcular $p(K, x_r, t)$ suponiendo conocida la composición del terreno, o sea $K^i(x)$. La solución a este problema es conocida y, a partir de la ecuación de propagación de onda en un medio acústico, se utiliza el *Método de Elementos Finitos* para resolverlo, al igual que en [FER/93a].

El segundo, llamado ***problema inverso***, supone que $p(x_r, t)$ es conocido, o sea: $p(x_r,t) = p^{obs}(x_r,t)$, y su objetivo es obtener $K=K^{i+1}(x)$. Esto se realiza mediante la optimización de la función de costo que se plantea comparando $p(K^i, x_r, t)$ y $p^{obs}(x_r, t)$.

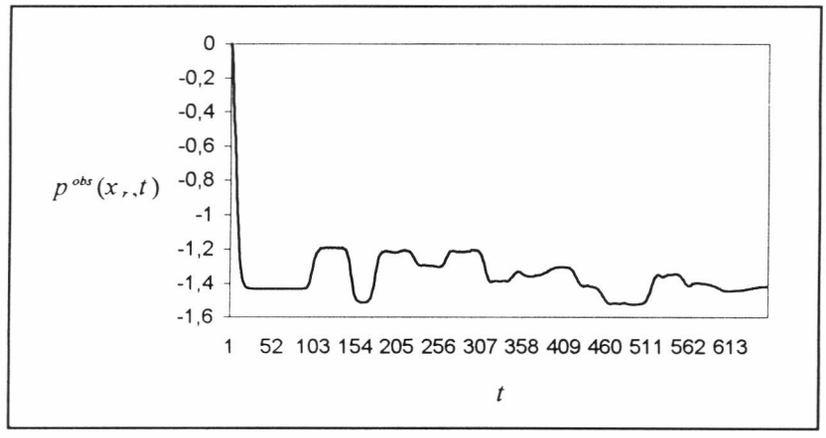

Figura 6-3. Secuencia de presiones observadas

En la [Figura 6-3] se observa, como ejemplo, la secuencia de presiones observadas $p^{obs}(x_r, t)$ para un perfil de dieciseis capas[1], dada la función fuente $S(x_s, t)$ de la [Figura 6-4].

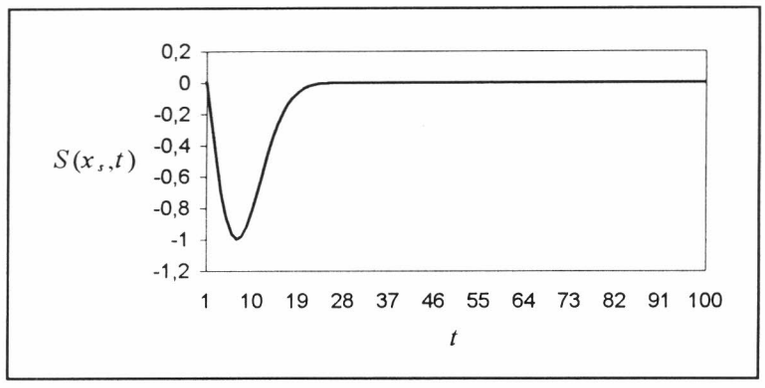

Figura 6-4. Secuencia generada por la fuente

En el resto de este Capítulo se describe con más detalle el problema directo, la resolución del problema de inversión mediante el Método de Cuasilinearización presentado en [FER/93], las características que presenta el problema, y las ventajas de AG como método de inversión.

6.2 Simulación del Experimento

El problema directo plantea la simulación de una explosión a una profundidad x_s de la corteza terrestre en el instante t=0, la propagación de la onda $S(x, t)$ junto con los rebotes originados en las interfaces entre capas durante el intervalo de tiempo I, y el muestreo de la señal $p(K, x, t)$ en un receptor ubicado a una profundidad x_r [Figura 6-1b]. Se asume que tanto x_s como x_r están cerca de la superficie. El resultado consiste en obtener la secuencia de presiones observadas durante la simulación. La influencia de cada capa en $p(K, x, t)$ está determinada por el módulo bulk $K(x)$ que la caracteriza, y que es dato conocido para el problema directo.

El modelo matemático, propuesto en [FER/93], es el siguiente. Dados:

$x \in \Omega = (0, 1)$	Segmento sobre el cual se realiza el experimento
$\Gamma = \delta\Omega = \{0, 1\}$	Bordes de Ω
$t \in I = (0, T)$	Tiempo de la simulación
$S(x, t)$	Función fuente
$\rho(x)$	Densidad del medio
$K(x)$	Módulo bulk o elasticidad del medio
$c(x) = \sqrt{\frac{K}{\rho}}$	Velocidad de onda
$\alpha = \sqrt{K\rho}$	Cambio de variable

[1] El ejemplo corresponde al juego de datos presentado como Test1 en el Capítulo 8.

se trata de calcular $p(K, x, t)$ que cumpla con la Ecuación de Onda Acústica:

$$\frac{1}{K} p_{tt}(K,x,t) - \frac{\partial}{\partial x}\left(\frac{1}{\rho}\frac{\partial}{\partial x} p(K,x,t)\right) = \frac{1}{K} S(x,t) \qquad x \in \Omega, t \in I$$

con condición inicial:

$$p(K,x,t=0) = p_t(K,x,t=0) = 0 \qquad x \in \Omega$$

y condición de bordes:

$$-\frac{1}{\rho}\frac{\partial p(K,x,t)}{\partial \nu} = \frac{1}{\alpha} p_t(K,x,t) \qquad x \in \Gamma, t \in I$$

Para resolver computacionalmente este sistema de ecuaciones es necesario transformar el problema continuo en una aproximación discreta. Esta se logra de la siguiente manera. Sea:

τ_h	Partición cuasi-regular (red no uniforme) en Ω, de elementos cuyo diámetro esta acotado por h
$\mathcal{M}_h$	Espacio de elementos finitos asociado con τ_h
M	Total de muestras p^{obs}
$(v,w) = \int_{\Omega} vw.dx$	Producto interno
$\langle v,w \rangle = \int_{\partial\Omega} vw.dx = v(0)w(0) + v(1)w(1)$	Producto interno en los bordes
$\Delta t = \frac{T}{M}$	Intervalo entre muestras
$u^n = u(n\Delta t)$	u en el instante $n\Delta t$
$\partial u^n = \frac{u^{n+1} - u^{n-1}}{2\Delta t}$	Derivada 1° de u en el instante $n\Delta t$
$\partial^2 u^n = \frac{u^{n+1} - 2u^n + u^{n-1}}{(\Delta t)^2}$	Derivada 2° de u en el instante $n\Delta t$

El método para obtener una aproximación a la solución $p(K, x, t)$ se define de la siguiente manera. Encontrar $p_h^n(K,x,t) \in \mathcal{M}_h$ tal que [FER/93]:

$$\left(\frac{1}{K}\partial^2 p_h^n, v\right) + \left(\frac{1}{\rho}\nabla p_h^n, \nabla v\right) + \left\langle \frac{1}{\alpha}\partial p_h^n, v\right\rangle = \left(\frac{1}{K} S^n, v\right)$$

con valores iniciales:

$$p_h^0 = p_h^1 = 0$$
$$v \in \mathcal{M}_h$$
$$n = 1,2,\ldots, M-1$$

Como la densidad del medio, $\rho(x)$, es conocida y dada la relación entre c, ρ y K, es indistinto trabajar con c o con K. Entonces, tanto para realizar la simulación como para optimizar la función de costo se puede utilizar c o K.

6.3 Optimización con el Método de Cuasilinearización

El método de inversión que se propone en [FER/93a] es el de Cuasilinearización. Como se mencionó anteriormente, la secuencia de presiones reales es conocida a priori, aunque no el módulo bulk $K(x)$ capaz de generar tal secuencia. Este valor es el que se busca estimar y, a partir de este, determinar el perfil sísmico vertical del terreno.

Suponiendo que se cuenta con una estimación de K, se define una función de costo $J(K)$, que mide el error cometido en la estimación p de p^{obs}:

$$J(K) = \frac{1}{2}\int_0^T \left| p(K, x_r, t) - p^{obs}(x_r, t) \right|^2 dt$$

El objetivo es minimizar el funcional $J(K)$, entonces se trata de encontrar K tal que haga cero la primera derivada de $J(K)$. Basta recorrer la superficie de costo ajustando la función variable $K(x)$ hasta encontrar el punto crítico de $J(K)$.

La principal dificultad que tiene la minimización de $J(K)$ es que este funcional es, en general, no lineal. Entonces la superficie de costo que se recorre tendrá mínimos locales que muy probablemente despisten al algoritmo de optimización.

La solución propuesta consiste en agregar un término regularizador $R^\beta(p,p^{obs},K)$ y minimizar:

$$J^\beta(K) = \frac{1}{2}\int_0^T \left| p(K, x_r, t) - p^{obs}(x_r, t) \right|^2 dt + R^\beta(p, p^{obs}, K)$$

con $R^\beta(p, p^{obs}, K) = \frac{1}{2}\beta \| K - K_0 \|^2$.

El parámetro β en R se elige de tal forma que transforme en positiva la segunda derivada de $J^\beta(K)$, con lo cual ahora sí, el problema consiste en buscar un mínimo en una superficie convexa en cierta vecindad del mismo. La determinación de β se efectúa al comienzo del algoritmo, y permanece fijo durante las sucesivas iteraciones, K cambia en cada iteración, y con éste p.

El planteo es el siguiente. Dado K^0 la estimación inicial de $K(x)$, se supone que $J^\beta(K)$ tiene un mínimo en $K^0+\delta K$ para algún δK dado. Entonces, se pretende calcular δK tal que:

$$\left.\frac{\partial J^\beta}{\partial K}\right|_{K^0+\delta K} = 0$$

Para esto, en cada iteración del algoritmo [Figura 6-2], el método de inversión debe estimar el nuevo K^{i+1} como:

$$K^{i+1} = K^i + \delta K$$

donde δK se determina a partir de la derivada de la presión respecto del parámetro $K(x)$.

6.4 Características del problema

La barrera principal que debe sortear el algoritmo de optimización es que la superficie de costo es no-lineal. Por este motivo el Método de Cuasilinearización resuelve, en realidad, una aproximación al problema original. Esta aproximación será buena dependiendo de dos factores:

- Una buena estimación K^0 de K.
- La elección adecuada del parámetro β.

Para que la estimación inicial de K sea buena, su valor debe ser tal que $J^\beta(K^0)$ se encuentre ubicado en la misma vecindad convexa que $J^\beta(K)$. Entonces K^0 debe estar suficientemente próximo al K buscado.

El parámetro β debe ser suficientemente grande como para que la vecindad convexa sea amplia, pero lo bastante pequeño como para mantener próximos los mínimos de $J(K)$ y $J^\beta(K)$.

Por último, aún cuando ambos parámetros se estimen correctamente, la solución del problema original muy probablemente se encuentre desplazada respecto a la del problema aproximado. Es difícil realizar una correcta elección de ambos parámetros de tal forma que la solución aproximada se encuentre próxima a la solución real y evitar, además, caer en un mínimo local.

6.5 El problema de inversión resuelto con AG

Las dificultades planteadas por el problema de inversión de la Ecuación de Onda Acústica, y las características propias de AG como método de optimización, hacen suponer que éste puede proveer de una solución adecuada al problema. Las

características de AG que se pueden explotar frente al problema de inversión son las siguientes:

- Se trata de un método de búsqueda global, adecuado para superficies multimodales.
- No necesita conocimiento de las derivadas de la función a optimizar.

La primera característica hace que AG tenga una probabilidad muy baja de caer en un mínimo local en una superficie no-lineal como $J(K)$. La segunda característica posibilita que AG no necesite disponer de información sobre las derivadas de la función de costo, entonces puede tratar el problema original: minimizar $J(K)$, y no una aproximación, $J^{\beta}(K)$. Por último, la forma en que AG genera sus estimaciones sobre el parámetro hace que no dependa de un buen valor inicial para tener éxito en la búsqueda.

Las consideraciones mencionadas hacen que AG se constituya en una alternativa prometedora como procedimiento de inversión.

Capítulo 7

El Problema de Inversión resuelto con AG

El presente capítulo plantea la resolución del problema de Inversión de la Ecuación de Onda Acústica utilizando AG como método de optimización. A partir del *Análisis del Problema* desarrollado en el Capítulo 6 se continúa con las tres etapas posteriores en la metodología de diseño presentada en el Capítulo 4. Estas son: *Diseño del Cromosoma*, *Elección de la Función de Aptitud* y *Elección de los Operadores*.

En primer término se presenta la ***Propuesta I***, que surge naturalmente del análisis inicial del problema. La ***Propuesta II*** replantea el diseño del cromosoma y logra una reducción considerable en el tamaño del espacio de búsqueda. En la ***Propuesta III*** se realiza un análisis más profundo del problema de inversión, que lleva a replantear la función de aptitud y los operadores genéticos.

Las etapas de *Análisis de Resultados* e *Implementación del Algoritmo* para cada una de las tres propuestas se verá en el Capítulo 8 y Apéndices A y B.

7.1 Propuesta I: Representación Basada en Velocidades

La primera, y más simple aproximación para lograr la inversión de la Ecuación de Onda Acústica consiste en partir del algoritmo básico planteado en el Capítulo 6, y reemplazar el Método de Cuasilinearización por Algoritmos Genéticos como método de optimización. De una manera muy simplificada, este reemplazo se muestra en la [Figura 7-1] tomando como base la [Figura 6-2].

El algoritmo comienza con la inicialización de una población de n estimaciones de K, $K^0_{1\le j\le n}(x)$. Una estimación $K^i_j(x)$ corresponde al individuo j en la generación i, y define un perfil posible para el terreno. Sobre cada $K^i_j(x)$ se simula la expansión de la onda $S(x_s, t)$ para calcular la presión que representa[1], y se compara esta presión con aquella observada en campo. Si ambas presiones son suficientemente parecidas, entonces $K^i_j(x)$ es la solución. Sino, se calcula la aptitud de cada individuo y se aplica AG para construir la nueva población K^{i+1}.

Este planteo básico del algoritmo es una primera aproximación conceptual. Para llegar a la implementación de un algoritmo es necesario diseñar el cromosoma, la función de aptitud, y elegir los operadores adecuados.

[1] Para simular el experimento se utiliza el mismo método que en [FER/93], Elementos Finitos.

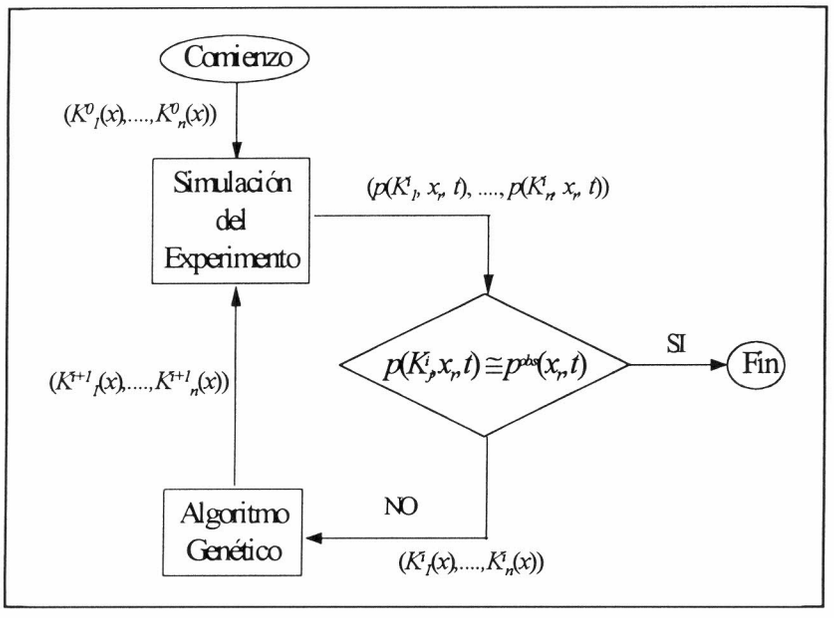

Figura 7-1. Inversión con AG

7.1.1 Diseño del Cromosoma

Un individuo de la población representa una estimación dada para la función $K(x)$. Dado que cada capa está compuesta por un único material, y que K depende del material, basta utilizar un único valor de K por cada capa. Entonces, el cromosoma tendrá tantos genes como capas tiene el terreno, y cada alelo representa el módulo bulk de la capa correspondiente. En [Figura 7-2] se grafica el cromosoma para el individuo j de la población en una generación i dada.

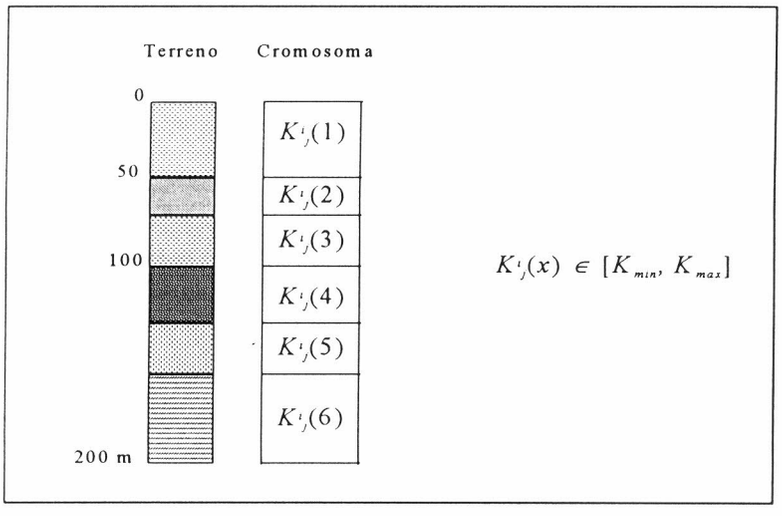

Figura 7-2. Cromosoma para la Propuesta I

Cada gen es un número real que debe ser representado por una cantidad fija de bits, entonces el valor $K^i_j(x)$ de cada alelo debe estar acotado. Para esto se utilizan los módulos bulk mínimo y máximo que se puede encontrar en un material.

$$K_j^i(x) \in [K_{\min}, K_{\max}]$$

Si n es la cantidad de capas y b es la cantidad de bits a utilizar por gen, se tiene que cada individuo es elegido de un total de $(2^b)^n$ posibilidades. Este es, en definitiva, el tamaño del espacio de búsqueda.

7.1.2 Definición de la Función de Aptitud

La función objetivo $g(K)$ se define como el error cuadrático medio entre la presión calculada para alguna estimación K y la presión observada en el campo. Esta es la misma que utiliza el método de Cuasilinearización [FER/93]. Tomando la función de aptitud igual a la función objetivo, se tiene que el algoritmo debe minimizar la función $f(K)$.

$$f(K) = g(K) = \frac{1}{M}\sum_{n=0}^{M-1}\left(p(K, x_r, n\Delta t) - p^{obs}(x_r, n\Delta t)\right)^2$$

Dado que la aptitud de un individuo depende de la presión que genera en el receptor, es necesario calcular esta presión antes de obtener la aptitud. Para esto se utiliza el método de Elementos Finitos, el mismo empleado en [FER/93].

El hecho de minimizar la función $f(K)$ hace que sea más intuitivo llamarla función de costo que función de aptitud, siguiendo la terminología habitual para algoritmos de optimización. Sin embargo, en este trabajo se sigue la convención tradicional para AG puesto que el principal objetivo de $f(K)$ es medir "la aptitud" del individuo que representa el K dado. Además, el significado de mejor o peor aptitud no tiene que estar relacionado en forma directa con un valor alto o bajo de la función f. En este caso, la relación es inversa pero igualmente válida. Entonces $f(K)$ será llamada "función de aptitud" y no función de costo.

7.1.3 Operadores Genéticos

El método de selección utilizado en esta primera propuesta es una combinación entre Selección con Control sobre el Número Esperado y Elitismo[1]. Este mecanismo, si bien es estocástico, garantiza que la cantidad de hijos de cada estructura es muy próximo al número esperado, el cual es proporcional a la aptitud relativa del individuo respecto del promedio de la población [GRE/90]. La combinación con Elitismo garantiza que no se pierden los avances logrados en generaciones pasadas.

Luego de la selección se aplica el operador de mutación. La mutación opera a nivel de bit. El método elegido es la Mutación Simple[2], con una probabilidad P_m de mutar cualquier bit de cualquier estructura de la población.

[1] Ver Secciones 2.1.2 y 2.1.3.
[2] Ver Sección 2.3.1.

Por último se realiza la cruza para construir la nueva generación. El método de cruza utilizado es la Cruza en Dos Puntos[1] operando a nivel de bit. Cada estructura participa de la cruza con probabilidad P_c. Para la elección de los dos puntos de cruza, la estructura es considerada un anillo, de esta manera, siempre es posible elegir la posición 0 como uno de dichos puntos.

7.1.4 Algoritmo para la Propuesta I

El AG a utilizar [Figura 7-3] consiste en generar una población inicial de n estimaciones de K; calcular la aptitud de cada individuo; si la condición de parada no es satisfecha, aplicar los operadores genéticos y crear la nueva generación de estimaciones de K, de lo contrario el algoritmo termina y el mejor individuo representa la solución. La condición de parada utilizada por el AG es la misma para las tres propuestas, y se define en el Capítulo 8, Sección 8.2.

El paso más complejo del algoritmo consiste en el cálculo de la aptitud. Para esto es necesario calcular la secuencia de presiones que representa cada una de los individuos de la población. Este cálculo es costoso pero imprescindible en las tres soluciones propuestas.

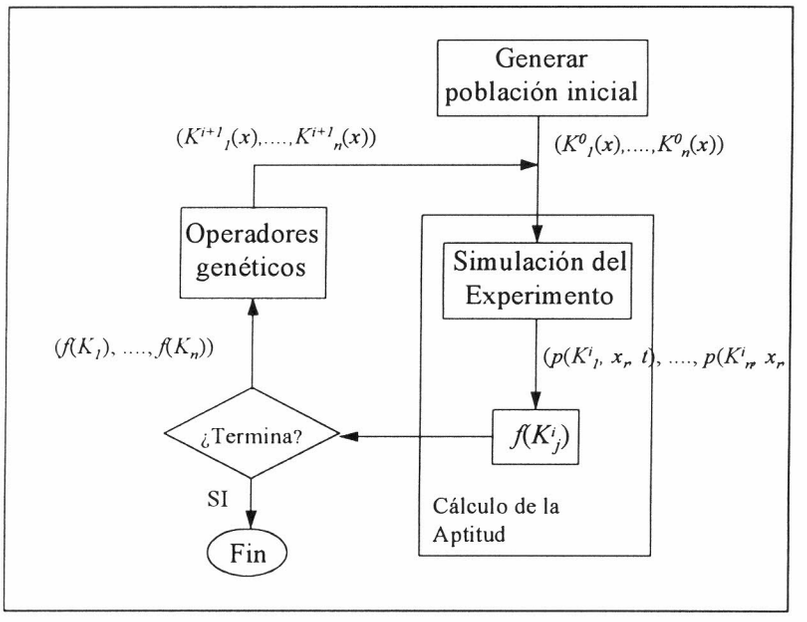

Figura 7-3. Inversión con AG: Propuesta I

El principal inconveniente que tiene esta propuesta es la amplitud del espacio de búsqueda, lo cual implica grandes tiempos de procesamiento. Esto está directamente relacionado con la cantidad de bits utilizados para representar cada gen y, en definitiva, con la precisión o granularidad para el parámetro K. En la medida en que se pretenda una granularidad más fina, la cantidad b de bits aumenta y, en consecuencia, el tamaño del espacio de búsqueda[2]. Otro inconveniente para esta

[1] Ver Sección 2.2.2. En la Sección 3.4 se describe las ventajas teóricas de este método.
[2] Para una discusión sobre el problema de granularidad ver la Sección 4.1.1.4.

propuesta consiste en la suposición de que la densidad $\rho(x)$ de cada capa es conocida, y esto no siempre es así.

Las desventajas mencionadas, sumadas a los resultados obtenidos poco satisfactorios[1], hacen que se vuelva a la etapa de *Análisis del Problema* y se estudie otra propuesta.

7.2 Propuesta II: Representación Basada en Materiales

Con el conocimiento adquirido en la *Propuesta I* y volviendo a analizar el problema, se observa que es posible mejorar el comportamiento del algoritmo con el agregado de una suposición adicional. Esta consiste en disponer de una lista Z de materiales suficientemente grande como para incluir aquellos que componen realmente el perfil del terreno. No es necesario que todos los materiales de la lista se encuentren en el terreno, de hecho no se conocen, basta con que todos aquellos que están en el terreno sí aparezcan en la lista. Para cada material que aparece en la lista se debe disponer del módulo bulk K y de su densidad ρ.

$$Z = \left\{(K,\rho)_0,\ldots,(K,\rho)_{\mu-1}\right\}$$

Dada la lista de materiales posibles, se busca determinar cuáles de estos se ajustan mejor a la composición real del terreno. El tamaño μ de la lista de materiales es pequeño respecto de los posibles valores del parámetro K, por lo tanto, el tamaño del espacio de búsqueda se reduce respecto de la *Propuesta I*. En las siguientes secciones se analiza en profundidad esta idea.

7.2.1 Diseño del Cromosoma

Luego de analizar nuevamente el problema se replantea la estructura del cromosoma. En la *Propuesta I* el alelo es un número real que representa el módulo bulk K estimado para una capa. En esta propuesta, el alelo representa a alguno de los materiales de Z. Para esto basta con que cada alelo sea un número entero z entre 0 y $\mu - 1$, que se representa con $\lceil \log_2 \mu \rceil$ bits. Esta definición tiene la ventaja de eliminar el problema de determinar la granularidad adecuada para todos los genes presentes en la *Propuesta I*.

Cada individuo [Figura 7-4] representa una posible permutación (con repetición) de materiales. Siendo n la cantidad de capas, se tiene que μ^n es el tamaño del espacio de búsqueda. Comparando este valor con el obtenido para la *Propuesta I* se tiene, en general, que:

$$\mu^n << (2^b)^n$$

puesto que para lograr una precisión razonable b tiene que ser relativamente grande.

[1] Ver Capítulo 8 para una discusión sobre los resultados de la Propuesta I.

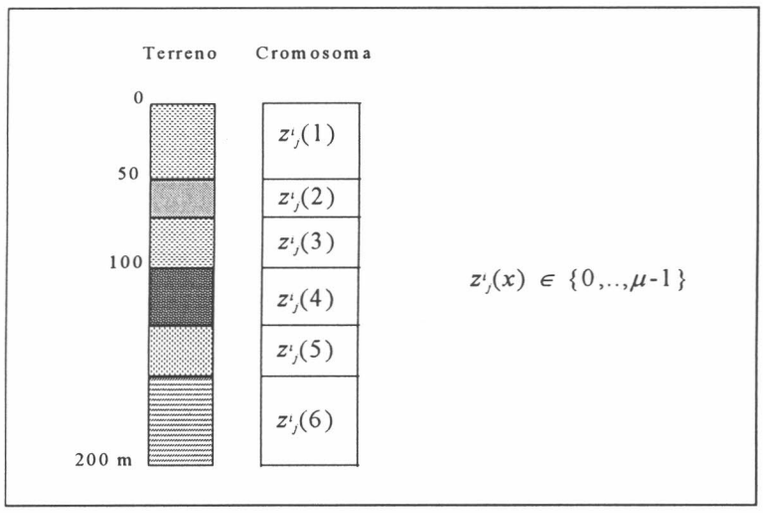

Figura 7-4. Cromosoma para la Propuesta II

7.2.2 Definición de la Función de Aptitud

La función de aptitud a utilizar es la misma $f(K)$ descripta para la *Propuesta I*. Sólo requiere para su cálculo de un paso previo. Como cada gen representa un código de material, es necesario obtener los valores $K(x)$ y $\rho(x)$ correspondientes antes de calcular la presión. Es decir, es necesario mapear el material $z(x)$ a los valores correspondientes de módulo bulk y densidad.

$$z^i_j(x) \rightarrow (K,\rho)^i_j(x)$$

Para que este mapeo siempre sea posible, se asume que:

Todo alelo z representa un material válido para la lista de materiales Z.

Para cumplir con la suposición, es necesario introducir restricciones para generar individuos útiles a la búsqueda. De las alternativas estudiadas para manejo de restricciones, se eligió restringir el espacio de búsqueda a aquellas estructuras que representan perfiles con materiales válidos[1]. Para lograr este objetivo es necesario que los operadores genéticos trabajen a nivel de gen y no de bit, y que eviten la construcción de individuos que no pertenezcan al espacio de búsqueda.

7.2.3 Operadores Genéticos

El método de selección utilizado no varía respecto de la *Propuesta I*, se utiliza Selección con Control sobre el Número Esperado combinado con Elitismo.

El operador de mutación elige cada gen con probabilidad P_m para mutar. El método sigue siendo Mutación Simple pero, a diferencia de la *Propuesta I*, el operador se aplica a nivel de gen y no de bit. Dado un gen elegido para mutar, el nuevo valor

[1] Ver Sección 3.3.1.3.

para el alelo se elige aleatoriamente del conjunto $\{0,\ldots,\mu-1\}$. Con esto se garantiza mantener el espacio de búsqueda restringido, como se supuso en la sección anterior.

Se utiliza Cruza en Dos Puntos, al igual que en la *Propuesta I*. Sin embargo, a diferencia de ésta, los puntos de cruza se eligen de tal forma de no particionar genes. Esto siginifica que no se puede elegir cualquier posición como punto de cruza, sino sólo aquellas que separan genes. Se impide así la destrucción de genes por efecto de la cruza, a la vez que se evita generar genes inválidos por combinación de bits que no corresponden a ningún material de Z. El primer punto de cruza se elige entre n posibilidades, y el segundo entre n-1.

7.2.4 Algoritmo para la Propuesta II

El algoritmo planteado como *Propuesta II* [Figura 7-5] es muy similar a aquel presentado como *Propuesta I* [Figura 7-3]. La principal innovación consiste en cambiar la semántica del cromosoma para lo cual se debe disponer, a priori, de una lista de materiales candidatos suficientemente grande como para incluir aquellos que verdaderamente componen el perfil. Para que el algoritmo explore regiones válidas del espacio de búsqueda se modifica ligeramente los operadores de mutación y cruza, de tal forma que operen a nivel de gen y no de bit. El cálculo de la función de aptitud, si bien es igual al realizado en la *Propuesta I*, requiere de un paso previo. Este consiste en mapear cada alelo del individuo en los correspondientes módulo bulk K y densidad ρ.

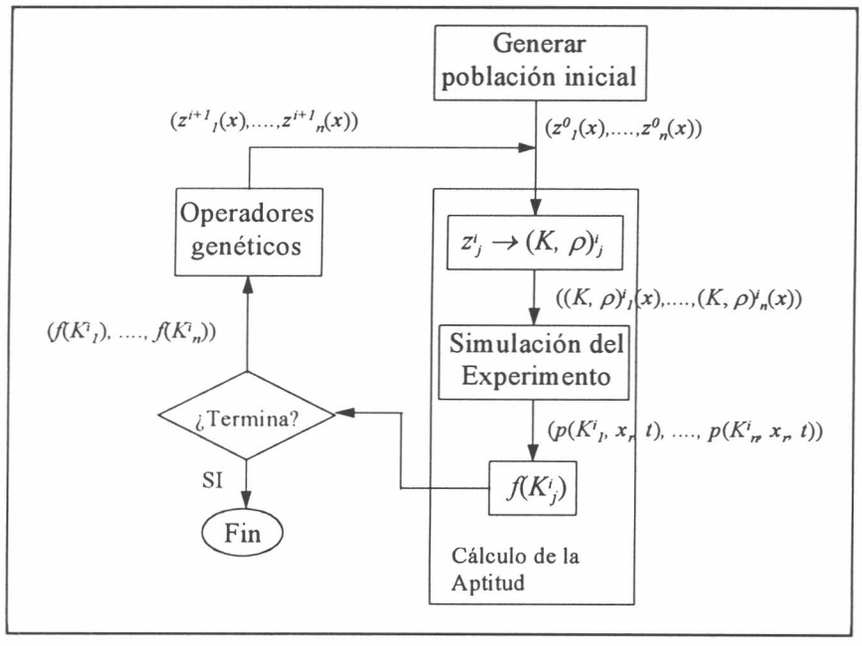

Figura 7-5. Inversión con AG: Propuesta II

Como se verá en el Capítulo 8, los resultados obtenidos con esta propuesta mejoran considerablemente respecto de la *Propuesta I*.

7.3 Propuesta III: Incorporación de Conocimiento del Problema

Considerando el avance logrado en la *Propuesta II* del algoritmo en cuanto a la representación del cromosoma, se trata ahora de explotar cierto conocimiento del problema para mejorar los operadores genéticos, y explorar las regiones más prometedoras del espacio de búsqueda. Se pretende incorporar mayor conocimiento al AG. Para lograr esto es necesario una vez más, analizar el problema.

Hay una relación muy estrecha entre el instante de tiempo en que se escucha un valor de presión y la capa de la que esta proviene. Simplificando, se puede decir que las primeras presiones escuchadas en el receptor corresponden a los rebotes de la onda en la primera capa. A continuación, se escuchan los rebotes de la primera y la segunda capa. Después, de la primera, la segunda y la tercera. Y así siguiendo [Figura 7-6].

Del análisis realizado se desprende la importancia que tiene una buena estimación del módulo bulk para las primeras capas. Las primeras capas influyen en toda la secuencia de presiones, mientras que las más profundas sólo alteran los últimos valores. Entonces, si no se logran buenas estimaciones para el módulo bulk en las primeras capas, difícilmente se logre alcanzar el óptimo.

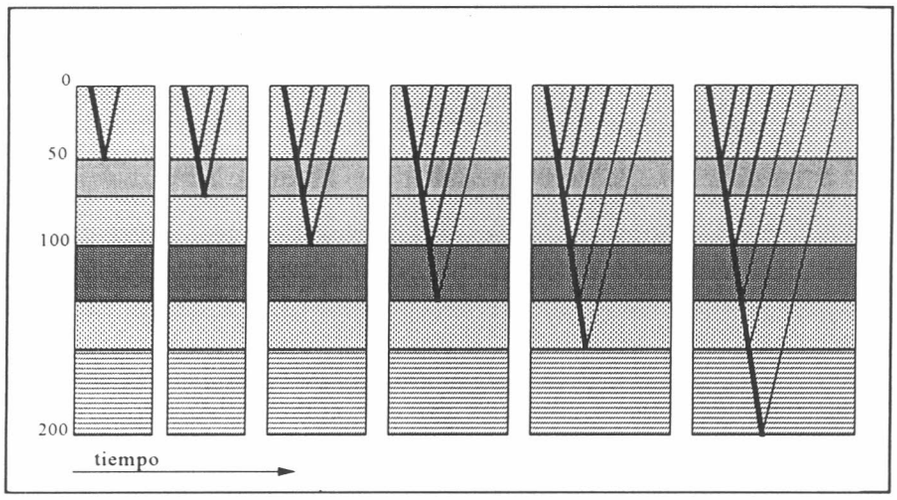

Figura 7-6. Propagación de la onda acústica

Si se compara la presión observada $p^{obs}(x_r, t)$ con la presión calculada $p(K,x_r,t)$ para alguna estimación de K, y se observa los instantes de tiempo en que se registran diferencias apreciables entre una y otra, se puede deducir - aproximadamente - cuáles son las capas en las que se estimó correctamente el módulo bulk y cuáles no. Las capas con una buena estimación serán aquellas en las que los valores calculados de presión coincidan aproximadamente con la presión observada. Es necesario tener en cuenta que si alguna capa se estima mal, afectará negativamente a las estimaciones que le siguen.

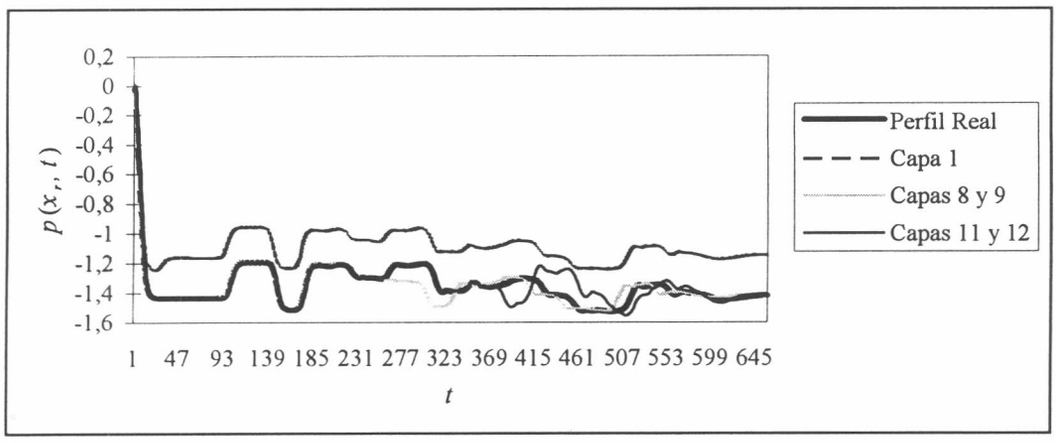

Figura 7-7. Presiones observada y estimadas para perfiles con una y dos capas incorrectas

La [Figura 7-7] muestra como ejemplo la presión observada para un perfil de dieciseis capas[1], y las alteraciones que sufre cuando se estima mal la primer capa, las capas ocho y nueve, y las once y doce. Se observa claramente que una mala estimación únicamente para la primer capa afecta la secuencia de presiones por completo, haciendo inútiles prácticamente todas las estimaciones, aunque éstas sean correctas. A medida que las capas mal estimadas se ubican más profundo, las primeras presiones hasta estas capas coinciden completamente con la secuencia real. Sin embargo, a partir de la primera capa mal estimada la secuencia se aparta de la real. En el ejemplo, cuando se estima mal las capas 8 y 9, la presión estimada difiere a partir de los 270ms aproximadamente. Mientras que, si la primer capa mal estimada es la 11, la diferencia se nota a partir de los 370ms.

Para aprovechar esta característica de la propagación de la onda, se propone ampliar el concepto de aptitud para calificar, también, a cada alelo de acuerdo a lo buena que sea la estimación de K que representa. La información provista por esta calificación es utilizada por los operadores genéticos para operar sobre la población de una manera más "inteligente". De esta manera se incorpora heurística al AG. En las siguientes secciones se detalla esta propuesta.

7.3.1 Diseño del Cromosoma

La estructura del cromosoma es similar a la utilizada en la *Propuesta II*. Basta agregar a cada gen un indicador de aptitud para su alelo. Asumiendo que esta aptitud toma sólo dos valores posibles, *Bueno* o *Malo*, se utiliza un bit adicional por cada uno de los genes. Esto no altera el tamaño del espacio de búsqueda, que sigue siendo μ^n, siendo μ la cantidad de materiales posibles y n el número de capas.

7.3.2 Definición de la Función de Aptitud

El concepto de aptitud se amplía para calificar no solamente a un individuo sino, también, a cada alelo que lo compone. La aptitud del individuo es, como siempre,

[1] El ejemplo corresponde al Test 1 presentado en el Capítulo 8.

el valor que se utiliza para determinar la capacidad de supervivencia durante el proceso de selección del AG. La aptitud del alelo es aprovechada por los operadores de cruza y mutación que la utilizan para alimentar la heurística que se explica a continuación.

La aptitud utilizada para calificar un individuo sigue siendo $f(K)$, la misma utilizada en las *Propuestas I* y *II*. El operador de selección se basa en esta aptitud para determinar la supervivencia de cromosomas. En esto no hay modificaciones respecto de las propuestas anteriores.

La aptitud que califica a un alelo puede tomar algun valor del conjunto $\{B, M\}$, donde B indica que el alelo es ***Bueno*** y M indica que es ***Malo***[1]. Se define la función γ aplicado a un cromosoma como una n-upla donde el elemento i indica la calificación obtenida por el alelo correspondiente al gen i. Entonces $\gamma: \mathrm{E}^n \rightarrow \{B, M\}^n$ donde E^n es el espacio de cromosomas de longitud n. Por ejemplo, la función γ aplicada al siguiente cromosoma:

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8

puede ser:

B	B	M	M	M	B	B	B

Para el problema de inversión en particular, el cromosoma representa una lista de materiales y, como se vió en la Sección 7.2.2, cada uno se mapea a un par (K, ρ). Como el cálculo de γ se realiza utilizando la velocidad de onda $c(x)$ y $c = \sqrt{K/\rho}$, entonces se define $\gamma: \Re^n \rightarrow \{B, M\}^n$ con $\gamma = \gamma(c)$.

La función $\gamma(c)$ utilizada se muestra en la [Figura 7-8]. Se determina la distancia d_i que debe recorrer la onda, ida y vuelta, para atravesar la capa i, es decir dos veces su espesor[2]. Dada la velocidad de onda estimada c_i para atravesar dicha capa[3], se calcula el tiempo $\tau_i = d_i / c_i$ necesario para atravesarla. Por último, se determina el intervalo de tiempo $U_i \subset I$, de longitud τ_i, durante el cual el receptor escucha las presiones correspondientes a esta capa.

El alelo será considerado bueno, si la presión calculada y la observada difieren en una cantidad suficientemente pequeña. De lo contrario, el alelo se considera malo. Esta comparación se realiza en el intervalo de tiempo determinado por U_i.

[1] Esta calificación no necesariamente tiene que resultar en dos valores posibles, para otros problemas puede resultar conveniente utilizar conjuntos más amplios, por ejemplo $\Re$.
[2] El espesor de cada capa es un dato conocido, ver Sección 6.1.
[3] Esta velocidad es la representada por el gen i del cromosoma.

Función $\gamma(c)$
i= 1
t_0= 0
Para cada gen i hacer
Sea:
c_i la velocidad representada por el alelo i
d_i = (2 * espesor de la capa i)
$\tau_i = d_i/c_i$
$U_i = [t_{i-1}, t_i]$ tal que $t_i = t_{i-1} + \tau_i$
Si $\left|p(K, x_r, U_i) - p^{obs}(K, x_r, U_i)\right| \leq \varepsilon$ entonces
el alelo es bueno
Sino
el alelo es malo
FinPara

Figura 7-8. Cálculo de la Función $\gamma(c)$

En resumen, se segmenta el tiempo total del experimento I= (0, T) en sub-intervalos que corresponden, cada uno, al lapso durante el cual se escucha la onda reflejada por la capa número i. Formalmente se busca particionar I de tal forma que:

$$I = \bigcup_{i=1}^{n} U_i = \bigcup_{i=1}^{n} [t_{i-1}, t_i] \quad \text{tal que} \quad t_i - t_{i-1} = \tau_i$$

7.3.3 Cruza Multipunto con Heurística

Para aprovechar la información brindada por la función $\gamma(c)$ se define una nueva variante del operador de cruza, se trata de la *Cruza Multipunto con Heurística*. La heurística consiste en:

Preservar los ***alelos buenos*** de ambos padres.

El criterio para determinar los alelos buenos se basa en la información brindada por la función $\gamma(c)$. La cruza utiliza esto para formar individuos que heredan los mejores alelos de ambos padres. Dados los cromosomas A y B, el procedimiento consiste simplemente en:

1. Copiar en el primer hijo los alelos buenos de A, y completar los genes que quedan con los alelos de B correspondientes a esas posiciones.

2. Copiar en el segundo hijo los alelos buenos de B, y completar los genes que quedan con los alelos de A correspondientes a esas posiciones.

El siguiente ejemplo muestra como opera esta variante de cruza. Dados los siguientes cromosomas, con la correspondiente evaluación de la función $\gamma(c)$:

Cromosoma A

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
B	B	M	M	M	B	B	B

Cromosoma B

b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8
B	B	B	M	M	M	B	B

luego de aplicar la cruza heurística se obtiene:

a_1	a_2	b_3	b_4	b_5	a_6	a_7	a_8

b_1	b_2	b_3	a_4	a_5	a_6	b_7	b_8

Nótese que los alelos a_3 y b_6 se han perdido. Esto no ocurre en la cruza convencional puesto que cada alelo de un cromosoma padre siempre sobrevive en alguno de los hijos. Esta característica particular de la cruza heurística se enuncia de la siguiente manera:

Propiedad Selectiva de la Cruza Heurística

- Si el alelo i de ambos cromosomas es calificado como bueno, entonces ambos alelos sobreviven.
- Si el alelo i de ambos cromosomas es calificado como malo, entonces ambos alelos sobreviven.
- Si el alelo i es calificado bueno para un cromosoma y malo para el otro, entonces solamente sobrevive el bueno, no el malo.

Para el problema de Inversión en particular, dos alelos a cruzar representan dos estimaciones de velocidad para una capa de terreno dada. La Propiedad Selectiva establece que si ambas estimaciones son buenas o ambas son malas, entonces se preservan ambas hasta tanto se tenga más información para decidir. Pero si una estimación es buena y la otra es mala, entonces sólo se preserva la buena, descartándose la mala.

7.3.4 Precisión de la Heurística Utilizada

Es necesario recalcar que la calificación de cada alelo será confiable en la medida en que la heurística utilizada sea confiable. Para la función $\gamma(c)$ definida en [Figura 7-8], esto está dado por el parámetro ε.

Para ε suficientemente pequeño, la calificación $\gamma(c)$ de los alelos es confiable hasta el primer material calificado como incorrecto inclusive. Esto se puede mostrar con el siguiente ejemplo. Sea A un cromosoma que estima la composición del terreno:

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
B	B	M	M	M	B	B	B

Sea V la verdadera composición del terreno. Esta es, obviamente, desconocida. Sin embargo, dada la calificación de cada alelo de A se puede suponer lo siguiente:

$v_1 = a_1$	$v_2 = a_2$	$v_3 \neq a_3$	?	?	?	?	?

Esto ocurre porque la presión de cada capa acumula los errores de las precedentes. Entonces, a partir de v_4 no se puede deducir nada puesto que v_3 no se pudo aún estimar de manera confiable. Con V a la vista se puede observar que A puede contener alelos realmente buenos calificados como malos, y alelos malos calificados como buenos. Se define:

Falsos Negativos son alelos buenos calificados erróneamente como malos.

y

Falsos Positivos son alelos malos calificados erróneamente como buenos.

Suponiendo que no es posible generar falsos positivos, los falsos negativos no presentan problemas. Por la Propiedad Selectiva de la cruza heurística, un falso negativo se pierde únicamente cuando su cromosoma se cruza con uno que tiene en esa posición un alelo calificado como bueno. Pero si este alelo está calificado como bueno, resulta que realmente es bueno puesto que se supuso la inexistencia de falsos positivos. Por lo tanto el falso negativo se pierde y sobrevive un alelo que sí es bueno.

Suponiendo que sí es posible generar falsos positivos, la abundancia de éstos puede llevar al algoritmo a converger prematuramente a una solución errónea. En el caso extremo en que la heurística califique siempre como buenos a todos los alelos, por definición de Cruza Heurística se tiene que la cruza de dos individuos siempre resulta en los mismos dos individuos, es decir que los hijos son iguales a sus padres. Cuando ocurre esto, la evolución depende exclusivamente de la selección y de la mutación, entonces el algoritmo converge prematuramente y difícilmente se llegue a la solución.

La aparición excesiva de falsos positivos o falsos negativos depende del valor de ε. Si ε es demasiado grande, un alelo muy alejado del valor correcto será considerado bueno cuando en realidad no lo es. Entonces aparecen los falsos positivos. Un valor demasiado pequeño de ε hará que se califique como malo a la mayoría de los alelos. Esto último traerá consigo la aparición excesiva de falsos negativos, lo cual tendrá el efecto de lentificar la convergencia del algoritmo.

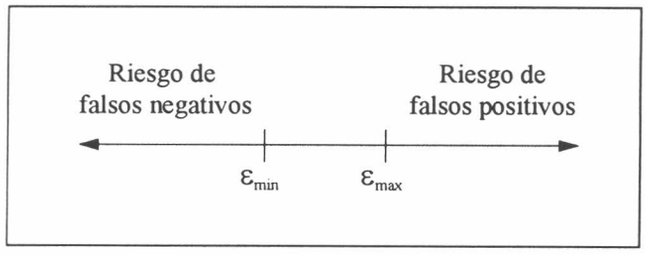

Figura 7-9. Cotas para el parámetro ε

La cruza multipunto con heurística mantiene la propiedad de independencia del problema a resolver, puesto que no utiliza información propia del mismo. La heurística que utiliza se la provee la nueva función de aptitud que califica a cada gen dentro del

cromosoma. Como se vió en el Capítulo 4, la función de aptitud sí depende del conocimiento del problema.

7.3.5 Mutación con Heurística

El determinismo de la cruza multipunto con heurística hace que el AG eventualmente alcance un punto fijo en el cual los primeros genes son buenos para toda la población. Como los puntos de cruza siempre ocurren entre genes, no entre bits, entonces no hay forma de generar nuevos genes a partir de los ya existentes. La mutación, entonces, cobra vital importancia para solucionar este problema, y con ello evitar la convergencia prematura.

Se define dos niveles de mutación: uno que afecta al cromosoma y otro que afecta a cada gen dentro de ese cromosoma. Cada cromosoma se muta con una determinada probabilidad P_m y, una vez seleccionado el cromosoma, se elige un único gen a mutar. El mecanismo es el siguiente:

1. Dado el cromosoma se define Q como el conjunto de todos los genes del cromosoma, y Q' como el conjunto de todos los genes a partir del primero malo inclusive.

   $$Q = \{q_i \,/\, q_i \text{ es un gen y } 1 \le i \le n\}$$
   $$Q' = \{q_i \,/\, q_i \text{ es un gen, } j \text{ es el primer gen malo y } j \le i \le n\}$$

2. Con probabilidad P_l tal que $0 < P_l < \frac{1}{2}$ la elección del gen se realiza sobre Q, mientras que con probabilidad $P_h = 1 - P_l$ se elige el gen en Q'.

3. Una vez determinado el conjunto del cual se elige el gen, todos los genes del mismo compiten con probabilidad uniforme.

La mutación baja P_l se aplica sobre todos los genes no solo para garantizar diversidad en la población sino, también, para impedir la supervivencia de los falsos positivos en el caso de los genes correspondientes a las primeras capas.

La mutación alta P_h se aplica a los genes posteriores al primer gen malo. Como se vió anteriormente, la calificación obtenida por estos genes es poco confiable dado que a partir del primer gen mal calificado se acumulan los errores. Por lo tanto, los esfuerzos de la mutación se concentran en estos genes, para mejorar la confianza en la calificación dada por la heurística.

Una vez determinado el conjunto de genes del cual se elige uno para mutar, se elige con probabilidad uniforme un gen dentro del mismo, y el nuevo valor del alelo se toma del conjunto de materiales posible Z.

7.3.6 Algoritmo para la Propuesta III

El algoritmo utilizado [Figura 7-10] en esta propuesta es similar al presentado en la [Figura 7-5] para la *Propuesta II*. Las modificaciones respecto de aquel son básicamente dos. Primero es necesario calcular la función $\gamma(c)$, lo cual se realiza en la etapa de cálculo de la función de aptitud. Segundo, los operadores genéticos de cruza y mutación utilizan la heurística provista por la función $\gamma(c)$ para crear la población correspondiente a la generación $i+1$.

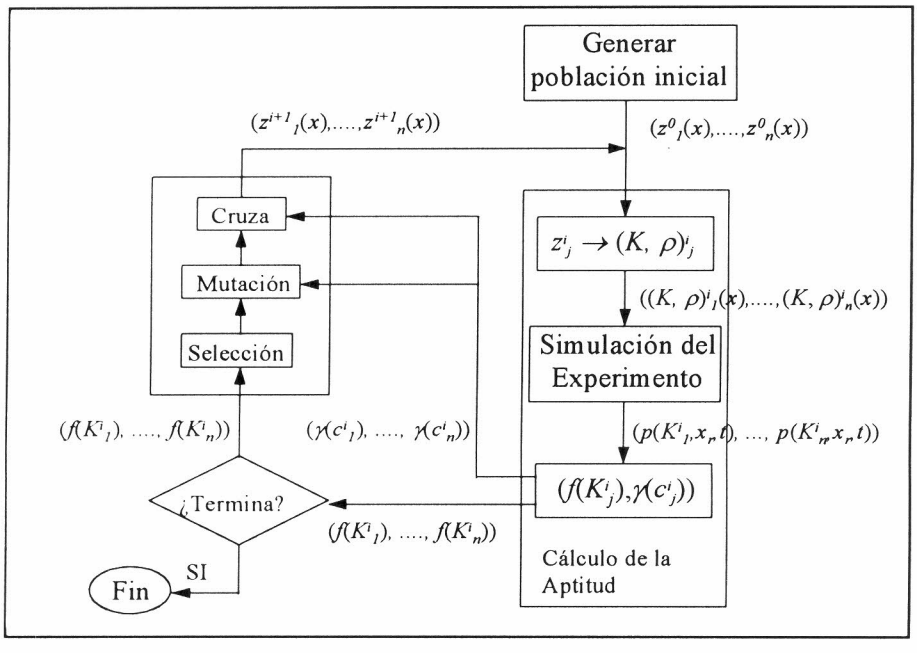

Figura 7-10. Inversión con AG: Propuesta III

El mecanismo de selección no cambia respecto de la *Propuesta II*, continúa utilizando la información provista por la función $f(K)$. La población intermedia que genera la selección es alterada por la mutación primero, y por la cruza luego. Estos operadores se mantienen independientes del problema, puesto que sólo necesitan información sobre cuán apto es un gen. Entonces, es posible cambiar la heurística que define la función $\gamma(c)$ sin que afecte la elección de los operadores, o se puede cambiar el método de cruza o mutación de tal forma de aprovechar de otra manera la heurística.

7.4 Conclusión

Como se ha visto a lo largo de este capítulo, las soluciones propuestas al problema de inversión evolucionan junto con el conocimiento que se adquiere del problema. Esto se logra ensayando primero una solución simple, *Propuesta I*, que surge a partir de una solución anterior no-genética. Las ventajas de esta propuesta respecto del *Método de Cuasilinearización* son principalmente dos: se realiza una búsqueda global sin depender de una buena estimación inicial y se minimiza la función objetivo original $J(K)$, no una aproximación $J^{\beta}(K)$.

Luego del análisis de los resultados obtenidos con la *Propuesta I* se profundiza el conocimiento del problema, se replantea alguno de los elementos que componen el AG y surge la *Propuesta II*. La novedad introducida en esta segunda propuesta consiste en un cambio en la representación del cromosoma que permite una reducción considerable en el espacio de búsqueda. Esto lleva a obtener poblaciones de mejor calidad en menor tiempo con individuos de mejor aptitud que los obtenidos con la *Propuesta I*. El cambio de representación trae consigo una importante ventaja adicional. Como el espacio de búsqueda ahora consiste en materiales, es decir pares $(K(x), \rho(x))$, no es necesario fijar la densidad en cada capa sino ésta pasa a ser incógnita sin afectar la eficiencia.

Los buenos resultados obtenidos con la *Propuesta II* sugieren que la dirección elegida para esta meta-búsqueda de un algoritmo que resuelva mejor el problema es la correcta. Entonces, a partir de la *Propuesta II*, sus resultados, y un tercer análisis del problema, tienen como consecuencia una ampliación del concepto de función de aptitud, y la consecuente incorporación de heurística al AG. Se definen métodos nuevos de cruza y mutación que aprovechan esta heurística y, al mismo tiempo, mantienen las ventajas logradas en la *Propuesta II*. Surge entonces la *Propuesta III* con una convergencia poblacional muy superior a la lograda con las propuestas anteriores. La heurística utilizada permite focalizar la búsqueda en las áreas más prometedoras del espacio de búsqueda menteniendo las ventajas logradas con la *Propuesta II*.

	Espacio de búsqueda	***K_{min} y K_{max} conocidos***	***$\rho(x)$ conocido por capa***	***Materiales candidatos***
Cuasilinearización	*módulo bulk*	*sí*	*sí*	*no*
Propuesta I	*módulo bulk*	*sí*	*sí*	*no*
Propuesta II	*material*	*no*	*no*	*sí*
Propuesta III	*material*	*no*	*no*	*sí*

Tabla 7-1. Comparación de hipótesis para los algoritmos de inversión

En la [Tabla 7-1] se resume las suposiciones en las que se basan el *Método de Cuasilinearización* visto en el Capítulo 6 [FER/93], y las *Propuestas I, II* y *III*. Tanto *Cuasilinearización* como la *Propuesta I* realizan una búsqueda en el espacio de módulos bulk o velocidades. Como este espacio de búsqueda es continuo, se hace necesario acotar los valores posibles para el parámetro a minimizar con valores K_{min} y K_{max}, y discretizarlo con cierta granularidad. Esto no es necesario en las *Propuestas II* y *III*, dado que el espacio de búsqueda es discreto. Sin embargo, en estos dos últimos casos es necesario disponer a priori de una lista de materiales candidatos a componer el perfil. Esta lista está compuesta de módulos bulk y densidades para cada material. Las características del espacio de búsqueda para las *Propuestas II* y *III* hacen innecesario fijar la densidad $\rho(x)$ para cada capa, como sucede en *Cuasilinearización* y la *Propuesta I*.

	Espacio de Búsqueda	*Función de Aptitud*	*Heurística*	*Selección*	*Cruza*	*Mutación*
Propuesta I	2^{bn}	*f(K)*	*no*	*Control sobre el número esperado*	*Multipunto*	*Simple*
Propuesta II	μ^n	*f(K)*	*no*	*Control sobre el número esperado*	*Multipunto*	*Simple*
Propuesta III	μ^n	*(f(K), γ(c))*	*si*	*Control sobre el número esperado*	*Multipunto con Heurística*	*Simple con Heurística*

Tabla 7-2. Características de las *Propuestas I, II* y *III*

En la [Tabla 7-2] se muestra las características principales de las *Propuestas I, II* y *III* desde el punto de vista de AG. En el Capítulo 8 se detallan los resultados obtenidos con cada propuesta en diversos experimentos y se muestra con ejemplos las mejoras logradas.

Capítulo 8

Análisis de Resultados

En este Capítulo se desarrolla la etapa de *Análisis de Resultados* para las pruebas realizadas con las *Propuestas I, II* y *III*. En primer término se describen los datos de campo, luego los parámetros propios de AG. Por último, se presentan los resultados obtenidos con una serie de cinco tests, que permite observar claramente las mejoras incorporadas en cada propuesta. Como las pruebas se realizan en condiciones ideales, los datos de campo no tienen ruido, hacia el final del capítulo se resume una serie de pruebas adicionales realizadas con datos de campo que tienen ruido incorporado.

Si bien cada propuesta surge como consecuencia de los resultados y el conocimiento obtenido a partir de la anterior, los resultados se presentan ordenados por test. De esta manera se puede observar claramente los avances logrados.

8.1 Datos de Campo

Se utilizan cinco conjuntos de datos, en adelante denominados *Test 1* a *Test 5*, con los cuales se realizan las pruebas de comportamiento de cada propuesta. Estos conjuntos son algunos de los utilizados en [FER/93] para testear el *Método de Cuasilinearización*. Cada uno de los test responden a perfiles posibles compuestos por datos de materiales reales. Las secuencias de presiones observadas se generan en forma sintética a partir de una simulación. Los datos más importantes que componen cada uno de los juegos se muestran en la [Tabla 8-1].

Dato	*Variable Representada*	*Unidad de Medida*	*Propuestas*
Capas	n	-	*I, II, III*
Espesor de cada capa	-	*m*	*I, II, III*
Ubicación de la fuente	x_s	*m*	*I, II, III*
Ubicación del receptor	x_r	*m*	*I, II, III*
Tiempo de simulación	T	*ms*	*I, II, III*
Presión observada	$p^{obs}(x_r, t)$	g/cm^2	*I, II, III*
Velocidad mínima	c_{min}	*m/ms*	*I*
Velocidad máxima	c_{max}	*m/ms*	*I*
Densidad de cada capa	$\rho(x)$	g/cm^3	*I*
Lista de materiales	Z	-	*II, III*
Módulo bulk	K	$dina/cm^2$	*I, II, III*
Velocidad	c	*m/ms*	*I, II, III*

Tabla 8-1. Principales datos de campo

La cantidad de capas n y el espesor de cada una debe ser estimada antes de la realización del experimento. La ubicación de la fuente x_s y el receptor x_r son datos

conocidos, habitualmente se asume que están ubicados a igual profundidad. El tiempo T de simulación se debe determinar suficientemente grande como para que la onda generada por la fuente tenga tiempo de recorrer ida y vuelta las n capas. La secuencia de presiones observadas $p^{obs}(x_r, t)$ como consecuencia de la expansión de la onda $S(x_s, t)$ es el principal dato de campo, que se utiliza para validar las estimaciones que se realicen.

La *Propuesta I* requiere estimaciones para las velocidades de onda mínima y máxima. El intervalo $[c_{\min}, c_{\max}]$ se debe estimar de tal forma que sea suficientemente amplio como para contener todas las velocidades de onda posibles para los materiales buscados.

Las *Propuestas II* y *III* requieren estimar una lista de materiales Z, suficientemente amplia como para contener aquellos materiales que realmente componen el perfil. La información que debe contener Z para cada material es su módulo bulk K y densidad ρ. Los juegos de datos utilizados, *Test 1* a *Test 5*, asignan valores a cada uno de los parámetros mencionados.

8.2 Parámetros del Algoritmo Genético

Cada propuesta de AG como método de inversión, *Propuesta I II* y *III*, se evaluó con cada uno de los cinco tests, *Test 1* a *Test 5*. Todas las pruebas se realizaron, excepto algunos casos particulares que se mencionarán específicamente, con iguales parámetros.

El tamaño de población es de 50 individuos para las *Propuestas I* y *II*. Para la *Propuesta III* se utiliza una población de 30 individuos ya que el conocimiento incorporado en esta propuesta permite reducir el tamaño de la población. El diseño de cada cromosoma depende de la *Propuesta* de que se trate. Para la *Propuesta I* se utilizan 10 bits por cada gen. Entonces, un alelo es uno de 1024 valores posibles, pertenecientes al intervalo $[c_{\min}, c_{\max}]$. En el caso de las *Propuestas II* y *III* se utiliza una lista Z de, a lo sumo, μ= 7 materiales. De esta manera, el alelo es un valor de 3 bits. En la [Figura 8-1] se observa el tamaño del espacio de búsqueda para cada propuesta en cada uno de los test.

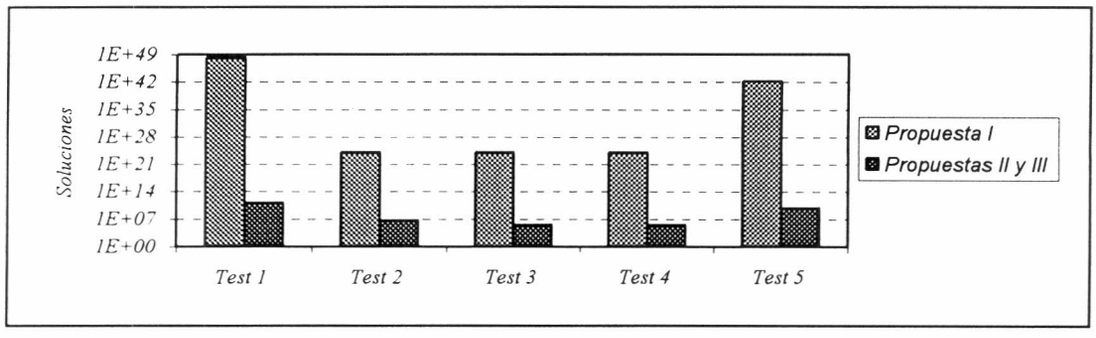

Figura 8-1. Tamaño del Espacio de Búsqueda (escala logarítmica)

Suponiendo que en el *Método de Cuasilinearización* [FER/93] se utiliza la misma cantidad de bits para representar cada parámetro que en la *Propuesta I*, se tiene que el tamaño del espacio de búsqueda coincide con este último.

Debido a las diferencias en la representación entre las propuestas, el grado de convergencia P_{eq} se calcula en base a la unidad de información sobre la que operan la cruza y la mutación. En la *Propuesta I* se opera a nivel de bit[1], entonces el valor mínimo de P_{eq} es $\frac{1}{2}$. En las *Propuestas II* y *III* el alelo puede tomar entre 5 y 7 valores posibles[2], entonces el valor mínimo de P_{eq} oscila entre $\frac{1}{5}$ y $\frac{1}{7}$.

Cada prueba consta de 3 experimentos, de tal forma de no depender exclusivamente de los resultados de uno. Se utilizan tres criterios de parada para un experimento. Para establecer el primer criterio de parada se define el concepto de evaluación:

Una ***evaluación*** es el cálculo de la aptitud para un individuo particular.

Un individuo puede estar repetido en una población dada, y puede sobrevivir en sucesivas generaciones. Como el cálculo de la aptitud es costoso computacionalmente, se pretende evaluar sólo una vez al individuo nuevo y no evaluar sus copias que, obviamente, tiene igual aptitud al original. Se evalúan entonces sólo los individuos que son distintos a sus padres. El primer criterio de parada se cumple al alcanzar un máximo de evaluaciones. El valor utilizado es de 2000 evaluaciones.

El segundo criterio de parada detiene el algoritmo si se produce una determinada cantidad de generaciones sucesivas sin evaluar individuos. Esto ocurre cuando la población es homogénea y los parámetros genéticos no son adecuados. Por ejemplo cuando las probabilidades de cruza y mutación son excesivamente bajas. Se acepta un máximo de 2 generaciones sucesivas sin evaluación.

El último criterio se cumple cuando una población está compuesta enteramente por el mismo individuo. En este caso, se dependería exclusivamente de la mutación para generar diversidad.

Como se describió en el Capítulo 7, se utiliza Selección con Control sobre el Número Esperado junto con Elitismo. Para las *Propuestas I* y *II* se usa Cruza de Dos Puntos, mientras que en la *Propuesta III* se utiliza Cruza de Dos Puntos con Heurística. En los tres casos $P_c = 0{,}6$. El método de mutación utilizado en las *Propuestas I* y *II* es Mutación Simple con $P_m = 0{,}001$, y en la *Propuesta III* se utiliza Mutación Heurística, con $P_m = 0{,}5$ y $P_h = 0{,}9$. La heurística se define con $\varepsilon = 0{,}001$.

---

[1] En la *Propuesta I* el alfabeto es una muestra de un intervalo en $\Re$. Dado que su cardinalidad es muy grande, el mínimo P_{eq} es muy pequeño, lo cual hace que la convergencia se mantenga en valores muy bajos. Es por esto que es mas conveniente calcular P_{eq} a nivel de bit.

[2] Si P_{eq} se calculara a nivel de bit para las *Propuestas II* y *III*, habría valores de bits que siempre serían iguales lo cual haría que P_{eq} tienda a 1 en forma ficticia.

En la [Tabla 8-2] se resumen los principales parámetros genéticos utilizados en las pruebas. En la mayoría de los casos se realizaron pruebas previas para determinar valores adecuados A lo largo de este capítulo se detalla el motivo por el cuál algunos parámetros no tienen el mismo valor para las tres propuestas.

	Propuesta I	***Propuesta II***	***Propuesta III***
Experimentos	*3*	*3*	*3*
Evaluaciones	*2000*	*2000*	*2000*
Generaciones sin Evaluación	*2*	*2*	*2*
Tamaño de Población	*50*	*50*	*30*
Granularidad	*1024*	*n/a*	*n/a*
Lista de Materiales	*n/a*	$5 \leq \mu \leq 7$	$5 \leq \mu \leq 7$
P_c	*0,6*	*0,6*	*0,6*
P_m	*0,001*	*0,001*	*0,5*
P_h	*n/a*	*n/a*	*0,9*
P_l	*n/a*	*n/a*	*0,1*
ε	*n/a*	*n/a*	*0,001*

Tabla 8-2. Parámetros Genéticos

8.3 Test 1: Resultados

El *Test 1* es un perfil de 16 capas, con la última finalizando a 960m de profundidad. La fuente y el receptor se encuentran ambos ubicados a 11m de la superficie. Se dispone de una lista *Z* de 5 materiales posibles, todos con igual densidad [Tabla 8-3].

Material	*Velocidad*	*Densidad*
0	*2,00*	*2,00*
1	*4,00*	*2,00*
2	*3,00*	*2,00*
3	*2,50*	*2,00*
4	*3,50*	*2,00*

Tabla 8-3. Lista de Materiales

En la [Figura 8-2] se observa la mejor estimación del perfil para cada una de las propuestas junto con el perfil real, expresado en velocidades. Suponiendo que la velocidad para un material dado se puede estimar con un error de ±0.25 m/ms, se tiene que las lineas punteadas delimitan el rango de velocidades que identifican a cada material. Entonces, la *Propuesta I* estima correctamente el material de 12 capas (75%), la *Propuesta II* estima 14 (87,5%), y la *Propuesta III* estima todos bien (100%).

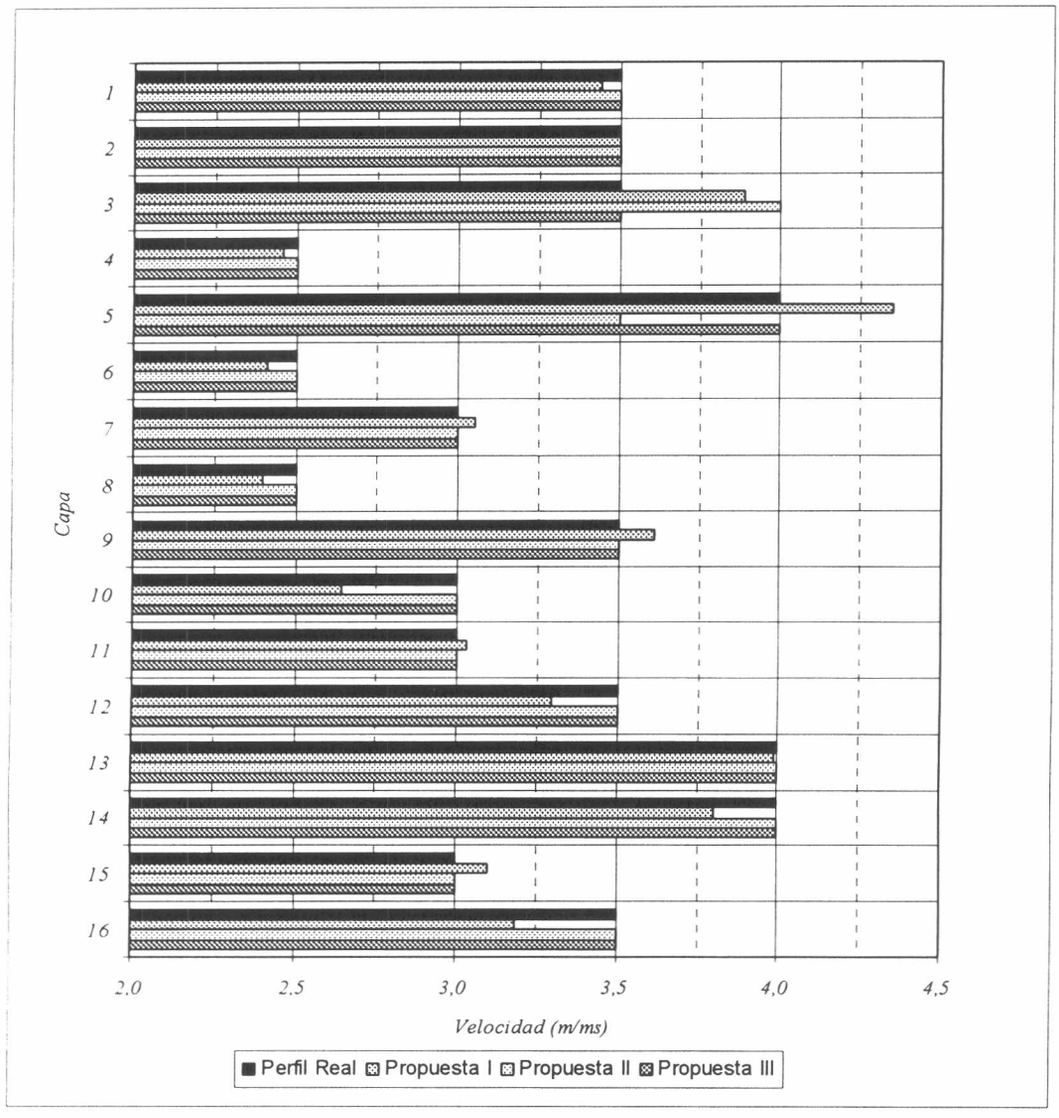

Figura 8-2. Test 1: Mejor perfil estimado por cada propuesta

En la [Tabla 8-4] se resume los resultados de los tres experimentos realizados para cada propuesta.

Propuesta	*Experimento*	*Gens*	*Evals*	P_{eq}	*Mejor Aptitud*	*Aptitud Promedio*
I	*1*	*67*	*2025*	*0,864*	*2,5435e-3*	*4,0728e-3*
	2	*65*	*2020*	*0,877*	*1,8523e-3*	*4,1725e-3*
	3	*64*	*2005*	*0,842*	*5,0831e-3*	*6,7846e-3*
II	*1*	*86*	*2000*	*0,929*	*3,6868e-3*	*4,5068e-3*
	2	*109*	*1876*	*0,986*	*8,6215e-4*	*1,0129e-3*
	3	*112*	*1965*	*0,974*	*3,1570e-3*	*3,5334e-3*
III	*1*	*43*	*573*	*0,998*	*1,7175e-13*	*2,6575e-4*
	2	*39*	*466*	*0,992*	*1,7175e-13*	*3,1596e-4*
	3	*43*	*515*	*0,996*	*1,7175e-13*	*6,5263e-5*

Tabla 8-4. Test 1: Resumen de resultados

La *Propuesta I* siempre termina al alcanzar 2000 evaluaciones, que es el máximo permitido. El total de evaluaciones no es exactamente 2000 porque la última generación siempre se evalúa completamente. La *Propuesta II* utiliza una cantidad de evaluaciones similar a la *I* pero logra un grado de convergencia P_{eq} superior. También logra mejores valores para la mejor aptitud y la aptitud promedio. En definitiva obtiene una población más homogénea y de mejor calidad. La cantidad de evaluaciones que la *Propuesta III* necesita para converger es aproximadamente la cuarta parte de la necesaria para las *Propuestas I* y *II*. El grado de convergencia es muy alto y los individuos de la población exhiben alta aptitud.

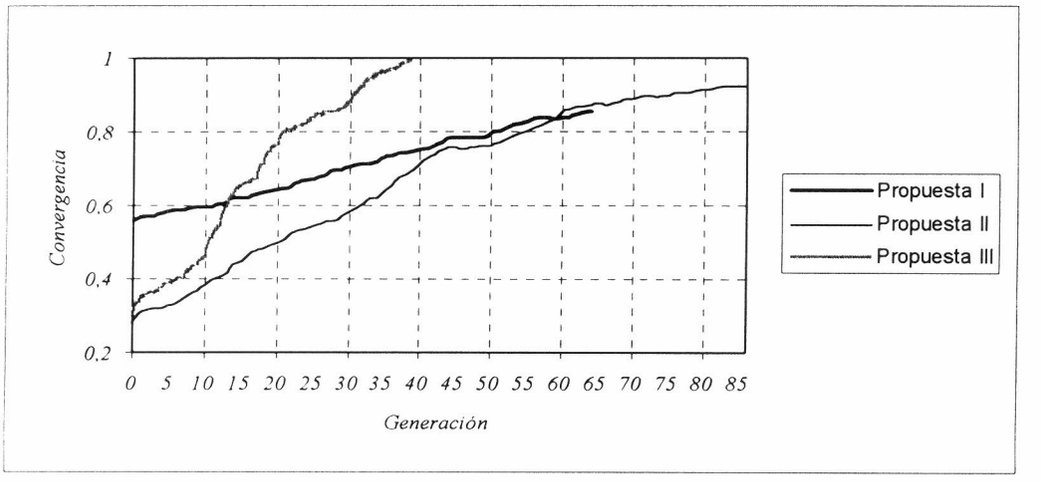

Figura 8-3. Test 1: Convergencia media poblacional

En la [Figura 8-3] se observa la evolución promedio de la población para los tres experimentos de cada propuesta. La *Propuesta I* converge muy lentamente respecto de la *Propuesta II*, y ésta respecto de la *Propuesta III*. Esta última no solo converge rápido, sino que también alcanza grados de convergencia mayores que los de las *Propuesta I* y *II*.

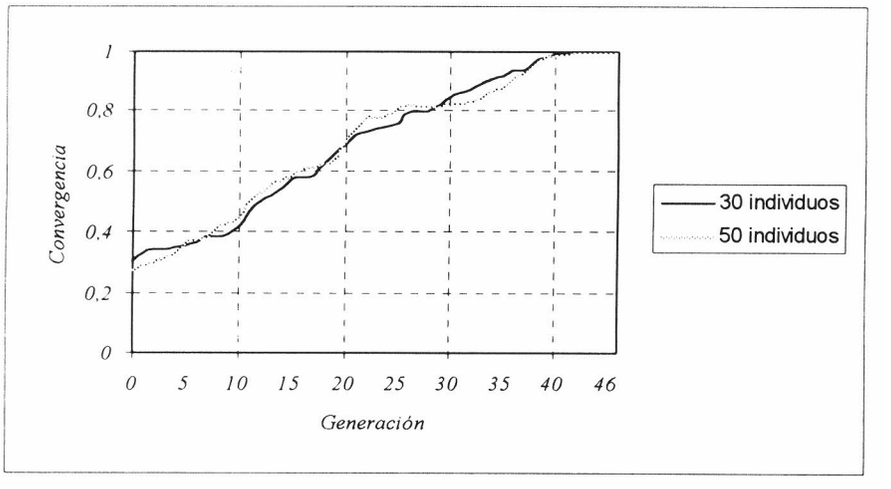

Figura 8-4. Test 1: Convergencia para 30 y 50 individuos

En la [Figura 8-4] se observa la convergencia de dos poblaciones utilizadas con la *Propuesta III*. Una con 50 individuos y la otra con 30. La velocidad de convergencia

de ambas poblaciones es similar, y con ambas se obtienen resultados idénticos. Es por esta razón que se utilizan solamente 30 individuos para las pruebas con la *Propuesta III*.

8.4 Test 2: Resultados

El *Test 2* presenta un perfil de sólo 8 capas, de 60m de espesor cada una. Tanto la fuente como el receptor se encuentran a 11m de profundidad. La característica distintiva de este test consiste en que pares de materiales de igual velocidad tienen distinta densidad. Este es el caso de los materiales {1, 6} y {2, 5} [Tabla 8-5]:

Material	*Velocidad*	*Densidad*
0	*2,00*	*2,00*
1	*4,00*	*2,00*
2	*3,00*	*2,00*
3	*2,50*	*2,00*
4	*3,50*	*2,00*
5	*3,00*	*1,00*
6	*4,00*	*1,00*

Tabla 8-5. Materiales para el Test 2

En la [Figura 8-5] se observa las estimaciones para cada propuesta, comparadas con el perfil real. Al igual que en el *Test 1*, si se asume un error de ±0,25 m/ms para identificar la velocidad correspondiente a cada material, se tiene que la *Propuesta I* identifica 7 de los 8 materiales (87,5%), aunque con un gran error en las capas 2, 4, 5 y 7. Las *Propuestas II* y *III* identifican correctamente los 8 materiales (100%).

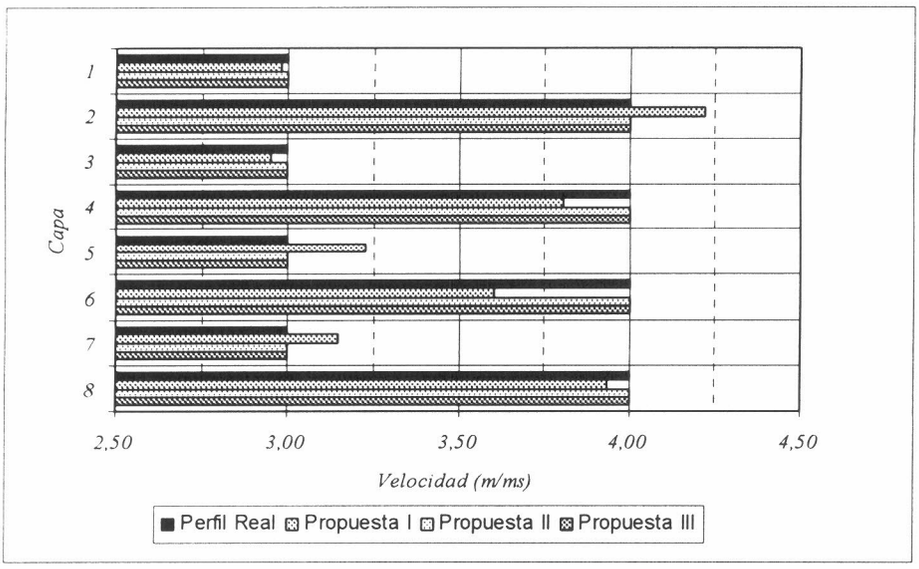

Figura 8-5. Test 2: Mejor perfil estimado por cada propuesta

Si bien los materiales {1, 6} y {2, 5} tienen igual velocidad, tanto la *Propuesta II* como la *III* determinan correctamente la densidad, por lo tanto el material queda completamente identificado.

Propuesta	*Experimento*	*Gens*	*Evals*	P_{eq}	*Mejor Aptitud*	*Aptitud Promedio*
I	*1*	*68*	*2005*	*0,753*	*1,0363e-3*	*2,9026e-3*
	2	*68*	*2018*	*0,784*	*9,1062e-4*	*2,5501e-3*
	3	*68*	*2017*	*0,861*	*2,9446e-4*	*6,8847e-4*
II	*1*	*87*	*1962*	*0,938*	*6,4678e-4*	*9,2322e-4*
	2	*81*	*1806*	*0,965*	*1,1199e-12*	*2,3547e-4*
	3	*75*	*1453*	*0,943*	*7,9144e-4*	*8,7618e-4*
III	*1*	*31*	*390*	*0,958*	*1,1199e-12*	*2,2668e-4*
	2	*27*	*319*	*0,967*	*1,1199e-12*	*5,3711e-4*
	3	*23*	*286*	*0,938*	*1,1199e-12*	*4,3174e-3*

Tabla 8-6. Test 2: Resumen de resultados

En la [Tabla 8-6] se detalla los resultados de las tres propuestas para el *Test 2*. Se observan claramente las mejoras de la *Propuesta II* respecto de la *I* y de la *III* respecto de la *II*. La *Propuesta II* logra una población bastante más homogénea que la *I*, utilizando menos evaluaciones y parando antes del máximo permitido. Los experimentos para la *Propuesta II* sugieren que el mínimo es bastante difícil de hallar. Sin embargo, la heurística de la *Propuesta III* permite llegar a él en los tres experimentos, con un grado de convergencia muy alto y con muy pocas evaluaciones respecto de las propuestas anteriores.

En la [Figura 8-6] se observa para las *Propuestas I* y *II* un comportamiento similar en la evolución de la población. La *Propuesta III* alcanza en pocas evaluaciones una población altamente homogénea.

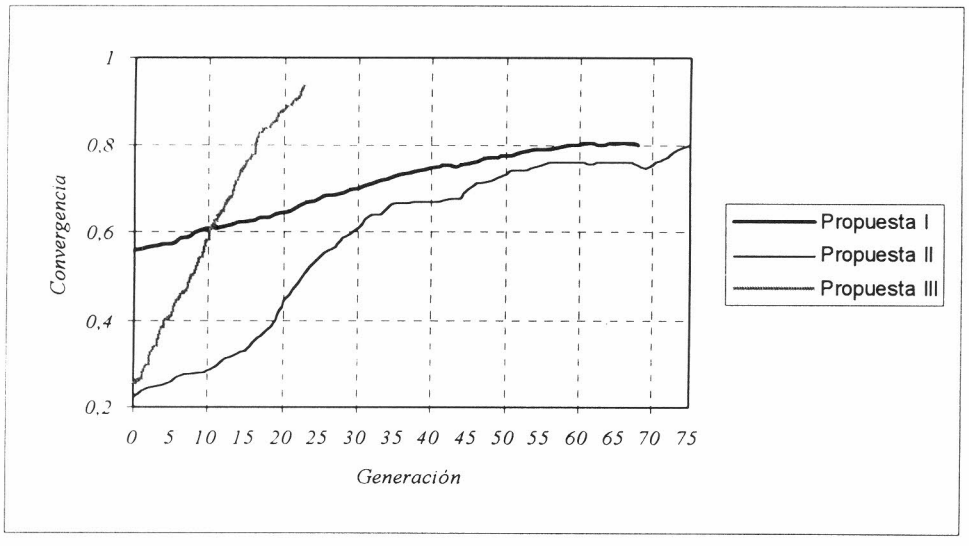

Figura 8-6. Test 2: Convergencia media poblacional

8.5 Test 3: Resultados

El *Test 3* consiste en un perfil de 8 capas, similar al *Test 2*, pero con densidad constante. La lista de materiales coincide con la utilizada en el *Test 1* [Tabla 8-3]. En la [Figura 8-7] se observa los perfiles estimados para cada propuesta. En este caso, las tres logran identificar correctamente los materiales asumiendo, como siempre, un error de ±0,25 m/ms para la *Propuesta I*.

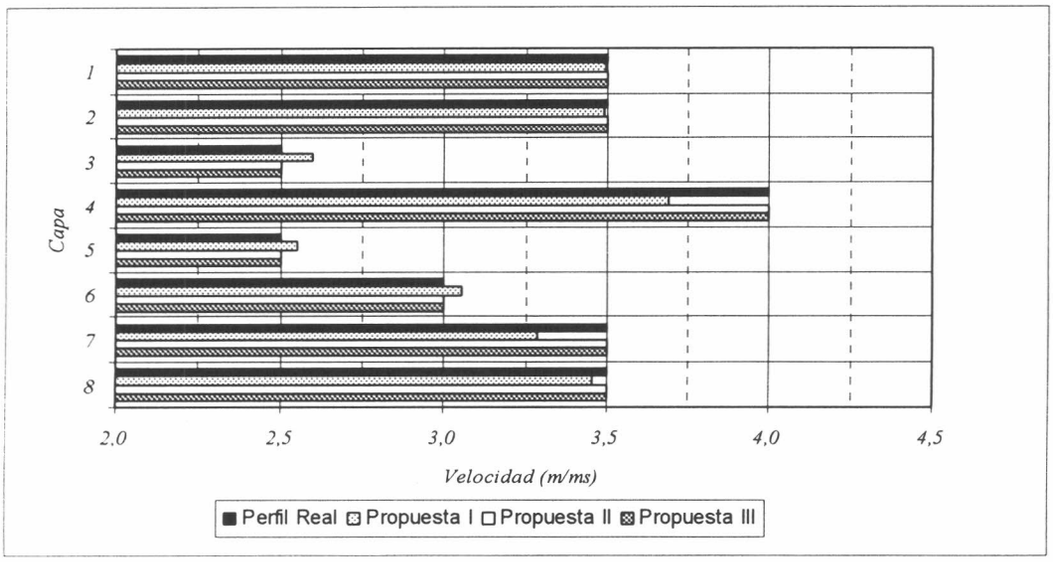

Figura 8-7. Test 3: Mejor perfil estimado por cada propuesta

La *Propuesta I* identifica 7 de las 8 capas con un error alto en la séptima capa. Las *Propuestas II* y *III* identifican todas las capas con un buen grado de convergencia.

Propuesta	*Experimento*	*Gens*	*Evals*	P_{eq}	*Mejor Aptitud*	*Aptitud Promedio*
I	*1*	*67*	*2003*	*0,832*	*1,1200e-4*	*1,1044e-3*
	2	*67*	*2006*	*0,818*	*1,4055e-4*	*9,1989e-4*
	3	*68*	*2012*	*0,850*	*1,9954e-4*	*5,7054e-4*
II	*1*	*80*	*1576*	*0,900*	*2,2199e-4*	*4,0018e-4*
	2	*91*	*1432*	*0,943*	*7,4153e-13*	*8,4684e-5*
	3	*110*	*1306*	*0,935*	*4,4146e-4*	*5,0983e-4*
III	*1*	*26*	*322*	*0,988*	*7,4153e-13*	*3,4686e-4*
	2	*28*	*273*	*0,992*	*7,4153e-13*	*1,2254e-3*
	3	*32*	*347*	*0,996*	*7,4153e-13*	*6,2608e-6*

Tabla 8-7. Test 3: Resumen de resultados

La cantidad de evaluaciones que requiere la *Propuesta III* para alcanzar el mínimo es muy baja respecto de las otras dos propuestas. La velocidad de convergencia de la población respecto de la *Propuesta II* es un argumento a favor de la heurística utilizada.

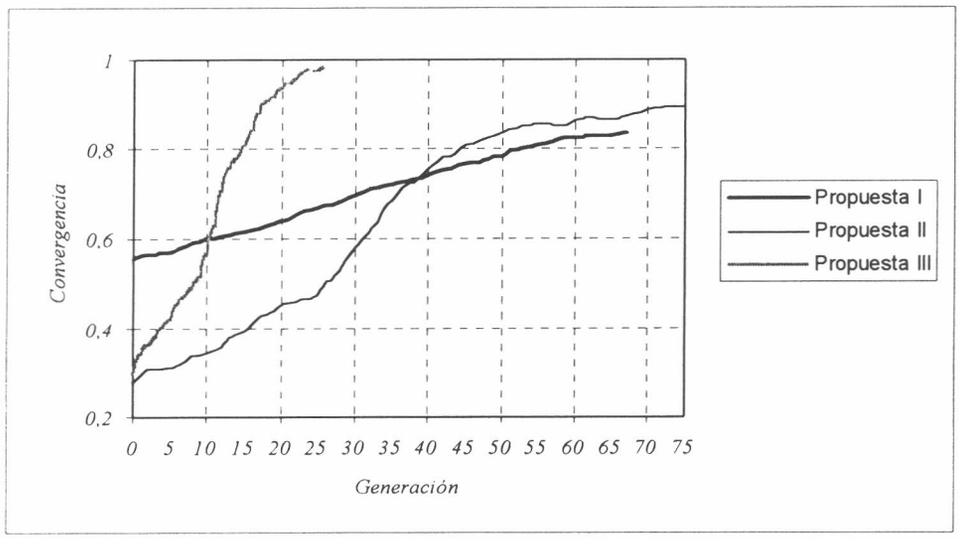

Figura 8-8. Test 3: Convergencia media poblacional

En la [Figura 8-8] se observa la gran ventaja de la *Propuesta III* respecto de las otras dos, al igual que lo observado en los test anteriores.

8.6 Test 4: Resultados

Los materiales utilizados para este test son los mismos que para los *Test 1* y *3*, [Tabla 8-3]. El emisor y el receptor están ubicados más cerca de la superficie, a 6m de profundidad. En la [Figura 8-9] se observa el perfil real y los estimados para este test. Con un error de ±0,25 m/ms para la *Propuesta I* se tiene que los materiales para las capas 6 y 7 no fueron identificados (75% de acierto). Las *Propuestas II* y *III* sí identifican correctamente todas las capas (100%).

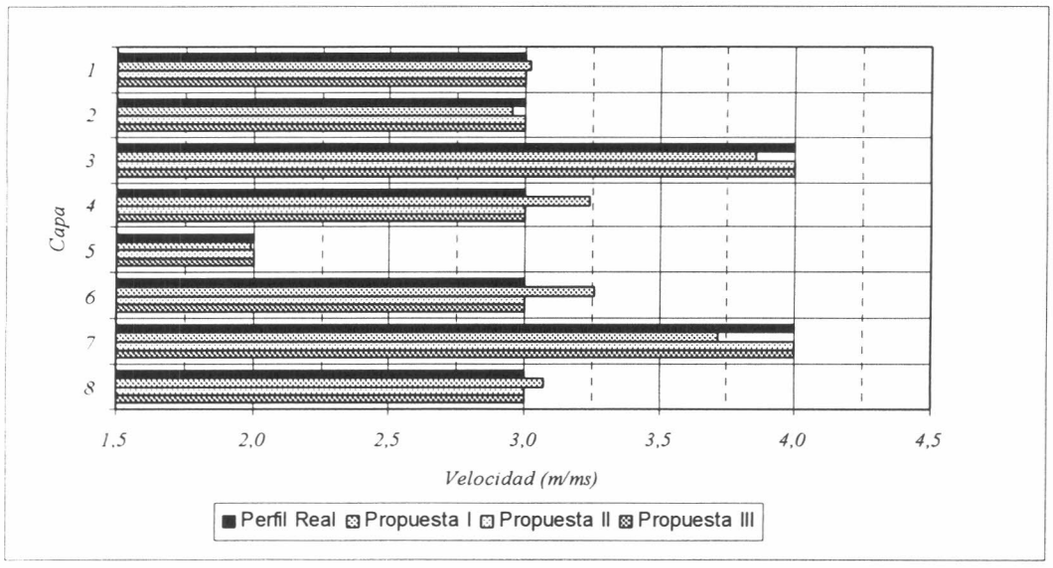

Figura 8-9. Test 4: Mejor perfil estimado por cada propuesta

En la [Tabla 8-8] se observan los resultados de las tres propuestas para el *Test 4*. La *Propuesta I* es la única que alcanza el máximo permitido de evaluaciones. La *Propuesta III* necesita aproximadamente la cuarta parte de evaluaciones que las necesarias en la *Propuesta II* y, aún así, logra un grado de convergencia superior.

Propuesta	*Experimento*	*Gens*	*Evals*	P_{eq}	*Mejor Aptitud*	*Aptitud Promedio*
I	*1*	*66*	*2012*	*0,737*	*3,5579e-3*	*1,6921e-2*
	2	*67*	*2015*	*0,738*	*2,0296e-3*	*1,1187e-2*
	3	*68*	*2012*	*0,791*	*7,3152e-4*	*5,7108e-3*
II	*1*	*98*	*1359*	*0,983*	*9,0431e-13*	*2,8870e-4*
	2	*76*	*1162*	*0,993*	*9,0431e-13*	*1,0455e-4*
	3	*50*	*963*	*0,988*	*9,0431e-13*	*4,0778e-4*
III	*1*	*33*	*339*	*0,983*	*9,0431e-13*	*9,0001e-4*
	2	*28*	*315*	*0,992*	*9,0431e-13*	*1,3658e-4*
	3	*28*	*308*	*0,992*	*9,0431e-13*	*1,1461e-3*

Tabla 8-8. Test 4: Resumen de resultados

La [Figura 8-10] confirma las tendencias observadas en los test anteriores respecto a la evolución de la población para cada propuesta. La *Propuesta III* presenta una velocidad de convergencia mucho mayor a la *Propuesta II*, y ésta mayor que la *Propuesta I*.

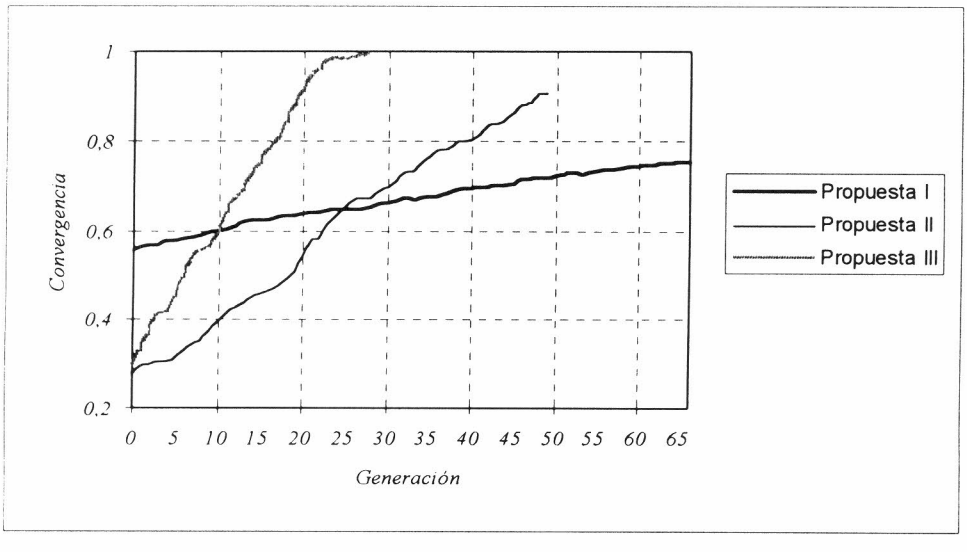

Figura 8-10. Test 4: Convergencia media poblacional

8.7 Test 5: Resultados

El *Test 5* es un perfil de 14 capas, con una lista de 5 materiales posibles [Tabla 8-3] y densidad constante. La profundidad del perfil es de 960m. La fuente y el receptor se ubican a 11m de profundidad. En la [Figura 8-11] se observa que la *Propuesta I* identifica 10 de las 14 capas (71%). La *Propuesta II* estima correctamente 13 de las 14 capas (92%), puesto que la capa 8 no logra identificarla. Y la *Propuesta III* estima bien las 14 capas (100%).

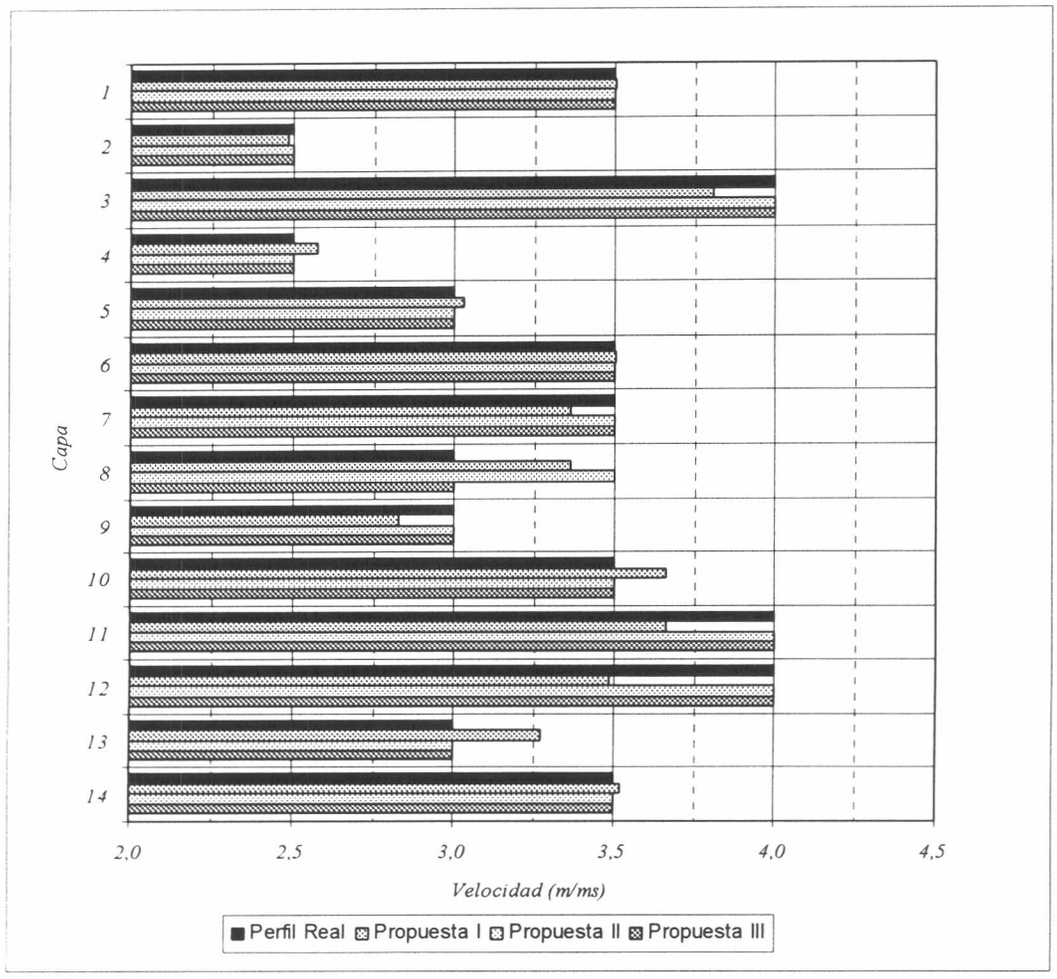

Figura 8-11. Test 5: Mejor perfil estimado por cada propuesta

La *Propuesta II* no logra alcanzar el mínimo global, aún cuando utiliza la mayor parte de las 2000 evaluaciones disponibles y alcanza un grado de convergencia alto [Tabla 8-9].

Propuesta	*Experimento*	*Gens*	*Evals*	P_{eq}	*Mejor Aptitud*	*Aptitud Promedio*
I	*1*	*67*	*2013*	*0,880*	*1,0733e-3*	*2,2268e-3*
	2	*66*	*2024*	*0,818*	*8,3316e-4*	*2,7059e-3*
	3	*66*	*2017*	*0,888*	*2,0159e-3*	*3,8072e-3*
II	*1*	*70*	*1347*	*0,949*	*1,1882e-3*	*1,6059e-3*
	2	*95*	*2009*	*0,886*	*5,8919e-4*	*1,3452e-3*
	3	*106*	*2001*	*0,926*	*6,9635e-4*	*1,4743e-3*
III	*1*	*35*	*418*	*0,998*	*6,7214e-13*	*7,9445e-4*
	2	*38*	*458*	*0,995*	*6,7214e-13*	*8,4233e-4*
	3	*47*	*543*	*0,998*	*6,7214e-13*	*6,1855e-5*

Tabla 8-9. Test 5: Resumen de resultados

En la [Figura 8-12] se observa la velocidad de convergencia muy alta de la *Propuesta III* respecto de la *II* y de la *I*.

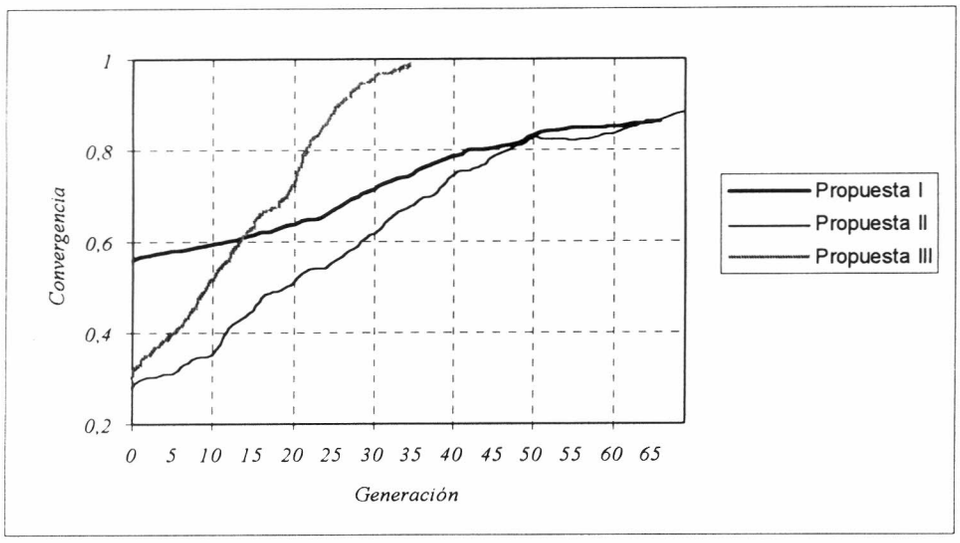

Figura 8-12. Test 5: Convergencia media poblacional

8.8 Simulaciones con Ruido

Dado los buenos resultados obtenidos con la *Propuesta III*, se realizaron los cinco test nuevamente pero incorporando ruido a la secuencia de presiones observadas. De esta manera se tiene una simulación más cercana a la realidad. Para lograr esto se incorporó un 1% de ruido blanco a la secuencia $p^{obs}(x_r, t)$.

Para una secuencia de presiones observadas con ruido se debe tener en cuenta que la mejor aptitud posible para cada test se aleja del valor observado cuando la secuencia no tiene ruido. Esto ocurre porque la aptitud se calcula comparando la presión observada (con ruido) y la calculada para un individuo/perfil dado. Aún cuando el individuo que se analiza sea el buscado, la presión calculada que éste representa no está alterada por ruido. Entonces siempre habrá diferencia entre ambas presiones y ésta se incrementa en la medida en que el ruido aumenta.

Para la *Propuesta III* en particular, el parámetro ε se debe ajustar de tal manera que sea menos exigente en la medida que haya más ruido en la secuencia $p^{obs}(x_r, t)$. Esto significa que el valor de ε a utilizar para una simulación con ruido debe ser mayor que el utilizado para la misma simulación pero sin ruido.

Test	*Mejor Experimento*	ε	*Gens*	*Evals*	P_{eq}	*Mejor Aptitud*	*Aptitud Promedio*
1	*1*	*0,055*	*31*	*402*	*0,990*	*1,8116e-4*	*2,1658e-4*
2	*2*	*0,01*	*28*	*277*	*0,954*	*5,7705e-5*	*3,4391e-4*
3	*3*	*0,01*	*32*	*328*	*1,000*	*5,2465e-5*	*5,2465e-5*
4	*3*	*0,01*	*151*	*2012*	*0,729*	*1,7394e-4*	*9,7996e-3*
5	*3*	*0,055*	*24*	*296*	*0,993*	*1,8831e-4*	*4,8360e-4*

Tabla 8-10. Resultados para la *Propuesta III* con ruido blanco

En la [Tabla 8-10] se observan los resultados para cada uno de los test, con un 1% de ruido blanco incorporado en $p^{obs}(x_r, t)$. En los cinco test el algoritmo determina correctamente los materiales para todas las capas.

En la [Figura 8-13] se observa la secuencia de presiones observadas con 1% de ruido blanco para el *Test 1*.

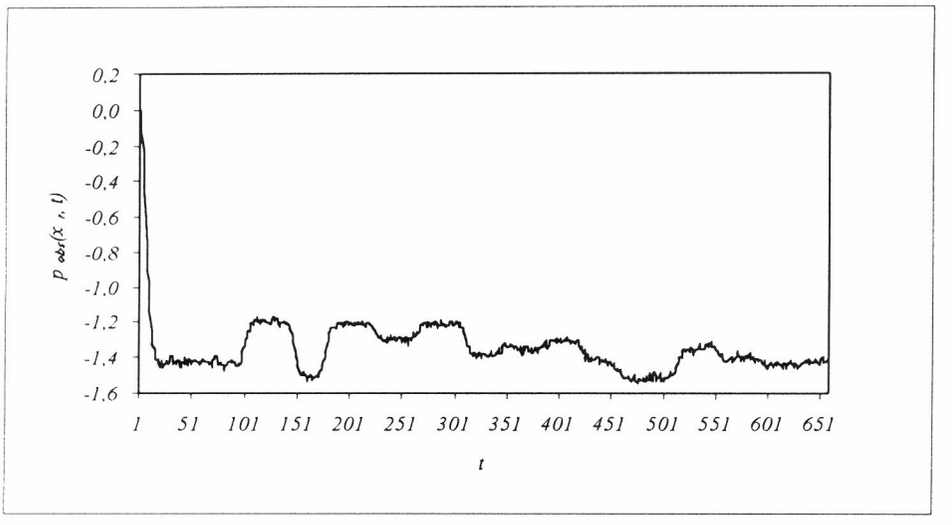

Figura 8-13. Test 1: Presión observada con 1% de ruido

En la [Figura 8-14] se observa la evolución de la población para el *Test 1* con distintos valores del parámetro ε. Cuando ε es demasiado grande, ε= 1, todos los alelos son considerados buenos y la cruza produce hijos exactamente iguales a sus padres. Esto tiene el efecto de converger prematuramente casi sin evolucionar. Si ε es demasiado pequeño, ε= 0,01, la evolución se torna lenta por la aparición excesiva de falsos negativos. Con valores de 0,1 y 0,055 se observa que la población evoluciona una cantidad razonable de generaciones, alcanzando altos grados de convergencia.

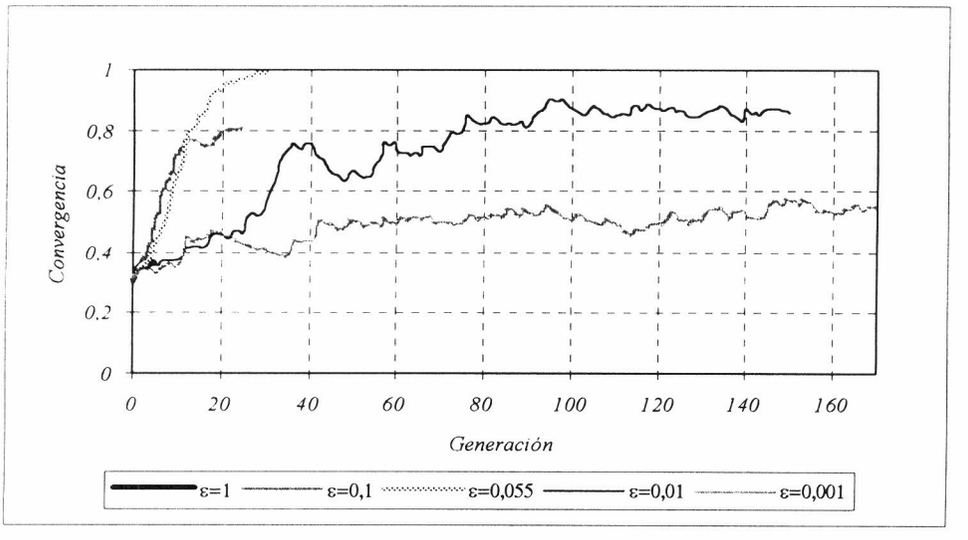

Figura 8-14. Test 1: Convergencia poblacional para distintos ε

8.9 Conclusión

Los resultados presentados en este capítulo confirman que AG es una muy buena alternativa para resolver el problema de inversión, como se propone en el Capítulo 6. Así mismo reafirman las suposiciones planteadas en el Capítulo 7 respecto de las características de cada propuesta. La *Propuesta I*, primer intento de solución, utiliza hipótesis similares a las supuestas por el Método de Cuasilinearización [FER/93] y logra resultados con una gran costo computacional. Esto se observa en la gran cantidad de evaluaciones necesarias que, aún así, no logran que la población alcance un alto grado de convergencia. Si bien la *Propuesta I* identifica la mayoría de las capas que componen el perfil, nunca logra identificar todas, y lo hace a expensas de un error bastante alto. La *Propuesta I* por sí sola no es adecuada para obtener una solución aceptable para el problema de inversión. Sin embargo, las soluciones obtenidas son suficientemente buenas como para que otro algoritmo de búsqueda local, como el Método de Cuasilinearización, las utilice como punto de partida para la búsqueda. Este último necesita cumplir restricciones mayores que la *Propuesta I*, por lo tanto una solución híbrida es bastante adecuada.

La *Propuesta II* utiliza otras restricciones respecto de la *Propuesta I* y cambia la semántica del cromosoma. Esto permite reducir el espacio de búsqueda, a la vez que se logra identificar correctamente el perfil en la mayoría de los tests. Al final de cada prueba el grado de convergencia de la población es alto, lo cual significa que la mayoría de los individuos constituyen soluciones potenciales de buena calidad.

La *Propuesta III* agrega conocimiento del problema a la *Propuesta II*. Esto logra reducir el costo computacional puesto que la cantidad de evaluaciones necesarias para alcanzar la solución se reduce a un 25% aproximadamente respecto de la *Propuesta II*. El grado de convergencia de la población es aún mayor, identificando correctamente el 100% de las capas en todas las pruebas. Las simulaciones realizadas con ruido confirman que la *Propuesta III* es una muy buena solución para el problema de inversión.

El método de resolución de problemas con AG presentado en el Capítulo 4 se puede aplicar con éxito a un problema considerado tradicionalmente como "difícil". Si bien no es un método riguroso, constituye una guía adecuada para diseñar un AG que resuelva un problema dado. El hecho de que se trate de un método iterativo hace que no sea imprescindible un conocimiento exhaustivo del problema previo al diseño del algoritmo.

Algoritmos Genéticos es un método adecuado para problemas de optimización, multimodales, con gran cantidad de variables, y espacios de búsqueda amplios, donde los métodos de búsqueda local encuentran dificultades para hallar la solución. En los capítulos 6, 7 y 8 se pretendió mostrar la efectividad de una solución genética para un problema de estas características, diseñada siguiendo el método presentado en el Capítulo 4.

Conclusión

El estudio de Algoritmos Genéticos realizado en este trabajo abarcó la descripción del algoritmo básico, las variantes más conocidas de los operadores genéticos y las propiedades de los mismos. Se resaltó la importancia de conocer las ventajas y desventajas de cada variante para lograr un buen diseño de algoritmo, apto para resolver cada problema particular en forma eficaz y eficiente. El estudio de la teoría de esquemas y el Teorema Fundamental permitió dar un marco formal a los Algoritmos Genéticos, a la vez que ayudó a comprender su funcionamiento. El análisis de Holland [HOL/75] se basa en un Algoritmo Genético simple, sin embargo su importancia radica en que constituye la base de toda la teoría desarrollada para otras variantes de operadores.

El comportamiento evolutivo y las variantes sobre los operadores son aquellos temas en los que se focaliza en la mayor parte de la bibliografía sobre Algoritmos Genéticos. De menor trascendencia, aunque no menos importante, son la representación de parámetros en un cromosoma y las características deseables para la función de aptitud. Ambos aspectos pueden influir decisivamente en la eficiencia y por esto se estudiaron en profundidad en este trabajo.

La codificación binaria para representar parámetros no necesariamente es la mejor, si bien es la más difundida. En el presente trabajo se mostró que pueden obtenerse importantes mejoras utilizando codificaciones que se adapten mejor al problema a resolver.

Un Algoritmo Genético es un algoritmo modular y, como tal, cada uno de sus componentes puede ser implementado de diferente manera. Estos componentes son: la estructura de los individuos de la población, los operadores genéticos y la función de aptitud. Una buena elección de cada componente para un problema particular que tome en cuenta las relaciones entre los mismos, llevará a un diseño adecuado para hallar la solución. Esta elección está fuertemente condicionada por el conocimiento del problema a resolver, lo cual no quita generalidad a Algoritmos Genéticos sino que lo convierte en un método a partir del cual es posible derivar múltiples algoritmos que se adecuen a los problemas particulares que se pretenda resolver.

En este trabajo se presentó un método de resolución de problemas mediante Algoritmos Genéticos. Este método es iterativo, donde en cada iteración se avanza en el conocimiento del problema y se evoluciona hacia una mejor solución. La iteración no es completa sino que se puede retroceder a una o varias etapas anteriores. El análisis del comportamiento del algoritmo y de los resultados obtenidos determinan a qué etapa previa de la iteración se retrocede, o si la solución obtenida es suficientemente buena.

En la segunda parte de esta tesis se mostró la resolución de un problema "difícil" mediante Algoritmos Genéticos, utilizando el método de diseño descripto. El problema de Inversión de la Ecuación de Onda Acústica plantea la minimización de una función multidimensional y multimodal, lo cual hace que sea difícilmente tratable

por los métodos de búsqueda local. Dadas estas características se propuso resolver dicho problema mediante Algoritmos Genéticos aplicando el método de diseño descripto en la primer parte. De esta manera se realizó un análisis inicial del problema y se planteó la *Propuesta I* como primer diseño de Algoritmo Genético destinado a resolverlo. Este primer diseño se derivó directamente del Método de Cuasilinearización [FER/93a] que realiza la inversión mediante un método de búsqueda local. Las ventajas de la *Propuesta I* respecto del Método de Cuasilinearización son basicamente dos: se realiza una búsqueda global sin depender de una buena estimación inicial y se minimiza la función objetivo original, no una aproximación. El análisis de los resultados obtenidos y una revisión de las características del problema llevaron a cambiar una de las componentes del Algoritmo Genético, la estructura del cromosoma, dando como resultado la *Propuesta II*. El cambio en la representación permitió reducir drásticamente el espacio de búsqueda, lográndose mejores resultados que en la *Propuesta I* si bien la convergencia del algoritmo es igualmente lenta. Una ventaja adicional lograda consiste en que no es necesario fijar la densidad de cada capa sino que ésta pasa a ser incógnita y se puede estimar sin afectar la eficiencia.

Una nueva revisión del problema, junto con la intención de mejorar la convergencia llevaron a incorporar conocimiento específico del problema en el algoritmo en la *Propuesta III*. Esto se realizó ampliando el concepto de función de aptitud de tal forma de calificar a cada gen que compone un individuo. De esta manera la aptitud de un individuo A es un par ordenado:

$$(f(A),\gamma(A)) = \left(f(A),\left(\gamma(A_1),\ldots,\gamma(A_l)\right)\right)$$

El cálculo de la función de aptitud extendida sigue siendo dependiente del problema a resolver. La función $f(A)$ es la utilizada por el operador de selección de la manera habitual. El conocimiento aportado por la función $\gamma(A)$ es explotado por los operadores de cruza y mutación. En este trabajo se propuso dos métodos nuevos de cruza y mutación que utilizan tal conocimiento: Cruza Multipunto con Heurística y Mutación Heurística. Por supuesto es posible diseñar nuevas variantes de operadores que utilicen de otra forma la función $\gamma(A)$. Como resultado de estas innovaciones el algoritmo resultante, la *Propuesta III*, presentó una velocidad de convergencia superior y resultados de mayor precisión que la *Propuesta II*.

Algoritmos Genéticos es, en definitiva, un método de búsqueda global sin un dominio de aplicación específico. Estas características hacen que no sea adecuado para aquellos problemas sobre el que se dispone de un método determinístico ad-hoc para resolverlo puesto que seguramente no puede competir en eficiencia. Las mismas características hacen, por otra parte, que Algoritmos Genéticos sea aplicable en una variedad muy amplia de problemas. La combinación con un método de búsqueda local permite ganar en precisión cuando el espacio de búsqueda ha sido suficientemente podado por un Algoritmo Genético. La potencia de un Algoritmo Genético lo convierte en un método que debe ser tenido en cuenta cuando se pretende realizar una búsqueda sobre un dominio de aplicación dado.

Una extensión posible al presente trabajo consiste en un estudio más amplio de los operadores heurísticos de cruza y mutación presentados. Una alternativa consiste

en asignar un "peso" a cada material para las *Propuestas II* y *III* de acuerdo a la probabilidad geológica de ocurrencia del mismo en un terreno dado. Este peso puede influir en la supervivencia de un material en determinada capa a dichos operadores. Otra posibilidad consiste en utilizar esta información en una función de penalización que afecte a la aptitud de cada individuo de la población. Esto implicaría que un individuo "geológicamente improbable" tenga pocas probabilidades de sobrevivir al operador de selección. Una extensión más sofisticada consiste en la resolución del problema de inversión para los casos de dos y tres dimensiones.

Bibliografía

La mayor parte del material sobre Algoritmos Genéticos fue extraída de los archivos del Illinois Genetic Algorithms Laboratory, University of Illinois (ftp://gal4.ge.uiuc.edu/pub/papers), y del Navy Center for Applied Research in AI, Naval Research Laboratory (ftp://ftp.aic.nrl.navy.mil/pub); ambos de Estados Unidos. A este último pertenece, también, la lista de correo electrónico dedicada a Algoritmos Genéticos GA-List@aic.nrl.navy.mil, la cual se ha seguido con sumo interés desde el año 1994 hasta la fecha. Otras publicaciones y archivos consultados corresponden a: IEEE (The Institute of Electrical and Electronics Engineers), ACM (Asociation for Computing Machine), MIT (Masachussets Institute of Technology).

La principal fuente de información para el problema de la Estimación de Parámetros en Medios Acústicos fue suministrada por Elena Fernández, del CONICET, División de Aplicaciones Científicas, C.C.C, Comisión Nacional de Energía Atómica.

[BAC/97] Thomas Bäck, Ulrich Hammel, Hans-Paul Schwefel
"Evolutionary Computation: Comments on the History and Current State"
IEEE Transactions on Evolutionary Computation, Vol 1, No. 1
April 1997

[BOS/96] Fabio Boschetti, Mike C. Dentith, Ron D. List
"Inversion of Seismic Refraction Data Using Genetic Algorithm"
Geophysics, Vol 61, No. 6, November-December 1996

[BOS/97] Fabio Boschetti, Mike C. Dentith, Ron D. List
"Inversion of Potential Field Data by Genetic Algorithms"
Geophysical Prospecting, 45, 461-478, 1997

[CAN/95] Erick Cantú-Paz
"A Summary of Research on Parallel Genetic Algorithms"
IlliGAL Report No. 95007, University of Illinois
July 1995

[DEJ/92] Kenneth A. De Jong, William M. Spears
"A Formal Analysis of the Role of Multi-Point Crossover in Genetic Algorithms"
Annals of Mathematics and Artificial Intelligence Journal, Vol 5, No. 1
1992

[FER/93a] Elena M. Fernandez, Patricia Gauzellino, Juan E. Santos, Dongwoo Sheen
"Parameter Estimation in Multidimensional Acoustic Media"
Technical Report #209
Center for Applied Mathematics, Purdue University, 1993

[FER/93b] Elena M. Fernandez, Patricia Gauzellino, Juan E. Santos
"An Algorithm for Parameter Estimation in Acoustic Media, Practical Issues"
1993

[FOG/94] David B. Fogel
"An Introduction to Simulated Evolutionary Optimization"
IEEE Transactions on Neural Networks, Vol 5, No. 1, pag 3-14, 1994

[GOL/89] David E. Goldberg
"Genetic Algorithms in Search, Optimization, and Machine Learning"
Addison-Wesley Publishing Company, 1989

[GOL/93] David E. Goldberg
"Making Genetic Algorithms fly: a lesson from the Wright brothers"
Advanced Technology for Developers, 2, 1-8
February 1993

[GOL/94] David E. Goldberg
"Genetic and Evolutionary Algorithms Come of Age"
Communications of the ACM, Vol. 37, No. 3, March 1994

[GOL/95] David E. Goldberg, Brad L. Miller
"Genetic Algorithms, Selection Schemes, and the Varying Effects of Noise"
IlliGAL Report No. 95009, University of Illinois, 1995

[GRE/90] John J. Grefenstette
A User's Guide to GENESIS V5.0
October 1990

[HOL/75] John H. Holland
"Adapatation in Natural and Artificial Systems"
The University of Michigan Press, 1975

[HOL/94] John H. Holland
"Schemata"
GA-List, GA Vol. 8 No. 26, 1994

[HOR/94] Jeffrey Horn, David E. Goldberg
"Genetic Algorithm Difficulty and the Modality of Fitness Landscapes"
IlliGAL Report No. 94006, University of Illinois, 1994

[MAH/93] Samir W. Mahfoud
"Simple Analytical Models of Genetic Algorithms for Multimodal Function Optimization"
IlliGAL Report No. 93001, University of Illinois, 1993

[MAH/95] Samir Mahfoud
"Niching Methods for Genetic Algorithms"
IlliGAL Report No. 95001, University of Illinois, 1995

[MIL/95] Brad L. Miller, David E. Goldberg
"Genetic Algorithms, Tournament Selection, and the Effects of Noise"
IlliGAL Report No. 95006, University of Illinois, 1995

[NOL/94] B. Nolte, L. N. Frazer
"Vertical Seismic Profile Inversion with Genetic Algorithms"
Geophysics J., Int. 117, 162-178, 1994

[SAM/86] A.A. Samarski
"Introducción a los métodos numéricos"
Editorial Mir Moscú, 1986

[SPE/91] William M. Spears, Vic Anand
"A study of Crossover Operators in Genetic Programming"
Navy Center for Applied Research in AI, Naval Research Laboratory
Sixth International Symposium on Methodologies for Intelligent Systems, Charlotte, NC, pag. 409-418, October 1991

[SPE/92] William M. Spears
"Crossover or Mutation?"
Proceedings of the Foundations of Genetic Algorithms Workshop
Vail, Colorado, pag. 221-237, July 1992

[SPE/93] William M. Spears, Keneth A. De Jong, Thomas Back, David B. Fogel, Hugo de Garis
"An Overview of Evolutionary Computation"
Artificial Intelligence Center Internal Report #AIC-92-030
Naval Research Laboratory, Washington DC, 1993

[SPE/94] William M. Spears
"Simple Subpopulation Schemes"
Artificial Intelligence Center Internal Report #AIC-93-020
Naval Research Laboratory, Washington DC, 1994

[SPE/95] William M. Spears
"Adapting Crossover in Evolutionary Algorithms"
Artificial Intelligence Center Internal Report #AIC-94-025
Naval Research Laboratory, Washington DC, 1995

[THI/94] Dirk Thierens, David E. Goldberg
"Elitist Recombination: an integrated selection recombination GA"
ICEC'94. 508-512
IlliGAL Publication [30], 1994

[WOL/97] David H. Wolpert, William G. Macready
"No Free Lunch Theorems for Optimization"
IEEE Transactions on Evolutionary Computation, Vol 1, No. 1
April 1997

Anexo A

Manual del usuario

El sistema *Algen* implementa la solución propuesta en esta tesis para resolver el problema de inversión de la ecuación de onda acústica. Para esto, el sistema permite simular datos de campo a partir de un modelo del terreno, estimar un perfil del mismo mediante alguna de las tres propuestas de Algoritmos Genéticos desarrolladas, analizar los resultados obtenidos, la evolución de la población y comparar los perfiles calculados con el perfil real.

A.1 Contenido del Diskette de Instalación

El diskette de instalación contiene un directorio con los ejecutables del sistema y otro directorio con juegos de datos de prueba.

En el directorio ***exes*** se encuentran los siguientes archivos:

algen.exe → *Programa principal*
ag1.exe → *Implementación de la Propuesta I*
ag2.exe → *Implementación de la Propuesta II*
ag3.exe → *Implementación de la Propuesta III*
recursos.rez → *Archivo de datos. Contiene las pantallas y menúes utilizados.*

En el directorio ***tests*** se encuentran los archivos con los modelos de terreno utilizados para el testeo del algoritmo. Las características de los mismos y los resultados obtenidos se detallan en el Capítulo 8.

Modelo.t1 → *Test 1*
Modelo.t2 → *Test 2*
Modelo.t3 → *Test 3*
Modelo.t4 → *Test 4*
Modelo.t5 → *Test 5*

En el segundo diskette se proveen los fuentes del sistema y los archivos con los resultados obtenidos en cada prueba.

A.2 Instalación

Para instalar el sistema *Algen* desde una disquetera, por ejemplo A:, al disco rígido C: ejecutar el programa de instalación incluído en el primer disquette:

A:> instala A: C:\Algen

El instalador copia los programas necesarios y archivos de prueba al directorio destino indicado.

A.3 Requerimientos

El programa se puede ejecutar sobre DOS versión 3.0 o superior. Requiere 580 K de memoria libre, un microprocesador 386 o superior y 700 KB de espacio libre en disco. Es recomendable disponer de cache de disco (tipo Smartdrv). La aplicación está desarrollada en lenguaje C++ (Borland C++) con librerías TurboVision para manejo de pantallas.

A.4 Descripción General del Sistema

La función del sistema Algen consiste en la estimación del perfil sísmico vertical tomando como entrada un conjunto de datos de campo y aplicando alguna de las tres propuestas de Algoritmo Genético desarrolladas en el Capítulo 7.

Dado que no siempre se dispone de datos reales de campo, estos se pueden sintetizar con el mismo sistema, a partir de un modelo de terreno. Para esto se definen las características del terreno y de la simulación en un archivo denominado *modelo.xxx*. Tomando como entrada el modelo del terreno se simula la generación y propagación de una onda, la cual se registra en un receptor ubicado cerca de la superficie. El resultado de la simulación, es decir los datos de campo, quedan registrados en el archivo *campo.xxx*.

La extensión *xxx* identifica un modelo dado así como todos los archivos que se generen para este modelo. El programa genera distintos archivos intermedios[1] para finalmente obtener los archivos de salida: *min.xxx*, *report.xxx* y *grafico.xxx*. El primero contiene las mejores soluciones obtenidas, el segundo datos sobre la evolución de la población y el tercero un gráfico comparativo de las secuencias de velocidad real y estimada por el algoritmo.

A.5 Ambiente del Sistema

Para ingresar al sistema hay que posicionarse en el directorio donde se ha instalado el sistema y ejecutar el programa *Algen*.

C:\Algen> algen.exe

Los elementos básicos que conforman la aplicación se indican a continuación en una imagen de la pantalla principal [Figura A-1].

[1] Ver Sección B-4.

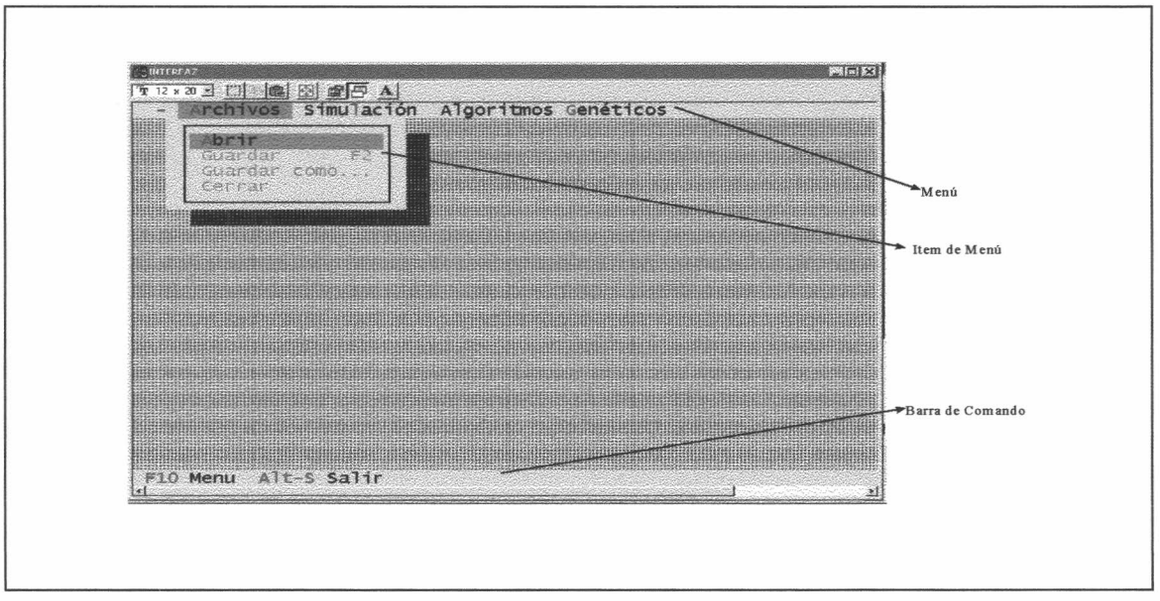

Figura A-1. Ambiente del Sistema

A.5.1 Opciones del Menú

El menú principal contiene las siguientes opciones:

- Archivos
- Simulación
- Algoritmos Genéticos

Para desplegar cualquiera de los menúes se puede seleccionar la opción deseada con el mouse, o bien con F10 ir a la barra de menú.

La opción **Archivos** permite editar un archivo cualquiera. Como se mencionó previamente, la extensión del archivo *modelo.xxx* caracteriza a un modelo en particular y se utiliza en los archivos intermedios y de salida del sistema para de este modo distinguir los distintos experimentos. También es posible abrir los archivos de gráficos y reporte obtenidos como resultados de las distintas ejecuciones, los cuales son descriptos en profundidad más adelante.

La opción **Simulación** del menú permite simular datos de campo a partir de un modelo del terreno. Se debe indicar el nombre del archivo que contiene el modelo del terreno con el que se desea trabajar. Como resultado de la simulación se generan dos archivos de salida: *campo.xxx* y *presetup.xxx*.

El archivo *campo.xxx* contiene los datos de campo (básicamente las presiones registradas) y el archivo *presetup.xxx* contiene los datos conocidos del modelo que indican cómo codificar el cromosoma (cantidad de capas, rango de velocidades o lista de materiales posibles).

El siguiente menú es **Algoritmos Genéticos**, donde se puede ejecutar cualquiera de las tres propuestas implementadas para este problema. Se debe indicar el archivo con datos de campo asociado al modelo a evaluar (*campo.xxx*), y algunos parámetros necesarios para el algoritmo, para los cuales se muestran valores por defecto. Genera los archivos de salida: *min.xxx*, *grafico.xxx* y *report.xxx*. El primero contiene las mejores estructuras obtenidas, el segundo datos sobre la evolución de la población y el tercero un gráfico donde se comparan las velocidades reales de las capas del terreno con las estimadas por el algoritmo.

A continuación se explica con mayor detalle cada uno de los menúes mencionados.

Archivos

Esta opción del menú permite Abrir, Grabar o Cerrar un archivo, así como Salir del Sistema *Algen*. Si se selecciona el ítem **Abrir**, se presenta un cuadro de diálogo que permite seleccionar el archivo [Figura A-2] a manipular. Se confirma con Open o se sale con Cancel. Al elegir la opción **Open** se ingresa a un editor con los datos del archivo seleccionado. Se puede **Cerrar** o **Grabar** (F2) el archivo desde el menú archivos seleccionando la opción correspondiente. La opción **Cerrar** permite salir del archivo sin grabarlo, en caso de que este haya sido modificado se pide confirmación. Por último la opción **Salir** (<ALT-S>) permite salir del sistema.

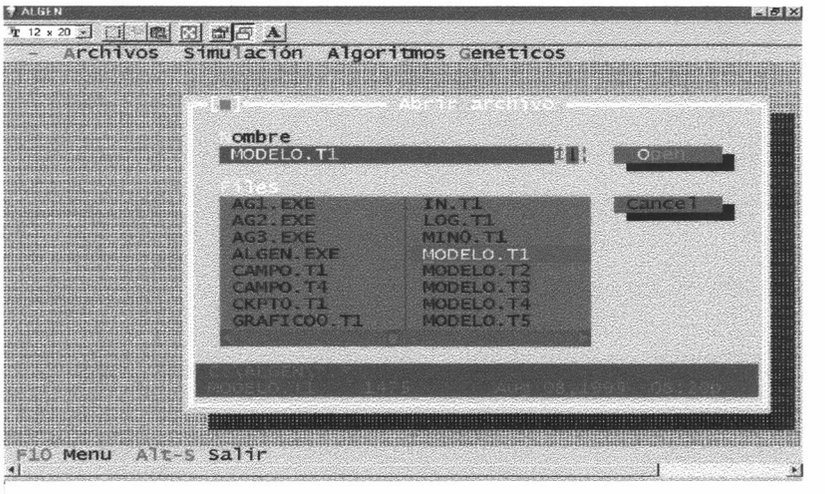

Figura A-2. Menú Archivos

Simulación

El menú **Simulación** permite generar datos de campo a partir de un modelo del terreno dado.

La opción **Editar Modelo** permite abrir un archivo *modelo.xxx* para edición. Una vez editado, el archivo puede grabarse con el mismo o distinto nombre usando las opciones del menú **Archivos**. Si se desea ejecutar AG con distintos parámetros sobre el mismo modelo, es conveniente editarlo y grabarlo con otro nombre para conservar los resultados con distintas extensiones. Es necesario respetar la estructura del archivo *modelo.xxx*, por lo tanto se recomienda editar alguno ya existente, modificarlo y guardarlo con la extensión correspondiente.

Para ejecutar la simulación se debe elegir un modelo existente mediante la opción **Elegir Modelo.** Al seleccionar esta opción un cuadro de diálogo muestra los archivos de modelos disponibles [Figura A-3]. Se debe seleccionar uno y confirmar con el botón **Open**. El proceso de simulación genera un archivo de nombre *campo.xxx*, con la misma extensión del modelo dado que contiene básicamente las presiones simuladas.

Una vez generado el archivo con los datos de campo, se está en condiciones de realizar la estimación del perfil del terreno ejecutando alguna de las propuestas de AG.

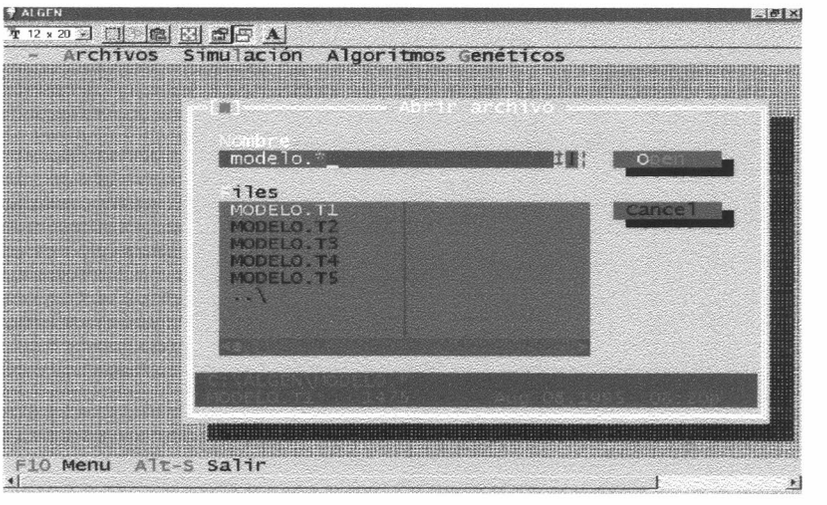

Figura A-3. Elección de un modelo para la simulación.

Algoritmos Genéticos

Al elegir esta opción se despliega un menú con las tres propuestas de AG implementadas. Luego de elegir alguna de las tres se despliega una ventana donde se puede seleccionar el archivo con los datos de campo a procesar. Se confirma con **Open** o se sale con **Cancel**.

El siguiente paso es completar los parámetros necesarios para ejecutar la propuesta de AG seleccionada [Figura A-4]. Los parámetros solicitados pueden ser completados por el usuario o se pueden dejar los sugeridos por defecto. Si se desea modificar alguno se puede desplazar con la tecla <TAB>.

Figura A-4. Parámetros para la Propuesta I

Algunos parámetros son comunes a las tres propuestas de AG y otros son particulares de cada una. Los parámetros comunes se describen a continuación:

Experimentos	Cantidad de experimentos que se desea ejecutar en una ejecución. Los experimentos son ejecuciones mutuamente independientes de AG, sobre los mismos parámetros y archivos de entrada. El único parámetro que varía es la semilla aleatoria que se utiliza para generar la población inicial, de otro modo todos los experimentos obtendrían resultados idénticos.
Evaluaciones	Cantidad de evaluaciones máximas en un experimento. Es utilizada como una condición de parada.
Población	Cantidad de individuos que formarán la población en cada generación.
Probabilidad de Cruza	Es la probabilidad de aplicar el operador de cruza[1].
Probabilidad de Mutación	Es la probabilidad de aplicar el operador de mutación[2].
Salto Generacional	Indica qué fracción de la población será reemplazada en cada generación. Si éste parámetro toma valor 1 entonces el 100 % de la población podrá ser reemplazada.
Ventana de Scaling	Se debe indicar el tamaño de la ventana de scaling que se utilizará. Si el valor ingresado es cero el sistema no tomará en cuenta esta opción[3].

[1] Ver Sección 2.2.
[2] Ver Sección 2.3.
[3] La necesidad de escalado se encuentra explicado en la Sección 3.4.2.

Max. Gens. sin Eval	Es la máxima cantidad de generaciones consecutivas sin evaluaciones. Se utiliza como condición de parada. Una generación no necesita evaluación si no presenta nuevos individuos con respecto a la generación anterior.
Semilla aleatoria	Es un número que se utilizará como semilla del generador aleatorio de la población inicial. Si se utiliza la misma semilla en distintas pruebas se generará la misma población inicial. Esto es útil para comparar distintos juegos de parámetros sobre la misma población.
Ranking mínimo	Es el mínimo número esperado de descendientes cuando se utiliza selección por Ranking[4].
Selección	Indica las características del método de selección a utilizar. La primera opción *Elitista* define si se garantizará o no el pasaje del mejor individuo a la próxima generación. La segunda opción define el método de selección a utilizar. Si se elige la opción *Ranking* se aplicará selección por ranking. Si no se elige esta opción se aplicará Selección proporcional con control del número esperado[5].

En la *Propuesta I* se solicitan los siguientes parámetros adicionales:

Codificación Gray	Se debe indicar si se quiere utilizar el código Gray en la codificación de los parámetros dentro del individuo[3].
Granularidad	Se debe ingresar el grado de discretización del espacio de búsqueda indicando la cantidad de bits a utilizar en la codificación de cada gen.

Para la *Propuesta III* se debe ingresar el siguiente parámetro adicional:

Error Presión	Es la máxima distancia aceptada (ε) entre dos presiones para calificar un alelo como "Bueno".

Una vez ingresados los parámetros necesarios, se confirma con **OK** o se sale con **Cancel**.

El tiempo de ejecución de la propuesta de AG elegida puede variar desde algunos minutos a varias horas dependiendo del modelo, propuesta y parámetros seleccionados. Durante la ejecución se muestra por pantalla un encabezado que indica:

- Máxima cantidad de evaluaciones a ejecutar en cada experimento.
- Extensión del modelo seleccionado.
- Propuesta de AG seleccionada.
- Número de experimento que se está procesando y cantidad total de experimentos requerida.

y los siguientes resultados parciales:

- Cantidad de generaciones evaluadas hasta el momento
- Cantidad de evaluaciones ejecutados hasta el momento
- Cantidad de alelos perdidos

[4] Para más detalle ver la Sección 2.1.4.
[5] Los métodos de Selección se explican en la Sección 2.1.
[3] Ver Sección 4.1.

- Cantidad de alelos que convergieron
- Sesgo o grado de convergencia de la población
- Performance online, es el promedio de todas las evaluaciones realizadas durante el experimento [GOL/89]
- Performance offline, es el promedio de las mejores evaluaciones del experimento [GOL/89]
- Aptitud de la mejor estructura evaluada hasta el momento
- Mejor estructura de la última generación evaluada
- En el caso de la *Propuesta III* se muestra la calificación de los alelos dada por la heurística, indicándose con 1 los alelos calificados como buenos y con 0 los calificados como malos.

A.5.2 Archivos de Salida

Al finalizar el proceso se generan los archivos *report.xxx* y los archivos *minN.xxx*, *graficoN.xxx*, donde *N* es el número de experimento. Este último contiene un gráfico [Figura A-5] donde se comparan las velocidades observadas contra las reales. Este archivo puede ser visualizado con la opción abrir del menú **Archivos**.

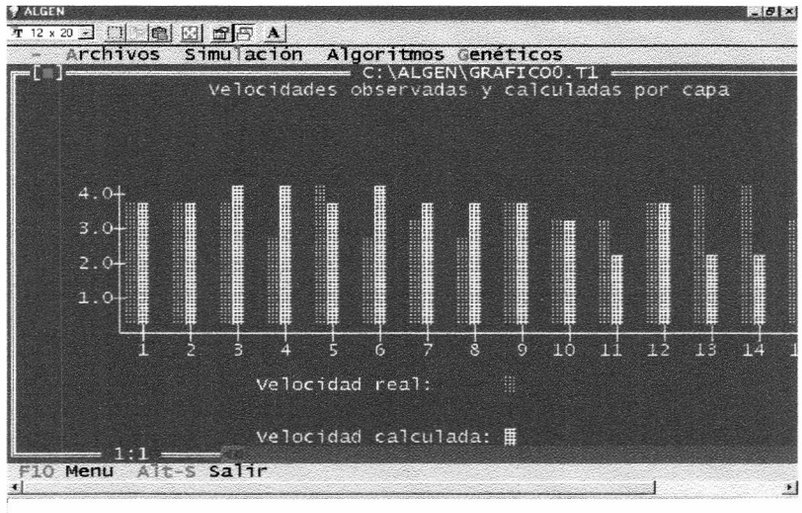

Figura A-5. Visualización del gráfico comparativo

A.6 Ejemplo

A continuación se desarrolla un ejemplo muy sencillo de ejecución del sistema *Algen* cuyo objetivo es guiar al usuario en el uso del mismo.

a) Se ingresa al sistema ejecutando el programa *algen* desde el directorio donde está instalado.

b) Mediante la opción **Editar Modelo** del menú **Simulación** se puede visualizar y modificar un modelo de terreno y generar uno nuevo grabándolo con otra extensión. A continuación se muestra el listado del archivo modelo [Figura A-6] para el Test1.

```
# cantidad de capas
16
# posicion de la interface de la superficie (metros)
0
# posicion de las interfaces entre capas (metros)
60
120
180
240
300
360
420
480
540
600
660
720
780
840
900
960
# cantidad de materiales
5
# densidad de cada material (gr/cm**3)
2.0000
2.0000
2.0000
2.0000
2.0000
# modulo bulk de cada material (dynes/cm**2)
8e10
32e10
18e10
12.5e10
24.5e10
# shear modulus (dynes/cm**2)
0.0
0.0
0.00000
0.00000
0.00000
# silenciar (1) o no (0) la fuente
1
# ruido blanco (%)
1
# tipo de fuente
#                   1: Tsang & Rader
#                   2: Gaussian deriv.
2
# frecuencia de la fuente (kHz)
.040
# tiempo total de la simulacion (msec)
675
# numero de puntos por longitud de onda
10
# materiales de cada capa
4
4
4
3
1
3
2
3
4
2
2
4
1
1
2
4
# posicion de la fuente (metros)
11
# posicion del receptor (metros)
11
# condicion de bordes
#                   0: free surface
#                   1: absorbing b.c.
#                   2: neuman
1
# tipo de contribucion de la fuente
#                   0: delta
#                   1: d/dx(delta)
0
# tiempo que enmudece la fuente
1
# parametro para filtro pasa-bajo (kHz)
0
# numero de puntos para Fourier
50
```

Figura A-6. Ejemplo de modelo de terreno (modelo.t1)

c) El siguiente paso consiste en generar los datos de campo a partir del modelo elegido en el paso anterior. Entre otras cosas se calcula la secuencia de presiones observadas para el perfil descripto en el modelo. Se utiliza la opción **Elegir Modelo** del menú **Simulación**. En este caso se eligió el archivo **modelo.t1**. El proceso de simulación genera el archivo **campo.t1** que se utilizará en la propuesta de AG que luego seleccione. La [Figura A-7] muestra dicho archivo.

```
#cantidad de intervalos de tiempo
658
#cantidad de intervalos de tiempo en que enmudece la fuente
0
#cantidad de intervalos en x
208
#duración de un intervalo de tiempo (ms)
1.025641
#longitud de intervalo para cada capa
4.615385
4.615385
```

```
4.615385
4.615385
4.615385
4.615385
4.615385
4.615385
4.615385
4.615385
4.615385
4.615385
4.615385
4.615385
4.615385
4.615385
#Propuesta I: ro de cada capa
2000000.000000
2000000.000000
2000000.000000
2000000.000000
2000000.000000
2000000.000000
2000000.000000
2000000.000000
2000000.000000
2000000.000000
2000000.000000
2000000.000000
2000000.000000
2000000.000000
2000000.000000
#posici¢n de la fuente (m)
11.000000
#nodo anterior a la fuente
3
#posici¢n del nodo anterior a la fuente (m)
9.230769
#posici¢n del nodo posterior a la fuente (m)
13.846154
#posici¢n del receptor (m)
11.000000
#nodo anterior al receptor
3
#posici¢n del nodo anterior al receptor (m)
9.230769
#posici¢n del nodo posterior al receptor (m)
13.846154
#intervalos acumulados en cada capa
0
13
26
39
52
65
```

```
78
91
104
117
130
143
156
169
182
195
208
#condicion de bordes
#0: free surface
#1: absorbing b.c.
#2: neuman
1
#tipo de contribucion de la fuente
#0: delta
#1: d/dx(delta)
0
#funci¢n fuente
0.013197
-0.254393
-0.501823
      :
      :
      :
0.000000
0.000000
#is
1.401375
#presiones observadas (g/m**2)
0.000000
0.000000
-0.060301
-0.165736
      :
      :
      :
-1.418758
-1.419134
#Datos para las versiones 2 y 3
#cantidad de materiales
5
#velocidad de cada material m/ms
2.000000
4.000000
3.000000
2.500000
3.500000
#densidad de cada material g/m**3
2000000.000000
2000000.000000
2000000.000000
2000000.000000
2000000.000000
```

Figura A-7. Ejemplo de datos de campo (campo.t1)

d) Se puede elegir cualquiera de las propuestas de Algoritmos Genéticos implementadas desde el menú de Algoritmos Genéticos. Para este ejemplo, se seleccionó la *Propuesta III* para correr sobre el archivo campo.t1 y se aceptaron los valores propuestos por defecto.

e) El sistema muestra los resultados parciales de la ejecución [Figura A-8].

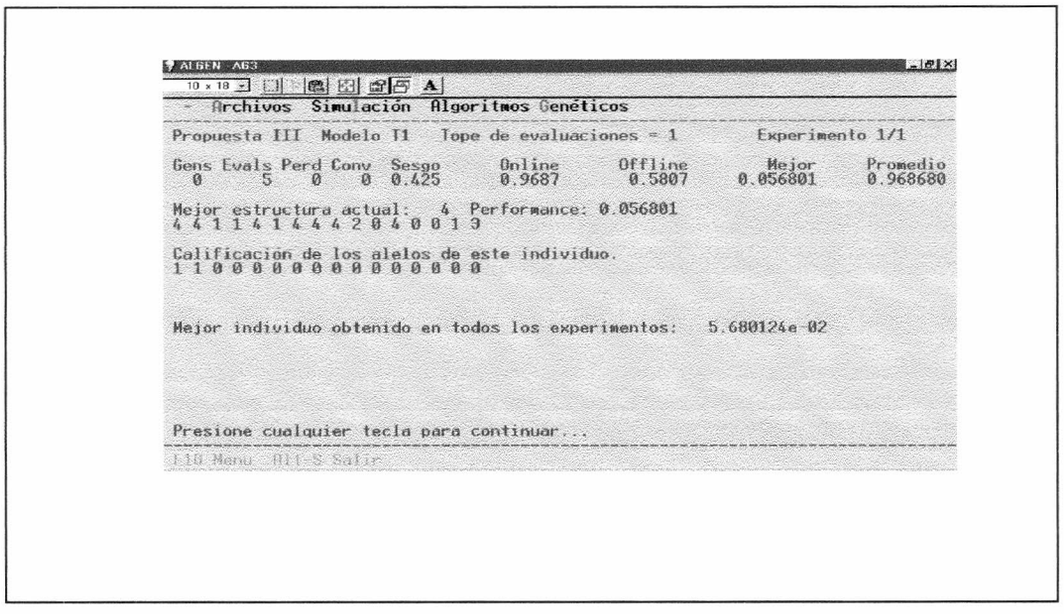

Figura A-8. Resultados parciales

f) Al finalizar el proceso se habrá generado el archivo de salida **report.t1** con un resumen de los parámetros utilizados y estadística referente a la evolución de la población en cada generación [Figura A-9]. El archivo **min0.t1** muestra las cinco mejores estructuras logradas en el primer experimento [Figura A-10].

Propuesta = 3
Exp. = 1
Tot. Eval. = 1
Pob. = 5
Long. cromosoma = 48
Prob. cruza = 0.600000
Prob. mutación = 0.001000
Salto generac. = 1.000000
Ventana scaling = 5
Interv. reporte = 1
Estruc. guardadas = 10
Max gens sin eval. = 2
Interv. entre vuelcos = 0
Vuelcos guardados = 0
Opciones = cDflLtboOe
Semilla aleatoria = 123456789
Ranking mínimo = 0.750000
Epsilon = 0.010
PROMEDIO

Gens	Evals	Perd	Conv	Sesgo	Online	Offline	Mejor	Promedio
0	5	0	0	0.425	9.687e-01	5.807e-01	5.680e-02	9.687e-01
1	81	0	0	0.564	4.748e-01	2.408e-02	1.492e-02	2.866e-01
:	:	:	:	:	:	:	:	:
63	1950	32	58	0.851	5.707e-02	8.055e-03	3.161e-03	5.444e-03
64	1980	33	58	0.858	5.630e-02	7.981e-03	3.160e-03	5.244e-03

Figura A-9. Ejemplo de reporte (report.t1)

0 4 3 4 4 3 1 0 1 2 1 1 4 3 1 0	2.5376e+00	0	1
1 3 1 3 1 4 0 4 2 4 1 0 1 4 0 1	1.1347e-01	0	2
2 1 1 0 3 4 0 1 4 1 1 2 3 2 0 0	9.7767e-02	0	3
0 3 3 3 0 0 4 1 2 3 0 4 0 1 1 3	2.0378e+00	0	4
4 4 1 1 4 1 4 4 4 2 0 4 0 0 1 3	5.6801e-02	0	5

Figura A-10. Ejemplo de las mejores cinco estructuras (min0.t1)

A.7 Archivos de Entrada y Salida del sistema

Se describe a continuación los distintos archivos utilizados por el sistema.

A.7.1 Datos del terreno: Modelo.*xxx*

Contiene los datos que caracterizan el modelo de terreno.

- Cantidad de capas
- Longitud de cada capa
- Tabla de materiales: (material, densidad, modulo bulk, shear modulus)
- Condición de bordes
- Tipo de fuente
- Frecuencia de la fuente
- Tipo de contribución de la fuente (como aplicar fuente al calculo de la presión)
- Tiempo total de la simulación
- Posición de la fuente
- Posición del receptor
- Silenciar o no la fuente
- Tiempo que enmudece la fuente
- Porcentaje de ruido blanco
- Filtro pasabajo
- Filtro para Fourier
- Número de puntos por longitud de onda.
- Composición del terreno: material de cada capa

A.7.2 Datos que determinan la codificación de la población: Presetup.*xxx*

- Cantidad de capas: determina la cantidad de genes
- Velocidad Mínima: determina rango en la *Propuesta I*
- Velocidad Máxima: determina rango en la *Propuesta I*
- Cantidad de materiales: determina rango en las *Propuestas II* y *III*

A.7.3 Datos sintéticos de campo: Campo.*xxx*

Contiene los datos resultantes de la simulación.

- Densidad de cada capa
- Condición de bordes
  - 0: superficie libre
  - 1: condición de bordes absorbente
  - 2: neuman
- Tipo de contribución de la fuente
  - 0: delta
  - 1: d/dx(delta)
- Posición en metros de la fuente
- Posición en metros del receptor
- Vector de presiones observada
- Intervalos en el tiempo
- Cantidad de intervalos de tiempo en que enmudece la fuente
- Total de intervalos en Ω

- Longitud del intervalo
- Longitud en metros del intervalo de cada capa
- Nodo anterior a la fuente
- Posición en metros del nodo anterior a la fuente
- Posición en metros del nodo posterior a la fuente
- Nodo anterior al receptor
- Posición en metros del nodo anterior al receptor
- Posición en metros del nodo posterior al receptor
- Intervalos acumulados por capa
- Función fuente
- Datos para las *Propuestas II* y *III*
    - Cantidad de materiales
    - Velocidad de cada material m/ms
    - Densidad de cada material g/m^3

A.7.4 Gráfico comparativo de velocidades: Grafico.*xxx*

Compara las velocidades estimadas por el algoritmo contra las velocidades reales provistas en el modelo mediante un gráfico de barras. En el eje vertical se representan las velocidades y en el eje horizontal las capas del terreno.

A.7.5 Resumen de resultados: Report.*xxx*

En este archivo se guardan los valores de los parámetros seleccionados por el usuario y la evolución de la población durante la ejecución.

Valores de los parámetros

- Propuesta de AG ejecutada.
- Cantidad de experimentos realizados
- Tope de evaluaciones
- Tamaño de la población
- Longitud de la estructura en bytes
- Probabilidad de cruza
- Probabilidad de mutación
- Salto generacional
- Tamaño de la ventana de scaling
- Intervalo entre generaciones informadas en el reporte
- Cantidad de mejores estructuras guardadas
- Máxima cantidad de generaciones sin evaluación admitidas
- Intervalo entre vuelcos
- Vuelcos guardados
- Opciones: indica entre otras cosas el método de selección usado [Anexo B]
- Semilla aleatoria
- Ranking mínimo

Datos sobre la evolución de la población

Informa el promedio y la varianza de las siguientes variables para cada generación sobre todos los experimentos efectuados en la ejecución:

- Número de generación

- Cantidad de evaluaciones acumuladas después de evaluar esta generación
- Cantidad de alelos perdidos
- Cantidad de alelos que convergieron
- Sesgo o grado de convergencia de la población
- Performance On line
- Performance Off line
- Aptitud de la mejor estructura de la generación
- Promedio de las aptitudes de las estructuras de la generación

A.7.6 Mejores estructuras obtenidas: Min.*xxx*

El archivo Min contiene las estructuras con mejor aptitud obtenidas en el experimento y presenta una línea por estructura. Los valores de cada línea son los alelos del cromosoma, la aptitud del mismo, la generación y la evaluación en la que apareció por primera vez.

Anexo B

Implementación del Sistema

B.1 Descripción general del sistema Algen

En [Figura B-1] se muestra de un modo general el flujo de datos entre los principales procesos del sistema *Algen*.

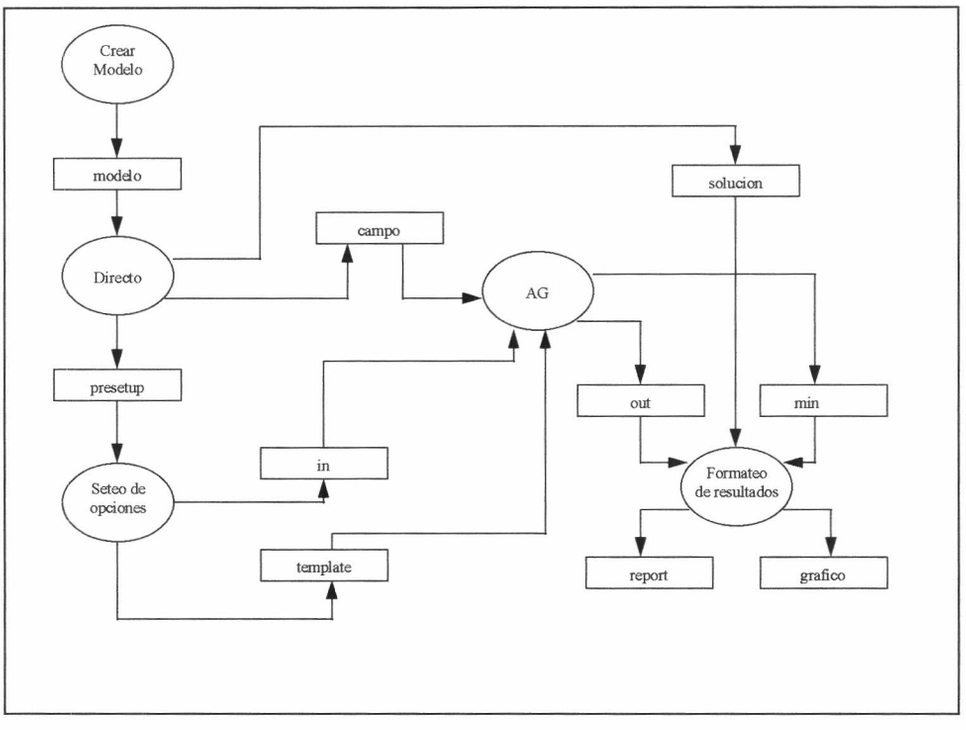

Figura B-1 Diagrama general del sistema

Mediante el proceso *Crear Modelo* se permite definir un modelo del terreno que será almacenado en el archivo *modelo* con la extensión elegida. Este archivo es tomado como entrada por el proceso *Directo* que simula la explosión en el terreno.

La salida principal de *Directo* es el archivo *campo* que contiene las presiones registradas. Otras salidas son el archivo *presetup* que contiene datos útiles para la codificación de los genes y el archivo *solución* que será utilizado para evaluar la calidad de la solución hallada por el Algoritmo Genético.

El proceso *Seteo de opciones* permite al usuario ajustar una serie de parámetros para la ejecución del Algoritmo Genético. Los parámetros referidos a la estructura de los genes se almacenan en el archivo *template* y el resto de las opciones elegidas se registra en el archivo *in*.

El *Algoritmo Genético* actúa en base a estos dos archivos de opciones y a los datos de campo provistos. Las mejores soluciones son guardadas en *min* y las estadísticas de performance del algoritmo en el archivo *out*.

En base a estas salidas y a la verdadera composición del terreno tomada del archivo *solución* se realiza el *Formateo de resultados* que muestra las estadísticas de performance en el archivo *report* y compara en el archivo *gráfico* la composición del terreno obtenida por el algoritmo con la composición dada en el modelo inicial.

B.2 Simulación para generar los datos de campo

La simulación del experimento se realizó en base al algoritmo descripto en [FER/93][1].

Para generar los datos de campo se leen los datos de un archivo modelo elegido, en base a esto se calculan datos auxiliares y a partir de los mismos se calcula la sucesión de presiones registradas en el receptor. Se alteran las mismas con el porcentaje de ruido blanco indicado y los resultados se graban en los archivos intermedios para ser tomados luego por el proceso de Algoritmos Genéticos [Figura B-2].

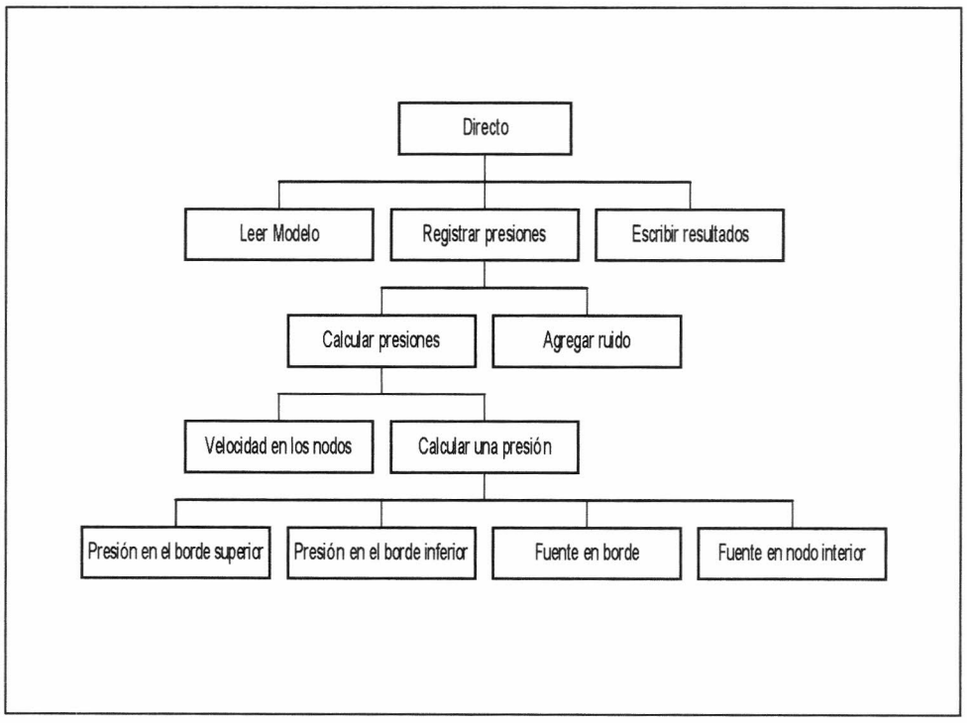

Figura B-2 Diagrama del problema directo

A continuación se describen las rutinas que componen el cálculo de la presión:

leerModelo: lee el archivo modelo elegido.

[1] Sección 6.2

registrarPresiones: Calcula datos auxiliares e invoca al cálculo de presiones.
Los datos auxiliares calculados son básicamente:

- Máxima y mínima velocidad de acuerdo a la lista de materiales posibles.
- Longitud de onda.
- Posición de los nodos en el eje X.
- Densidad en cada nodo.
- Tamaño del intervalo en el tiempo.
- Total de intervalos en el tiempo
- Influencia de la fuente a lo largo del tiempo.
- Determinación de los nodos más próximos a la fuente y al receptor.
- Enmudecimiento de la fuente durante los primeros intervalos de tiempo.
- Velocidad de cada material en base al modulo bulk y a la densidad.
- Velocidad de cada capa en base al material y a la velocidad de cada material.

presiones: Calcula la serie de presiones registradas en el receptor a lo largo del tiempo en base a las velocidades de los materiales que componen el terreno.

agregarRuido: Modifica las presiones calculadas alterándolas con un porcentaje de ruido.

vel_en_nodos: Calcula la velocidad en cada uno de los nodos del eje X.

una_presion: Calcula la presión en el receptor para un instante de tiempo dado, basándose en las presiones en todos los nodos en los instantes anteriores.

borde_sup: Calcula la presión ejercida por los rebotes de la onda sobre el borde superior.

borde_inf: Calcula la presión ejercida por los rebotes de la onda sobre el borde inferior.

fuente_en_borde: Calcula la presión ejercida directamente por la fuente en el borde.

fuente_interior: Calcula la presión ejercida directamente por la fuente en un nodo interior.

escribirResultados: Genera los archivos presetup, campo y solución.

B.3 Implementación de Algoritmos Genéticos

La implementación de Algoritmos Genéticos fue desarrollada a partir del programa *Genesis 5.0* de dominio público[1] de John J. Grefenstette.

B.3.1 Estructura del programa

A continuación se describen los módulos del programa:

[1] Copyright 1990. Su uso está permitido para fines educativos y de investigación.

- MAIN.C

El módulo principal *main* ejecuta un ciclo que consiste en crear la nueva generación y ver si se cumple la condición de parada.

- GENERATE.C

La función *Generate* forma la nueva población, la evalúa y toma estadísticas de performace. La primera generación se crea aleatoriamente mediante la rutina *Initialize*. Las siguientes generaciones se forman por la aplicación sucesiva de selección, mutación y cruza.

- INIT.C

La rutina *Initialize* inicializa la población al azar a partir de una semilla aleatoria provista por el usuario.

- SELECT.C

La rutina S*elect* implementa el método de selección elegido: Selección proporcional con control del número esperado o bien Selección por Ranking. Escala la aptitud comparando la performance de cada individuo contra la peor de las últimas W generaciones, siendo W el tamaño de la ventana de escalado.

- MUTATE.C

La rutina M*utate* realiza mutación a nivel de bit. Se decide el bit a mutar calculando el intervalo entre mutaciones de acuerdo a la probabilidad de mutación. La próxima posición a mutar se calcula como:

$$\frac{\ln(r)}{\ln(1-P_m)}$$

donde r es un número elegido con probabilidad uniforme en el intervalo $(0,1)$ y P_m es la probabilidad de mutación seleccionada.

- CROSS.C

La rutina C*rossover* produce dos hijos aplicando cruza en dos puntos a nivel de bit.

- ELITIST.C

Si el mejor individuo de la generación anterior no está presente en la nueva generación, la rutina *Elitist* lo incluye reemplazando a alguno de los recientemente generados.

- EVALUATE.C

La rutina *Evaluate* aplica la función objetivo *eval* a cada individuo de la población.

- EVAL.C

La función *eval* es la función objetivo. Es dependiente del problema.

- MEASURE.C

La rutina *Measure* obtiene medidas de performance como: mejor aptitud de la generación, aptitud promedio, peor aptitud de la ventana de escalado, performance on line y off line.

- CONVERGE.C

Esta rutina recoge estadísticas sobre el estado de convergencia de una población y las refleja en las siguientes tres medidas:

a) *Cantidad de alelos perdidos*: Un alelo se dice perdido si todos los cromosomas tienen el mismo valor en una determinada posición.

b) *Cantidad de genes que convergieron*: Se considera que un gen convergió si el 95% de los cromosomas tienen el mismo valor en ese gen.

c) *Valor del sesgo global o grado de convergencia*: Se calcula promediando el porcentaje que ocupa el alelo dominante para cada gen.

- DONE.C

La rutina *Done* determina si se cumple la condición de parada. Las condiciones de parada son: cantidad de evaluaciones, cantidad de generaciones sin nuevos individuos y uniformidad total de la población.

- BEST.C

Las rutinas *Savebest*, *Readbest*, *Printbest* guardan, recuperan e imprimen las mejores soluciones respectivamente.

- CHECKPNT.C

Esta rutina guarda todos los datos de la población para un eventual reinicio.

- CONVERT.C

Contiene distintas rutinas de conversión entre representación interna y externa. El programa trabaja con tres niveles de representación. Internamente, para optimizar espacio y tiempo de ejecución, utiliza una representación "empaquetada" donde cada alelo es un bit que puede tomar valores 0 ó 1. Los operadores genéticos de cruza y mutación trabajan a este nivel.

A nivel de usuario provee dos opciones de representación: cadena de caracteres o punto flotante. En el primer caso cada gen puede tomar valores '0' o '1', en el segundo, los genes representan números reales que toman valores dentro de un rango con una determinada granularidad. El operador de selección y la entrada y salida de datos trabajan en este nivel de representación más elevado. Para la presente aplicación utilizamos la representación en punto flotante.

- DISPLAY.C

Contiene rutinas para el manejo de pantalla .

- ERROR.C

Se encarga del manejo de errores.

- INPUT.C

  La rutina *Input* inicializa variables. Lee de los archivos in y template.

- RESTART.C

  Restart toma la población inicial y los datos del experimento de un archivo *ckpt* generado en una ejecución anterior, para poder reinicializar el programa desde un punto anterior.

B.3.2 Propuesta I

A continuación se detallan las modificaciones introducidas por la *Propuesta I* sobre el programa original.

- EVAL.C

  La función de objetivo *eval* se calcula como la suma de la diferencia de cuadrados entre la serie de presiones escuchadas en el receptor (dato tomado del archivo *campo*) y las presiones generadas por el modelo de terreno representado por el individuo a evaluar.

- PRESION.C

  La rutina *presiones* calcula las presiones generadas por el modelo de terreno que representa un individuo.

- IN_EVAL.C

  La rutina *In*_eval lee los datos del archivo *campo*.

B.3.3 Propuesta II

A continuación se detallan las modificaciones introducidas por la *Propuesta II*.

- CROSS.C

  Se realiza la cruza a nivel de gen en lugar de hacerlo a nivel de bit. Esto significa que los puntos de cruza sólo pueden ubicarse en los puntos de separación entre genes.

- EVAL.C

  Se modifica la función *eval* para adaptarla a la nueva representación de los cromosomas. La modificación consiste en traducir los códigos de material a los parámetros físicos correspondientes (velocidad y densidad del medio) como paso previo para poder calcular las presiones generadas por los mismos.

- IN_EVAL.C

  El valor de la densidad del medio, ρ, ya no se considera fijo para cada capa sino que se asocia a un material dado y afecta a la capa que contiene ese material.

- INIT.C

  Se modifica para generar la población inicial con la nueva codificación del cromosoma.

- INPUT.C

Elimina la restricción sobre la cantidad de valores que puede tomar un gen. Ya no se exige que la misma sea potencia de 2.

- MUTATE.C
  En esta propuesta la mutación se realiza a nivel de gen.

- MEASURE
  Por el cambio de representación en esta propuesta, las estadísticas sobre la convergencia de la población se calculan a nivel de gen.

B.3.4 Propuesta III

A continuación se detallan las modificaciones introducidas por la *Propuesta III*. Se modifica la estructura para almacenar la calificación de los alelos de acuerdo a la heurística.

- CROSS
  Implementa la cruza multipunto con heurística.

- EVAL
  Se le agrega una nueva función a la rutina *eval* para que califique los alelos de un cromosoma de acuerdo a la heurística y los almacene en la estructura para ser utilizados por la cruza y la mutación heurísticas.

- MUTATE
  Implementa la mutación heurística.

B.4 Archivos intermedios

B.4.1 In: Parámetros del algoritmo genético

Este archivo contiene los siguientes parámetros:

- Cantidad de experimentos.
- Cantidad máxima de evaluaciones a realizar.
- Tamaño de la población.
- Longitud del cromosoma en bytes.
- Probabilidad de cruza.
- Probabilidad de mutación.
- Salto generacional.
- Tamaño de la ventana de Scaling.
- Intervalo de reporte.
- Cantidad de estructuras guardadas
- Máxima cantidad de generaciones permitida sin evaluaciones.
- Intervalo entre vuelcos.
- Cantidad de vuelcos grabados.
- Opciones.
- Semilla aleatoria.

- Ranking mínimo.

La mayor parte de estos parámetros son provistos por el usuario y están definidos en el Anexo A. A continuación se explican los parámetros no ingresados por el usuario:

- Longitud del cromosoma: En la propuesta I se calcula a partir de la granularidad definida por el usuario. En las propuestas II y III en base a la cantidad de materiales que se va a representar.
- Intervalo de reporte: Indica cada cuántas generaciones se calculan estadísticas. El valor para este parámetro está fijado en 1.
- Cantidad de estructuras guardadas: Las mejores 10 estructuras de la población se guardan en el archivo *min*.
- Intervalo entre vuelcos: Indica cada cuántas generaciones se hace un vuelco de la población. Este valor fue fijado en 0. No se realizan vuelcos periódicos de la población.
- Cantidad de vuelcos grabados: 0.
- Opciones: Las opciones referidas al método de selección son definidas por el usuario. El resto de las opciones que son fijadas por el programa son las siguientes:
  - c: Llevar estadísticas de convergencia de la población.
  - D: Mostrar estadísticas de performance por pantalla.
  - f: Usar representación de punto flotante.
  - l: Registrar las ejecuciones efectuadas en un log.
  - L: Volcar la última generación en un archivo ckpt.
  - b: Escribir el promedio de los mejores valores de cada experimento.
  - o: Escribir el promedio de la performance on-line de todos los experimentos. La performance on-line es el promedio de todas las evaluaciones durante el experimento.
  - O: Escribir el promedio de la performance off-line de todos los experimentos. La performance on-line es el promedio de los mejores individuos de cada generación.

B.4.2 Presetup: Datos para generar la población inicial

El archivo presetup contiene los siguientes datos:

- Cantidad de capas
- Velocidad Mínima
- Velocidad Máxima
- Cantidad de materiales

La cantidad de capas determina la cantidad de genes, habrá un gen por cada capa.

En la *Propuesta I* las velocidades mínima y máxima determinan el rango de valores que puede adoptar un gen.

En las *Propuestas II* y *III* esto esta determinado por la cantidad de materiales. Los valores que puede adoptar un gen en este caso son los enteros que pertenecen al intervalo [0, cantidad de materiales – 1].

B.4.3 Solución: Verdadera composición del terreno

En el archivo solución se guarda la siguiente información:

- Cantidad de capas
- Cantidad de materiales
- Material de cada capa
- Velocidad de cada material
- Velocidad de cada capa

B.4.4 Template: Estructura de los cromosomas

La estructura del archivo template es la siguiente:

- Cantidad de genes

Por cada gen:

- Posición del gen dentro del cromosoma
- Valor mínimo y máximo que puede adoptar
- Cantidad de valores que puede adoptar (granularidad)
- Formato para la salida del gen

B.4.5 Out: Performance del algoritmo genético

Es la salida del sistema, que luego será formateada por la rutina report. Muestra los siguientes datos por cada generación:

- Número de generación.
- Cantidad de evaluaciones.
- Cantidad de alelos perdidos.
- Cantidad de alelos que convergieron.
- Valor del sesgo global.
- Performance On-line.
- Performance Off-line.
- Valor de la función objetivo para el mejor individuo.
- Promedio de la función objetivo para toda la población.

B.4.6 Checkpnt: Población final

En este archivo se guardan valores de algunas variables importantes, así como los individuos de la población en formato binario junto con su aptitud. Los individuos son presentados también en punto flotante.

Anexo C

Listado de Fuentes

C.1 Programa Algen

C.1.1 Módulo AG.H

```
/*  AG.H
        Define las llamadas a la simulaci¢n, al
algoritmo gen,tico y a las
        rutinas de exhibicion de resultados.
*/

/* ag: corre el algoritmo gen,tico para una versi¢n
dada */
void ag(int version);

/* simulaci¢n: Pide un modelo y llama a la rutina
directo para correr la
simulaci¢n de la explosi¢n */
void simulacion();

/* graficar: hace un gr fico de barras comparando las
velocidades reales
del terreno y las calculadas por el AG */
int graficar(char *ext, int version, unsigned long
cantExperimentos);

extern "C" {
/* directo: a partir del modelo del terreno simula la
explosi¢n y en
base a esa simulaci¢n calcula las presiones que se
hubieran registrado */
void directo(const char *ext);

/* report: formatea la informaci¢n estad¡stica
provista por ag */
void report(const char *ext, int version);

};
```

C.1.2 Módulo APP.H

```
/* APP.H
   Definicion del objeto TApp que hereda de
TApplication de Turbo Vision.
   Es el motor de la aplicaci¢n. Maneja los men£es y
llama a las rutinas
   correspondientes de acuerdo a las opciones
elegidas.
*/
#define Uses_TApplication
#include <tv.h>

class TMenuBar;
class TStatusLine;
class TEditWindow;
class TDialog;

const
  cmOpen       = 100,
  cmVersion1   = 150,
  cmAbrir    = 151,
  cmModelo     = 152,
  cmSalir    = 153,
  cmSimulacion = 154,
  cmVersion2   = 155,
  cmVersion3   = 156,
  cmAbout    = 157;
```

```
class TApp : public TApplication
{

public:

          TApp();

          virtual void handleEvent( TEvent& event );
          static TMenuBar *initMenuBar( TRect );
          static TStatusLine *initStatusLine( TRect );
          virtual void outOfMemory();
          void setearComandos(Boolean archivoAbierto);


private:
          TEditWindow *ventanaActiva;
          TEditWindow *openEditor( const char
*fileName, Boolean visible );
          void fileOpen(const char *mascara);


};

extern TEditWindow *clipWindow;

ushort execDialog( TDialog *d, void *data );
```

C.1.3 Módulo D_EXTERN.H

```
/* D_EXTERN.H
   Declaraciones de variables globales como externas
para compartir entre
   m¢dulos del problema directo.
*/
#ifndef EXTERN_H
#define EXTERN_H

/* Variables globales */
extern int l;                                   /*
cantidad de intervalos de la malla */
extern int keyb;                        /* tipo de
borde:

                    0 -> free surface

                    1 -> absorbing boundary condition

                    2 -> newman
                                                */
extern int kdel;                        /* tipo de
fuente:
                    1 -> delta de Dirac

                    2 -> derivada de la delta de Dirac

                    3 -> 2 delta de Dirac
                                                */
extern double *source;          /* funcion fuente
muestreada */
extern double *h;               /* longitud en metros de
cada intervalo */
extern int ntmute;                      /* cantidad de
intervalos en t que ignoro la presion escuchada */
extern int *kx;                         /* cantidad
acumulada de intervalos en x de una capa */
extern double *ronodo;    /* densidad en cada nodo */
extern double *xnod;      /* posicion de cada nodo */
extern double xsou;      /* posicion de la fuente */
extern int jsou;                /* nodo
inmediatamente superior a la fuente */
extern double *h;            /* longitud en metros del
cada intervalo */
```

```
extern double dt;                    /* longitud del
intervalito de tiempo */
extern double is;                          /* par metro
auxiliar */
extern double *recmed;   /* presiones observadas en el
receptor */
```

```
extern int jrec;              /* nodo inmediatamente
superior al receptor */
extern double xrec;          /* posicion del receptor */
extern int nt;                     /* cantidad de
intervalos en el tiempo */

#endif
```

C.1.4 Módulo D_GLOBAL.H

```
/* D_GLOBAL.H
   Declaraciones de variables globales del problema
directo.
*/
#ifndef GLOBAL_H
#define GLOBAL_H

/* Variables globales */
int l;                                  /* cantidad de
intervalos de la malla */
int keyb;                       /* tipo de borde:

                    0 -> free surface

                    1 -> absorbing boundary condition

                    2 -> newman
                                                    */
int kdel;                       /* tipo de fuente:

                    1 -> delta de Dirac

                    2 -> derivada de la delta de Dirac

                    3 -> 2 delta de Dirac
                                                    */
double *source;     /* funcion fuente muestreada */
```

```
double *h;                        /* longitud en metros de
cada intervalo */
int ntmute;                                /* cantidad de
intervalos en t que ignoro la presion escuchada */
int *kx;                          /* cantidad acumulada de
intervalos en x de una capa */
double *ronodo;   /* densidad en cada nodo */
double *xnod;     /* posicion de cada nodo */
double xsou;            /* posicion de la fuente */
int jsou;               /* nodo inmediatamente
superior a la fuente */
double *h;        /* longitud en metros del cada
intervalo */
double dt;                          /* longitud del
intervalito de tiempo */
double is;                                 /* par metro
auxiliar */
double *recmed;   /* presiones observadas en el
receptor */
int jrec;         /* nodo inmediatamente superior al
receptor */
double xrec;        /* posicion del receptor */
int nt;                 /* cantidad de intervalos en
el tiempo */

#endif
```

C.1.5 Módulo D_INPUT.H

```
/* D_INPUT.H
   Manejo de datos de entrada y salida para el
problema directo.
*/
#ifndef ENTRADA_H
#define ENTRADA_H

/* leerModelo: lee el archivo modelo elegido */
void leerModelo(char *archivo);

/* registrarPresiones: arma la malla para la
aplicaci¢n de elementos finitos
y llama a la rutina presiones para que calcule las
presiones */
void registrarPresiones(void);
```

```

/* escribirResultados:  Escritura de resultados del
problema directo.
   Se escriben los archivos presetup, campo y solucion
*/
void escribirResultados(char *archivo);

/* liberarMemoria: libera la memoria asignada a las
variables utilizadas
en el problema directo
*/
void liberarMemoria(void);

#endif
```

C.1.6 Módulo D_PRESIO.H

```
/* D_PRESIO.H
   C lculo de las presiones (problema directo)
*/
#ifndef PRESION_H
#define PRESION_H
/* una_presion : La funci¢n devuelve la presi¢n en el
receptor y calcula las
presiones en todos los nodos para ser utilizadas en
pasos posteriores*/
double una_presion(double *unm1, double *un, double
*unp1, int paso, double *constk, double *velnodo);

/* presiones: Se calculan las presiones a partir de
las velocidades */
void presiones(double *velocidad, int capas, double
*presion, int nt);

/* vel_en_nodos: Calcula la velocidad en cada nodo de
la malla */
void vel_en_nodos(double *velocidad, int capas, double
*velnodo);
```

```
/*borde_sup:  Calcula la presion en el borde superior
en el tiempo n+1
(sin contar la influencia directa de la fuente) */
double borde_sup(double unm1, double unx1, double
unx2, double cte);

/*borde_inf:  Calcula la presion en el borde inferior
en el tiempo n=1
(sin contar la influencia directa de la fuente) */
double borde_inf(double unm1, double unx1, double
unx2, double cte);

/* fuente_en_borde: Contribucion de la fuente en el
borde indicado por
nodo en el paso 'paso' */
double fuente_en_borde(int paso, int nodo);

/* fuente_interior: Contribucion de la fuente en el un
nodo interior */
double fuente_interior(int paso, int nodo, double cte,
double *velnodo);

#endif
```

C.1.7 Módulo D_RUIDO.H

```
/* D_RUIDO.H
   Simulaci¢n de ruido en las presiones generadas.
*/
/* agregarRuido: modifica las presiones alter ndolas
con un porcentaje de
```

```
ruido */
void agregarRuido(double *recmed,int nt,double
wnperc);
```

C.1.8 Módulo EDITOR.H

```
/* EDITOR.H
   Rutinas utilizadas por el objeto TEditor
```

```
*/
ushort doEditDialog( int dialog, ... );
```

C.1.9 Módulo FIELDS.H

```
/* FIELDS.H
Definici¢n de distintos tipos de datos utilizados para
el ingreso por
pantalla. Ampliaci¢n de las clases provistas en los
fuentes de Turbo Vision.
*/
#if !defined( __FIELDS_H )
#define __FIELDS_H

#define Uses_TInputLine
#define Uses_TStreamable
#include <tv.h>

// Accepts only valid numeric input between Min and
Max

class TNumInputLine : public TInputLine
{
public:
          TNumInputLine( const TRect&, int, long, long
);
          virtual ushort dataSize();
          virtual void getData( void *);
          virtual void setData( void *);
          virtual Boolean valid( ushort );
          long min;
          long max;
protected:
          TNumInputLine( StreamableInit ) :
TInputLine( streamableInit ) {};
          virtual void write( opstream& );
          virtual void *read( ipstream& );
private:
          virtual const char *streamableName() const
                    { return name; }
public:
          static const char * const name;
          static TStreamable *build();
};

inline ipstream& operator >> ( ipstream& is,
TNumInputLine& cl )
          { return is >> (TStreamable&)cl; }
inline ipstream& operator >> ( ipstream& is,
TNumInputLine*& cl )
          { return is >> (void *&)cl; }
inline opstream& operator << ( opstream& os,
TNumInputLine& cl )
          { return os << (TStreamable&)cl; }
inline opstream& operator << ( opstream& os,
TNumInputLine* cl )
          { return os << (TStreamable *)cl; }

/*-------------------------------------------------------
------------------*/
class TRealNumInputLine : public TInputLine
{
public:
          TRealNumInputLine( const TRect&, int,
double, double);
          virtual ushort dataSize();
          virtual void getData( void *);
          void setData(void *rec);
          virtual Boolean valid( ushort );
          double min;
          double max;
protected:
          TRealNumInputLine( StreamableInit ) :
TInputLine( streamableInit ) {};
          virtual void write( opstream& );
          virtual void *read( ipstream& );
private:
          virtual const char *streamableName() const
                    { return name; }
public:
          static const char * const name;
          static TStreamable *build();
};

inline ipstream& operator >> ( ipstream& is,
TRealNumInputLine& cl )
          { return is >> (TStreamable&)cl; }
inline ipstream& operator >> ( ipstream& is,
TRealNumInputLine*& cl )
          { return is >> (void *&)cl; }
inline opstream& operator << ( opstream& os,
TRealNumInputLine& cl )
    { return os << (TStreamable&)cl; }
inline opstream& operator << ( opstream& os,
TRealNumInputLine* cl )
          { return os << (TStreamable *)cl; }
/*-------------------------------------------------------
------------------*/
class TPot2NumInputLine : public TInputLine
{
public:
          TPot2NumInputLine( const TRect&, int, long,
long);
          virtual ushort dataSize();
          virtual void getData( void *);
          virtual void setData( void *);
          virtual Boolean valid( ushort );
          long min;
          long max;

protected:
          TPot2NumInputLine( StreamableInit ) :
TInputLine( streamableInit ) {};
          virtual void write( opstream& );
          virtual void *read( ipstream& );
private:
          virtual const char *streamableName() const
                    { return name; }
public:
          static const char * const name;
          static TStreamable *build();
};

inline ipstream& operator >> ( ipstream& is,
TPot2NumInputLine& cl )
          { return is >> (TStreamable&)cl; }
inline ipstream& operator >> ( ipstream& is,
TPot2NumInputLine*& cl )
          { return is >> (void *&)cl; }
inline opstream& operator << ( opstream& os,
TPot2NumInputLine& cl )
          { return os << (TStreamable&)cl; }
inline opstream& operator << ( opstream& os,
TPot2NumInputLine* cl )
          { return os << (TStreamable *)cl; }

#endif  // __FIELDS_H
```

C.1.10 Módulo FORMAT.H

```
/*
 *  archivo:        format.h
 *
 *  objetivo:       especificar los formatos para
archivos de entrada y salida
 */

/* el archivo in se lee de acuerdo a  IN_FORMAT e
IN_VARS */

#define IN_FORMAT " \
                                        Exp. = %d \
                    Tot. Eval. = %d \
                                        Pob. = %d \
          Long. cromosoma = %d \
                    Prob. cruza = %lf \
           Prob. mutaci¢n = %lf \
           Salto generac. = %lf \
          Ventana scaling = %d \
          Interv. reporte = %d \
         Estruc. guardadas = %d \
   Max gens sin eval. = %d \
Interv. entre vuelcos = %d \
          Vuelcos guardados = %d \
                           Opciones = %s \
          Semilla aleatoria = %lu \
             Ranking m¡nimo = %lf "
```

```
#define IN_VARS
&Totalexperiments,&Totaltrials,&Popsize,&Length,\
          &C_rate,&M_rate,&Gapsize,\
          &Windowsize,&Interval,\
          &Savesize,&Maxspin,\
          &Dump_freq,&Num_dumps,\
          Options,\
          &OrigSeed, &Rank_min



/*        LINE_FIN es el formato de cada linea del
archivo out tal como          */
/*        es leida por la rutina report
                              */
#define LINE_FIN "%lf %lf %lf %lf %lf %lf %lf %lf %lf"

#define LINE_VIN
&line[0],&line[1],&line[2],&line[3],&line[4],\
          &line[5],&line[6],&line[7],&line[8]


/*        formatos de salida.           */
```

```
/* OUT_FORMAT es el formato en que se imprimen los
parametros de entrada */

#define OUT_FORMAT "\
                Exp. = %d\n\
          Tot. Eval. = %d\n\
                Pob. = %d\n\
      Long. cromosoma = %d\n\
          Prob. cruza = %lf\n\
       Prob. mutaci¢n = %lf\n\
       Salto generac. = %lf\n\
      Ventana scaling = %d\n\
      Interv. reporte = %d\n\
    Estruc. guardadas = %d\n\
   Max gens sin eval. = %d\n\
Interv. entre vuelcos = %d\n\
    Vuelcos guardados = %d\n\
              Opciones = %s\n\
    Semilla aleatoria = %lu\n\
       Ranking m¡nimo = %lf\n"

/* OUT_VARS son los par metros que se imprimir n de
acuerdo con OUT_FORMAT */
```

```
#define OUT_VARS
Totalexperiments,Totaltrials,Popsize,Length,\
          C_rate,M_rate,Gapsize,\
          Windowsize,Interval,Savesize,Maxspin,\
          Dump_freq,Num_dumps,Options,OrigSeed,Rank_mi
n


/* OUT_F2 es el formato para los datos producidos por
'Measure'.
 * OUT_V2 describe las variables.
 */
#define OUT_F2 "%5d %5d %2d %2d %5.3f %.6e %.6e %.6e
%.6e\n"

#define OUT_V2 Gen,Trials,Lost,Conv,Bias,Online,\
          Offline,Best,Ave_current_perf



/** fin de archivo **/
```

C.1.11 Módulo GLOBAL.H

```
/*
 *  archivo:        global.h
 *
 *  objetivo:       variables globales
 */


/* El archivo in especifica estos par metros */
int    Totalexperiments; /* cantidad de experimentos
                        */
int    Totaltrials; /* evaluaciones por experimentos
                        */
int    Popsize;                 /* tama¤o de la
poblaci¢n */
int    Length;                  /* longitud en bits de
la estructura */
double  C_rate;                 /* probabilidad de
cruza*/
double  M_rate;                 /* probabilidad de
mutacion */
double  Gapsize;    /* fracci¢n de la poblaci¢n que
ser  reemplazada cada generaci¢n */
int    Windowsize;  /* usado para actualizar la peor
performance */
int    Interval;    /* cantidad de evaluaciones entre
estad¡sticas        */
int    Savesize;    /* n£mero de estructuras que se
guardan en minfile  */
int    Maxspin;                 /* max gens sin
evaluaciones                    */
int    Dump_freq;   /* gens entre checkpoints
          */
int    Num_dumps;   /* cantidad de archivos de
checkpoint que se guardan       */
char   Options[40]; /* opciones
                        */
unsigned long    Seed;          /* semilla para el
generador aleatorio */
unsigned long    OrigSeed; /* semilla inicial     */

/* datos estad¡sticos y variables auxiliares */
double  Ave_current_perf;/* aptitud promedio de la
generaci¢n actual   */
double  Best;                   /* la mejor aptitud
hasta ahora         */
double  Best_current_perf;/* la mejor aptitud en la
generaci¢n actual   */
int    Best_guy;    /* numero de estructura con
best_current_perf               */
int    Bestsize;    /* cantidad de mejores estructuras
grabadas */
```

```
double  Bias;                   /* promedio de la
dominaci¢n de alelos                    */
int    Bytes;                   /* longitud en bytes de
la estructura empaquetada */
int    Conv;                    /* n£mero de genes que
convergieron parcialmente       */
char   Doneflag;                /* se setea cuando se
cumple la condici¢n de parada */
int    Experiment;  /* contador de experimentos
                        */
int    Gen;                     /* contador         de
generaciones                    */
unsigned long    Initseed; /* semilla inicial del
experimento         */
int    Lost;                    /* cantidad de
posiciones que convergieron totalmente */
int    Mu_next;                 /* pr¢xima posici¢n a
mutar               */
double  Offline;    /* performance offline
                        */
double  Offsum;                 /* acumulador para la
performance offline             */
double  Online;                 /* performance online
                                */
double  Onsum;                  /* acumulador para
performance online  */
int    Plateau;                 /* contador de
evaluaciones para la siguiente salida   */
double Rank_min;    /* minima tasa de muestreo para
seleccion por ranking           */
int    Spin;                    /* cantidad de
generaciones desde la £ltima evaluaci¢n */
double  Totbest;    /* mejor de todos los experimentos
                                */
double  Totoffline; /* offline de todos los
experimentos                            */
double  Totonline;  /* online de todos los
experimentos                            */
int    Trials;                  /* contador de
evaluaciones */
double  *Window;    /* cola circular de los peores de
las £ltimas generaciones        */
double  Worst;                  /* peor performance
hasta ahora         */
double  Worst_current_perf;/* peor perf en la
generaci¢n actual */


/** fin de archivo **/
```

C.1.12 Módulo SETUP.H

```
/*
SETUP.H
Lectura de par metros para correr el algoritmo
gen,tico.
Lee del archivo presetup y del input del usuario.
```

```
Escribe en los archivos in y en template.
*/
Boolean setup (int version, char *ext, unsigned long&
experimentos);
```

C.1.13 Módulo SETUPDAT.H

```
/*-------------------------------------------------------
------------------*
SETUPDAT.H
Objeto para manejo de datos de setup.
De esta clase derivan stupdat1, stupdat2 y stupdat3
que implementan
los objetos para las tres versiones de ag.
/*-------------------------------------------------------
------------------*/
#define __SETUPDAT_H
/*-------------------------------------------------------
-----------*/
class SetupData
```

```
{
public:
struct {


          unsigned long experimentos;
          unsigned long trials;
          unsigned long poblacion;
          double pcruza;
          double pmut;
          double gap;
          unsigned long scalingWindow;
          unsigned long maxGenSinEval;
          unsigned long randomSeed;
```

```
          double rankMin;
          unsigned int checkBoxSeleccion;
          unsigned int  checkBoxCodificacion; //
oculto para versiones 2 y 3
          unsigned long valores;  // oculto para
versiones 2 y 3
          int bitlength;      // oculto
          int genes;     // oculto
          double min;         // oculto
          double max;         // oculto
          char options[20];  // oculto
          char formatoGen[6]; // oculto
} datos;
          char dialogName[20];
          virtual void setDefault();
          virtual void leerPresetup(char *file);
          virtual void escribirParametros(char *ext);
          virtual void derivarDatos();
          virtual unsigned short obtenerParametros();
          virtual unsigned long getExperimentos();
};
```

C.1.14 Módulo STUPDAT1.H

```
/*-------------------------------------------------------
------------------*
STUPDAT1.H
Objeto para manejo de datos de setup en la propuesta 1
de ag.
/*-------------------------------------------------------
------------------*/
#if !defined( __SETUPDAT_H )
#include "setupdat.h"
#endif
/*-------------------------------------------------------
-----------*/
class SetupDataV1 : public SetupData
{
public:
          void derivarDatos();
          void setDefault();
          SetupDataV1::SetupDataV1();
};
```

C.1.15 Módulo STUPDAT2.H

```
/*-------------------------------------------------------
------------------*
STUPDAT2.H
Objeto para manejo de datos de setup en la propuesta 2
de ag.
/*-------------------------------------------------------
------------------*/
#define __SETUPDAT2_H
#if !defined( __SETUPDAT_H )
#include "setupdat.h"
#endif
/*-------------------------------------------------------
-----------*/
class SetupDataV2 : public SetupData
{
public:
          virtual void derivarDatos();
          virtual void setDefault();
          SetupDataV2();
};
```

C.1.16 Módulo STUPDAT3.H

```
/*-------------------------------------------------------
------------------*
STUPDAT3.H
Objeto para manejo de datos de setup en la propuesta 3
de ag.
/*-------------------------------------------------------
------------------*/
#if !defined( __SETUPDAT_H )
#include "setupdat.h"
#endif
class SetupDataV3 : public SetupData
{

public:

struct {
          unsigned long experimentos;
          unsigned long trials;
          unsigned long poblacion;
          double pcruza;
          double pmut;
          double gap;
          unsigned long scalingWindow;
          unsigned long maxGenSinEval;
          unsigned long randomSeed;
          double rankMin;
          unsigned int checkBoxSeleccion;
          double errorPresion;  // s¢lo para versi¢n 3
          unsigned long valores;  // oculto para
versiones 2 y 3
          int bitlength;      // oculto
          int genes;     // oculto
          double min;         // oculto
          double max;         // oculto
          char options[20]; // oculto
          char formatoGen[6];
} datos;

public:
          void setDefault();
          unsigned short obtenerParametros();
          void escribirParametros(char *ext);
          void derivarDatos();
          void leerPresetup(char *file);
          SetupDataV3();
          unsigned long getExperimentos();
};
```

C.1.17 Módulo TVUTILS.H

```
/* TVUTILS.H
   Rutinas que muestran o reciben datos de pantalla.
*/
#define Uses_TDialog
#define Uses_TParamText
#include <tv.h>

/* Clase TCartel: cartel en pantalla */
class TCartel : public TDialog
{
public:
          TCartel( const TRect& bounds, const char
*aTitle, const char *aText, int aParamCount= 0);
};

/* execDialog: Ejecuta una ventana de dialogo y
retorna la condicion de
salida (cmOk o cmCancel) */
ushort execDialog( TDialog *d, void *data );

/* makeEspere: crea una ventana de dialogo que muestra
un cartel mientras
se simula el experimento */

TDialog *makeEspere();
/* elegioExtensionArchivo:
 recibe una nombre de archivo con wildcards,
 permite que el usuario elija un archivo validando que
el mismo responda
 a la m scara indicada y se encuentre en el directorio
activo
 y devuelve la extensi¢n del archivo elegido.
 Retorna true si  eligio y falso si cancelo.
*/
Boolean eligioExtensionArchivo(const char* nombre,
char* ext);
```

C.1.18 Módulo UTILS.H

```
/* UTILS.H
```

```
   Rutinas de acceso a archivos y rutinas de c lculo
utilizadas en el
   programa
*/
#include <stdio.h>
/* readFile: lee una linea de un archivo ignorando los
comentarios */
char *readFile(FILE *fp, char *line);

/* ilog2: calcula el logaritmo en base 2 de un n£mero
potencia de 2 */
int ilog2(unsigned long n);
```

```
/* cantbits: cantidad de bits necesarios para
almacenar n valores */
int cantbits(unsigned long n);

/* fileExists: indica si el archivo dado existe */
int fileExists(char *fname);

/* current_drive: devuelve el drive activo */
char *current_drive(char *path);

/* current_dir: devuelve el directorio actual */
char *current_dir(char *path);
```

C.1.19 Módulo AG.CPP

```
/*
 *  archivo:        ag.cpp
 *
 *  objetivo:   disparar simulacion del experimento y
las distintas
 *                                      propuestas de
algoritmos geneticos.
 */
#define Uses_TProgram
#define Uses_TDeskTop
#define Uses_MsgBox
#include <tv.h>
#include <process.h>
#include <dir.h>
#include <string.h>
#include <stdio.h>
#include <errno.h>
#include "ag.h"
#include "tvutils.h"
#include "setup.h"
#include "app.h"
#include "utils.h"


void ag(int version) {

int result;
char ext[6];
char errMsg[80];
extern int errno;
unsigned long cantExperimentos;
if (setup(version, ext, cantExperimentos)) {
   ((TApp *)TProgram::application)-
>setearComandos(False);
   ((TApp *)TProgram::application)-
>disableCommand(cmSalir);
   ((TApp *)TProgram::application)-
>disableCommand(cmMenu);
   TProgram::application->redraw();

   switch (version) {
          case 1:
                    result = spawnl(P_WAIT, "ag1.exe",
"ag1", ext+1);
                    break;
          case 2:
                    result = spawnl(P_WAIT, "ag2.exe",
"ag2", ext+1);
                    break;
          case 3:
                    result = spawnl(P_WAIT, "ag3.exe",
"ag3", ext+1);
                    break;
   }

   if (result == -1)
   {
                    // informar error
                    if (errno == ENOMEM)
                              sprintf(errMsg,"No hay
suficiente memoria para ejecutar el algoritmo
gen,tico");
                    else
                              sprintf(errMsg,"No se
pudo ejecutar el algoritmo gen,tico. Error %d",errno);
                    messageBox(errMsg,mfError);
   }
   else {
                    report(ext+1,version);

          graficar(ext,version,cantExperimentos);
   }
   ((TApp *)TProgram::application)-
>setearComandos(True);
   ((TApp *)TProgram::application)-
>enableCommand(cmSalir);
   ((TApp *)TProgram::application)-
>enableCommand(cmMenu);
   TProgram::application->redraw();
}
}
/*-------------------------------------------------
--------------*/
void simulacion() {
          // abrir dialog para que elija modelo.*
          // si eligio -> llamar a la rutina directo
TDialog *cartel;
char ext[MAXEXT];
   if (eligioExtensionArchivo("modelo", ext) ) {
                    cartel = makeEspere();
                    TProgram::deskTop->insert(cartel);
                    directo(&ext[1]);    // extensi¢n
sin punto
                    TProgram::deskTop-
>destroy(cartel);
   }
}

/** fin de archivo **/
```

C.1.20 Módulo APP.CPP

```
/*
 *  archivo:        app.cpp
 *
 *  objetivo:   modulo principal. Definicion del
objeto TApp para el manejo
 *              de la aplicacion.
 */
#define Uses_TApplication
#define Uses_TEditWindow
#define Uses_TDeskTop
#define Uses_TRect
#define Uses_TEditor
#define Uses_TFileEditor
#define Uses_TFileDialog
#define Uses_TResourceFile
#define Uses_fpstream
#define Uses_MsgBox
#define Uses_TStatusLine
#define Uses_TStatusItem
#define Uses_TMemo
#define Uses_TMenu
#define Uses_TMenuBar
#define Uses_TSubMenu
#define Uses_TMenuItem
#include <tv.h>

#include "app.h"
#include "ag.h"
#include "editor.h"
#include "tvutils.h"
#include <stdlib.h>
#include <stdarg.h>
#include <strstrea.h>
```

```
#include <iomanip.h>

__link (RResourceCollection)
__link (RStatusLine)
__link (RMenuBar)
__link (RDialog)
__link (RLabel)
__link (RInputLine)
__link (RButton)
__link (RCheckBoxes)
__link (RNumInputLine)
__link (RRealNumInputLine)
__link (RPot2NumInputLine)
__link (RCluster)
__link (RMenuView)
__link (RMenuBox)
TEditWindow *clipWindow;
TResourceFile *rez; // archivo de recursos (menues,
dialog boxes)

/*-------------------------------------------------
------------------*/
void TApp::setearComandos(Boolean habilitar) {

TCommandSet ts;

          ts.enableCmd( cmAbrir );
          ts.enableCmd( cmModelo );
          ts.enableCmd( cmVersion1 );
          ts.enableCmd( cmVersion2 );
          ts.enableCmd( cmVersion3 );
          ts.enableCmd( cmSimulacion );
```

```
          if (habilitar) {
                    enableCommands(ts);
          }
          else {
                    disableCommands(ts);
          }
}
/*-------------------------------------------------------
------------------*/
TEditWindow *TApp::openEditor( const char *fileName,
Boolean visible )
{
          TRect r = deskTop->getExtent();
          TView *p = validView( new TEditWindow( r,
fileName, wnNoNumber ) );
          if( !visible )
                    p->hide();
          deskTop->insert( p );
          return (TEditWindow *)p;
}
/*-------------------------------------------------------
------------------*/
TApp::TApp() :
          TProgInit( TApp::initStatusLine,
                              TApp::initMenuBar,
                              TApp::initDeskTop
                            ),
          TApplication()
{

          ventanaActiva = NULL;

          TCommandSet ts;
          ts.enableCmd( cmSave );
          ts.enableCmd( cmSaveAs );
          disableCommands( ts );

          setearComandos(True);
          redraw();


          TEditor::editorDialog = doEditDialog;

          clipWindow = openEditor( 0, False );
          if( clipWindow != 0 )
                    {
                    TEditor::clipboard = clipWindow-
>editor;
                    TEditor::clipboard->canUndo =
False;
                    }
}
/*-------------------------------------------------------
------------------*/
void TApp::fileOpen(const char *mascara)
{
          char fileName[MAXPATH];
          strcpy( fileName, mascara );

          if( execDialog( new TFileDialog( mascara,
"Abrir archivo",
                              "~N~ombre",
fdOpenButton, 100 ), fileName) != cmCancel ) {
                    ventanaActiva = openEditor(
fileName, True );
                    setearComandos(False);
    }
}
/*-------------------------------------------------------
------------------*/
void TApp::handleEvent( TEvent& event )
{
          if( event.what == evCommand ) {
                    switch( event.message.command )
                              {
                              case cmAbout:
                                        TDialog *pd =
(TDialog *)rez->get("aboutDialog");
                                        if ( pd ) {

          TProgram::deskTop->execView( pd );
                                        }

          TProgram::deskTop->destroy( pd );
                                        break;
                              case cmModelo:

          fileOpen("modelo.*");
                                        break;
                              case cmAbrir:

          fileOpen("*.*");
                                        break;
                              case cmClose:
                                        ventanaActiva-
>close();
                                        ventanaActiva
= NULL;

          setearComandos(True);
                                        clearEvent(
event );
                                        break;
                              case cmVersion1:
                                        ag(1);
                                        break;
                              case cmVersion2:
                                        ag(2);
                                        break;
                              case cmVersion3:
                                        ag(3);
                                        break;
                              case cmSalir:
                                        if
(ventanaActiva)

          ventanaActiva->close();

          endModal(cmQuit);
                                        break;

                              case cmSimulacion:
                                        simulacion();
                                        break;
                              }
          }
          TApplication::handleEvent( event );
          clearEvent( event );
}

int main()
{
          fpstream *ofps;     // streamable object
para utilizar recursos
          const char rezFileName[]= "recursos.rez"; //
nombre del archivo de recursos
          ofps= new fpstream(rezFileName,
ios::in|ios::binary);
          if (!ofps->good()) {
             messageBox("Error al inicializar la
stream del sistema.",mfInformation);
             return 1;
          }
          rez= new TResourceFile(ofps);
          if (!rez) {
             messageBox("Error al abrir los recursos
del sistema.",mfInformation);
             return 1;
          }

          TApp agApp;
          agApp.run();
          return 0;
}
/*-------------------------------------------------------
------------------*/
TStatusLine *TApp::initStatusLine(TRect)
{
          return (TStatusLine *)rez-
>get("statusLine");
}
/*-------------------------------------------------------
---------------*/
TMenuBar *TApp::initMenuBar(TRect)
{
          return (TMenuBar *)rez->get("menuBar");
}
/*-------------------------------------------------------
---------------*/
void TApp::outOfMemory()
{
          messageBox("No hay suficiente memoria para
esta operaci¢n.", mfError | mfOKButton );
}

/** fin de archivo **/
```

C.1.21 Módulo DIBU.CPP

```
/*
 *  archivo:        dibu.cpp
 *
 *  objetivo: Hace un gr fico de barras comparando las
velocidades reales
 *                              y calculadas
 */

#include <math.h>
#include <stdio.h>
#include <dir.h>
#include <alloc.h>
#include "utils.h"
int obtener_solucion(const char *extension,
                                                  int
&cantMat, double **velMaterial,
                                                  int
&cantCapas, double **velReal) {
/* Leer archivo solucion para obtener velocidades
verdaderas. */
          FILE *fp;
          char file[13], line[200];
          int i;
          float auxin;
          int dummy;

          sprintf(file, "solucion%s", extension);
          if ((fp = fopen(file, "r")) == NULL)
                    return 1;
```

```
          /* cantidad de capas */
          readFile(fp, line);
          sscanf(line,"%d", &cantCapas);
          /* cantidad de materiales */
          readFile(fp, line);
          sscanf(line,"%d", &cantMat);
          /* material de cada capa  (no me interesa)*/
          for (i=0; i < cantCapas; i++) {
                    readFile(fp, line);
                    sscanf(line, "%d", &dummy);
          }

          /* velocidad de cada material */
          *velMaterial = new double[cantMat];
          for (i=0; i < cantMat; i++) {
                    readFile(fp, line);
                    sscanf(line,"%f", &auxin);
                    (*velMaterial)[i] = auxin;
          }

          /* velocidad real del terreno */
          *velReal = new double[cantCapas];
          for (i=0; i < cantCapas; i++) {
                    readFile(fp, line);
                    sscanf(line,"%f", &auxin);
                    (*velReal)[i] = auxin;
          }

          fclose(fp);

          return 0;

}
/*-------------------------------------------------------
------------------*/
int obtener_veloc_calculada(int nroExp, const char
*extension, int version,
 int cantMat, double *velMaterial,
 int cantCapas, double *velCalc) {
/* Leer archivo min para obtener materiales o
velocidades calculados */
          FILE *fp;
          char file[13];
          int i, resul;
          float auxin;
          int dummy;
          double *velAux;
          double errMin, error;


          /* levantar de min.x la solucion */
          sprintf(file, "min%d%s", nroExp,extension);
          if ((fp = fopen(file, "r")) == NULL) {
                    return 3;
          }

          errMin = -1;
          velAux = new double[cantCapas];

          /* Buscar la mejor soluci¢n de entre las
registradas en el archivo
             min */
          resul = fscanf(fp,"%f", &auxin);
          while (resul != 0 && resul != EOF) {
                    velAux[0] = auxin;
                    for (i=1; i < cantCapas; i++) {
                              fscanf(fp,"%f", &auxin);
                              velAux[i] = auxin;
                    }
                    fscanf(fp, "%f", &auxin);
                    error = auxin;
                    if (errMin == -1 || error <
errMin) {
                              // actualizar velocidad
optima
                              errMin = error;
                              for (i=0; i < cantCapas;
i++)
                                        velCalc[i] =
velAux[i];
                    }
                    // leer el par de dummy que hay
atras
                    fscanf(fp, "%d",dummy);
                    fscanf(fp, "%d",dummy);
                    // leer el primer valor del
siguiente renglon
                    resul = fscanf(fp,"%f", &auxin);
          }

          delete velAux;
          fclose(fp);
          /* Si no es la version 1, trabaj, con
materiales, debo traducir */
          if (version > 1) {
                    for (i=0; i < cantCapas; i++)
                                        velCalc[i] =
velMaterial[velCalc[i]];
          }
          return 0;

}

/*-------------------------------------------------------
--------------*/
void hacer_grafico(int nroExp, const char *ext, int
cant_capas, double *vel_real, double *vel_calc) {
          int i, j;
          FILE *fp;
          char graffile[MAXPATH];
          // Escala 0.1 = 1 caracter

          sprintf(graffile, "grafico%i%s",nroExp,ext);
          fp = fopen(graffile, "w");

/* Vertical Archivo */
// Buscar la maxima velocidad a ser representada
          double max=0;
          for (i=0;i < cant_capas; i++) {
                    max = vel_real[i]>max? vel_real[i]
: max;
                    max = vel_calc[i]>max? vel_calc[i]
: max;
          }

int margen_izq = 5;
int margen_sup = 5;
int u = 2; // tama¤o de la unidad de velocidad en el
gr fico
int gap = 4; // distancia entre capa y capa
int k;

fprintf(fp,"               Velocidades observadas y
calculadas por capa\n");

          // dejar renglones arriba
          for (i=0;i < margen_sup; i++)
fprintf(fp,"\n");

          for (i=ceil(max)*u; i > 0; i--) {
                    // margen izquierdo
                    for (j=0;j < margen_izq; j++)
fprintf(fp," ");

                    // raya vertical
                    if (i % u == 0)
                              fprintf(fp,"%1.1fÅ",1.0
* i / u);
                    else
                              fprintf(fp,"   ³");

                    // datos
                    for (j=0; j < cant_capas; j++) {
                              if (vel_real[j] >= 1.0 *
i / u)

          fprintf(fp,"°");
                              else
                                        fprintf(fp,"
");

                              if (vel_calc[j] >= 1.0 *
i / u)

          fprintf(fp,"²");
                              else
                                        fprintf(fp,"
");

                              for (k= 0; k < gap - 2;
k++) fprintf(fp," ");
                    }

                    fprintf(fp,"\n");
          }
          // rayita de abajo
          for (j=0;j < margen_izq+3; j++) fprintf(fp,"
");
          fprintf(fp,"À");
          for (j=0; j < cant_capas; j++) {
                    fprintf(fp,"ÄÄÄÄ");
          }
          fprintf(fp,"\n");
          for (j=0;j < margen_izq+4; j++) fprintf(fp,"
");
          for (j=0; j < cant_capas; j++) {
                    fprintf(fp,"%2d  ",j+1);
          }
          fprintf(fp,"\n\n");
          fprintf(fp,"                         Velocidad
real:      °\n");
          fprintf(fp,"\r\n");
          fprintf(fp,"                         Velocidad
calculada: ²\n");

          fclose(fp);
}
/*-------------------------------------------------------
---------------*/
int graficar(char *ext, int version, unsigned long
cantExperimentos) {
int cantCapas, cantMat;
double **velReal, **velMaterial;
double *velCalc;
int resul=0;
/* Hace un archivo por cada experimento */
          velReal = new double *;
          velMaterial = new double *;
          resul = obtener_solucion(ext, cantMat,
velMaterial, cantCapas, velReal);
          velCalc = new double[cantCapas];
          if (resul == 0)
                    for (int i=0; i <
cantExperimentos; i++) {
                              resul =
obtener_veloc_calculada(i,ext, version,
                               cantMat, *velMaterial,
cantCapas, velCalc);
```

```
                                hacer_grafico(i,
ext,cantCapas, *velReal, velCalc);
                        }
          delete *velReal;
          delete *velMaterial;
          delete velReal;
          delete velCalc;
          delete velMaterial;
          return resul;
}

/* fin de archivo */
```

C.1.22 Módulo EDITOR.CPP

```
/*
 * archivo editor.cpp
 *
 * objetivo:   Rutinas utilizadas por el objeto
TEditor
 */
#define Uses_MsgBox
#define Uses_TApplication
#define Uses_TButton
#define Uses_TCheckBoxes
#define Uses_TDeskTop
#define Uses_TDialog
#define Uses_TEditor
#define Uses_TFileDialog
#define Uses_THistory
#define Uses_TInputLine
#define Uses_TLabel
#define Uses_TObject
#define Uses_TPoint
#define Uses_TProgram
#define Uses_TRect
#define Uses_TSItem

#include <tv.h>

#include "editor.h"
#include "app.h"
#include "tvutils.h"
#include <stdlib.h>
#include <stdarg.h>
#include <strstrea.h>
#include <iomanip.h>
/*-------------------------------------------------------
-------------------*/
typedef char *_charPtr;
typedef TPoint *PPoint;

#pragma warn -rvl

ushort doEditDialog( int dialog, ... )
{
          va_list arg;

          char buf[80];
          ostrstream os( buf, sizeof( buf ) );
          switch( dialog )
                    {
                    case edOutOfMemory:
                              return messageBox( "No
hay suficiente memoria para esta operaci¢n",

                       mfError | mfOKButton );
                    case edReadError:
                              {
                              va_start( arg, dialog );
                              os << "Error leyendo
archivo " << va_arg( arg, _charPtr )
                                 << "." << ends;
                              va_end( arg );
                              return messageBox( buf,
mfError | mfOKButton );
                              }

                    case edWriteError:
                              {
                              va_start( arg, dialog );
                              os << "Error escribiendo
archivo " << va_arg( arg,_charPtr )
                                 << "." << ends;
                              va_end( arg );
                              return messageBox( buf,
mfError | mfOKButton );
                              }

                    case edSaveModify:
                              {
                              va_start( arg, dialog );
                              os << va_arg( arg,
_charPtr )
                                 << " ha sido
modificado. ¨Desea grabarlo?" << ends;
                              va_end( arg );
                              return messageBox( buf,
mfInformation | mfYesNoCancel );
                              }

                    case edSaveAs:
                              {
                              va_start( arg, dialog );
                              return execDialog( new
TFileDialog( "modelo.*",


                    "Grabar archivo como",


                    "~N~ombre",


                    fdOKButton,


                    101 ), va_arg( arg, _charPtr ) );
                              }

                    }
}

#pragma warn .rvl

/** fin de archivo **/
```

C.1.23 Módulo FIELDS.CPP

```
/*
 * archivo: fields.cpp
 *
 * Objetivo: definici¢n de distintos tipos de datos
utilizados para el ingreso
 * por pantalla. Ampliaci¢n de las clases provistas en
los fuentes de
 * Turbo Vision.
 */
#define Uses_TStreamableClass
#define Uses_TInputLine
#define Uses_TStreamable
#define Uses_MsgBox
#include <tv.h>
__link( RInputLine )

#if !defined( __FIELDS_H )
#include "fields.h"
#endif  // __FIELDS_H

#if !defined( __STRING_H )
#include <string.h>
#endif  // __STRING_H

#if !defined( __STDLIB_H )
#include <stdlib.h>
#endif  // __STDLIB_H

#if !defined( __STRSTREAM_H )
#include <strstream.h>
#endif  // __STRSTREAM_H

#if !defined( __STDIO_H )
#include <stdio.h>
#endif  // __STDIO_H
```

```
#include "utils.h"

// TNumInputLine: n£mero entero
/* -------------------------------------------------
-- */
const char * const TNumInputLine::name =
"TNumInputLine";

void TNumInputLine::write( opstream& os )
{
          TInputLine::write( os );
          os << min;
          os << max;

}

void *TNumInputLine::read( ipstream& is )
{
          TInputLine::read( is );
          is >> min;
          is >> max;
          return this;
}

TStreamable *TNumInputLine::build()
{
          return new TNumInputLine( streamableInit );
}


TStreamableClass RNumInputLine( TNumInputLine::name,

                                TNumInputLine::build,
```

```
                                        __DELTA(TNumInputLine)
                                    );

TNumInputLine::TNumInputLine( const TRect& bounds,
                                            int aMaxLen,
                                            long aMin,
                                            long aMax )
:
          TInputLine(bounds, aMaxLen)
{
          min = aMin;
          max = aMax;
}


ushort TNumInputLine::dataSize()
{
          return sizeof(long);
}

void TNumInputLine::getData( void *rec )
{
          *(long *)rec = atol(data);
}

void TNumInputLine::setData( void *rec )
{
          ltoa(*(long *)rec, data, 10);
          selectAll(True);
}

Boolean TNumInputLine::valid( ushort command )
{
          long value;
          Boolean ok;
          char msg[80];
          ostrstream os(msg, 80);

          ok = True;
          if ( (command != cmCancel) && (command !=
cmValid) )
                    {
                    if (strlen(data) == 0)
                              strcpy(data,"0");
                    value = atol(data);
                    if ( (value == 0) || (value < min)
|| (value > max) )
                              {
                              select();
                              os << "El n£mero debe
estar entre" << min << " y " << max << "." << ends;
                              messageBox(os.str(),
mfError + mfOKButton);
                              selectAll(True);
                              ok = False;
                              }
                    }
          if (ok)
                    return TInputLine::valid(command);
          else
                    return False;
}

// TRealNumInputLine : n£mero real
/* ------------------------------------------------
-- */
const char * const TRealNumInputLine::name =
"TRealNumInputLine";

void TRealNumInputLine::write( opstream& os )
{
          TInputLine::write( os );
          os << min;
          os << max;

}

void *TRealNumInputLine::read( ipstream& is )
{
          TInputLine::read( is );
          is >> min;
          is >> max;
          return this;
}

TStreamable *TRealNumInputLine::build()
{
          return new TRealNumInputLine( streamableInit
);
}


TStreamableClass RRealNumInputLine(
TRealNumInputLine::name,


TRealNumInputLine::build,


__DELTA(TRealNumInputLine)
                                    );

TRealNumInputLine::TRealNumInputLine( const TRect&
bounds,
                                        int aMaxLen,
                                              double aMin,
                                              double aMax
) :
          TInputLine(bounds, aMaxLen)
{
          min = aMin;
          max = aMax;
}


ushort TRealNumInputLine::dataSize()
{
            return sizeof( double);
}

void TRealNumInputLine::getData( void *rec )
{
          *(double *)rec = atof(data);
}

void TRealNumInputLine::setData(void *rec)
{
          sprintf(data,"%g",*(double *)rec);
          selectAll(True);
}

Boolean TRealNumInputLine::valid( ushort command )
{
          double value;
          Boolean ok;
          char msg[80];
          ostrstream os(msg, 80);

          ok = True;
          if ( (command != cmCancel) && (command !=
cmValid) )
                    {
                    if (strlen(data) == 0)
                              strcpy(data,"0");
                    value = atof(data);
                    if ( (value == 0) || (value < min)
|| (value > max) )
                              {
                              select();
                              os << "El n£mero debe
estar entre" << min << " y " << max << "." << ends;
                              messageBox(os.str(),
mfError + mfOKButton);
                              selectAll(True);
                              ok = False;
                              }
                    }
          if (ok)
                    return TInputLine::valid(command);
          else
                    return False;
}
// TPot2NumInputLine : n£mero potencia de 2
/* ------------------------------------------------
-- */
const char * const TPot2NumInputLine::name =
"TPot2NumInputLine";

////
void TPot2NumInputLine::write( opstream& os )
{
          TInputLine::write( os );
          os << min;
          os << max;

}

void *TPot2NumInputLine::read( ipstream& is )
{
          TInputLine::read( is );
          is >> min;
          is >> max;
          return this;
}

TStreamable *TPot2NumInputLine::build()
{
          return new TPot2NumInputLine( streamableInit
);
}


TStreamableClass RPot2NumInputLine(
TPot2NumInputLine::name,


TPot2NumInputLine::build,


__DELTA(TPot2NumInputLine)

                                    );

TPot2NumInputLine::TPot2NumInputLine( const TRect&
bounds,
                                        int aMaxLen,
                                              long aMin,
                                        long aMax ) :
          TInputLine(bounds, aMaxLen)
{
    min = aMin;
          max = aMax;
}
```

```
ushort TPot2NumInputLine::dataSize()
{
    return sizeof(long);
}

void TPot2NumInputLine::getData( void *rec )
{
          *(long *)rec = atol(data);
}

void TPot2NumInputLine::setData( void *rec )
{
          ltoa(*(long *)rec, data, 10);
          selectAll(True);
}

Boolean TPot2NumInputLine::valid( ushort command )
{
          long value;
          Boolean ok;
          char msg[80];
          ostrstream os(msg, 80);

          ok = True;
          if ( (command != cmCancel) && (command !=
cmValid) )
                    {
                    if (strlen(data) == 0)
                              strcpy(data,"0");
                    value = atol(data);
                    if ( (value == 0) || (value < min)
|| (value > max) )
                              {
                              select();
                              os << "Number must be
from " << min << " to " << max << "." << ends;
                              messageBox(os.str(),
mfError + mfOKButton);
                              selectAll(True);
                              ok = False;
                              }
                              else {
                                if ( (1L <<
ilog2(value)) != value) {;
                                        select();
                                        os << "El
n£mero debe ser potencia de 2." << ends;
          messageBox(os.str(), mfError + mfOKButton);

          selectAll(True);
                                        ok = False;
                                }
                              }
                    }
          if (ok)
                    return TInputLine::valid(command);
          else
                    return False;
}
/** fin de archivo **/
```

C.1.24 Módulo SETUP.CPP

```
/*
 * archivo: setup.cpp
 *
 * objetivo: Lectura de par metros para correr el
algoritmo gen,tico.
 *                              Lee del archivo
presetup y del input del usuario.
 *                              Escribe en los archivos
in y en template.
 */
#include <dir.h>
#include <string.h>
#include <stdio.h>
#include "tvutils.h"
#include "setup.h"
#include "stupdat1.h"
#include "stupdat2.h"
#include "stupdat3.h"
/*-------------------------------------------------------
------------------*/
/* setup:
Par metros de entrada:
version
ext (extensi¢n de los archivos)
Par metros de salida:
cantExperimentos: cantidad de veces que se correr  el
experimento.
*/
Boolean setup (int version, char *ext, unsigned long&
cantExperimentos) {

          char presetupfile[MAXPATH];
          ushort eleccion;
          SetupData *setupData;
          Boolean seteado;

          if (eligioExtensionArchivo("campo",ext)) {
                    switch (version) {
                              case 1: setupData = new
SetupDataV1();

          break;
                              case 2: setupData = new
SetupDataV2();

          break;
                              case 3: setupData = new
SetupDataV3();

          break;
                    }

          sprintf(presetupfile,"presetup%s",ext);
                    setupData-
>leerPresetup(presetupfile);
                    eleccion = setupData-
>obtenerParametros();

                    if (eleccion != cmCancel ) {
                              setupData-
>derivarDatos();
                              setupData-
>escribirParametros(ext); /*escribir in y template*/
                              cantExperimentos =
setupData->getExperimentos();
             }
             delete setupData;
             seteado = (eleccion!= cmCancel);
          }
          else {
                    seteado = False;
    }
          return seteado;
}
/** fin de archivo **/
```

C.1.25 Módulo SETUPDAT.CPP

```
/*
 * archivo setupdat.cpp
 *
 * objetivo: Objeto para manejo de datos de setup.
 * De esta clase derivan stupdat1, stupdat2 y stupdat3
que implementan
 * los objetos para las tres versiones de ag.
 */
#define Uses_TDeskTop
#define Uses_TProgram
#define Uses_TDialog
#define Uses_TResourceFile
#include <tv.h>
#include <dir.h>
#include <string.h>
#include "utils.h"
#include "setupdat.h"
extern TResourceFile *rez;
/*-------------------------------------------------------
-----------*/
void SetupData::setDefault() {
          datos.experimentos = 1;
          datos.trials = 1;
          datos.poblacion = 5;
          datos.pcruza = 0.6;
          datos.pmut =0.001;
          datos.gap = 1.0;
          datos.scalingWindow = 5;
          datos.maxGenSinEval = 2;
          strcpy(datos.options,"cDflLtboO"); /*
opciones para genesis */
          /* c: lleva estad¡sticas de performance
             D: Muestra estad¡sticas de performance
por pantalla
             f: Usa representaci¢n de punto flotante
para los genes.
             l: Guarda en un log la hora de comienzo y
fin de cada experimento.
             L: guarda la ultima generacion en un
archivo de checkpoint ckpt.
             t: Muestra que proceso est  ejecutando a
cada momento.
             b: Si se hacen varios experimentos,
escribe el promedio de los mejores
                valores sobre todos los
experimentos.
             o: Al final de los experimentos escribe
la performance on-line promedio
             O: Al final de los experimentos escribe
la performance off-line promedio
          */
          strcpy(datos.formatoGen,"%8.5f");
          datos.randomSeed = 123456789;
          datos.rankMin = 0.75;
          datos.checkBoxSeleccion = 1;
          datos.checkBoxCodificacion = 0;
```

```
}
/*-------------------------------------------------------
-----------------*/
void SetupData::leerPresetup(char *file) {
          char line[200];
          FILE *fpre;

          fpre = fopen(file, "r");

          // leer cantidad de genes
          readFile(fpre, line);
          sscanf(line,"%d", &datos.genes);

          // leer el rango de valores posibles para
los genes
          readFile(fpre, line);
          sscanf(line,"%lf", &datos.min);
          readFile(fpre, line);
          sscanf(line,"%lf", &datos.max);

          // granularidad
          readFile(fpre, line);
          sscanf(line,"%lu", &datos.valores);
          fclose(fpre);
}
/*-------------------------------------------------------
------------------*/
void SetupData::escribirParametros(char *ext) {
          char infile[MAXPATH];
          char templatefile[MAXPATH];
          FILE *fp;
          int  j;

          sprintf(infile, "in%s", ext);
          sprintf(templatefile, "template%s",ext);

          fp = fopen(templatefile, "w");
          fprintf(fp, "genes: %d\n\n", datos.genes);
          printf("\n");


          for (j=0; j < datos.genes ; j++)
          {
                    fprintf(fp, "gene %d\n", j);
                    fprintf(fp, "min: %g\n",
datos.min);
                    fprintf(fp, "max: %g\n",
datos.max);
                    fprintf(fp, "values: %lu\n",
datos.valores);
                    fprintf(fp, "format:
%s\n",datos.formatoGen);
                    fprintf(fp, "\n");
          }
          fclose(fp);

          fp = fopen(infile, "w");

          fprintf(fp, "%21s = %d\n", "Exp.",
datos.experimentos);
          fprintf(fp, "%21s = %d\n", "Tot. Eval.",
datos.trials);
          fprintf(fp, "%21s = %d\n", "Pob.",
datos.poblacion);
          fprintf(fp, "%21s = %d\n", "Long.
cromosoma",datos.bitlength);
          fprintf(fp, "%21s = %g\n", "Prob.
cruza",datos.pcruza);
          fprintf(fp, "%21s = %g\n", "Prob.
mutaci¢n",datos.pmut);
          fprintf(fp, "%21s = %g\n", "Salto
generac.",datos.gap);
          fprintf(fp, "%21s = %d\n", "Ventana
scaling",datos.scalingWindow);
          fprintf(fp, "%21s = %s\n", "Interv.
reporte","1");
          fprintf(fp, "%21s = %s\n", "Estruc.
guardadas", "10");
          fprintf(fp, "%21s = %d\n", "Max gens sin
eval.", datos.maxGenSinEval);
          fprintf(fp, "%21s = %s\n", "Interv. entre
vuelcos", "0");
          fprintf(fp, "%21s = %s\n", "Vuelcos
guardados", "0");
          fprintf(fp, "%21s = %s\n", "Opciones",
datos.options);
          fprintf(fp, "%21s = %lu\n","Semilla
aleatoria", datos.randomSeed);
          fprintf(fp, "%21s = %g\n", "Ranking m¡nimo",
datos.rankMin);

          fclose(fp);
}
/*-------------------------------------------------------
------------------*/
void SetupData::derivarDatos() {
          // Se agregan a las opciones para correr
genesis las elegidas para
          // seleccion
          if (datos.checkBoxSeleccion & 1) {
                    strcat(datos.options,"e");
          // elitista
          }
          if (datos.checkBoxSeleccion & 2) {
                    strcat(datos.options,"R");      //
ranking
          }
}
/*-------------------------------------------------------
------------------*/
// obtenerParametros : lee parametros del input del
usuario
unsigned short SetupData::obtenerParametros() {
          unsigned short control = cmCancel;
          TDialog *pd = (TDialog *)rez-
>get(dialogName);
          if ( pd ) {

                    setDefault();
                    pd->setData( &datos );


                    control = TProgram::deskTop-
>execView( pd );

                    if( control != cmCancel ) {
                              pd->getData( &datos );
                    }
          }
          TProgram::deskTop->destroy( pd );
          return control;

}
/*-------------------------------------------------------
------------------*/
unsigned long SetupData::getExperimentos() {return
datos.experimentos;}

/** fin de archivo **/
```

C.1.26 Módulo STUPDAT1.CPP

```
/*
 * archivo stupdat1.cpp
 *
 * objetivo: Objeto para manejo de datos de setup en
la propuesta 1 de ag.
 */
#include <values.h>
#include <string.h>
#include "fields.h"
#include "utils.h"
#include "stupdat1.h"
/*-------------------------------------------------------
-----------*/
void SetupDataV1::derivarDatos() {

          SetupData::derivarDatos();
          // calculo la cantidad de bits que ocupa
cada gen.
          datos.bitlength = datos.genes *
ilog2(datos.valores);
          // agrego opciones elegidas para Genesis
          if (datos.checkBoxCodificacion) {
                    strcat(datos.options,"g");
          // Gray code
          }
}
/*-------------------------------------------------------
------------------*/
void SetupDataV1::setDefault() {
          SetupData::setDefault();
          datos.valores = 1024;         //
granularidad default
}
/*-------------------------------------------------------
------------------*/
SetupDataV1::SetupDataV1() {
  strcpy(dialogName,"v1Dialog");
}

/** fin de archivo **/
```

C.1.27 Módulo STUPDAT2.CPP

```
/*
 * archivo stupdat2.cpp
 *
 * objetivo: Objeto para manejo de datos de setup en
la propuesta 2 de ag.
 */
#include <values.h>
#include <string.h>
#include "fields.h"
#include "utils.h"
#include "stupdat2.h"
/*-------------------------------------------------------
-----------*/
void SetupDataV2::derivarDatos() {
          SetupData::derivarDatos();
```

```
          datos.min = 0;
          datos.max = datos.valores -1;
          datos.bitlength = datos.genes *
cantbits(datos.valores);
}
/*-------------------------------------------------
-----------*/
SetupDataV2::SetupDataV2() {
  strcpy(dialogName,"v2Dialog");
}
```

```
/*-------------------------------------------------
-----------*/
void SetupDataV2::setDefault() {
  SetupData::setDefault();
  strcpy(datos.formatoGen,"%2.0f");
}

/** fin de archivo **/
```

C.1.28 Módulo STUPDAT3.CPP

```
/*
 * archivo stupdat2.cpp
 *
 * objetivo: Objeto para manejo de datos de setup en
la propuesta 3 de ag.
 */
#define Uses_TDeskTop
#define Uses_TDialog
#define Uses_TProgram
#define Uses_TResourceFile
#include <tv.h>

#include <values.h>
#include "fields.h"
#include "utils.h"
#include "stupdat3.h"
#include <dir.h>
#include <string.h>

extern TResourceFile *rez;
/*-------------------------------------------------
-----------*/
void SetupDataV3::setDefault() {

          datos.experimentos = 1;
          datos.trials = 1;
          datos.poblacion = 5;
          datos.pcruza = 0.6;
          datos.pmut =0.001;
          datos.gap = 1.0;
          datos.scalingWindow = 5;
          datos.maxGenSinEval = 2;
          strcpy(datos.options,"cDflLtboO");
          datos.randomSeed = 123456789;
          datos.rankMin = 0.75;
          datos.checkBoxSeleccion = 1;
          datos.errorPresion = 0.01;
          strcpy(datos.formatoGen,"%2.0f");
}
/*-------------------------------------------------
-----------*/
unsigned short SetupDataV3::obtenerParametros()
{
          ushort control = cmCancel;
          TDialog *pd = (TDialog *)rez-
>get("v3Dialog");
          if ( pd ) {
                    setDefault();
                    pd->setData( &datos );
                    control = TProgram::deskTop-
>execView( pd );
                    if( control != cmCancel ) {
                              pd->getData( &datos );
                    }
          }
          TProgram::deskTop->destroy( pd );
          return control;
}
/*-------------------------------------------------
-----------*/
void SetupDataV3::escribirParametros(char *ext) {
          char infile[MAXPATH];
          char templatefile[MAXPATH];
          FILE *fp;
          int  j;

          sprintf(infile, "in%s", ext);
          sprintf(templatefile, "template%s",ext);

          fp = fopen(templatefile, "w");
          fprintf(fp, "genes: %d\n\n", datos.genes);
          printf("\n");


          for (j=0; j < datos.genes ; j++)
          {
                    fprintf(fp, "gene %d\n", j);
                    fprintf(fp, "min: %g\n",
datos.min);
                    fprintf(fp, "max: %g\n",
datos.max);
                    fprintf(fp, "values: %lu\n",
datos.valores);
                    fprintf(fp, "format:
%s\n",datos.formatoGen);
                    fprintf(fp, "\n");
          }
          fclose(fp);

          fp = fopen(infile, "w");
```

```
          fprintf(fp, "%21s = %d\n", "Exp.",
datos.experimentos);
          fprintf(fp, "%21s = %d\n", "Tot. Eval.",
datos.trials);
          fprintf(fp, "%21s = %d\n", "Pob.",
datos.poblacion);
          fprintf(fp, "%21s = %d\n", "Long.
cromosoma",datos.bitlength);
          fprintf(fp, "%21s = %g\n", "Prob.
cruza",datos.pcruza);
          fprintf(fp, "%21s = %g\n", "Prob.
mutaci¢n",datos.pmut);
          fprintf(fp, "%21s = %g\n", "Salto
generac.",datos.gap);
          fprintf(fp, "%21s = %d\n", "Ventana
scaling",datos.scalingWindow);
          fprintf(fp, "%21s = %s\n", "Interv.
reporte","1");
          fprintf(fp, "%21s = %s\n", "Estruc.
guardadas", "10");
          fprintf(fp, "%21s = %d\n", "Max gens sin
eval.", datos.maxGenSinEval);
          fprintf(fp, "%21s = %s\n", "Interv. entre
vuelcos", "0");
          fprintf(fp, "%21s = %s\n", "Vuelcos
guardados", "0");
          fprintf(fp, "%21s = %s\n", "Opciones",
datos.options);
          fprintf(fp, "%21s = %lu\n","Semilla
aleatoria", datos.randomSeed);
          fprintf(fp, "%21s = %g\n", "Ranking m¡nimo",
datos.rankMin);
          fprintf(fp, "%21s = %g\n", "Epsilon",
datos.errorPresion);

          fclose(fp);
}
/*-------------------------------------------------
-----------*/
void SetupDataV3::derivarDatos() {
          datos.min = 0;
          datos.max = datos.valores -1;
          datos.bitlength = datos.genes *
cantbits(datos.valores);
          if (datos.checkBoxSeleccion & 1) {
                    strcat(datos.options,"e");
          }
          if (datos.checkBoxSeleccion & 2) {
                    strcat(datos.options,"R");
          }
}
/*-------------------------------------------------
-----------*/
void SetupDataV3::leerPresetup(char *file) {
          char line[200];
          FILE *fpre;

          fpre = fopen(file, "r");

          readFile(fpre, line);
          sscanf(line,"%d", &datos.genes);

          // leer el rango de valores posibles para
los genes
          readFile(fpre, line);
          sscanf(line,"%lf", &datos.min);
          readFile(fpre, line);
          sscanf(line,"%lf", &datos.max);

          // en la  version 2 y 3 leo valores del
presetup
          readFile(fpre, line);
          sscanf(line,"%lu", &datos.valores);
          fclose(fpre);
}
/*-------------------------------------------------
-----------*/
SetupDataV3::SetupDataV3() {
  strcpy(dialogName,"v3Dialog");
}

/*-------------------------------------------------
-----------*/
unsigned long SetupDataV3::getExperimentos()
{
 return datos.experimentos;
}

/** fin de archivo **/
```

C.1.29 Módulo TVUTILS.CPP

```
/*
 * archivo: TVUTILS.CPP
 *
 * objetivo: Rutinas que muestran o reciben datos de
pantalla.
 */
#define Uses_TDeskTop
#define Uses_TProgram
#define Uses_MsgBox
#define Uses_TFileDialog
#include <tv.h>
#include <dir.h>
#include <string.h>
#include "tvutils.h"
#include "utils.h"

__link (RParamText) // para TCartel
/*-------------------------------------------------
---*/
/* constructor del objeto TCartel */
TCartel::TCartel(const TRect& bounds, const char
*aTitle, const char *aText, int aParamCount) :

          TDialog( bounds, aTitle ),

          TWindowInit( &TCartel::initFrame )
{
          flags &= ~wfClose;
          options|= ofCentered;
          insert(new TParamText(TRect(3, 2, size.x -
2, size.y - 3), aText, aParamCount) );
}

/*-------------------------------------------------
*/
/* makeEspere: crea una ventana de dialogo que muestra
un cartel mientras
se simula el experimento */
TDialog *makeEspere() {
            TCartel *cartel = new TCartel(TRect(0, 0,
50, 6),

"Informaci¢n", "Simulando experimento...");
           return (cartel);
}
/*-------------------------------------------------
*/
/* execDialog: Ejecuta una ventana de dialogo y
retorna la condicion de
salida (cmOk o cmCancel) */
ushort execDialog( TDialog *d, void *data )
{
          TView *p = TProgram::application->validView(
d );
          if( p == 0 )
                    return cmCancel;
          else
                    {
                    if( data != 0 )
                              p->setData( data );
                    ushort result = TProgram::deskTop-
>execView( p );
                    if( result != cmCancel && data !=
0 )
                              p->getData( data );
                    TObject::destroy( p );
                    return result;
                    }
}
/*-------------------------------------------------
------------------*/
/* elegioExtensionArchivo:
 recibe una nombre de archivo con wildcards,
 permite que el usuario elija un archivo validando que
el mismo responda
 a la m scara indicada y se encuentre en el directorio
activo
 y devuelve la extensi¢n del archivo elegido.
 Retorna true si  eligio y falso si cancelo.
*/
Boolean eligioExtensionArchivo(const char* nombre,
char* ext) {
char fileName[MAXPATH];
char drive[MAXDRIVE];
char dir[MAXDIR];
char file[MAXFILE];
char directorio[MAXDIR];
char unidad[MAXDRIVE];

Boolean ok;
char errMsg[80];


   do {
            ok = True;
            sprintf( fileName, "%s.*",nombre);
            if( execDialog( new TFileDialog( fileName,
"Abrir archivo",
                                "~N~ombre",
fdOpenButton, 100 ), fileName) == cmCancel ) {
                                return (False);
            }
            else {

          fnsplit(fileName,drive,dir,file,ext);
                    current_drive(unidad);
                    current_dir(directorio);
                    if (strcmpi(drive,unidad) == 0 &&
strcmpi(dir,directorio) == 0) {
                      if (strcmpi(file,nombre) == 0) {
                                if
(!fileExists(fileName)) {
                                  messageBox("El archivo
no existe.",mfInformation);
                                  ok = False;
                                }
                      }
                      else {
                                  sprintf(errMsg,"Debe
elegir un archivo '%s.*'.",nombre);

          messageBox(errMsg,mfInformation);
                                ok = False;
                      }
                    }
                    else {
                      messageBox("Debe elegir un
archivo en este directorio",mfInformation);
                      ok = False;
                    }
            }
          } while  (!ok);
          return (True);
}
/** fin de archivo **/
```

C.1.30 Módulo UTILS.CPP

```
/*
 * archivo utils.cpp
 *
 * objetivo: Rutinas de acceso a archivos y rutinas de
c lculo utilizadas
 *                                en el programa
 */
#include <dir.h>
#include <stdio.h>
#include <string.h>
#include "utils.h"
/*-------------------------------------------------
------------------*/
/* readFile: lee una linea de un archivo ignorando los
comentarios */
char *readFile(FILE *fp, char *line)
{
          char *c;

          do {
                    c= fgets(line, 100, fp);
          } while ( c!= NULL && (line[0]== '#' ||
line[0] == 10));
          return c;
}
/*-------------------------------------------------
------------------*/
/* ilog2: calcula el logaritmo en base 2 de un n£mero
potencia de 2 */
int ilog2(unsigned long n)
{
          int i;

          i = 0;
          while ((int) (n & 1) == 0)
          {
                    n >>= 1;
                    i++;
          }
          return(i);
}
/*-------------------------------------------------
------------------*/
/* cantbits: cantidad de bits necesarios para
almacenar n valores */
int cantbits(unsigned long n)
{
          unsigned long mask;
          int otro1, bits, i, totbits;

          otro1=0;
          n = n - 1;  /* tengo que representar de 0 a
n-1 */
          totbits = sizeof(long) * 8;

          /* busca el primer 1 contando de la
izquierda */
          mask = (long)1 << (totbits - 1);
          for (i=0; !(n & mask); i++, mask >>=1);
          bits = totbits -i;

          /* se fija si hay otros unos */
```

```
        for (i++, mask>>=1; i < totbits; i++, mask
>>=1) {
                if (n & mask) otro1 = 1;
        }

        if (otro1) bits++;
        return(bits);

}


/*-----------------------------------------------------
------------------*/
/* fileExists: indica si el archivo dado existe */
int fileExists(char *fname)
{
        struct ffblk ffblk;

        return ( findfirst(fname, &ffblk,0) == 0 );
}
/*-----------------------------------------------------
------------------*/
```

```
/* current_drive: devuelve el drive activo */
char *current_drive(char *path)
{
   strcpy(path, "X:");      /* fill string with form
of response: X:\ */
   path[0] = 'A' + getdisk();    /* replace X with
current drive letter */
   return(path);
}
/*-----------------------------------------------------
------------------*/
/* current_dir: devuelve el directorio actual */
char *current_dir(char *path)
{
        strcpy(path,"\\");
        getcurdir(0, path+1);
        strcat(path,"\\");
        return(path);
}

/** fin de archivo **/
```

C.1.31 Módulo D_INPUT.C

```
/*
 * archivo d_input.c
 *
 * objetivo: Lectura de los datos del modelo para el
problema directo.
 *                          Armado de la malla para
elementos finitos.
 *                          Escritura de resultados
del problema directo.
 */
#include <stdio.h>
#include <alloc.h>
#include <assert.h>
#include <math.h>
#include <values.h>
#include <process.h>
#include "d_extern.h"         /* variables globales */
#include "d_presio.h"
#include "d_ruido.h"
#define epsilon             1e-14
#define PI          3.14159265358979323846
#define calcVel(bulk, shear, ro)          sqrt((bulk +
4.0 * shear / 3.0) / ro)


        double wnperc; /* porcentaje de ruido blanco
*/
        double frecFuente; /* frecuencia de la
fuente en kHz */
        double tiempoTotal;  /* tiempo total de la
simulaci¢n */
        int ptosPorLongOnda; /* cantidad de puntos
por longitud de onda */
        int tipoFuente;  /* tipo de la fuente.
Valores posibles: 1 o 2 */

        int cantMateriales;
        double *roMaterial;  /* se lee en gr/cm**3
=> pasa a gr/m**3 */
        double *bulkMaterial; /* en dinas/cm**2 =>
pasa a g/(msec**2 * m) */
        double *shearMaterial; /* en dinas/cm**2 =>
pasa a g/(msec**2 * m) */
        double *velMaterial;   /* en m/msec */

        int    *materialCapa;  /* material verdadero
de cada capa */
        double *velocidad;  /* verdadera velocidad
de cada capa */

        double velMax, velMin; /* Maxima y minima
velocidad */

        /* Datos calculados */

        double longOnda;
        double longIntervX; /* longitud de un
intervalo en x. */
        int *cantIntervalos; /* Cantidad de
intervalos por capa. Comienza en 1 */
        double *longIntervalos; /* longitud de los
intervalos para cada capa */

        int cantCapas;
        double *interfaces; /* posicion de las
interfaces entre capas en metros*/


        int silenciar;  /* indica si hay o no que
silenciar la fuente */

        double tmute;
/* tiempo que se enmudece la fuente */
        double f4;    /* filtro pasa bajos */
        double fny;                   /* frecuencia
de Nyquist */
        double delf;
        int npf;            /* numero de puntos para
Fourier */
```

```
        int lpulse;                                /*
"       "                   "           "       "
*/

        double maxFuente; /* maximo valor de source
*/
/*-----------------------------------------------------
------------------*/
/* readFile: lee una linea de un archivo ignorando los
comentarios */
char *readFile(FILE *fp, char *line)
{
        char *c;

        do {
                c= fgets(line, 100, fp);
        } while ( c!= NULL && (line[0]== '#' ||
line[0] == 10));
        return c;
}
/*-----------------------------------------------------
------------------*/
/* leerModelo: lee el archivo modelo elegido */
void leerModelo(char *extension)
{
/* lee las variables */
        FILE *fp;
        char file[13], line[200];
        int i;
        float auxin;
        int nroMaterial;

        sprintf(file, "modelo.%s", extension);
        if ((fp = fopen(file, "r")) == NULL)
        {
                printf("Entrada: no pude abrir
%s", file);
                exit(1);
        }
        /* cantidad de capas */
        readFile(fp, line);
        sscanf(line,"%d", &cantCapas);

        /* posicion de las capas */
        interfaces = malloc((cantCapas+1) *
sizeof(double) );
        /* x capas, x+1 interfaces. Leo las
posiciones de las interfaces */
        for (i=0; i <= cantCapas; i++) {
                readFile(fp, line);
                sscanf(line,"%f", &auxin);
                interfaces[i] = auxin;
        }

        /* cantidad de materiales */
        readFile(fp, line);
        sscanf(line,"%d", &cantMateriales);

        /* ro de los materiales */
        roMaterial = malloc(cantMateriales *
sizeof(double));
        for (i=0; i < cantMateriales; i++) {
                readFile(fp, line);
                sscanf(line,"%f", &auxin);
                roMaterial[i]= auxin * 1e6;
        }

        /* bulk de los materiales */
        bulkMaterial = malloc(cantMateriales *
sizeof(double));
        for (i=0; i < cantMateriales; i++) {
                readFile(fp, line);
                sscanf(line,"%f", &auxin);
                bulkMaterial[i]= auxin  / 1e4; /*
cambio unidades */
        }

        /* shear modulus de los materiales */
        shearMaterial = malloc(cantMateriales *
sizeof(double));
        for (i=0; i < cantMateriales; i++) {
```

```
                    readFile(fp, line);
                    sscanf(line,"%f", &auxin);
                    shearMaterial[i] = auxin / 1e4; /*
cambio de unidades */
          }

          /* silenciar o no la fuente al comienzo */
          readFile(fp, line);
          sscanf(line,"%d", &silenciar);

          /* porcentaje de ruido blanco */
          readFile(fp, line);
          sscanf(line,"%lf", &wnperc);
          wnperc = wnperc * 0.01;

          /* tipo de fuente */
          readFile(fp, line);
          sscanf(line, "%d", &tipoFuente);

          /* frecuencia de la fuente en kHz */
          readFile(fp, line);
          sscanf(line, "%f", &auxin);
          frecFuente=auxin;

          /* tiempo que dura la simulacion en msec */
          readFile(fp, line);
          sscanf(line, "%f", &auxin);
          tiempoTotal = auxin;

          /* puntos por longitud de onda */
          readFile(fp, line);
          sscanf(line, "%d", &ptosPorLongOnda);

          /* Verdadera composicion de la tierra */
          materialCapa = malloc(cantCapas *
sizeof(double));
          for (i=0; i < cantCapas; i++) {
                    readFile(fp, line);
                    sscanf(line, "%d", &nroMaterial);
                    materialCapa[i]= nroMaterial;
          }

          /* Ubicacion de la fuente */
          readFile(fp, line);
          sscanf(line, "%f", &auxin);
          xsou = auxin;

          /* Ubicaci¢n del receptor*/
          readFile(fp, line);
          sscanf(line, "%f", &auxin);
          xrec= auxin;

          /* Condicion de bordes usada */
          readFile(fp, line);
          sscanf(line, "%d", &keyb);

          /* Funcion que determina la contribuci¢n de
la fuente en los distintos
             puntos del espacio: delta es la distancia
a la fuente */
          /* 0 -> delta ,1  -> d/dx(delta)  */
          readFile(fp, line);
          sscanf(line, "%d", &kdel);

          /* tiempo que enmudece la fuente */
          readFile(fp, line);
          sscanf(line, "%f", &auxin);
          tmute = auxin;

          /* parametro para filtro pasa bajos en kHz*/
          readFile(fp, line);
          sscanf(line, "%f", &auxin);
          f4=auxin;

          /* numero de puntos para Fourier */
          readFile(fp, line);
          sscanf(line, "%d", &npf);

          fclose(fp);
          assert(heapcheck()==_HEAPOK);
}

/*-------------------------------------------------------
-----------------*/
/* registrarPresiones: arma la malla para la
aplicaci¢n de elementos finitos
y llama a la rutina presiones para que calcule las
presiones */
void registrarPresiones() {


          double velAux;
          int i, j;
          double hmin;  /* minimo de los h */
          double if4;
          double psi,width,totpul,ts;  /* vbles
auxiliares para calcular source */
          double tt, alfa, w0;                    /*
"         "                    "            "       "
*/
          double aux;                  /*  "         "
                    "           "       "    */

          /* Maxima y minima velocidad POSIBLE de
acuerdo a los materiales posibles */

          velMax = calcVel(bulkMaterial[0],
shearMaterial[0], roMaterial[0]);
          velMin = velMax;
          for( i=1; i < cantMateriales; i++) {
                    velAux = calcVel(bulkMaterial[i],
shearMaterial[i], roMaterial[i]);
                    if (velAux > velMax) velMax =
velAux;
                    if (velAux < velMin) velMin =
velAux;
          }

          /* Dejamos un poco de margen */
          velMax *= 1.125;
          velMin *= 0.95;


          /* Calculamos la longitud de onda */
          assert(frecFuente);
          longOnda = velMin / frecFuente;

          /* Calculamos el tama¤o de un intervalo en X
*/
          assert(ptosPorLongOnda);
          longIntervX= longOnda / ptosPorLongOnda;

          /* Calculamos la cantidad de intervalos por
capa */
          cantIntervalos = malloc((cantCapas+1) *
sizeof(double) );
          longIntervalos = malloc((cantCapas+1) *
sizeof(double) );
          hmin = MAXDOUBLE;
          for(i=1; i <= cantCapas; i++) {
                    cantIntervalos[i] = (interfaces[i]
- interfaces[i-1]) / longIntervX + 1;
                    longIntervalos[i] = (interfaces[i]
- interfaces[i-1])/cantIntervalos[i];
                    if (longIntervalos[i] < hmin) hmin
= longIntervalos[i];
          }

          /* kx cantidad de intervalos sobre c/
interface */
          kx = malloc((cantCapas+1) * sizeof(int));
          kx[0]=0;
          for (i=1; i <=cantCapas; i++) {
                    kx[i] = kx[i-1] +
cantIntervalos[i];
          }

          /* cantidad total de intervalos */
          l = kx[cantCapas];

          /* calcular xnod. */

          xnod = malloc((l+2) * sizeof(double) );

          for (i=1; i<=cantCapas; i++) {
                    xnod[ kx[i-1] + 1] = interfaces[i-
1];
                    for (j=kx[i-1]+2; j<= kx[i]; j++)
{
                              xnod[ j ] = xnod[j-1] +
longIntervalos[i];
                    }
          }
          xnod[l+1] = interfaces[cantCapas];

          /* calculo h */
          h = malloc((l+1) * sizeof(double) );
          for (i=1; i<=l; i++) {
                    h[i] = xnod[i+1] - xnod[i];
          }

          /* calculo ronodo */
          ronodo = malloc((l+2) * sizeof(double));
          for (i=0; i <cantCapas; i++)
                    for (j=kx[i]+1; j <= kx[i+1]; j++)
                              ronodo[j] =
roMaterial[materialCapa[i]];
          ronodo[l+1]=
roMaterial[materialCapa[cantCapas-1]];


          /* calculo intervalito de tiempo */
          dt = 1.0 * hmin / velMax;

          /* calculo cantidad de intervalos de tiempo
*/
          nt = tiempoTotal / dt;

          /* calculo de source */

          /* vbles auxiliares usadas en caso 1 y 2 */
          psi = 8 * pow( frecFuente, 2);
          width = 0.5 * frecFuente;
          ts = 2.5 * width;
          totpul = 6 * ts;
          lpulse = totpul/dt + 1.0001;

          switch(tipoFuente) {
                    case 1:
                              w0 = 2 * PI *
frecFuente;
                              alfa = 0.79 * w0 /
PI;
                              assert(nt>lpulse);
/* error. ? */
                              source =
malloc((nt+1) * sizeof(double));
```

```
                                        for (i=1; i <= nt;
i++) {
                                                tt = (i-1) *
dt;
                                                source[i] =
4 * alfa * tt * exp( -1 * alfa * tt) * sin(w0 * tt);
                                        }

                        case 2:
                                        if (nt < lpulse) nt =
lpulse;  /*  nt = max(nt,lpulse) */
                                        source =
malloc((nt+1) * sizeof(double));
                                        for (i=1; i<=nt; i++)
{
                                                tt = (i-1) *
dt;
                                                aux = psi *
pow(tt - ts, 2);
                                                source[i] =
-2 * psi * (tt - ts) * exp(-1 * aux);
                                        }
                                        break;
                }

                /* calculo el maximo valor de la fuente para
normalizar la misma */
                maxFuente = fabs(source[1]);
                for (i=2; i <= nt; i++)
                   if (fabs(source[i]) > maxFuente)
maxFuente= fabs(source[i]);

                /* normalizo la fuente */

                for (i=1; i <=nt; i++) {
                   source[i] = source[i] / maxFuente;
                   /* si fuente < 10 ** -14 le pone 0. */
                   if ( fabs(source[i]) < 1e-14) source[i]=
0;
                }

                /* calcular fny (nyquist frequency) */
                fny= 0.5 * dt;
                if ( f4 > fny || f4 == 0 ) f4= fny;
                delf= f4 / npf;
                if4= f4 / delf + 1;
                if ( if4 == 1 ) {
                        f4= fny;
                        if4= f4 / delf + 1;
                }

                /* Calcular jsou y jrec */
                for ( i= 1; i <= l+1; i++) {
                        if ( xsou > xnod[i] ) jsou= i;
                        if ( xrec > xnod[i] ) jrec= i;
                }
                /* No se puede tener una condici¢n de bordes
Dirichlet si la fuente esta
                en la superficie */
                assert( jsou != 1 || keyb != 0 );

                /* Chequea que la posici¢n de la fuente xsou
no coincida con la posicion
                   de un nodo  k*h */
                assert( (xsou - xnod[jsou]) >= epsilon &&
(xnod[jsou+1] - xsou) >= epsilon);

                /* Calcular cantidad de pasos que no escucho
la fuente */
                ntmute= tmute / dt;

                /* Calcular is
                   2.*is=number of points per shortest
wavelength.
                   it is used when the source is two deltas
with opposite
                   signs separated for the shortest
wavelength */
                if ( f4 < 1e-13 )
                        is= 1;
                else
                        is= velMin / f4 / hmin / 2 + 1;

                if ( kdel == 2 ) {
                        assert( jsou + is <= l+1 );
                        assert( jsou - is >= 1 );
                }

                /* Calcular velocidad de cada material */
                velMaterial=
malloc(cantMateriales*sizeof(double));
                for ( i=0; i < cantMateriales; i++)
                        velMaterial[i]=
calcVel(bulkMaterial[i],shearMaterial[i],roMaterial[i]
);

                /* Calcular las velocidades reales */
                velocidad= malloc(cantCapas*sizeof(double));
                for ( i=0; i < cantCapas; i++)
                        velocidad[i]=
velMaterial[materialCapa[i]];

                /* Ahora hay que calcular las presiones
OBSERVADAS */
                recmed= malloc((nt+1)*sizeof(double));
                presiones(velocidad, cantCapas, recmed, nt);
                if (wnperc)
                        agregarRuido(recmed,nt,wnperc);
```

```
                assert(heapcheck()==_HEAPOK);
}

/*-----------------------------------------------------
----------------*/
/* escribirResultados:  Escritura de resultados del
problema directo.
   Se escriben los archivos presetup, campo y solucion
*/
void escribirResultados(char *extension) {

FILE *fp;
char file[13];
int i;

                /* Archivo: presetup.* */
                sprintf(file, "presetup.%s", extension);
                if ((fp = fopen(file, "w")) == NULL)
                {
                        printf("escribirResultados: no
pude abrir %s", file);
                }
                /* cantidad de capas */
                fprintf(fp,"#cantidad de capas\n");
                fprintf(fp, "%d\n\n", cantCapas);
                /* velocidad minima */
                fprintf(fp,"#velocidad m¡nima (m/ms)\n");
                fprintf(fp, "%f\n\n", velMin);
                /* velocidad maxima */
                fprintf(fp,"#velocidad m xima (m/ms)\n");
                fprintf(fp, "%f\n\n", velMax);
                /* cantidad de materiales (es utilizado por
versiones 2 y 3)*/
                fprintf(fp,"#cantidad de materiales\n");
                fprintf(fp, "%d\n\n", cantMateriales);
                fclose( fp );

                /* Archivo: campo.* */
                sprintf(file, "campo.%s", extension);
                if ((fp = fopen(file, "w")) == NULL)
                {
                        printf("escribirResultados: no
pude abrir %s", file);
                }
                fprintf(fp,"#cantidad de intervalos de
tiempo\n");
                fprintf(fp, "%d\n\n", nt);
                fprintf(fp,"#cantidad de intervalos de
tiempo en que enmudece la fuente\n");
                fprintf(fp, "%d\n\n", ntmute);
                fprintf(fp,"#cantidad de intervalos en
x\n");
                fprintf(fp, "%d\n\n", l);
                fprintf(fp,"#duraci¢n de un intervalo de
tiempo (ms)\n");
                fprintf(fp, "%f\n\n", dt);

                fprintf(fp,"#longitud de intervalo para cada
capa\n");
                for (i= 1; i<= cantCapas; i++)
                        fprintf(fp, "%f\n", h[kx[i]]);
/* long. del interv. de c/capa */
                fprintf(fp,"\n");


                /* Para la versi¢n 1 hay que especificar el
ro de cada capa */
                fprintf(fp,"#Propuesta I: ro de cada
capa\n");
                for (i= 1; i<= cantCapas; i++)
                        fprintf(fp, "%f\n",
roMaterial[materialCapa[i-1]]);
                fprintf(fp,"\n");

                fprintf(fp,"#posici¢n de la fuente (m)\n");
                fprintf(fp, "%f\n\n", xsou);

                fprintf(fp,"#nodo anterior a la fuente\n");
                fprintf(fp, "%d\n\n", jsou);

                fprintf(fp,"#posici¢n del nodo anterior a la
fuente (m)\n");
                fprintf(fp, "%f\n\n", xnod[jsou]);

                fprintf(fp,"#posici¢n del nodo posterior a
la fuente (m)\n");
                fprintf(fp, "%f\n\n", xnod[jsou+1]);

                fprintf(fp,"#posici¢n del receptor (m)\n");
                fprintf(fp, "%f\n\n", xrec);

                fprintf(fp,"#nodo anterior al receptor\n");
                fprintf(fp, "%d\n\n", jrec);
                fprintf(fp,"#posici¢n del nodo anterior al
receptor (m)\n");
                fprintf(fp, "%f\n\n", xnod[jrec]);
                fprintf(fp,"#posici¢n del nodo posterior al
receptor (m)\n");
                fprintf(fp, "%f\n\n", xnod[jrec+1]);

                fprintf(fp,"#intervalos acumulados en cada
capa\n");
                for (i= 0; i<= cantCapas; i++)
                        fprintf(fp, "%d\n", kx[i]);
                fprintf(fp, "\n");

                fprintf(fp,"#condicion de bordes\n");
                fprintf(fp,"#0: free surface\n");
                fprintf(fp,"#1: absorbing b.c.\n");
```

```
          fprintf(fp,"#2: neuman\n");
          fprintf(fp, "%d\n\n", keyb);

          fprintf(fp,"#tipo de contribucion de la
fuente\n");
          fprintf(fp,"#0: delta\n");
          fprintf(fp,"#1: d/dx(delta)\n");
          fprintf(fp, "%d\n\n", kdel);

          fprintf(fp,"#funci¢n fuente\n");
          for (i= 1; i<= nt; i++)
                    fprintf(fp, "%f\n", source[i]);
          fprintf(fp, "\n");

          fprintf(fp,"#is\n");
          fprintf(fp, "%f\n\n", is);

          fprintf(fp,"#presiones observadas
(g/m**2)\n");
          for (i=1; i <= nt; i++)
                    fprintf(fp, "%f\n", recmed[i]);
          fprintf(fp, "\n");
          /* Escribo en el archivo campo la cantidad
de materiales y la
          velocidad y el ro de cada material (son
utilizados por las versiones 2 y 3)*/
          fprintf(fp, "#Datos para las versiones 2 y
3\n");
          fprintf(fp,"#cantidad de materiales\n");
          fprintf(fp, "%d\n\n", cantMateriales);

          fprintf(fp,"#velocidad de cada material
m/ms\n");
          for ( i=0; i < cantMateriales; i++)
                    fprintf(fp, "%f\n",
velMaterial[i]);
          fprintf(fp, "\n");
          fprintf(fp,"#densidad de cada material
g/m**3\n");
          for ( i=0; i < cantMateriales; i++)
                    fprintf(fp, "%f\n",
roMaterial[i]);

          fclose(fp);

          /* Archivo: solucion.* */
          sprintf(file, "solucion.%s", extension);
          if ((fp = fopen(file, "w")) == NULL)
          {
                    printf("escribirResultados: no
pude abrir %s", file);
          }

          fprintf(fp,"#cantidad de capas\n");
          fprintf(fp, "%d\n\n", cantCapas);

          fprintf(fp,"#cantidad de materiales\n");
          fprintf(fp, "%d\n\n", cantMateriales);

          /* material de c/capa */
          fprintf(fp,"#material de cada capa\n");
          for ( i=0; i < cantCapas; i++)
                    fprintf(fp, "%d\n",
materialCapa[i]);
          fprintf(fp,"\n");

          /* velocidad de c/material */
          fprintf(fp,"#velocidad de cada material
m/ms\n");
          for ( i=0; i < cantMateriales; i++)
                    fprintf(fp, "%f\n",
velMaterial[i]);
          fprintf(fp,"\n");

          /* velocidad real de c/capa */
          fprintf(fp,"#velocidad de real de cada capa
m/ms\n");
          for ( i=0; i < cantCapas; i++)
                    fprintf(fp, "%8.5f\n",
velocidad[i]);
          fprintf(fp,"\n");

          fclose(fp);


}

/*-------------------------------------------------
---------------*/
/* liberarMemoria: libera la memoria asignada a las
variables utilizadas
en el problema directo
*/
void liberarMemoria() {

          free(interfaces);
          free(materialCapa);
          free(velMaterial);
          free(roMaterial);
          free(bulkMaterial);
          free(shearMaterial);
          free(cantIntervalos);
          free(longIntervalos);
          free(velocidad);
}

/** fin de archivo **/
```

C.1.32 Módulo D_MAIN.C

```
/*
 *  archivo d_main.c
 *
 *  objetivo: Resoluci¢n del problema directo.
 */
#include "d_global.h"
#include "d_input.h"
#include <stdio.h>
```

```
void directo(char *ext) {
          leerModelo(ext);
          registrarPresiones();
          escribirResultados(ext);
          liberarMemoria();
}

/** fin de archivo **/
```

C.1.33 Módulo D_PRESIO.C

```
/*
 *  archivo d_presio.c
 *
 *  objetivo: C lculo de presiones (problema directo)
 */
#include "d_extern.h"
#include <alloc.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include "d_presio.h"
/*-------------------------------------------------
---------------*/
/* una_presion : La funci¢n devuelve la presi¢n en el
receptor y calcula las
presiones en todos los nodos para ser utilizadas en
pasos posteriores
Par metros de entrada:
   unm1: presiones en los nodos en el tiempo n-1,
   un: presiones en los nodos en el tiempo n
   paso = numero de instante del tiempo en el que se
esta trabajando
   constk: constantes calculadas
   velnodo: velocidades en los nodos
Par metros de salida
   unp1: presiones en los nodos en el paso n+1
*/
double una_presion(unm1, un, unp1, paso, constk,
velnodo)
          double *unm1, *un, *unp1, *constk, *velnodo;
          int paso;
{
          double aux1, aux2;
          int k;
```

```
          assert(heapcheck()==_HEAPOK);

          /* condicion de bordes */
          unp1[1]= borde_sup(unm1[1], un[1], un[2],
constk[1]);
          unp1[l+1]= borde_inf( unm1[l+1], un[l],
un[l+1], constk[l+1]);

          /* agregar la contribucion de la fuente a
cada borde */
          /* primero al superior */
          unp1[1]+= fuente_en_borde(paso, 1);
          unp1[1]/= 1 + constk[1];
          /* y ahora al inferior */
          unp1[l+1]+= fuente_en_borde(paso, l+1);
          unp1[l+1]/= 1 + constk[l+1];

          /* calculo de la presion de los nodos
interiores */
          for (k=2; k <= l; k++) {
                    unp1[k]= 2 * un[k] - unm1[k] + 2 *
constk[k] * pow(dt, 2) *
                                        ( (un[k+1] -
un[k])     / ( h[k] * ronodo[k] ) -
                                        ( un[k] -
un[k-1] ) / ( h[k-1] * ronodo[k-1] ));

                    /* agregar la contribucion de la
fuente */
                    unp1[k]+= fuente_interior( paso,
k, constk[k], velnodo);
          }
```

```
          /* Interpolacion lineal de las presiones en
los dos nodos que rodean al
                    receptor */
          aux1= (xnod[jrec + 1] - xrec) / h[jrec] *
unp1[jrec];
          aux2= (xrec - xnod[jrec]) / h[jrec] *
unp1[jrec+1];

          assert(heapcheck()==_HEAPOK);
          return (aux1 + aux2);
}

/*-------------------------------------------------
----------------*/
/* presiones: Se calculan las presiones a partir de
las velocidades */
void presiones(velocidad, capas, presion, nt)
          /* velocidades por cada capa, parametro de
entrada */
          double *velocidad;
          int capas;  /* cantidad de capas */

          /* presiones calculadas a partir de las
velocidades */
          double *presion;
          int nt;             /* cantidad de
intervalos en el tiempo */
{
          int i;              /* auxiliar */
          double *aux;  /* puntero auxiliar para hacer
swap */
          double *unm1; /* presiones en el paso n-1 */
          double *un; /* presiones en el paso n   */
          double *unp1; /* presiones en el paso n+1 */
          double *velnodo;  /* velocidades en nodos */
          double *constk; /* constantes auxiliares */

          unm1= malloc((l+2)*sizeof(double)); /*
presiones en el paso n-1 */
          un= malloc((l+2)*sizeof(double));   /*
presiones en el paso n   */
          unp1= malloc((l+2)*sizeof(double)); /*
presiones en el paso n+1 */
          velnodo= malloc((l+2)*sizeof(double));  /*
velocidades en nodos */
          constk= malloc((l+2)*sizeof(double)); /*
constantes auxiliares */
          vel_en_nodos(velocidad, capas, velnodo);

          assert(heapcheck()==_HEAPOK);

          for ( i=1; i <= (l+1); i++) {
                    unm1[i]= 0;
                    un[i]= 0;
          }

          /* Condiciones iniciales */
          presion[1]= 0;
          presion[2]= 0;

          /* Tiempo de espera desde que comienza la
explosion hasta que se comienza
             a escuchar (mute) */
          for ( i= 3; i <= ntmute; i++)
                    presion[i]= 0;

          /* constantes auxiliares */
          constk[1]= velnodo[1] * dt / h[1];
          constk[l+1]= velnodo[l] * dt / h[l];
          for ( i = 2; i <= l; i++)
                    constk[i]= ( pow(velnodo[i], 2) *
pow(velnodo[i-1], 2) * ronodo[i] * ronodo[i-1]) /

          (h[i-1] * pow(velnodo[i], 2) * ronodo[i] +
h[i] * pow(velnodo[i-1], 2) * ronodo[i-1]);

          /* calcula la presion en el paso i */
          for ( i= max(ntmute+1, 3); i <= nt; i++) {
                    presion[i]= una_presion(unm1, un,
unp1, i, constk, velnodo);

                    aux= unm1;   /* lo guardo para
despues */
                    unm1= un;    /* avanza un paso en
el tiempo */
                    un= unp1;    /*    "    "   "   "
"    "   */
                    unp1= aux;   /* no interesan los
valores pero si el espacio de memoria */
          }

          free(unm1);
          free(un);
          free(unp1);
          free(velnodo);
          free(constk);
          /* el resultado va en 'presion' */
          assert(heapcheck()==_HEAPOK);
}
/*-------------------------------------------------
------------------*/
/* vel_en_nodos: Calcula la velocidad en cada nodo de
la malla */
void vel_en_nodos(velocidad, capas, velnodo)
double *velocidad;
int capas;
double *velnodo;
{
          int c, nodo;
          for (c=0; c <capas; c++)
                    for (nodo=kx[c]+1; nodo <=
kx[c+1]; nodo++)
                              velnodo[nodo] =
velocidad[c];
          velnodo[l+1]= velocidad[capas-1];
}
/*-------------------------------------------------
------------------*/
/*borde_sup:  Calcula la presion en el borde superior
en el tiempo n+1
(sin contar la influencia directa de la fuente) */
double borde_sup(unm1, unx1, unx2, cte)

double unm1; /* presion en el nodo 1 en el tiempo n-1
*/
double unx1; /* presion en el nodo 1 en el tiempo n */
double unx2; /* presion en el nodo 2 en el tiempo n */
double cte;  /* constante auxiliar */
{
          double presion;
          switch (keyb) {
                    case 0: /* dirichlet b.c. (free
surface) */
                                        presion = 0;
                                        break;
                    case 1: /* absorbing b.c. */
                                        presion = 2 *
unx1 - unm1 + 2 * pow(cte, 2) * (unx2 - unx1) +

          cte * unm1;
                                        break;
                    case 2: /* neumann b.c. */
                                        presion = 2 *
unx1 - unm1 + 2 * pow(cte, 2) * (unx2 - unx1);
                                        break;
          }
          return presion;
}
/*-------------------------------------------------
------------------*/
/*borde_inf:  Calcula la presion en el borde inferior
en el tiempo n=1
(sin contar la influencia directa de la fuente) */

double borde_inf(unm1, unx1, unx2, cte)

double unm1; /* presion en el paso anterior */
double unx1; /* presion en el anteultimo nodo */
double unx2; /* presion en el ultimo nodo */
double cte;  /* constante ad hoc */
{
  double presion;
  presion = 2 * unx2 - unm1 + 2 * pow(cte, 2) * (unx1
- unx2) +

          cte * unm1;
  return presion;
}
/*-------------------------------------------------
------------------*/
/* fuente_en_borde: Contribucion de la fuente en el
borde indicado por
nodo en el paso 'paso' */
double fuente_en_borde(paso, nodo)

int paso;
int nodo;
{
          double eval, presion;
          /* segun el tipo de fuente */
          switch (kdel) {
                    case 0: /* dirac delta at xsou
times source[n] */
                              if (1 == jsou)
                                        eval =
(xnod[jsou+1] - xsou) / h[jsou];
                              else if (1 == jsou+1)
                                                  eval
= (xsou - xnod[jsou]) / h[jsou];
                                         else
                                                  eval
= 0;
                              presion = 2 * pow(dt, 2)
* source[paso] * eval / h[nodo];
                              break;

                    case 1: /* derivative of dirac
delta at .5 * (xsou + xsou + h) times source[n] */
                              presion = 2 * pow(dt, 2)
* source[paso] / pow(h[nodo], 2);
                              if (1 == jsou) {
                                        /* no hacemos
nada */
                              }
                              else if (1 == jsou + 1)
{
                                        presion *= -
1;
                                        }
                                        else {

          presion = 0;
                                        }
                              break;

                    case 2: /* 2 dirac delta at xsou
times source[n] */
```

```
                                presion = 2 * pow(dt, 2)
* source[paso] * h[nodo] / 2.0;
                                if (1 == jsou - is) {
                                        /* no hacemos
nada */
                                }
                                else if (1 == jsou + is)
{
                                        presion *= -
1;
                                        }
                                        else {

          presion = 0;
                                        }
                                break;
          }
          return presion;
}

/*-------------------------------------------------
-------------------*/
/* fuente_interior: Contribucion de la fuente en el un
nodo interior */
double fuente_interior(paso, nodo, cte, velnodo)

int paso;
int nodo;
double cte; /* constante auxiliar */
double *velnodo;
{
          double eval, presion;
          switch (kdel) {
                    case 0: /* dirac delta at xsou
times source[n] */

                                if (nodo == jsou)
                                        eval =
(xnod[jsou+1] - xsou) / h[jsou];
                                else if (nodo == jsou+1)
                                                eval
= (xsou - xnod[jsou]) / h[jsou];
                                        else
                                                eval
= 0;

                                presion = 2 * pow(dt, 2)
* cte * source[paso] * eval /

(pow(velnodo[jsou], 2) * ronodo[jsou]);
                                break;

                    case 1: /* derivative of dirac
delta at .5 * (xsou + xsou + h) times source[n] */
```

```
                                if (nodo == jsou) {
                                        presion = 2 *
pow(dt, 2) * cte * source[paso] /

(pow(h[jsou], 2) * pow(velnodo[jsou], 2) *
ronodo[jsou]);
                                }
                                else if (nodo == jsou +
1) {
                                        presion = 2 *
pow(dt, 2) * cte * source[paso] /

(pow(h[jsou], 2) * pow(velnodo[jsou], 2) *
ronodo[jsou]);
                                        presion *= -
1;
                                        }
                                        else {

          presion = 0;
                                        }
                                break;

                    case 2: /* 2 dirac delta at xsou
times source[n] */

                                if (nodo == jsou - is) {
                                        presion = 2 *
pow(dt, 2) * cte * source[paso] *

(pow(velnodo[jsou], 2) * ronodo[jsou]) / 2.0;
                                }
                                else if (nodo == jsou +
is) {
                                        presion = 2 *
pow(dt, 2) * cte * source[paso] *

(pow(velnodo[jsou], 2) * ronodo[jsou]) / 2.0;
                                        presion *= -
1;
                                        }
                                        else {

          presion = 0;
                                        }
                                break;
          }
          return presion;
}

/** fin de archivo **/
```

C.1.34 Módulo D_RUIDO.C

```
/*
 * archivo d_ruido.c
 *
 * objetivo: Simulaci¢n de ruido en las presiones
generadas.
 */
#include <stdlib.h>
#include <math.h>
#define frandom()       ( random(32767) / 32766.0 )
/*-------------------------------------------------
---------*/
/* gasdev: calcula un nro a partir de una distribucion
N(0,1) */
double gasdev()
{
          static char iset= 0;
          static double gset;
          double fac, r, v1, v2;

          if (iset == 0) {
             do {
                       v1= 2.0 * frandom() - 1.0;
                       v2= 2.0 * frandom() - 1.0;
                       r= v1 * v1 + v2 * v2;
             } while (r >= 1.0 || r == 0.0);
             fac= sqrt(-2.0 * log(r) / r);
             gset= v1 * fac;
             iset= 1;
             return v2 * fac;
          }
```

```
          else {
                    iset= 0;
                    return gset;
          }
}
/*-------------------------------------------------
------------------*/
/* agregarRuido: modifica las presiones alter ndolas
con un porcentaje de
ruido */
void agregarRuido(double *recmed,int nt,double wnperc)
{
          double maxTraza;
          int i;

          /* obtener el maximo valor absoluto de la
traza */
          maxTraza=0;
          for (i=1; i <= nt; i++)
                    if (fabs(recmed[i]) > maxTraza)
maxTraza = fabs(recmed[i]);

   /* agregar ruido a la traza */
   for (i=1; i<=nt; i++)
          recmed[i] += wnperc * gasdev() * recmed[i];

}

/** fin de archivo **/
```

C.1.35 Módulo REPORT.C

```
/*
 *  archivo:        report.c
 *
 *  objetivo:       generar un reporte que resuma la
media y la varianza de una
 *                  serie de mediciones de performance
de una corrida de AG.
 */
#include <stdio.h>
#include "global.h"
#include "format.h"

#define ROWS 200
#define COLUMNS 9
```

```
double average[ROWS][COLUMNS];
double variance[ROWS][COLUMNS];
double line[COLUMNS];


report(ext, version)
char *ext;
int version;
{
          register int row, col, i;
          double oldgens;               /* contador de
la generacion previa */
```

```
        double tmp;
        int cutoff;                     /* los datos
fueron truncados? ver abajo */
        char Outfile[15];   /* archivo de salida (en
bruto) producido por el AG */
        char Infile[15];    /* archivo de entrada de
AG */
        char Reportfile[15]; /* Archivo de salida
del reporte */
        char errmsg[40];    /* mensaje de error */
        int lines;                      /* cantidad de
lineas del reporte */
        int eof;            /* indica cuando se
termino Outfile */
        int expn;           /* cantidad de
experimentos */
        double epsilon;

        FILE *fp, *fopen();

        /* Con el fin de calcular las medias y
varianzas, la                */
        /* cantidad de l¡neas se toma como el m¡nimo
n£mero de                    */
        /* l¡neas producidas por un experimento. Si
esto implicara  */
        /* descartar datos, el flag cutoff se setea,
de modo que     */
        /* se imprima una advertencia.
*/

        cutoff = 0;

        for (row = 0; row < ROWS; row++)
                for (col = 0; col < COLUMNS;
col++)
                {
                        average[row][col] = 0.0;
                        variance[row][col] =
0.0;
                }

        /* setear los nombres de archivos */

        sprintf(Infile, "in.%s", ext);
        sprintf(Outfile, "out.%s", ext);
        sprintf(Reportfile, "report.%s",ext);


        /* leer los parametros de Infile */
        fp = fopen(Infile, "r");

        fscanf(fp, IN_FORMAT, IN_VARS);
        if (version == 3) fscanf(fp,"Epsilon =
%lf",&epsilon);
        fclose(fp);

        /* leer los datos del Outfile */
        fp = fopen(Outfile, "r");

        lines = 0;
        oldgens = -1.0;

        /* leer una l¡nea */
        fscanf(fp, LINE_FIN, LINE_VIN);

        eof = 0;
        for (expn = 0; (!eof); expn++)
        {
                row = 0;

                /* oldgens > line[0] indica que
comienzan */
                /* los datos de un nuevo
experimento      */
                while ( !(eof) && (oldgens <=
line[0]))
                {
                        /* si oldgens = line[0],
significa que esta l¡nea repite */
                        /* la anterior (cosa que
a veces sucede despues de un    */
                        /* reinicio. En este
caso se ignora la l¡nea                 */
                        if (oldgens < line[0])
                        {
                                /* acumular
los valores de los experimentos */
                                for (col = 0;
col < COLUMNS; col++)
                                {

        average[row][col] += line[col];

        variance[row][col] += line[col] * line[col];
                                }
                                row++;
                        }
                        oldgens = line[0];

                        /* leer una l¡nea */
                        eof = (fscanf(fp,
LINE_FIN, LINE_VIN) == EOF);
                }

                oldgens = -1.0;
                if (expn == 0) lines = row;
                else
                {
                        if (row < lines)
                        {
                                lines = row;
                                cutoff = 1;
                        }
                }
        }
        fclose(fp);

        /* calcular el promedio y la varianza */
        for (row = 0; row < ROWS; row++)
        {
                for (col = 0; col < COLUMNS;
col++)
                {
                        tmp = average[row][col]
* average[row][col];
                        tmp /= expn;
                        variance[row][col] -=
tmp;
                        if (expn > 1)

        variance[row][col] /= (expn-1);
                        average[row][col] /=
expn;
                }
        }

        fp =  fopen (Reportfile,"w");
        /* escribir la tabla */
        fprintf(fp, "%21s = %d\n", "Propuesta",
version);
        fprintf(fp,OUT_FORMAT, OUT_VARS);
        if (version == 3) fprintf(fp,"%21s =
%5.3f","Epsilon",epsilon);

        /* imprimir los promedios */
        fprintf(fp,"\nPROMEDIO\n");
        fprintf(fp,"Gens   Evals  Perd  ");
        fprintf(fp,"Conv  Sesgo    Online     ");
        fprintf(fp,"Offline     Mejor
Promedio\n");

        for (i = 0; i < lines; i++)
        {
                fprintf(fp,"%4.0f  %6.0f  %4.0f
",

        average[i][0],average[i][1],average[i][2]);
                fprintf(fp,"%4.0f  %5.3f  %9.3e
",

        average[i][3],average[i][4],average[i][5]);
                fprintf(fp,"%9.3e  %9.3e
%9.3e\n",

        average[i][6],average[i][7],average[i][8]);
        }

        /* imprimir las varianzas */
        if (expn > 1)
        {
                fprintf(fp,"\nVARIANZA\n");
                fprintf(fp,"Gens   Evals  Perd
");
                fprintf(fp,"Conv  Sesgo    Online
");
                fprintf(fp,"Offline     Mejor
Promedio\n");
                for (i=0; i<lines; i++)
                {
                        fprintf(fp,"%4u  %6u
%4u  ",
                        (unsigned int)
variance[i][0], (unsigned int) variance[i][1],
                        (unsigned int)
variance[i][2]);
                        fprintf(fp,"%4u  %5.3f
%9.3e  ",
                        (unsigned int)
variance[i][3],variance[i][4],variance[i][5]);
                        fprintf(fp,"%9.3e  %9.3e
%9.3e\n",

        variance[i][6],variance[i][7],variance[i][8]
);
                }
        }

        /* escribir warnings si los hubiera */

        if (cutoff)
        {
                fprintf(fp,"\nNOTA: Algunos
experimentos produjeron m s");
                fprintf(fp," datos que otros.\n");
                fprintf(fp,"       Los datos extra
no fueron reportados arriba.\n");
        }

        if (expn < Totalexperiments)
        {
                fprintf(fp,"\nADVERTENCIA: Hay muy
pocos datos para el n£mero dado de ");
                fprintf(fp,"experimentos.\n");
        }
        if (expn > Totalexperiments)
        {
```

```
                    fprintf(fp,"\nADVERTENCIA:
Demasiados datos para el n£mero dado de
experimentos.\n");
          }
```

```
          fclose(fp);
}

/*** fin de archivo ***/
```

C.2 Programa AG1

C.2.1 Módulo DEFINE.H

```
/*
 *  archivo:       define.h
 *
 *  objetivo:    definiciones globales
 */

#include <math.h>
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include "format.h"
#include <conio.h>

#define clrtoeol        clreol
#define refresh()       fflush(stdout)
#define printw          printf
#define scanw           scanf
#define getw            gets

/********************** CONSTANTES
***************************/

#define CHARSIZE 8

/* usados en generador aleatorio de n£meros */
#define MASK 2147483647
#define PRIME 65539
#define SCALE 0.4656612875e-9



/************************ TIPOS
*****************************/

/* tipo para almacenar cada individuo de la poblaci¢n
*/
typedef struct {
          char *Gene;
          double Perf;
          int Needs_evaluation;
} STRUCTURE;

/* tipo para almacenar las mejores estructuras */
typedef struct {
          char *Gene;
          double Perf;
          int Gen;
          int Trials;
} BESTSTRUCT;

/* registro para interpretar el formato empaquetado de
acuerdo con el template definido */
typedef struct {
          double min;
          double max;
          unsigned long values;
          char format[16];
          double incr;
          int bitlength;
} GENESTRUCT;

/************************ MACROS
*****************************/

/* Comparaci¢n de dos valores de performance */
#define BETTER(X,Y)      (Maxflag ? (X) > (Y) : (X) <
(Y))

/* Un alelo converge si todos los individuos - FEW
/* tiene el mismo valor en esa posici¢n.  */
#define FEW              (Popsize/20)

/*************************************************
***********/
/*       Rand obtiene un n£mero de una serie psuedo
aleatoria */
/*       en el intervalo [0,1).

          */
/*                Randint devuelve un entero en el
intervalo [low,high]          */
/*************************************************
***********/

#define Rand()  (( Seed = ( (Seed * PRIME) & MASK) ) *
SCALE )

#define Randint(low,high) ( (int) (low + (high-low+1)
* Rand()))

/** fin de archivo **/
```

C.2.2 Módulo EXTERN.H

```
/*
 *  archivo:        extern.h
 *
 *  objetivo:       declaraciones externas.
 */

#include "define.h"

/* El archivo in especifica estos par metros */

extern int    Totalexperiments;         /* cantidad de
experimentos                  */
extern int    Totaltrials;    /* evaluaciones por
experimentos                  */
extern int    Popsize;                  /* tama¤o de
la poblaci¢n */
extern int    Length;                   /* longitud en
bits de la estructura */
extern double  C_rate;                  /*
probabilidad de cruza*/
extern double  M_rate;                  /*
probabilidad de mutacion */
extern double  Gapsize;                 /* fracci¢n de
la poblaci¢n que ser  reemplazada cada generaci¢n */
extern int    Windowsize;     /* usado para actualizar
la peor performance */
extern int    Interval;                 /* cantidad de
evaluaciones entre estad¡sticas         */
extern int    Savesize;                 /* n£mero de
estructuras que se guardan en minfile   */
extern int    Maxspin;                  /* max gens
sin evaluaciones              */
extern int    Dump_freq;      /* gens entre
checkpoints                   */
extern int    Num_dumps;      /* cantidad de archivos
de checkpoint que se guardan  */
extern char   Options[];      /* opciones
                              */
extern unsigned long Seed;    /* semilla para el
generador aleatorio */
extern unsigned long OrigSeed;          /* semilla
inicial   */
```

```


/*****************************************  de
global.h   ******/

/*        Variables globales. */


/* Nombres de archivos */
extern char   Bestfile[];     /* archivo con las
mejores estructuras */
extern char   Ckptfile[];     /* archivo de check
point                         */
extern int    Curr_dump;      /* sufijo del £ltimo
dumpfile  */
extern char   Dumpfile[];     /* dumpfile actual (si
hay m s de uno)     */
extern char   Initfile[];     /* archivo de
estructuras iniciales                   */
extern char   Infile[];                 /* parametros
globales                      */
extern char   Logfile[];      /* log de inicio y
reinicio            */
extern char   Minfile[];      /* prefijo de Bestfile
(se guarda un bestfile por cada experimento       */
extern char   Outfile[];      /* salida "en bruto" de
estad¡sticas                                    */
extern char   Schemafile[];   /* archivo para guardar
la historia de un esquema     */
extern char   Templatefile[]; /* archivo que describe
la interpretacion de los genes          */

extern char   *Bitstring;     /* representaci¢n en
cadena de '1's y '0's         */
extern double *Vector;                  /*
representaci¢n de punto flotante */
extern int Genes;             /* Cantidad de genes
interpretados                 */
extern GENESTRUCT *Gene;      /* puntero a unidades de
template  */
extern STRUCTURE  *Old;                 /* puntero a
la generaci¢n anterior        */
```

```
extern STRUCTURE  *New;                 /* puntero a
la nueva generaci¢n */
extern BESTSTRUCT *Bestset;   /* conjunto de las
mejores estructuras           */

/* datos estad¡sticos y variables auxiliares */
extern double  Ave_current_perf;/* aptitud promedio de
la generaci¢n actual          */
extern double  Best;                    /* la mejor
aptitud hasta ahora */
extern double  Best_current_perf;/* la mejor aptitud
en la generaci¢n actual       */
extern int     Best_guy;                /* numero de
estructura con best_current_perf                  */
extern int     Bestsize;                /* cantidad de
mejores estructuras grabadas */
extern double  Bias;                    /* promedio de
la dominaci¢n de alelos       */
extern int     Bytes;                   /* longitud en
bytes de la estructura empaquetada */
extern int     Conv;            /* n£mero de genes que
convergieron parcialmente     */
extern char    Doneflag;                /* se setea
cuando se cumple la condici¢n de parada */
extern int     Experiment;      /* contador de
experimentos                  */
extern int     Gen;             /* contador         de
generaciones                  */
extern unsigned long    Initseed; /* semilla inicial
del experimento     */
extern int     Lost;            /* cantidad de
posiciones que convergieron totalmente */
extern int     Mu_next;                 /* pr¢xima
posici¢n a mutar              */
extern double  Offline;                 /* performance
offline                       */
extern double  Offsum;                  /* acumulador
para la performance offline   */
extern double  Online;                  /* performance
online                        */
extern double  Onsum;                   /* acumulador
para performance online       */
extern int     Plateau;                 /* contador de
evaluaciones para la siguiente salida   */
extern double Rank_min;                 /* minima tasa
de muestreo para seleccion por ranking  */
extern double  Totbest;                 /* mejor de
todos los experimentos                        */
extern double  Totoffline;     /* offline de todos los
experimentos                  */
extern double  Totonline;      /* online  de todos los
experimentos                  */
extern int     Trials;                  /* contador de
evaluaciones */
```

```
extern double  *Window;                 /* cola
circular de los peores de las £ltimas generaciones
                */
extern double  Worst;                   /* peor
performance hasta ahora       */
extern double  Worst_current_perf;/* peor perf en la
generaci¢n actual */
extern int          Spin;               /* cantidad de
generaciones desde la £ltima evaluaci¢n */

/* flags seteados de acuerdo a Options */
extern char    Allflag;                 /*  evaluar
todas las estructuras         */
extern char    Bestflag;                /*  imprimir
el mejor valor      final     */
extern char    Collectflag;     /*  llevar estad¡sticas
de performance */
extern char    Convflag;                /*  llevar
estadisticas de convergencia */
extern char    Displayflag;     /*  mostrar estadisticas
de cada generacion */
extern char    Dumpflag;                /*  hacer un
vuelco despues de cada evalueacion                */
extern char    Eliteflag;       /*  usar la estrategia
de seleccion elitista         */
extern char    Floatflag;       /*  convertir cadenas a
punto flotante      */
extern char    Grayflag;                /*  usar gray
code                          */
extern char    Initflag;                /*  leer
estructuras iniciales de un archivo               */
extern char    Interflag;       /*  modo interactivo
                              */
extern char    Lastflag;                /*  vuelco de
la ultima generacion */
extern char    Logflag;                 /*  log de
inicios y reinicios           */
extern char    Maxflag;                 /*  maximizar
en lugar de minimizar         */
extern char    Offlnflag;       /*  imprimir medida de
offline final                 */
extern char    Onlnflag;                /*  imprimir
medida de online final        */
extern char    Rankflag;                /*  usar
seleccion por ranking         */
extern char    Restartflag;     /*  recomenzar una
corrida                       */
extern char    Schemflag;       /*  seguir la historia
de un esquema       */
extern char    Traceflag;       /*  mostrar las rutinas
que se ejecutan               */

/** fin de archivo **/
```

C.2.3 Módulo FORMAT.H

```
/*
 *  archivo:        format.h
 *
 *  objetivo:       especificar los formatos para
archivos de entrada y salida
 */

/* el archivo in se lee de acuerdo a  IN_FORMAT e
IN_VARS */

#define IN_FORMAT " \
                                        Exp. = %d \
                        Tot. Eval. = %d \
                                        Pob. = %d \
            Long. cromosoma = %d \
                        Prob. cruza = %lf \
             Prob. mutaci¢n = %lf \
             Salto generac. = %lf \
            Ventana scaling = %d \
            Interv. reporte = %d \
           Estruc. guardadas = %d \
   Max gens sin eval. = %d \
Interv. entre vuelcos = %d \
           Vuelcos guardados = %d \
                            Opciones = %s \
           Semilla aleatoria = %lu \
              Ranking m¡nimo = %lf "



#define IN_VARS
&Totalexperiments,&Totaltrials,&Popsize,&Length,\
          &C_rate,&M_rate,&Gapsize,\
          &Windowsize,&Interval,\
          &Savesize,&Maxspin,\
          &Dump_freq,&Num_dumps,\
          Options,\
          &OrigSeed, &Rank_min



/*        LINE_FIN es el formato de cada linea del
archivo out tal como          */
/*        es leida por la rutina report
                              */
#define LINE_FIN "%lf %lf %lf %lf %lf %lf %lf %lf %lf"

#define LINE_VIN
&line[0],&line[1],&line[2],&line[3],&line[4],\
          &line[5],&line[6],&line[7],&line[8]


/*        formatos de salida.           */

/* OUT_FORMAT es el formato en que se imprimen los
parametros de entrada */

#define OUT_FORMAT " \
                                        Exp. = %d\n\
                        Tot. Eval. = %d\n\
                                        Pob. = %d\n\
            Long. cromosoma = %d\n\
                        Prob. cruza = %lf\n\
             Prob. mutaci¢n = %lf\n\
             Salto generac. = %lf\n\
            Ventana scaling = %d\n\
            Interv. reporte = %d\n\
           Estruc. guardadas = %d\n\
   Max gens sin eval. = %d\n\
Interv. entre vuelcos = %d\n\
           Vuelcos guardados = %d\n\
                            Opciones = %s\n\
           Semilla aleatoria = %lu\n\
              Ranking m¡nimo = %lf\n"

/* OUT_VARS son los par metros que se imprimir n de
acuerdo con OUT_FORMAT */

#define OUT_VARS
Totalexperiments,Totaltrials,Popsize,Length,\
          C_rate,M_rate,Gapsize,\
          Windowsize,Interval,Savesize,Maxspin,\
          Dump_freq,Num_dumps,Options,OrigSeed,Rank_mi
n


/* OUT_F2 es el formato para los datos producidos por
'Measure'.
 * OUT_V2 describe las variables.
 */
#define OUT_F2 "%5d %5d %2d %2d %5.3f %.6e %.6e %.6e
%.6e\n"

#define OUT_V2 Gen,Trials,Lost,Conv,Bias,Online,\
          Offline,Best,Ave_current_perf
```

```
/** fin de archivo **/
```

C.2.4 Módulo GLOBAL.H

```
/*
 *  archivo:        global.h
 *
 *  objetivo:       variables globales
 */

#include "define.h"


/* Nombre de archivos */
char      Bestfile[40];       /* archivo con las
mejores estructuras */
char   Ckptfile[40];          /* archivo de check
point                   */
int    Curr_dump;   /* sufijo del ultimo vuelco   */
char   Dumpfile[40];          /* vuelco actual (si hay
mas de uno)             */
char   Initfile[40];          /* archivo de
estructuras iniciales                 */
char   Infile[40];  /* parametros globales
                    */
char   Logfile[40]; /* logs de inicio y reinicio
          */
char   Minfile[40]; /* prefijo de Bestfile (se guarda
un bestfile por cada experimento        */
char   Outfile[40]; /* salida "en bruto" de
estad¡sticas                                    */
char   Schemafile[40];        /* archivo para guardar
la historia de un esquema     */
char   Templatefile[40]; /* archivo que describe la
interpretacion de los genes   */

char   *Bitstring;  /* representaci¢n en cadena de
'1's y '0's           */
double *Vector;               /* representacion de
punto flotante */
int Genes;                    /* Cantidad de genes
interpretados                 */
GENESTRUCT *Gene;  /* puntero a unidades de template
          */
STRUCTURE  *Old;    /* puntero a la generaci¢n
anterior  */
STRUCTURE  *New;    /* puntero a la nueva generaci¢n
          */
BESTSTRUCT *Bestset;/* conjunto de las mejores
estructuras                   */


/* El archivo in especifica estos par metros */
int    Totalexperiments; /* cantidad de experimentos
                    */
int    Totaltrials; /* evaluaciones por experimentos
                    */
int    Popsize;               /* tama¤o de la
poblaci¢n */
int    Length;                /* longitud en bits de
la estructura */
double  C_rate;               /* probabilidad de
cruza*/
double  M_rate;               /* probabilidad de
mutacion */
double  Gapsize;    /* fracci¢n de la poblaci¢n que
ser  reemplazada cada generaci¢n */
int    Windowsize;  /* usado para actualizar la peor
performance */
int    Interval;    /* cantidad de evaluaciones entre
estad¡sticas          */
int    Savesize;    /* n£mero de estructuras que se
guardan en minfile  */
int    Maxspin;               /* max gens sin
evaluaciones                  */
int    Dump_freq;   /* gens entre checkpoints
          */
int    Num_dumps;   /* cantidad de archivos de
checkpoint que se guardan     */
char   Options[40]; /* opciones
                    */
unsigned long    Seed;        /* semilla para el
generador aleatorio */
unsigned long    OrigSeed; /* semilla inicial     */

/* datos estad¡sticos y variables auxiliares */
double  Ave_current_perf;/* aptitud promedio de la
generaci¢n actual   */
double  Best;                 /* la mejor aptitud
hasta ahora           */
double  Best_current_perf;/* la mejor aptitud en la
generaci¢n actual   */
int    Best_guy;    /* numero de estructura con
best_current_perf             */
int    Bestsize;    /* cantidad de mejores estructuras
grabadas */
double  Bias;                 /* promedio de la
dominaci¢n de alelos                    */
int    Bytes;                 /* longitud en bytes de
la estructura empaquetada */
int    Conv;                  /* n£mero de genes que
convergieron parcialmente     */
char   Doneflag;              /* se setea cuando se
cumple la condici¢n de parada */
int    Experiment;  /* contador de experimentos
                    */
int    Gen;                   /* contador         de
generaciones                  */
unsigned long    Initseed; /* semilla inicial del
experimento           */
int    Lost;                  /* cantidad de
posiciones que convergieron totalmente */
int    Mu_next;               /* pr¢xima posici¢n a
mutar                 */
double  Offline;    /* performance offline
                    */
double  Offsum;               /* acumulador para la
performance offline           */
double  Online;               /* performance online
                              */
double  Onsum;                /* acumulador para
performance online  */
int    Plateau;               /* contador de
evaluaciones para la siguiente salida   */
double Rank_min;    /* minima tasa de muestreo para
seleccion por ranking         */
int    Spin;                  /* cantidad de
generaciones desde la £ltima evaluaci¢n */
double  Totbest;    /* mejor de todos los experimentos
                              */
double  Totoffline; /* offline de todos los
experimentos                  */
double  Totonline;  /* online de todos los
experimentos                  */
int    Trials;                /* contador de
evaluaciones */
double  *Window;    /* cola circular de los peores de
las £ltimas generaciones      */
double  Worst;                /* peor performance
hasta ahora           */
double  Worst_current_perf;/* peor perf en la
generaci¢n actual */

/* flags seteados de acuerdo a Options */
char   Allflag;               /*  evaluar todas las
estructuras                   */
char   Bestflag;              /*  imprimir el mejor
valor     final     */
char   Collectflag;           /*  llevar estad¡sticas
de performance */
char   Convflag;              /*  llevar estadisticas
de convergencia */
char   Displayflag; /*  mostrar estadisticas de cada
generacion */
char   Dumpflag;              /*  hacer un vuelco
despues de cada evalueacion             */
char   Eliteflag;   /*  usar la estrategia de
seleccion elitista  */
char   Floatflag;   /*  convertir cadenas a punto
flotante  */
char   Grayflag;              /*  usar gray code
                    */
char   Initflag;              /*  leer estructuras
iniciales de un archivo                 */
char   Interflag;   /*  modo interactivo
                    */
char   Lastflag;              /*  vuelco de la ultima
generacion */
char   Logflag;               /*  log de inicios y
reinicios           */
char   Maxflag;               /*  maximizar en lugar
de minimizar        */
char   Offlnflag;   /*  imprimir medida de offline
final               */
char   Onlnflag;              /*  imprimir medida de
online final                  */
char   Rankflag;              /*  usar seleccion por
ranking             */
char   Restartflag; /*  recomenzar una corrida
                    */
char   Schemflag;   /*  seguir la historia de un
esquema   */
char   Traceflag;   /*  mostrar las rutinas que se
ejecutan            */

/** fin de archivo **/
```

C.2.5 Módulo IN_EVAL.H

```
/*
 *  archivo:        in_eval.h
 *
 *  definicion de las variables para los parametros
del problema
 */
```

```
 int nt;                /* cantidad de intervalos en
el tiempo */
 int ntmute;                              /* cantidad de
intervalos en t que ignoro la presion escuchada */
 int l;                                   /* cantidad de
intervalos de la malla */
 double dt;                       /* longitud del
intervalito de tiempo */
 double *h;                     /* longitud en metros de
cada intervalo (por capa) */
 double *ronodo;   /* densidad de cada nodo */
 double xsou;           /* posicion de la fuente */
 int jsou;                        /* nodo
inmediatamente superior a la fuente */
 double xjsou;     /*posicion en mts. del nodo
anterior a la fuente */
 double xjsoup1;   /*posicion en mts. del nodo
posterior a la fuente */
 double xrec;            /*posicion en mts. del
receptor  */
 int jrec;                    /*nodo anterior al
receptor  */
 double xjrec;       /*posicion en mts. del nodo
anterior al receptor */
 double xjrecp1;   /*posicion en mts. del nodo
posterior al receptor */
 int *kx;                     /* cantidad acumulada de
intervalos en x de una capa */
```

```
 int keyb;                                /* tipo de
borde:

                                 0: superficie libre

                                 1: condicion de bordes
absorbente

                                 2: neuman

                        */
 int kdel;                                /* tipo de
fuente:

                        1 -> delta de Dirac

                        2 -> derivada de la delta de Dirac

                        3 -> 2 delta de Dirac

                        */
 double *source;    /* funcion fuente muestreada */
 double is;
 double *recmed;   /* presiones observadas en el
receptor */

 /** fin de archivo **/
```

C.2.6 Módulo IN_EVAL2.H

```
/*
 *  archivo:        in_eval2.h
 *
 *  declaraciones externas para los parametros del
problema
 */

extern int nt;                    /* cantidad de
intervalos en el tiempo */
extern int ntmute;                        /* cantidad de
intervalos en t que ignoro la presion escuchada */
extern int l;                                        /*
cantidad de intervalos de la malla */
extern double dt;                 /* longitud del
intervalito de tiempo */
extern double *h;              /* longitud en metros de
cada intervalo (por capa) */
extern double *ronodo;   /* densidad de cada nodo */
extern double xsou;     /* posicion de la fuente */
extern int jsou;                  /* nodo
inmediatamente superior a la fuente */
extern double xjsou;     /*posicion en mts. del nodo
anterior a la fuente */
extern double xjsoup1;   /*posicion en mts. del nodo
posterior a la fuente */
extern double xrec;      /*posicion en mts. del
receptor  */
extern int jrec;               /*nodo anterior al
receptor  */
extern double xjrec;           /*posicion en mts. del
nodo anterior al receptor */
```

```
extern double xjrecp1;   /*posicion en mts. del nodo
posterior al receptor */
extern int *kx;                           /* cantidad
acumulada de intervalos en x de una capa */
extern int keyb;                          /* tipo de
borde:

                                 0: superficie libre

                                 1: condici¢n de bordes
absorbente

                                 2: neuman

                        */
extern int kdel;                          /* tipo de
fuente:

                        1 -> delta de Dirac

                        2 -> derivada de la delta de Dirac

                        3 -> 2 delta de Dirac

                        */
extern double *source;        /* funcion fuente
muestreada */
extern double is;
extern double *recmed;   /* presiones observadas en el
receptor */

 /** fin de archivo **/
```

C.2.7 Módulo PRESION.H

```
/*
 *  archivo:        presion.h
 *
 *  c lculo de las presiones
 */

#ifndef PRESION_H
#define PRESION_H

double una_presion(double *unm1, double *un, double
*unp1, int paso, double *constk, double *velnodo);
void presiones(double *velocidad, int capas, double
*presion, int nt);
```

```
void vel_en_nodos(double *velocidad, int capas, double
*velnodo);
double borde_sup(double unm1, double unx1, double
unx2, double cte);
double borde_inf(double unm1, double unx1, double
unx2, double cte);
double fuente_en_borde(int paso, int nodo);
double fuente_interior(int paso, int nodo, double cte,
double *velnodo);

#endif

/** fin de archivo **/
```

C.2.8 Módulo MAIN.C

```
/*
 *  archivo:        main.c
 *
 *  objetivo:       programa principal.
 *
 */

#include "global.h"
#include "in_eval.h"

main(argc,argv)
int argc;
char *argv[];
{
          FILE *fp, *fopen();
```

```
          long clock;
          long time();
          char *ctime();
          extern void die();  /* manejador de se¤ales
*/
          int i;


          /* ver el uso de parametros en la linea de
comandos en input.c */
          Input(argc,argv);
          In_eval(argc,argv);

          if (Displayflag) {
                    initscr();
```

```
                    signal(SIGINT, die);
                    for (i=1;i<=23;i++) {
                              move(i,0);
                              clrtoeol();
                    }
                    refresh();

                    if (Interflag)
                              Interactive(); /* nunca
retorna */

                    /* este punto se alcanza solo si
Interflag esta en OFF */
                    /* l¡nea superior */
                    move(1,0);
                    for (i=0;i<=79;i++) {
                              printw("-");
                    }
                    /* l¡nea inferior */
                    move(23,0);
                    for (i=0;i<=79;i++) {
                              printw("-");
                    }

                    move(2,0);
                    printw(" Propuesta   I  Modelo %s
Tope de evaluaciones = %d", argv[1], Totaltrials);
                    refresh();
          }

          do                  /* un experimento */
                    {

                    move(2,60);
                    printw("Experimento %d/%d",
Experiment+1,Totalexperiments);
                    /* limpiar de la pantalla los
resultados del experimento anterior */
                    for (i=5;i<=22;i++) {
                              move(i,0);
                              clrtoeol();
                    }
                    refresh();

                    do        /* ver el ciclo
principal de AG en generate.c */
                              {
                              Generate();
                    }
                    while (!Doneflag);

                    if (Traceflag)
```

```
                    printf("Online %e   Offline %e
Mejor %e\n",
                              Online, Offline, Best);

                    /* acumular mediciones de
performance */
                    Totonline += Online;
                    Totoffline += Offline;
                    Totbest += Best;

                    /* preparar el proximo experimento
*/
                    Experiment++;
                    Gen = 0;

          }
          while (Experiment < Totalexperiments);

          /* calcular e imprimir las mediciones
finales de performance */

          Totonline /= Totalexperiments;
          Totoffline /= Totalexperiments;
          Totbest /= Totalexperiments;

          if (Displayflag) {
                    move(15,0);
                    printf(" Mejor individuo obtenido
en todos los experimentos:   %e\n", Totbest);
                    refresh();
          }
          if (Logflag)
          {
                    fp = fopen(Logfile, "a");
                    fprintf(fp, "Mejor %e\n",
Totbest);
                    time(&clock);
                    fprintf(fp, "%s\n",
ctime(&clock));
                    fclose(fp);
          }

          if (Displayflag) {
                    move(22,0);
                    printf(" Presione cualquier tecla
para continuar...");
                    getch();
                    die();
          }
}

/** fin de archivo **/
```

C.2.9 Módulo BEST.C

```
/*
 *  archivo:        best.c
 *
 *  objetivo:       entrada, mantenimiento y salida de
las mejores estructuras
 *
 */

#include "extern.h"

static double worst_value;    /* peor valor en
Bestset              */
static int worst;             /* puntero al peor
elemento (elemento con worst_value)     */


Savebest(i)
register int i;               /* indice de estructura
en  Bestset         */
{
          /*  Grabar la i-esima estructura de la
poblacion actual */
          /*  La cantidad de estructuras a grabar es
Savesize */

          register int j;
          register int k;
          int found;

          if (Bestsize < Savesize)
          {
                    /* Bestsize es la cantidad grabada
hasta ahora, entonces */
                    /* todavia hay lugar en Bestset
                    */
                    /* Ver si ya existe una estructura
identica en Bestset */
                    for (j=0, found=0; j<Bestsize &&
(!found); j++)
                              for (k=0, found=1;
(k<Bytes) && (found); k++)
                                        found =
(New[i].Gene[k]
                                                  ==
Bestset[j].Gene[k]);
                    if (found) return;

                    /* insertar la i-esima estructura
*/
                    for (k=0; k<Bytes; k++)
                    {
```

```
          Bestset[Bestsize].Gene[k] = New[i].Gene[k];
                    }
                    Bestset[Bestsize].Perf =
New[i].Perf;
                    Bestset[Bestsize].Gen = Gen;
                    Bestset[Bestsize].Trials = Trials;
                    Bestsize++;
                    if (Bestsize == Savesize)
                    {
                              /* buscar el peor
elemento en Bestset */
                              worst_value =
Bestset[0].Perf;
                              worst = 0;
                              for (j=1; j<Savesize;
j++)
                              {
                                        if
(BETTER(worst_value,Bestset[j].Perf))
                                        {

          worst_value = Bestset[j].Perf;

          worst = j;
                                        }
                              }
                    }
          }
          else
          {
                    /* Ya estan ocupados todos los
lugares, ver si hay que desplazar
                       a alguno */
                    if (BETTER(New[i].Perf,
worst_value))       /* hay que grabar New[i] */
                    {
                              /* a menos que ya este
representada */
                              for (j=0, found=0;
j<Bestsize && (!found); j++)
                                        for (k=0,
found=1; (k<Bytes) && (found); k++)

          found = (New[i].Gene[k]

          == Bestset[j].Gene[k]);
                              if (found) return;

                              /* sobreescribir la peor
estructura */
```

```
                                for (k=0; k<Bytes; k++)
                                {

          Bestset[worst].Gene[k] = New[i].Gene[k];
                                }
                                Bestset[worst].Perf =
New[i].Perf;
                                Bestset[worst].Gen =
Gen;
                                Bestset[worst].Trials =
Trials;
                                /* buscar el peor
elemento Bestset */
                                worst_value =
Bestset[0].Perf;
                                worst = 0;
                                for (j=1; j<Savesize;
j++)
                                {
                                        if
(BETTER(worst_value, Bestset[j].Perf))
                                        {

          worst_value = Bestset[j].Perf;

          worst = j;
                                        }
                                }
                        }
          }
}


Printbest()
{
          /*        Escribir las mejores estructuras
en Bestfile.        */

          register int i;
          register int j;
          register int k;
          FILE *fp, *fopen();

          fp = fopen(Bestfile, "w");

          for (i=0; i<Bestsize; i++)
          {
                    Unpack(Bestset[i].Gene, Bitstring,
Length);
                    if (Floatflag)
                    {
                              FloatRep(Bitstring,
Vector, Genes);
                              for (j=0; j<Genes; j++)
                              {
                                        fprintf(fp,
Gene[j].format, Vector[j]);
                                        fprintf(fp, "
");
                              }
                    }
                    else
                    {
                              for (j=0; j<Length; j++)
                              {
                                        fprintf(fp,
"%c", Bitstring[j]);
                              }
                    }
                    fprintf(fp, "  %11.4e ",
Bestset[i].Perf);
                    fprintf(fp, " %4d  %4d\n",
Bestset[i].Gen, Bestset[i].Trials);
          }

          fclose(fp);
}


Readbest()
{
```

```
          /*   Leer las mejores estructuras de
Bestfile    */
          /*   durante un Restart
*/

          int i,j,k;
          FILE *fp, *fopen();
          int status;

          if (Savesize)
          {

                    fp = fopen(Bestfile, "r");
                    if (fp == NULL)
                    {
                              Bestsize = 0;
                              return;
                    }

                    Bestsize = 0;
                    status = 1;
                    if (Floatflag)
                    {
                              for (i = 0; i < Genes &&
status != EOF; i++)
                              {
                                        status =
fscanf(fp, "%lf", &Vector[i]);
                              }
                    }
                    else
                              status = fscanf(fp,
"%s", Bitstring);

                    while (status != EOF)
                    {
                              if (Floatflag)

          StringRep(Vector, Bitstring, Genes);

                              Pack(Bitstring,
Bestset[Bestsize].Gene, Length);

                              fscanf(fp, "%lf ",
&Bestset[Bestsize].Perf);
                              fscanf(fp, "%d  %d",
&Bestset[Bestsize].Gen,
          &Bestset[Bestsize].Trials);
                              Bestsize++;

                              /* Tomar la proxima
estructura */
                              if (Floatflag)
                                        for (i = 0; i
< Genes && status != EOF; i++)
          status = fscanf(fp, "%lf", &Vector[i]);
                              else
                                        status =
fscanf(fp, "%s", Bitstring);
                    }
                    fclose(fp);

                    /* buscar el peor elemento en
Bestset */
                    worst_value = Bestset[0].Perf;
                    worst = 0;
                    for (i=1; i<Bestsize; i++)
                    {
                              if (BETTER(worst_value,
Bestset[i].Perf))
                              {
                                        worst_value =
Bestset[i].Perf;
                                        worst = i;
                              }
                    }
          }
}

/**** fin de archivo ****/
```

C.2.10 Módulo CHECKPNT.C

```
/*
 *  archivo:        checkpoint.c
 *
 *  objetivo:       grabar las variables globales para
un eventual reinicio.
 *
 */
#include <dir.h>
#include "extern.h"


Checkpoint(ckptfile)
char ckptfile[];
{
          FILE *fp, *fopen();
          int i,j;
          char CkptfileExp[MAXPATH], drive[MAXDRIVE],
dir[MAXDIR], file[MAXFILE], ext[MAXEXT];
```

```
          /* Archivo de checkpoint */
          fnsplit(ckptfile, drive, dir, file, ext);
          sprintf(CkptfileExp, "%s%d%s", file,
Experiment, ext);
          fp = fopen(CkptfileExp, "w");

          fprintf(fp, "Experimentos %d\n",
Experiment);
          fprintf(fp, "On_line total   %12.6e\n",
Totonline);
          fprintf(fp, "Off_line total   %12.6e\n",
Totoffline);
          fprintf(fp, "Generaci¢n %d\n", Gen);
          fprintf(fp, "Performance On_line  %12.6e\n",
Onsum);
          fprintf(fp, "Performance Off_line %12.6e\n",
Offsum);
          fprintf(fp, "Evaluaciones  %d\n", Trials);
```

```
          fprintf(fp, "Pr¢xima estad¡stica %d\n",
Plateau);
          fprintf(fp, "Mejor  %12.6e\n", Best);
          fprintf(fp, "Peor   %12.6e\n", Worst);
          fprintf(fp, "Cantidad de generaciones desde
la £ltima evaluaci¢n %d\n", Spin);
          fprintf(fp, "Ultimo vuelco %d\n",
Curr_dump);
          fprintf(fp, "Mu_prox %d\n", Mu_next);
          fprintf(fp, "Semilla Aleatoria %lu\n",
Seed);
          fprintf(fp, "Semilla Inicializadora %lu\n",
Initseed);

          fprintf(fp,"\n");
          fprintf(fp, "Window\n");
          for (i=0; i<Windowsize; i++) fprintf(fp,
"%12.6e\n", Window[i]);
          fprintf(fp,"\n");


          for (i=0; i<Popsize; i++)
          {
                    Unpack(New[i].Gene, Bitstring,
Length);
                    fprintf(fp, "%s", Bitstring);
                    fprintf(fp, " %12.8e ",
New[i].Perf);

                    fprintf(fp, "%1d   ",
New[i].Needs_evaluation);

                    fprintf(fp, "\n");
          }
```

```
          if (Floatflag)      /* imprimir en punto
flotante */
          {
                    fprintf(fp, "\n");
                    for (i=0; i<Popsize; i++)
                    {
                              Unpack(New[i].Gene,
Bitstring, Length);
                              FloatRep(Bitstring,
Vector, Genes);
                              for (j=0; j < Genes;
j++)
                              {
                                        fprintf(fp,
Gene[j].format, Vector[j]);
                                        fprintf(fp, "
");
                              }
                              fprintf(fp, " %10.4f",
New[i].Perf);
                              fprintf(fp, "\n");
                    }
          }

          fclose(fp);

          /*  guardar las mejores estructuras en
Bestfile */
          if (Savesize)
                    Printbest();

}

/** fin de archivo **/
```

C.2.11 Módulo CONVERT.C

```
/*
 *  archivo:        convert.c
 *
 *  objetivo:       funciones que traducen entre
distinas representaciones
 *
 */

#include "extern.h"

static char BIT[CHARSIZE] ={ '\200', '\100', '\040',
'\020',
                                        '\010',
'\004', '\002', '\001'};


/* Itoc and Ctoi traducen ints a strings y viceversa
*/


unsigned long int Ctoi(instring, length)
          char *instring;               /* cadena de
'0's y '1's         */
          int length;                   /* longitud de
instring            */
{
          register int i;
          unsigned long n;    /* acumulador para el
valor a retornar */

          n = (unsigned long) 0;
          for (i=0; i<length; i++)
          {
                    n <<= 1;
                    n += (*instring++ - (int) '0');
          }
          return(n);
}



Itoc(n, outstring, length)
          unsigned long int n;          /* entero a
convertir           */
          char *outstring;    /* cadena de '0's y '1's
resultante */
          int length;                   /* longitud de
outstring           */
{
          register int i;

          for (i=length-1; i>=0; i--)
          {
                    outstring[i] = '0' + (n & 1);
                    n >>= 1;
          }
}


/* Pack y Unpack traducen entre cadenas de '0's y '1's
y arreglos de bits
   empaquetados */

Pack(instring, outstring, length)
          char *instring;               /* cadena de
'0's y '1's */
          char *outstring;    /* representacion
empaquetada de instring */
```

```
          int length;                   /* longitud de
instring   */
{
          static firstflag = 1 ;
          static full;        /* numero de bytes
completamente usados en outstring       */
          static slop;        /* numbero de bits
usados en el ultimo byte de outstring */
          register i,j;

          if (firstflag)
          {
                    full = length / CHARSIZE;
                    slop = length % CHARSIZE;
                    firstflag = 0;
          }

          for (i=0; i<full; i++, outstring++)
          {
                    *outstring = '\0';
                    for (j=0; j < CHARSIZE; j++)
                              if (*instring++ == '1')
*outstring |= BIT[j];
          }
          if (slop)
          {
                    *outstring = '\0';
                    for (j=0; j < slop; j++)
                              if (*instring++ == '1')
*outstring |= BIT[j];
          }
}


Unpack(instring, outstring, length)
          char *instring;               /*
representacion binaria empaquetada      */
          char *outstring;    /* representacion en
instring en cadena de '0's y '1's */
          int length;                   /* longitud de
outstring                   */
{
          static firstflag = 1 ;
          static full;        /* numbero de bytes
completamente usados en instring */
          static slop;        /* numbero de bits
usados en el ultimo byte de instring */
          register i,j;

          if (firstflag)
          {
                    full = length / CHARSIZE;
                    slop = length % CHARSIZE;
                    firstflag = 0;
          }

          for (i=0; i<full; i++, instring++)
          {
                    for (j=0; j < CHARSIZE; j++)
                              if (*instring & BIT[j])
                                        *outstring++ =
'1';
                              else
                                        *outstring++ =
'0';
          }

          if (slop)
```

```
        {
                for (j=0; j < slop; j++)
                        if (*instring & BIT[j])
                                *outstring++ =
'1';
                        else
                                *outstring++ =
'0';
        }
        *outstring = '\0';
}


/* Traducciones entre representaciones binarias
tradicionales de enteros
   (punto fijo) y Gray code */


Gray(instring, outstring, length)
char *instring;                 /* string que representa
un entero en punto fijo */
char *outstring;    /* string que representa el entero
en Gray code        */
register int length;            /* longitud de strings
                                */
{
        register int i;
        register char last;

        last = '0';
        for (i=0; i<length; i++)
        {
                outstring[i] = '0' + (instring[i]
!= last);
                last = instring[i];
        }
}


Degray(instring, outstring, length)
char *instring;                 /* string que representa
el entero en Gray code          */
char *outstring;    /* string que representa el entero
en punto fijo */
register int length;            /* longitud de strings
                                */
{
        register int i;
        register int last;

        last = 0;
        for (i=0; i<length; i++)
        {
                if (instring[i] == '1')
                        outstring[i] = '0' +
(!last);
                else
                        outstring[i] = '0' +
last;
                last = outstring[i] - '0';
        }
}


/* Traducciones entre cadenas de '0's y '1's y
vectores de puntos flotantes
 */


FloatRep(instring, vect, length)
        char instring[];    /* cadena de '0's y '1's
                */
        double vect[];                  /*
representacion en punto flotante */
        int length;                     /* longitud de
vect (arreglo de salida)        */
{
        register int i;
        unsigned long int n;            /* Valor
entero decodificado                     */
        register int pos;   /* posicion para
comenzar a decodificar                  */
        char tmpstring[80]; /* usado para la
interpretacion en Gray code */

        pos = 0;
        for (i=0; i < length; i++)
        {
                if (Grayflag)
                {
                        Degray(&instring[pos],
tmpstring, Gene[i].bitlength);
                        n = Ctoi(tmpstring,
Gene[i].bitlength);
                }
                else
                {
                        n = Ctoi(&instring[pos],
Gene[i].bitlength);
                }
                vect[i] = Gene[i].min +
n*Gene[i].incr;
                pos += Gene[i].bitlength;
        }
}


StringRep(vect, outstring, length)
        double *vect;                   /*
representacion en punto flotante        */
        char *outstring;    /* cadena de '0's y '1's
                            */
        int length;                     /* longitud de
vect                            */
{
        register int i;
        unsigned long int n;            /* valor
entero (indice) con que se representa vect[i]*/
        register int pos;   /* proxima posicion para
llenar  outstring   */
        char tmpstring[80]; /* usado para la
traduccion a gray code          */

        pos = 0;
        for (i=0; i < length; i++)
        {
                /* Convertir valor de punto
flotante a un indice */
                n = (int) ((vect[i] - Gene[i].min)
/ Gene[i].incr + 0.5);

                /* codificar n como cadena de
caracteres */
                if (Grayflag)
                {
                        /* convertir a Gray code
*/
                        Itoc(n, tmpstring,
Gene[i].bitlength);
                        Gray( tmpstring,
&outstring[pos], Gene[i].bitlength);
                }
                else
                {
                        Itoc(n, &outstring[pos],
Gene[i].bitlength);
                }
                pos += Gene[i].bitlength;
        }
        outstring[pos] = '\0';
}


/*** fin de archivo ***/
```

C.2.12 Módulo CROSS.C

```
/*
 *  archivo:        cross.c
 *
 *  objetivo:       efectuar la cruza en dos puntos
sobre toda la poblaci¢n
 *              La cruza se realiza a nivel de bit.
 */

#include <alloc.h>
#include "extern.h"

/**** las mascaras estan en octal ****/
char premask[CHARSIZE] = { '\000', '\200', '\300',
'\340',
                                        '\360',
'\370', '\374', '\376' };

char postmask[CHARSIZE] = { '\377', '\177', '\077',
'\037',
                                        '\017',
'\007', '\003', '\001'};


Crossover()
```

```
/**** Si los padres no difieren en los segmentos
externos =>
         no cruza & no evalua
      Sino
          cruza, compara los segmentos internos
          Si los segmentos internos son iguales =>
             no evalua
****/
{
        register int mom, dad;          /*
participantes en la cruza */
        register int xpoint1;           /* primer
punto de cruza con respecto a la estructura */
        register int xpoint2;   /* segundo punto de
cruza con respecto a la estructura */
        register int xbyte1;            /* primer byte
a cruzar */
        register int xbit1; /* primer bit a cruzar
en el byte xbyte1 */
        register int xbyte2;            /* ultimo byte
a cruzar */
        register int xbit2; /* ultimo bit a cruzar
en xbyte2 */
        register int i;
```

```
        register char temp; /* usado para
intercambiar alelos */
        static int last;    /* ultimo individuo en
sufrir la cruza */
        int diff;           /* indica si los padres
difieren de los hijos */
        char *kid1;                 /* punteros a
los hijos */
        char *kid2;
        static int firstflag = 1;

        if (firstflag)
          {
            last = (C_rate*Popsize*Gapsize) - 0.5 ;
            firstflag = 0;
          }

        for (mom=0; mom < last ; mom += 2)
        {
                dad = mom + 1;

                /* los hijos comienzan siendo
copias identicas de los padres */
                kid1 = New[mom].Gene;
                kid2 = New[dad].Gene;

                /* se eligen dos puntos de cruza
*/
                xpoint1 = Randint(0,Length);
                xpoint2 = Randint(0,Length-1);

                /* garantizo que xpoint1 < xpoint2
*/
                if (xpoint2 >= xpoint1)
                        xpoint2++;
                else
                {
                        i = xpoint1;
                        xpoint1 = xpoint2;
                        xpoint2 = i;
                }

                xbyte1 = xpoint1 / CHARSIZE;
                xbit1 = xpoint1 % CHARSIZE;
                xbyte2 = xpoint2 / CHARSIZE;
                xbit2 = xpoint2 % CHARSIZE;

                /* Me fijo si los padres difieren
en los segmentos externos
                   ya que si los mismos son
iguales, la cruza no tiene sentido
                   pues los hijos saldrian iguales
a los padres */
                diff = 0;
/**** comparo bytes enteros de la izq. ****/
                for (i=0; i < xbyte1; i++) diff +=
(kid1[i] != kid2[i]);
/**** comparo fraccion de byte de la izq. ****/
                diff += ( (kid1[xbyte1] &
premask[xbit1]) !=
                          (kid2[xbyte1] &
premask[xbit1]) );
/**** comparo fraccion de byte de la der. ****/
                diff += ( (kid1[xbyte2] &
postmask[xbit2]) !=
                          (kid2[xbyte2] &
postmask[xbit2]) );
/**** comparo bytes enteros de la der. ****/
                for (i=xbyte2+1; i < Bytes; i++)
diff += (kid1[i] != kid2[i]);

                if (diff) /* los padres difieren
en los segmentos externos */
                {
                        /* cruzarlos */
                        temp = kid1[xbyte1];
/**** cruzo el xbyte1 ****/
                        kid1[xbyte1] =
(kid1[xbyte1] & premask[xbit1]) |

        (kid2[xbyte1] & postmask[xbit1]);

                        kid2[xbyte1] =
(kid2[xbyte1] & premask[xbit1]) |

        (temp & postmask[xbit1]);

                        diff  = ((kid1[xbyte1] &
postmask[xbit1]) !=
                                 (kid2[xbyte1]
& postmask[xbit1]) );

                        for (i=xbyte1 + 1; i <
xbyte2; i++)
/**** cruzo el segmento de bytes enteros, pero no el
xbyte2 ****/
                        {
                                temp =
kid1[i];
                                kid1[i] =
kid2[i];
                                kid2[i] =
temp;
                                diff +=
(kid1[i] != kid2[i]);
                        }

                        if (xbyte1 < xbyte2)
/**** si la cruza abarca mas de un byte.... ****/
                        {
/**** cruzo el xbyte2 ****/
                                temp =
kid1[xbyte2];
                                kid1[xbyte2] =
(kid1[xbyte2] & postmask[xbit2]) |

        (kid2[xbyte2] & premask[xbit2]);

                                kid2[xbyte2] =
(kid2[xbyte2] & postmask[xbit2]) |

        (temp & premask[xbit2]);

                                diff  +=
((kid1[xbyte2] & premask[xbit2]) !=

        (kid2[xbyte2] & premask[xbit2]) );
                        }
                        else
                        {
                                temp =
kid1[xbyte2];
                                kid1[xbyte2] =
(kid1[xbyte2] & premask[xbit2]) |

        (kid2[xbyte2] & postmask[xbit2]);

                                kid2[xbyte2] =
(kid2[xbyte2] & premask[xbit2]) |

        (temp & postmask[xbit2]);

                                diff  =
((kid1[xbyte2] & postmask[xbit1] & premask[xbit2]) !=

        (kid2[xbyte2] & postmask[xbit1] &
premask[xbit2]) );

                        }

                        if (diff) /* los hijos
difieren de los padres */
                        {
                                /* entonces,
los hijos deben ser evaluados */

        New[mom].Needs_evaluation = 1;

        New[dad].Needs_evaluation = 1;
                        }
                }
        }
}

/** fin de archivo **/
```

C.2.13 Módulo DISPLAY.C

```
/*
 *  archivo:        display.c
 *
 *  objetivo:       maneja la pantalla
 *
 */

#include "extern.h"

Interactive() {
        char cmd[40];
        char opt[40];
        register int i;
        int ncycles;
        int ok;

        ncycles = 1;
        while (1) {
                ok = 1;
                move(22,0);
                clrtoeol();
                move(22,35);
                printw("q (clear & exit), x
(exit), <n> (do n gens)");
                move(22,0);
                printw("
");
                refresh();
                move(22,0);
                printw("gens[%d]: ", ncycles);
                refresh();
                getstr(cmd);
                if (strcmp(cmd, "q") == 0) {
                        if (Lastflag)

        Checkpoint(Ckptfile);
                        else
                                if (Savesize)

        Printbest();
```

```
                                clear();
                                die();
                        }
                        if (strcmp(cmd, "x") == 0) {
                                if (Lastflag)

        Checkpoint(Ckptfile);
                                else
                                        if (Savesize)

        Printbest();
                                move(23,0);
                                die();
                        }
                        if (strcmp(cmd, "") != 0) {
                                if (sscanf(cmd, "%d",
&ncycles) !=1)
                                {
                                        move(23,0);
                                        clrtoeol();

        printw("unknown command: %s", cmd);
                                        ok = 0;
                                        refresh();
                                }
                        }
                        if (ok)
                        {
                                move(23,0);
                                clrtoeol();
                                move(1,0);
                                printw("run until Gens =
%d", Gen + ncycles -1);
                                move(1,35);
                                printw("executing: ");
                                refresh();
                                for (i=0; i < ncycles;
i++)
                                        Generate();
                        }
                }
}
```

```
die(sig)
int sig;
{
        sig++;
        signal(SIGINT, SIG_IGN);
        move(23,0);
        clrtoeol();
        refresh();
        endwin();
        exit(0);
}


move(row, col)
        int row, col;
{
        /* mover a las coordenadas fila, columna de
la pantalla */
        /* (0,0) = esquina superior izquierda;
           (23,79) = esquina inferior derecha
        */

        gotoxy(col+1, row+1);
}

clear()
{
        /* limpiar la pantalla */
        clrscr();
}


getstr(s)
     char *s;
{
  getw(s);
}

initscr() {}

endwin() {}

/**** fin de archivo ****/
```

C.2.14 Módulo DONE.C

```
/*
 *  archivo:        done.c
 *
 *  objetivo:       ver si se cumple la condicion de
parada.
 *
 */

#include "extern.h"

int Done()
```

```
{

/**** por demasiadas evaluaciones, o toda la poblacion
es igual, o
        demasiadas generaciones sin evaluar ****/
        return ((Trials >= Totaltrials)  || ( Lost
>= Length)
                || (Spin >= Maxspin));
}

/* fin de archivo */
```

C.2.15 Módulo ELITIST.C

```
/*
 *  archivo:        elitist.c
 *
 *  objetivo:       La politica elitista estipula que
el mejor individuo
 *                  siempre sobrevive en la nueva
generacion. El individuo de elite
 *                  es ubicado en la ultima posicion
de la nueva poblacion
 *            y no es afectado por cruza o mutacion ya
que se pasa a la nueva
 *                  generacion despues de haber
aplicado estos operadores.
 */

#include "extern.h"

Elitist()
{
        register int i;
        register int k;
        register int found; /* se setea si ya existe
el individuo de elite en

            la nueva generacion */


        /* Hay algun elemento en la poblacion actual
                */
        /* que sea identico al mejor de la
generacion anterior?        */
```

```
        for (i=0, found=0; i<Popsize && (!found);
i++)
                for (k=0, found=1; (k<Bytes) &&
(found); k++)
                        found = (New[i].Gene[k]
                                        ==
Old[Best_guy].Gene[k]);

        if (!found)         /* No se encontro el
individuo de elite */
        {
                /* reemplazar el ultimo individuo
por el individuo de elite */
                for (k=0; k<Bytes; k++)
                        New[Popsize-1].Gene[k] =
Old[Best_guy].Gene[k];
                New[Popsize-1].Perf =
Old[Best_guy].Perf;
                New[Popsize-1].Needs_evaluation =
0;

                if (Traceflag)
                {
                        printf("perf: %e\n",
New[Popsize-1].Perf);
                }
        }

}


/*** fin de archivo ***/
```

C.2.16 Módulo ERROR.C

```
/*
 *  archivo:        error.c
 *
 *  objetivo:       imprimir mensaje de error y
abortar. El error queda tambien
```

```
 *                              en un log de
errores.
 *
 */

#include <stdio.h>
```

```
Error(s)
char *s;
{
          FILE *fopen(), *fp;
          long clock;
          long time();
          char *ctime();

          fp = fopen("error", "a");
          fprintf(fp, "%s\n", s);
          time(&clock);
          fprintf(fp, "%s\n", ctime(&clock));
          fclose(fp);
          fprintf(stderr, "%s\n", s);

          exit(1);
}

/*** fin de archivo ***/
```

C.2.17 Módulo EVAL.C

```
/*
 * archivo: eval.c
 *
 * objetivo: calcular la aptitud del individuo.
 */
#include <alloc.h>
#include "extern.h"
#include "in_eval2.h"

double costo(v1, v2, N)
   double v1[], v2[];
   int N;
/*                N
                                        --
                                1    \              2
                                -    /   ( v1 - v2 )
                                n         --      i
i
                               i=1
*/
{
          int i;
          double s= 0;

          for (i=1; i <= N; i++)
                    s+= pow(( v1[i] - v2[i] ),2);
          return s/N;
}
```

```
double eval(str, length, vect, genes)
char str[]; /* representacion en cadena de '0's y '1's
               */
int length;         /* longitud de la cadena
                    */

/* vect: velocidades de cada una de las "genes" capas
*/
double vect[];      /* representacion en punto
flotante            */

/* genes: cantidad de capas */
int genes;          /* cantidad de elementos en el
vector                        */
{
          double resul;
          /* presiones calculadas a partir de las
velocidades, una por cada
          instante de tiempo */
          double
*recest=malloc((nt+1)*sizeof(double));

          presiones(vect, genes, recest, nt);
          resul =costo(recest, recmed, nt);
          free(recest);
          return resul;

}
```

C.2.18 Módulo EVALUATE.C

```
/*
 *  archivo:        evaluate.c
 *
 *  objetivo:       evaluar la poblacion actual
mediante la funcion "eval"
 *
 */

#include "extern.h"
extern double eval();

Evaluate()
{
          register double performance;
          register int i;

          for (i=0; i<Popsize; i++)
          {
                    if (Displayflag) {
                              move(20,0);
                              clrtoeol();
                              printf(" Evaluando
individuo %d de la poblaci¢n",i);
                              refresh();
                    }

                    if ( New[i].Needs_evaluation )
                    {
                              Unpack(New[i].Gene,
Bitstring, Length);
                              if (Floatflag)

          FloatRep(Bitstring, Vector, Genes);
                              New[i].Perf = eval
(Bitstring, Length,
                                        Vector,
Genes);
                              performance =
New[i].Perf;
                              New[i].Needs_evaluation
= 0;
                              Trials++;
                              Spin = 0;   /* Hubo una
evaluacion, spin vuelve a 0 */
                              if (Savesize)
Savebest(i);

                              if (Trials == 1)
                                        Best =
performance;
                              if (BETTER(performance,
Best))
                              {
                                        Best =
performance;
                              }

                              Onsum += performance;
                              Offsum += Best;
                              if (Dumpflag)
Checkpoint(Ckptfile);
                    }
          }
          move(20,0);
          clrtoeol();
}


/** fin de archivo **/
```

C.2.19 Módulo GENERATE.C

```
/*
 *  archivo:        generate.c
 *
 *  objetivo:       Una generacion consiste de
 *                  (1) formar una nueva
poblacion de estructuras
 *                  (2) evaluar la poblacion
 *                  (3) recoger estadisticas
de performance
 *
 */
```

```
#include "extern.h"
extern int  Done();


Generate()
{
          static int rsflag = 1;        /* flag
reseteado despues de restart            */
```

```
        STRUCTURE *temp;    /* para swapear punteros
de la poblacion     */
        register int i;             /* para marcar
estructuras                 */

        if (Traceflag)
                printf("                    Gen
%d\n",Gen);

        /* crear una nueva poblacion */

        if (Restartflag && rsflag)
        {
                /* si es un restart leer archivo
de checkpoint */
                Restart();
                rsflag = 0;         /* Ya hicimos
este restart. */
                Converge();
        }

        else if (Gen == 0)
                /* Si recien comienza el
experimento */
        {
                Initialize();       /* formar una
poblacion inicial */
                Spin++;
        }

        else
                /* formar una nueva poblacion a
partir de */
                /* la vieja via operadores
geneticos */
        {
                Select();
                Mutate();
                Crossover();
                if (Eliteflag)
                        Elitist();
                if (Allflag)        /* marcar las
estructuras para evaluacion */
                        for (i=0; i<Popsize;
i++) New[i].Needs_evaluation = 1;
                Spin++;
        }

        /* evaluar la poblacion recientemente
formada */
        Evaluate();

        /* recoger estadisticas de performance */
        Measure();

        /* chequear la condicion de parada para este
experimento */
        Doneflag = Done();

        /* Hacer vuelco si corresponde */
        if (Num_dumps && Dump_freq && Gen %
Dump_freq == 0)
        {
                if (Num_dumps > 1)
                {
                        sprintf(Dumpfile,
"dump.%d", Curr_dump);
                        Curr_dump = (Curr_dump +
1) % Num_dumps;
                        Checkpoint(Dumpfile);
                }
                Checkpoint(Ckptfile);
        }
        else
        {
                if (Doneflag)
                {
                        if (Lastflag)

        Checkpoint(Ckptfile);
                        else
                                if (Savesize)

        Printbest();
                }
        }


        /* Intercambiar punteros para la proxima
generacion */
        temp = Old;
        Old = New;
        New = temp;

        /* actualizar contador de generaciones */
        Gen++;

}

/*  fin de archivo */
```

C.2.20 Módulo IN_EVAL.C

```
/*
 *  archivo:        in_eval.c
 *
 *  objetivo:       Lectura de parametros del modelo a
ser tratado por eval
 *              Este archivo es problema-dependiente
 *                                  Es decir debe
modificarse (o eliminarse) de trabajarse
 *                      con otro problema.
 *
 */
#include <alloc.h>
#include "extern.h"   /* Hay que incluirlo para
obtener Genes */
#include "in_eval2.h" /* parametros de campo */
/*-------------------------------------------------
-----------------*/
char *readFile(FILE *fp, char *line)
{
        char *c;

        do {
                c= fgets(line, 100, fp);
        } while ( c!= NULL && (line[0]== '#' ||
line[0] == 10));
        return c;
}
/*-------------------------------------------------
-----------------*/
In_eval(argc,argv)
int argc;
char *argv[];
{
        char In_evalfile[40],line[200];
        FILE *fopen(), *fp;
        double *hcapa, *rocapa;

        int i,j;
        char msg[40];

        /* Determinar los nombres de los archivos */

        if (argc < 2)
        {
                strcpy(In_evalfile,"campo");
        }
        else
        {
                sprintf(In_evalfile, "campo.%s",
argv[1]);
        }

        /* leer parametros de campo */
        if ((fp = fopen(In_evalfile, "r")) == NULL)
        {
                sprintf(msg, "In_eval: No puede
abrirse %s", In_evalfile);
                Error(msg);
        }

        readFile(fp, line);
        sscanf(line, "%d", &nt);

        readFile(fp, line);
        sscanf(line, "%d", &ntmute);

        readFile(fp, line);
        sscanf(line, "%d", &l);

        readFile(fp, line);
        sscanf(line, "%lf",&dt);

        hcapa = malloc(Genes * sizeof(double) );  /*
cantGenes = cantCapas */
        if (hcapa == NULL) {
                printf("in_eval: No hay memoria
para h\n");
                abort();
        }
        for (i=0; i < Genes; i++) {
                readFile(fp, line);
                sscanf(line,"%lf", &hcapa[i]);
        }

    /* leo la densidad de cada capa */
        rocapa = malloc(Genes * sizeof(double) );
/* cantGenes = cantCapas */
        if (rocapa == NULL) {
                printf("in_eval: No hay memoria
para rocapa\n");
                abort();
        }
        for (i=0; i < Genes; i++) {
                readFile(fp, line);
                sscanf(line,"%lf", &rocapa[i]);
        }

        readFile(fp, line);
        sscanf(line,"%lf", &xsou);

        readFile(fp, line);
        sscanf(line,"%d", &jsou);

        readFile(fp, line);
        sscanf(line, "%lf", &xjsou);
```

```
        readFile(fp, line);
        sscanf(line, "%lf", &xjsoup1);

        readFile(fp, line);
        sscanf(line, "%lf", &xrec);

        readFile(fp, line);
        sscanf(line, "%d", &jrec);

        readFile(fp, line);
        sscanf(line, "%lf", &xjrec);

        readFile(fp, line);
        sscanf(line, "%lf", &xjrecp1);

        kx = malloc((Genes+1) * sizeof(double) );
/* cantGenes = cantCapas */
        if (kx == NULL) {
                printf("in_eval: No hay memoria
para kx\n");
                abort();
        }
        for (i=0; i <= Genes; i++) {
                readFile(fp, line);
                sscanf(line,"%d", &kx[i]);
        }

        readFile(fp, line);
        sscanf(line, "%d", &keyb);

        readFile(fp, line);
        sscanf(line, "%d", &kdel);

        source = malloc((nt+1) * sizeof(double));
        if (source == NULL) {
                printf("in_eval: No hay memoria
para source\n");
                abort();
        }
```

```
        for (i=1; i <= nt; i++) {
                readFile(fp, line);
                sscanf(line, "%lf", &source[i]);
        }

        readFile(fp, line);
        sscanf(line, "%lf", &is);

        recmed = malloc((nt+1) * sizeof(double));
        if (recmed == NULL) {
                printf("in_eval: No hay memoria
para recmed\n");
                abort();
        }
        for (i=1; i <= nt; i++) {
                readFile(fp, line);
                sscanf(line, "%lf", &recmed[i]);
        }

        fclose(fp);

        /* convertimos rocapa en ronodo */
        ronodo = malloc((l+2) * sizeof(double));
        for (i=0; i < Genes; i++)
                for (j=kx[i]+1; j <= kx[i+1]; j++)
                        ronodo[j] = rocapa[i];
        ronodo[l+1]= rocapa[Genes-1];
        free(rocapa);

        /* convertimos hcapa en h */
        h = malloc((l+1) * sizeof(double));
        for (i=0; i < Genes; i++)
                for (j=kx[i]+1; j <= kx[i+1]; j++)
                        h[j] = hcapa[i];
        free(hcapa);

}

/** fin de archivo **/
```

C.2.21 Módulo INIT.C

```
/*
 *  archivo:        init.c
 *
 *  objetivo:       crear la poblacion inicial de
estructuras,
 *                  e inicilizar variables de
performance.
 *                  Esta rutina es llamada al inicio
de cada experimento.
 *
 */


#include "extern.h"
#include "dir.h"
#include "in_eval2.h"

Initialize()
{
        FILE *fp, *fopen();
        register int i, j;
        int status;                     /* indicates
fin de archivo en initfile    */
        char drive[MAXDRIVE], dir[MAXDIR],
file[MAXFILE], ext[MAXEXT];

        fnsplit(Minfile, drive, dir, file, ext);
        sprintf(Bestfile, "%s%d%s", file,
Experiment, ext);


        /* preparo para el nuevo experimento */
        Doneflag = 0;
        Curr_dump = 0;
        Bestsize = 0;
        /* setear proximo a mutar */
        if (M_rate < 1.0)
          Mu_next = ceil (log(Rand()) / log(1.0 -
M_rate));
        else
          Mu_next = 1;

        Trials = Gen = 0;
        Lost = Conv = 0;
        Plateau = 0;
        Spin = 0;
        Onsum = Offsum = 0.0;
        for (i=0; i<Windowsize; i++) Window[i] =
0.0;

        /* Inicializar poblacion */

        i = 0;                          /* estructura
actual */

        if (Initflag)                   /* tomar
algunas estructuras de Initfile */
        {
                if ((fp = fopen(Initfile, "r")) ==
NULL)
                {
                        char msg[40];
                        sprintf(msg,
"Initialize: no puede abrirse %s", Initfile);
```

```
                        Error(msg);
                }

                status = 1;
                if (Floatflag)
                {
                        for (j = 0; j < Genes &&
status != EOF; j++)
                        {
                                status =
fscanf(fp, "%lf", &Vector[j]);
                        }
                }
                else
                        status = fscanf(fp,
"%s", Bitstring);

                while (status != EOF && i <
Popsize)
                {
                        if (Floatflag)

        StringRep(Vector, Bitstring, Genes);

                        Pack(Bitstring,
New[i].Gene, Length);
                        New[i].Needs_evaluation
= 1;
                        i++;

                        /* leer la proxima
estructura */
                        if (Floatflag)
                                for (j = 0; j
< Genes && status != EOF; j++)

        status = fscanf(fp, "%lf", &Vector[j]);
                        else
                                status =
fscanf(fp, "%s", Bitstring);
                }
                fclose(fp);
        }

        /*****************************************
************/
        /* La semilla del generador de numeros
aleatorios se       */
        /* graba despues de la inicializacion de la
primera             */
        /* poblacion en cada experimento. El valor
grabado es          */
        /* usado como semilla en los experimentos
siguientes.         */
        /* La razon es permitir correr varios
experimentos con  */
        /* el mismo conjunto de parametros, y
comparar los        */
        /* resultados de corridas de un conjunto de
parametros          */
        /* con corridas realizadas con otro
conjunto.                   */
```

```
        /*****************************************
*************/

        if ( Experiment > 0 ) Seed = Initseed;

        for (; i < Popsize; i++)    /* inicializar el
resto de la poblacion al azar */
        {
                for (j = 0; j < Length; j++)
                {
                        if (Randint(0,1))
                                Bitstring[j] =
'1';
                        else
                                Bitstring[j] =
'0';
                }
                Pack(Bitstring , New[i].Gene ,
Length);
                New[i].Needs_evaluation = 1;
        }

        Initseed = Seed;

}

/** fin de archivo **/
```

C.2.22 Módulo INPUT.C

```
/*
 *  archivo:        input.c
 *
 *  purpose:        Inicializar nombres de archivos,
leer archivos de entrada
 *                  e inicializar variables para esta
corrida.
 *
 *                  Ver en init.c la inicializacion de
variables para cada experimento
 */

#include <alloc.h>
#include "extern.h"
#include "in_eval2.h"

Input(argc,argv)
int argc;
char *argv[];
{
        FILE *fopen(), *fp;

        int i;
        char msg[40];
        long clock;                     /* fecha
actual                                  */

        long time();
        char *ctime();
        int ilog2();

        /* establecer nombres de archivos */

        if (argc < 2)
        {
                strcpy(Infile,"in");
                strcpy(Outfile,"out");
                strcpy(Ckptfile,"ckpt");
                strcpy(Minfile,"min");
                strcpy(Logfile, "log");
                strcpy(Initfile, "init");
                strcpy(Schemafile, "schema");
                strcpy(Templatefile, "template");
        }
        else
        {
                sprintf(Infile, "in.%s", argv[1]);
                sprintf(Outfile, "out.%s",
argv[1]);
                sprintf(Ckptfile, "ckpt.%s",
argv[1]);
                sprintf(Minfile, "min.%s",
argv[1]);
                sprintf(Logfile, "log.%s",
argv[1]);
                sprintf(Initfile, "init.%s",
argv[1]);
                sprintf(Schemafile, "schema.%s",
argv[1]);
                sprintf(Templatefile,
"template.%s", argv[1]);
        }

        strcpy(Bestfile, Minfile);

        /* leer los parametros de infile */

        if ((fp = fopen(Infile, "r")) == NULL)
        {
                sprintf(msg, "Input: no puede
abrirse %s", Infile);
                Error(msg);
        }
        fscanf(fp,IN_FORMAT,IN_VARS);
        Seed = OrigSeed;
        fclose(fp);

        /* activar las opciones */
        for (i=0; Options[i] != '\0'; i++)
                Setflag(Options[i]);
        if (Displayflag)
                Traceflag = 0;

        /* Bytes es el tama¤o de cada cromosoma
empaquetado */
        Bytes = Length / CHARSIZE;
        if (Length % CHARSIZE) Bytes++;

        /* leer el archivo template. Solo se usa en
la representacion de punto
           flotante */
        if (Floatflag)
        {
                if ((fp = fopen(Templatefile,
"r")) == NULL)
                {
                        sprintf(msg, "Input: no
puede abrirse %s", Templatefile);
                        Error(msg);
                }
                fscanf(fp, "genes: %d ", &Genes);
                Gene = (GENESTRUCT *)
calloc((unsigned) Genes,
                        sizeof(GENESTRUCT));

                for (i=0; i<Genes; i++)
                {
                        fscanf(fp, " gene %*d");
                        fscanf(fp, " min: %lf",
&Gene[i].min);
                        fscanf(fp, " max: %lf",
&Gene[i].max);
                        fscanf(fp, " values:
%lu", &Gene[i].values);
                        fscanf(fp, " format:
%s", Gene[i].format);
                        Gene[i].bitlength =
ilog2(Gene[i].values);
                        Gene[i].incr =
(Gene[i].max - Gene[i].min) /

        (Gene[i].values - 1);
                }
                fclose(fp);
        }
        /* reservar memoria para las estructuras de
tama¤o variable */

        /* usado para representacion de punto
flotante del cromosoma */
        Vector = (double *) calloc((unsigned) Genes,
sizeof(double));

        /* usado para la representacion en cadena de
'0's y '1's */
        Bitstring = malloc((unsigned) (Length+1));
        Bitstring[Length] = '\0';

        if (Bitstring == NULL) {
                printf("input: No hay memoria para
Bitstring\n");
                abort();
        }
        /* arreglos de la poblacion */
        Old = (STRUCTURE *) calloc((unsigned)
Popsize, sizeof(STRUCTURE));
        New = (STRUCTURE *) calloc((unsigned)
Popsize, sizeof(STRUCTURE));

        for (i=0; i<Popsize; i++)
        {
                Old[i].Gene = malloc((unsigned)
Bytes);
                New[i].Gene = malloc((unsigned)
Bytes);

        }

        /* usado para computar Worst.
          (El peor de las ultimas Windowsize
generaciones) */
        if (Windowsize)
        Window = (double *) calloc((unsigned)
Windowsize, sizeof(double));

        /* usado para guardar las mejores
estructuras */
        if (Savesize)
         Bestset = (BESTSTRUCT *) calloc((unsigned)
Savesize, sizeof(BESTSTRUCT));

        for (i=0; i<Savesize; i++)
```

```
                    Bestset[i].Gene =
malloc((unsigned) Bytes);

          /* mostrar los parametros de entrada */
          if (Traceflag) printf(OUT_FORMAT, OUT_VARS);

          /* Sobreescribir el archivo de salida a
menos que sea un reinicio */
          if (!Restartflag)
          {
                    if ((fp = fopen(Outfile, "w")) ==
NULL)
                    {
                              sprintf(msg, "Input: No
puede abrirse %s", Outfile);
                              Error(msg);
                    }
                    fclose(fp);
          }

          /* guardar esta activacion en un log */
          if (Restartflag)
          {
                    if (Logflag)
                    {
                              if ((fp = fopen(Logfile,
"a")) == NULL)
                              {

          sprintf(msg,"Input: No puede abrirse %s",
Logfile);
                                        Error(msg);
                              }
                              fprintf(fp, "%s
Recomenz$ ", argv[0]);
                              time(&clock);
                              fprintf(fp, "%s",
ctime(&clock));
                              fclose(fp);
                    }
          }
          else
          {
                    if (Logflag)
                    {
                              if ((fp = fopen(Logfile,
"a")) == NULL)
                              {

          sprintf(msg,"Input: No puede abrirse %s",
Logfile);
                                        Error(msg);
                              }
                              fprintf(fp, "%s comenz$
", argv[0]);
                              time(&clock);
                              fprintf(fp, "%s",
ctime(&clock));
                              fclose(fp);
                    }
          }
}


int ilog2(n)
          unsigned long n;
{
          register int i;

          if (n <= 0)
          {
                    printf("La cantidad de valores es
%d, debe ser positiva!\n", n);
                    abort();
          }

          i = 0;
          while ((int) (n & 1) == 0)
          {
                    n >>= 1;
                    i++;
          }
          return(i);
}


Setflag(c)
char c;
{
          switch (c) {
          case 'a' :
                    Allflag = 1;
                    break;
          case 'b' :
                    Bestflag = 1;
                    break;
          case 'c' :
                    Collectflag = 1;
                    Convflag = 1;
                    break;
          case 'C' :
                    Collectflag = 1;
                    break;
          case 'd' :
                    Dumpflag = 1;
                    break;
          case 'D' :
                    Displayflag = 1;
                    break;
          case 'e' :
                    Eliteflag = 1;
                    break;
          case 'f' :
                    Floatflag = 1;
                    break;
          case 'g' :
                    Grayflag = 1;
                    break;
          case 'i' :
                    Initflag = 1;
                    break;
          case 'I' :
                    Interflag = 1;
                    Displayflag = 1;
                    break;
          case 'l' :
                    Logflag = 1;
                    break;
          case 'L' :
                    Lastflag = 1;
                    break;
          case 'M' :
                    Maxflag = 1;
                    break;
          case 'o' :
                    Onlnflag = 1;
                    break;
          case 'O' :
                    Offlnflag = 1;
                    break;
          case 'r' :
                    Restartflag = 1;
                    break;
          case 'R' :
                    Rankflag = 1;
                    break;
          case 's' :
                    Schemflag = 1;
                    break;
          case 't' :
                    Traceflag = 1;
                    break;
          }
}

/** fin de archivo **/
```

C.2.23 Módulo MEASURE.C

```
/*
 *  archivo:        measure.c
 *
 *  objetivo:       calcular mediciones de performance
y adjuntarlas al archivo
 *                  output.
 *
 */

#include "extern.h"

#define DISPMEAS 5

Measure()
{
          double New_worst();
          FILE *fp, *fopen();
          register int i;
          register int w;
          register double performance;
          int j;

          for (i=0; i<Popsize; i++)
          {
                    /* estadisticas de la poblacion
actual */
                    performance = New[i].Perf;
                    if (i>0)
                    {
                              Ave_current_perf +=
performance;
                              if (BETTER(performance,
Best_current_perf))
                              {

          Best_current_perf = performance;
                                        Best_guy = i;
                              }
                              if
(BETTER(Worst_current_perf, performance))

          Worst_current_perf = performance;
                    }
                    else
                    {
                              Best_current_perf =
performance;
                              Worst_current_perf =
performance;
```

```
                                Ave_current_perf =
performance;
                                Best_guy = 0;
                        }

        }

        Ave_current_perf /= Popsize;

        /* actualizar Worst */
        if (Windowsize)
        {
                        /* Worst = worst en las ultimas
(Windowsize) generaciones */
                        w = Gen % Windowsize;
                        Window[w] = New_worst();
                        Worst = Window[0];
                        for (i=1; i < Windowsize; i++)
                                if (BETTER(Worst,
Window[i])) Worst = Window[i];
        }
        else
                        if (BETTER(Worst,
Worst_current_perf))
                                Worst = New_worst();

        /* actualizar medidas de performance global
*/
        Online = Onsum / Trials;
        Offline = Offsum / Trials;

        if (Traceflag)
        {
                        printf("     Gen %d     Evals
%d\n",Gen,Trials);
                        if (Onlnflag) printf("     Online
%e\n", Online);
                        if (Offlnflag) printf("
Offline %e\n", Offline);
                        }

        if (Displayflag)
        {
                        static firstflag = 1;
                        if (firstflag)
                        {
                                firstflag = 0;
                                move(DISPMEAS - 1,0);
                                printw(" Gens Evals Perd
");
                                printw("Conv  Sesgo
Online      ");
                                printw("Offline
Mejor     Promedio");
                        }

                        move(DISPMEAS,0);
                        clrtoeol();
                        Converge();
                        printw("%4d %6d %4d ",
                                Gen, Trials, Lost);
                        printw("%4d  %5.3f %11.4f  ",
                                Conv, Bias, Online);
                        printw("%11.4f  %11.6f  %11.6f",
                                Offline, Best,
Ave_current_perf);

                        move(DISPMEAS+2,0);
                        clrtoeol();
                        printw(" Mejor estructura actual:
%3d  ", Best_guy);
                        printw("Performance: %0.6f ",
New[Best_guy].Perf);
                        move(DISPMEAS+3,0);
                        clrtoeol();
                        Unpack(New[Best_guy].Gene,
Bitstring, Length);
                        if (Floatflag)
                        {
                                FloatRep(Bitstring,
Vector, Genes);
                                for (j=0; j<Genes; j++)

        printw(Gene[j].format, Vector[j]);
                        }
                        else
                        {
                                printw("%s", Bitstring);
                        }
                        refresh();
        }

        if ( Interval && Collectflag && ((Trials >=
Plateau) || Doneflag))
        {
                        /* agregar mediciones al archivo
output */
                        Converge();
                        fp = fopen(Outfile, "a");
                        fprintf(fp,OUT_F2, OUT_V2);

                        fclose(fp);
                        Plateau =
(Trials/Interval)*Interval + Interval;
        }
```

```
        if (Logflag  && (Spin >= Maxspin))
        {
                        fp = fopen(Logfile, "a");
                        fprintf(fp, "Experimento %1d  ",
Experiment);
                        fprintf(fp, "SPINNING en la Gen
%1d, ",Gen);
                        fprintf(fp, "despu,s de  %1d
Evaluaciones\n", Trials);
                        fclose(fp);
        }

        if ( Interflag && (Spin >= Maxspin))
        {
                        move(22,0);
                        clrtoeol();
                        printw("SPINNING en la Gen %1d,
",Gen);
                        printw("despu,s de %1d
Evaluaciones\n", Trials);
                        refresh();
        }

}


double New_worst()
{
        double delta;

        /* Devolver un valor un poquito peor que
than Worst_current_perf */
/**** un poquito peor porque si comparo contra el peor
la distancia daria 0 ****/

        if (Maxflag)
                        delta = 1.0e-4;
        else
                        delta = -1.0e-4;

        if (Worst_current_perf == 0.0) return (-
delta);

        if (Worst_current_perf > 0.0)
                        return (Worst_current_perf*(1.0 -
delta));

        return (Worst_current_perf*(1.0 + delta));
}



static char BIT[CHARSIZE] ={ '\200', '\100', '\040',
'\020',
                                '\010', '\004', '\002',
'\001'};


Converge()                              /* medir la
convergencia de la poblacion  */
{
        register int i,j;
        register int ones;  /* cantidad de unos en
una posicion        dada*/
        int focus;                      /* indice del
byte actual                     */
        int bit;                        /* indice del bit actual
                                        */
        FILE *fp, *fopen();

        Bias = 0.0;
        Lost = Conv = 0;
        if (!Convflag) return;

        for (j = 0; j < Length; j++)
        {
                        focus = j / CHARSIZE;
                        bit = j % CHARSIZE;
                        ones = 0;
                        for (i=0; i < Popsize; i++)
                                        ones +=
((New[i].Gene[focus] & BIT[bit]) != 0);
                        Lost += (ones == 0) || (ones ==
Popsize);
                        Conv += (ones <= FEW) || (ones >=
Popsize - FEW);
                        Bias += (ones > Popsize/2) ? ones
: (Popsize - ones);
        }

        Bias /= (Popsize*Length);

        if (Logflag && (Lost==Length))
        {
                        fp = fopen(Logfile, "a");
                        fprintf(fp, "CONVERGIO en Gen %1d,
",Gen);
                        fprintf(fp, "despu,s de %1d
Evaluaciones\n", Trials);
                        fclose(fp);
        }

}


/** fin de archivo **/
```

C.2.24 Módulo MUTATE.C

```
/*
 *  archivo:        mutate.c
 *
 *  purpose:        Mutar la poblacion
 *
 *        La variable global Mu_next indica el proximo
bit a mutar tratando
 *        la poblacion como un arreglo lineal de bits.
 */

#include "extern.h"


Mutate()
{
          static int bits;     /* cantidad de bits
total en la poblacion */
          register int i;               /* indice del
individuo a mutar */
          register int j;      /* bit a mutar dentro de
la estructura */
          register char k;      /* alelo al azar */
          register int open;  /* indica si el
cromosoma ya fue desempaquetado */
          static int firstflag = 1;

          if (firstflag)
          {
              bits = Gapsize*Popsize*Length + 0.5;
/**** 0.5 para redondear ****/
              firstflag = 0;
          }

          if (M_rate > 0.0)
          {
                    open = -1;
/**** Mu_next se inicializa al ppio. del pgm. o en la
generacion anterior ****/
                    while (Mu_next<bits)
                    {
                              i = Mu_next/ Length;
          /* estructura a mutar */
                              j = Mu_next % Length;
          /* bit a mutar dentro de la estructura */
                              if (open != i)    /* hay
que desempaquetar la estructura i */
                              {
                                        Unpack
(New[i].Gene , Bitstring, Length);
                                        open = i;
                              }

                              /* elegir un alelo al
azar */
                              if (Randint(0,1))
                                        k = '1';
                              else
                                        k = '0';

                              if (k != Bitstring[j])
/* es una mutacion verdadera */
                              {
                                        Bitstring[j] =
k;

          New[i].Needs_evaluation = 1;
                              }
                              if
(New[i].Needs_evaluation)
                                        Pack (
Bitstring , New[i].Gene , Length);

                              /* calcular proximo bit
a mutar */
                              if (M_rate < 1.0)
                                        Mu_next +=
ceil (log(Rand()) / log(1.0 - M_rate));
                              else
                                        Mu_next += 1;
                    }
                    /* ajustar Mu_next para la proxima
generacion */
                    Mu_next -= bits;
          }
}


/** fin de archivo **/
```

C.2.25 Módulo PRESION.C

```
/*
 *  archivo:        presion.c
 *
 *  objetivo:   Calcular las presiones generadas por
un juego de velocidades.
 *
 */
#include <alloc.h>
#include <math.h>
#include <stdio.h>
#include <assert.h>
#include <stdlib.h>
#include "presion.h"
#include "in_eval2.h"
/*-------------------------------------------------------
----------------*/
double una_presion(unm1, un, unp1, paso, constk,
velnodo)
          double *unm1, *un, *unp1, *constk, *velnodo;
          int paso;
{
          double aux1, aux2;
          int k;

          assert(heapcheck()==_HEAPOK);

          /* condicion de bordes */
          unp1[1]= borde_sup(unm1[1], un[1], un[2],
constk[1]);
          unp1[l+1]= borde_inf( unm1[l+1], un[l],
un[l+1], constk[l+1]);

          /* agregar la contribucion de la fuente a
cada borde */
          /* primero al superior */
          unp1[1]+= fuente_en_borde(paso, 1);
          unp1[1]/= 1 + constk[1];
          /* y ahora al inferior */
          unp1[l+1]+= fuente_en_borde(paso, l+1);
          unp1[l+1]/= 1 + constk[l+1];

          /* calculo de la presion de los nodos
interiores */
          for (k=2; k <= l; k++) {
                    unp1[k]= 2 * un[k] - unm1[k] + 2 *
constk[k] * pow(dt, 2) *
                    ( (un[k+1] - un[k]) / ( h[k] *
ronodo[k] ) -
                    ( un[k] - un[k-1] ) / ( h[k-1] *
ronodo[k-1] ));
                    /* agregar la contribucion de la
fuente */
                    unp1[k]+= fuente_interior( paso,
k, constk[k], velnodo);
          }

          /* Interpolacion lineal de las presiones en
los dos nodos que rodean al
                    receptor */
          aux1= (xjrecp1 - xrec) / h[jrec] *
unp1[jrec];
          aux2= (xrec - xjrec) / h[jrec] *
unp1[jrec+1];

          assert(heapcheck()==_HEAPOK);
          return (aux1 + aux2);
}

/*-------------------------------------------------------
----------------*/
void presiones(velocidad, capas, presion, nt)
          /* velocidades por cada capa, parametro de
entrada */
          double *velocidad;
          int capas;  /* cantidad de capas */

          /* presiones calculadas a partir de las
velocidades */
          double *presion;
          int nt;             /* cantidad de
intervalos en el tiempo */
{

          int i;              /* auxiliar */
          double *aux;  /* puntero auxiliar para hacer
swap */
          double *unm1; /* presiones en el paso n-1 */
          double *un; /* presiones en el paso n   */
          double *unp1; /* presiones en el paso n+1 */
          double *velnodo;  /* velocidades en nodos */
          double *constk; /* constantes auxiliares */

          unm1= malloc((l+2)*sizeof(double)); /*
presiones en el paso n-1 */
          un= malloc((l+2)*sizeof(double));   /*
presiones en el paso n   */
          unp1= malloc((l+2)*sizeof(double)); /*
presiones en el paso n+1 */
          velnodo= malloc((l+2)*sizeof(double));  /*
velocidades en nodos */
          constk= malloc((l+2)*sizeof(double)); /*
constantes auxiliares */
          vel_en_nodos(velocidad, capas, velnodo);
```

```
          assert(heapcheck()==_HEAPOK);

          for ( i=1; i <= (l+1); i++) {
                    unm1[i]= 0;
                    un[i]= 0;
          }

          /* Condiciones iniciales */
          presion[1]= 0;
          presion[2]= 0;

          /* Tiempo de espera desde que comienza la
explosion hasta que se comienza
             a escuchar (mute) */
          for ( i= 3; i <= ntmute; i++)
                    presion[i]= 0;

          /* constantes auxiliares */
          constk[1]= velnodo[1] * dt / h[1];
          constk[l+1]= velnodo[l] * dt / h[l];
          for ( i = 2; i <= l; i++)
                    constk[i]= ( pow(velnodo[i], 2) *
pow(velnodo[i-1], 2) * ronodo[i] * ronodo[i-1]) /

          (h[i-1] * pow(velnodo[i], 2) * ronodo[i] +
h[i] * pow(velnodo[i-1], 2) * ronodo[i-1]);

          /* calcula la presion en el paso i */
          for ( i= max(ntmute+1,3); i <= nt; i++) {
                    presion[i]= una_presion(unm1, un,
unp1, i, constk, velnodo);
                    aux= unm1;    /* lo guardo para
despues */
                    unm1= un;     /* avanza un paso en
el tiempo */
                    un= unp1;     /*    "    "    "    "
"    "    */
                    unp1= aux;    /* no interesan los
valores pero si el espacio de memoria */
          }

          free(unm1);
          free(un);
          free(unp1);
          free(velnodo);
          free(constk);
          /* el resultado va en 'presion' */

          assert(heapcheck()==_HEAPOK);
}
/*-------------------------------------------------
-----------------*/
void vel_en_nodos(velocidad, capas, velnodo)
double *velocidad;
int capas;
double *velnodo;
{
          int c, nodo;
          for (c=0; c <capas; c++)
                    for (nodo=kx[c]+1; nodo <=
kx[c+1]; nodo++)
                              velnodo[nodo] =
velocidad[c];
          velnodo[l+1]= velocidad[capas-1];
}
/*-------------------------------------------------
-----------------*/
double borde_sup(unm1, unx1, unx2, cte)
/* Calcula la presion en el borde superior (sin
fuente) */
double unm1;
double unx1;
double unx2;
double cte;
{
          double presion;
          switch (keyb) {
                    case 0: /* Condicion de bordes de
Dirichlet (superficie libre) */
                                        presion = 0;
                                        break;
                    case 1: /* Condicion de bordes
absorbente */
                                        presion = 2 *
unx1 - unm1 + 2 * pow(cte, 2) * (unx2 - unx1) +

          cte * unm1;
                                        break;
                    case 2: /* Condicion de bordes de
Neumann  */
                                        presion = 2 *
unx1 - unm1 + 2 * pow(cte, 2) * (unx2 - unx1);
                                        break;
          }
          return presion;
}
/*-------------------------------------------------
-----------------*/
double borde_inf(unm1, unx1, unx2, cte)
/* Calcula la presion en el borde inferior (sin
fuente) */
double unm1; /* presion en el paso anterior */
double unx1; /* presion en el anteultimo nodo */
double unx2; /* presion en el ultimo nodo */
double cte;  /* constante ad hoc */
{
  double presion;
  presion = 2 * unx2 - unm1 + 2 * pow(cte, 2) * (unx1
- unx2) +
          cte * unm1;
  return presion;
}
/*-------------------------------------------------
-----------------*/
/* Las tres rutinas de fuente podrian unificarse */
double fuente_en_borde(paso, nodo)
/* Contribucion de la fuente en el borde indicado por
nodo en el paso 'paso' */
int paso;
int nodo;
{
          double eval, presion;
          switch (kdel) {
                    case 0: /* delta de Dirac en xsou
* source[n] */
                                        if (1 == jsou)
                                                  eval =
(xjsoup1 - xsou) / h[jsou];
                                        else if (1 == jsou+1)
                                                            eval
= (xsou - xjsou) / h[jsou];
                                                  else
                                                            eval
= 0;
                                        presion = 2 * pow(dt, 2)
* source[paso] * eval / h[nodo];
                                        break;

                    case 1: /* derivada del delta de
Dirac en
                                                            .5 * (xsou
+ xsou + h) * source[n] */
                                        presion = 2 * pow(dt, 2)
* source[paso] / pow(h[nodo], 2);
                                        if (1 == jsou) {
                                                  /* no hacemos
nada */
                                        }
                                        else if (1 == jsou + 1)
{
                                                  presion *= -
1;
                                                  }
                                                  else {

          presion = 0;
                                                  }
                                        break;

                    case 2: /* 2 delta de Dirac en
xsou * source[n] */
                                        presion = 2 * pow(dt, 2)
* source[paso] * h[nodo] / 2.0;
                                        if (1 == jsou - is) {
                                                  /* no hacemos
nada */
                                        }
                                        else if (1 == jsou + is)
{
                                                  presion *= -
1;
                                                  }
                                                  else {

          presion = 0;
                                                  }
                                        break;
          }
          return presion;
}

/*-------------------------------------------------
-----------------*/
double fuente_interior(paso, nodo, cte, velnodo)
/* Contribucion de la fuente en el un nodo interior */
int paso;
int nodo;
double cte; /* constante auxiliar */
double *velnodo;
{
          double eval, presion;
          switch (kdel) {
                    case 0: /* delta de Dirac en xsou
* source[n] */

                                        if (nodo == jsou)
                                                  eval =
(xjsoup1 - xsou) / h[jsou];
                                        else if (nodo == jsou+1)
                                                            eval
= (xsou - xjsou) / h[jsou];
                                                  else
                                                            eval
= 0;

                                        presion = 2 * pow(dt, 2)
* cte * source[paso] * eval /

(pow(velnodo[jsou], 2) * ronodo[jsou]);
                                        break;

                    case 1: /* derivada del delta de
Dirac en
                                                            .5 * (xsou
+ xsou + h) * source[n] */

                                        if (nodo == jsou) {
```

```
                                        presion = 2 *
pow(dt, 2) * cte * source[paso] /
(pow(h[jsou], 2) * pow(velnodo[jsou], 2) *
ronodo[jsou]);
                                }
                                else if (nodo == jsou +
1) {
                                        presion = 2 *
pow(dt, 2) * cte * source[paso] /
(pow(h[jsou], 2) * pow(velnodo[jsou], 2) *
ronodo[jsou]);
                                        presion *= -
1;
                                }
                                else {

        presion = 0;
                                }
                                break;

                        case 2: /* 2 delta de Dirac en
xsou * source[n] */
```

```
                                if (nodo == jsou - is) {
                                        presion = 2 *
pow(dt, 2) * cte * source[paso] *
(pow(velnodo[jsou], 2) * ronodo[jsou]) / 2.0;
                                }
                                else if (nodo == jsou +
is) {
                                        presion = 2 *
pow(dt, 2) * cte * source[paso] *
(pow(velnodo[jsou], 2) * ronodo[jsou]) / 2.0;
                                        presion *= -
1;
                                }
                                else {

        presion = 0;
                                }
                                break;
        }
        return presion;
}

/*** fin de archivo ***/
```

C.2.26 Módulo RESTART.C

```
/*
 *  archivo:        restart.c
 *
 *  objetivo:       correr AG a partir de una corrida
anterior
 *
 */

#include <dir.h>
#include "extern.h"

extern void Readbest();


Restart()
{
        FILE *fp, *fopen();
        int i;
        char msg[40];
        char drive[MAXDRIVE], dir[MAXDIR],
file[MAXFILE], ext[MAXEXT];

        fp = fopen(Ckptfile, "r");
        if (fp == NULL)
        {
                sprintf(msg,"Restart: Ckptfile %s
no se encuentra",
                        Ckptfile);
                Error(msg);
        }

        fscanf(fp, "Experimentos %d ", &Experiment);
        fscanf(fp, "On_line total   %lf ",
&Totonline);
        fscanf(fp, "Off_line total   %lf ",
&Totoffline);
        fscanf(fp, "Generaci¢n %d ", &Gen);
        fscanf(fp, "Performance On_line  %lf ",
&Onsum);
```

```
        fscanf(fp, "Performance Off_line %lf ",
&Offsum);
        fscanf(fp, "Evaluaciones  %d ", &Trials);
        fscanf(fp, "Pr¢xima estad¡stica %d ",
&Plateau);
        fscanf(fp, "Mejor %lf ", &Best);
        fscanf(fp, "Peor   %lf ", &Worst);
        fscanf(fp, "Cantidad de generaciones desde
la £ltima evaluaci¢n %d ", &Spin);
        fscanf(fp, "Ultimo vuelco %d ", &Curr_dump);
        fscanf(fp, "Mu_prox %d ", &Mu_next);
        fscanf(fp, "Semilla Aleatoria %lu ", &Seed);
        fscanf(fp, "Semilla Inicializadora %lu ",
&Initseed);

        fscanf(fp, " Window ");
        for (i=0; i<Windowsize; i++) fscanf(fp,
"%lf", &Window[i]);

        for (i=0; i<Popsize; i++)
        {
                fscanf(fp, "%s", Bitstring);
                fscanf(fp, " %lf ", &New[i].Perf);
                Pack(Bitstring, New[i].Gene,
Length);
                fscanf(fp, " %d ",
&New[i].Needs_evaluation);
        }
        fclose(fp);

        fnsplit(Minfile, drive, dir, file, ext);
        sprintf(Bestfile, "%s%d%s", file,
Experiment, ext);

        Readbest();

}

/** fin de archivo **/
```

C.2.27 Módulo SELECT.C

```
/*
 *  archivo:        select.c
 *
 *  objetivo:       seleccionar los individuos para la
nueva generacion
 *
 */
#include <alloc.h>
#include "extern.h"

Select()
{
        static firstflag = 1;           /* indica
primera ejecucion               */
        static int *sample; /* apunta a las
estructuras seleccionadas       */
        double expected;    /* numero especrado de
hijos                   */
        double factor;                  /*
normalizador para valor esperado        */
        double perf;                    /* mejor
performance siguiente (para ranking)                */
        double ptr;                     /* determina
la distribucion de la parte fraccionaria */
        double rank_max;    /* max numero de hijos
(en ranking) */
        double sum;
        int best;           /* indice a la mejor
estructura siguiente                    */
        register int i;
        register int j;
        register int k;
```

```
        register int temp;  /* usado para
intercambiar punteros                   */

        if (firstflag)
        {
                sample = (int *) calloc((unsigned)
Popsize, sizeof(int));
                firstflag = 0;
        }

        if (Rankflag)
        {
                /* Asignar a cada estructura su
nro de orden (rank) dentro de la poblacion. */
                /* rank = Popsize-1 para el mejor,
rank = 0 para el peor           */
                /**** Se usa el campo
Needs_evaluation para guardar el valor de rank ****/

                /* limpiar el campo donde se
guarda el valor de rank */
                for (i=0; i<Popsize; i++)
                        Old[i].Needs_evaluation
= 0;

                for (i=0; i < Popsize-1; i++)
                {
                        /* buscar la mejor i-
esima estructura */
                        best = -1;
                        perf = 0.0;
                        for (j=0; j<Popsize;
j++)
```

```
                                {
                                        if
(Old[j].Needs_evaluation == 0 &&

(best == -1 || BETTER(Old[j].Perf,perf)))
                                        {
                                                perf
= Old[j].Perf;
                                                best
= j;
                                        }
                                }
                                /* ponerle el orden a la
estructura */

          Old[best].Needs_evaluation = Popsize -1 - i;
                    }
                    /* normalizador para las
probabilidades de seleccion por ranking */
                    rank_max = 2.0 - Rank_min;
                    /**** Calcula el tamaño del slot
patron ****/
                    factor = (rank_max - Rank_min) /
(double) (Popsize -1);
          }
          else
          {
                    /* normalizador para las
probabilidades de seleccion proporcional */
                    /**** Calcula el tamaño del slot
patron ****/
                    factor = Maxflag ?
1.0/(Ave_current_perf - Worst) :
                                                1.0/(Worst
- Ave_current_perf);
          }

          /* Algoritmo de muestreo estocastico de
James E. Baker */

          k=0;                /* indice de la proxima
estructura a elegir */

          ptr = Rand();   /* girar la ruleta una vez
*/

          for (sum=i=0; i < Popsize; i++)
          {
                    if (Rankflag)
                    {
                                /**** Mapea los
individuos en forma lineal entre
                                  Rank_min y Rank_max
****/
                                expected = Rank_min +
Old[i].Needs_evaluation * factor;
                    }
                    else
                    {
                                /**** Mapea en forma
proporcional a la distancia con
                                  el peor individuo
****/
                                if (Maxflag) {
                                        if
(Old[i].Perf > Worst)
                                          expected =
(Old[i].Perf - Worst) * factor;
                                        else expected
= 0.0;
                                }
                                else {
                                        if
(Old[i].Perf < Worst)
                                          expected =
(Worst - Old[i].Perf) * factor;
                                        else expected
= 0.0;
                                }
                    }
/**** Asigna entre floor y ceil de la cantidad
esperada de hijos ****/
                    for (sum += expected; sum > ptr;
ptr++){
                      sample[k++] = i;
                    }
          }

          if (k != Popsize) {
            printf("select: Se seleccionaron %d
individuos en lugar de %d\n", k, Popsize);
            abort();
          }

          /* mezclar al azar los punteros de las
nuevas estructuras */
/**** Pensar en que pasa si un individuo recibe mas de
una copia en la
            seleccion. Luego, en la cruza, el padre y
la madre serian muy probablemente
            el mismo. Por eso se mezcla! ****/
          for (i=0; i<Popsize; i++)
          {
                    j = Randint(i,Popsize-1);
                    temp = sample[j];
                    sample[j] = sample[i];
                    sample[i] = temp;
          }

          if (Gapsize<1.0)              /* Salto
generacional */
                    Gap(sample);

          /* finalmente, formar la nueva poblacion */
          for (i=0; i<Popsize; i++)
          {
                    k = sample[i];
                    for (j=0; j<Bytes; j++)
                    {
                                New[i].Gene[j] =
Old[k].Gene[j];
                    }
                    New[i].Perf = Old[k].Perf;
                    New[i].Needs_evaluation = 0;
          }
}

/* Elegir sobrevivientes de la vieja poblacion
uniformemente sin reposicion */

Gap(sample)
int sample[];
/**** (Gap == 0) => no hay seleccion
                              no hay cruza
            Se destruye una porcion de la poblacion
seleccionada y se reemplaza
            por indiv. de la poblacion ORIGINAL sin
tener en cuenta su aptitud ****/
{
          static firstflag = 1;
          static int *survivors;        /* permutacion
al azar de 0 .. Popsize-1 */
          register int i,j;
          int temp;             /* para swapear
                                   */

          if (firstflag)
          {
                    survivors = (int *)
calloc((unsigned) Popsize, sizeof(int));
                    firstflag = 0;
          }

          /* Mezclar uniformemente */
          for (j=0; j<Popsize; j++) survivors[j]=j;
          for (j=0; j<Popsize; j++)
          {
                    i = Randint(j, Popsize-1);
                    temp = survivors[i];
                    survivors[i] = survivors[j];
                    survivors[j] = temp;
          }

          /* nuevos sobrevivientes elegidos */
          for (i=Gapsize*Popsize; i<Popsize; i++)
                    sample[i] = survivors[i];
}
/*** fin de archivo ***/
```

C.3 Programa AG2

C.3.1 Módulo DEFINE.H

```
/*
 *  archivo:        define.h
 *
 *  objetivo:    definiciones globales
 */

#include <math.h>
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include "format.h"
#include <conio.h>
```

```
#define clrtoeol        clreol
#define refresh()       fflush(stdout)
#define printw          printf
#define scanw           scanf
#define getw            gets

/********************** CONSTANTES
***************************/

#define CHARSIZE 8

/* usados en generador aleatorio de nfmeros */
```

```
#define MASK 2147483647
#define PRIME 65539
#define SCALE 0.4656612875e-9



/********************** TIPOS
*****************************/

/* tipo para almacenar cada individuo de la poblaci¢n
*/
typedef struct {
          char *Gene;
          double Perf;
          int Needs_evaluation;
} STRUCTURE;

/* tipo para almacenar las mejores estructuras */
typedef struct {
          char *Gene;
          double Perf;
          int Gen;
          int Trials;
} BESTSTRUCT;

/* registro para interpretar el formato empaquetado de
acuerdo con el template definido */
typedef struct {
          double min;
          double max;
          unsigned long values;
          char format[16];
          double incr;
          int bitlength;
} GENESTRUCT;

/********************** MACROS
*****************************/

/* Comparaci¢n de dos valores de performance */
#define BETTER(X,Y)      (Maxflag ? (X) > (Y) : (X) <
(Y))

/* Un alelo converge si todos los individuos - FEW
/* tiene el mismo valor en esa posici¢n.  */
#define FEW              (Popsize/20)

/*************************************************
**********/
/*       Rand obtiene un n£mero de una serie psuedo
aleatoria */
/*       en el intervalo [0,1).

          */
/*                    Randint devuelve un entero en el
intervalo [low,high]          */
/*************************************************
**********/

#define Rand()   (( Seed = ( (Seed * PRIME) & MASK) ) *
SCALE )

#define Randint(low,high) ( (int) (low + (high-low+1)
* Rand()))

/** fin de archivo **/
```

C.3.2 Módulo EXTERN.H

```
/*
 *  archivo:        extern.h
 *
 *  objetivo:       declaraciones externas.
 */

#include "define.h"

/* El archivo in especifica estos par metros */

extern int    Totalexperiments;         /* cantidad de
experimentos                    */
extern int    Totaltrials;    /* evaluaciones por
experimentos                    */
extern int    Popsize;                  /* tama¤o de
la poblaci¢n */
extern int    Length;                   /* longitud en
bits de la estructura */
extern double  C_rate;                  /*
probabilidad de cruza*/
extern double  M_rate;                  /*
probabilidad de mutacion */
extern double  Gapsize;                 /* fracci¢n de
la poblaci¢n que ser  reemplazada cada generaci¢n */
extern int    Windowsize;     /* usado para actualizar
la peor performance */
extern int    Interval;                 /* cantidad de
evaluaciones entre estad¡sticas         */
extern int    Savesize;                 /* n£mero de
estructuras que se guardan en minfile   */
extern int    Maxspin;                  /* max gens
sin evaluaciones                */
extern int    Dump_freq;      /* gens entre
checkpoints                     */
extern int    Num_dumps;      /* cantidad de archivos
de checkpoint que se guardan  */
extern char   Options[];      /* opciones
                                */
extern unsigned long Seed;    /* semilla para el
generador aleatorio */
extern unsigned long OrigSeed;          /* semilla
inicial   */



/*****************************************  de
global.h   ******/

/*        Variables globales. */


/* Nombres de archivos */
extern char   Bestfile[];     /* archivo con las
mejores estructuras */
extern char   Ckptfile[];     /* archivo de check
point                           */
extern int    Curr_dump;      /* sufijo del £ltimo
dumpfile  */
extern char   Dumpfile[];     /* dumpfile actual (si
hay m s de uno)     */
extern char   Initfile[];     /* archivo de
estructuras iniciales                   */
extern char   Infile[];                 /* parametros
globales                        */
extern char   Logfile[];      /* log de inicio y
reinicio            */
extern char   Minfile[];      /* prefijo de Bestfile
(se guarda un bestfile por cada experimento       */
extern char   Outfile[];      /* salida "en bruto" de
estad¡sticas                                    */
extern char   Schemafile[];   /* archivo para guardar
la historia de un esquema     */
extern char   Templatefile[]; /* archivo que describe
la interpretacion de los genes          */

extern char   *Bitstring;     /* representaci¢n en
cadena de '1's y '0's         */
extern double *Vector;                  /*
representaci¢n de punto flotante */
extern int Genes;             /* Cantidad de genes
interpretados                   */
extern GENESTRUCT *Gene;      /* puntero a unidades de
template  */
extern STRUCTURE  *Old;                 /* puntero a
la generaci¢n anterior        */
extern STRUCTURE  *New;                 /* puntero a
la nueva generaci¢n */
extern BESTSTRUCT *Bestset;   /* conjunto de las
mejores estructuras           */

/* datos estad¡sticos y variables auxiliares */
extern double  Ave_current_perf;/* aptitud promedio de
la generaci¢n actual          */
extern double  Best;                    /* la mejor
aptitud hasta ahora */
extern double  Best_current_perf;/* la mejor aptitud
en la generaci¢n actual       */
extern int    Best_guy;                 /* numero de
estructura con best_current_perf                */
extern int    Bestsize;                 /* cantidad de
mejores estructuras grabadas */
extern double  Bias;                    /* promedio de
la dominaci¢n de alelos               */
extern int    Bytes;                    /* longitud en
bytes de la estructura empaquetada */
extern int    Conv;           /* n£mero de genes que
convergieron parcialmente     */
extern char   Doneflag;                 /* se setea
cuando se cumple la condici¢n de parada */
extern int    Experiment;     /* contador de
experimentos                            */
extern int    Gen;            /* contador         de
generaciones                  */
extern unsigned long    Initseed; /* semilla inicial
del experimento     */
extern int    Lost;           /* cantidad de
posiciones que convergieron totalmente */
extern int    Mu_next;                  /* pr¢xima
posici¢n a mutar              */
extern double  Offline;                 /* performance
offline                       */
extern double  Offsum;                  /* acumulador
para la performance offline   */
extern double  Online;                  /* performance
online                        */
extern double  Onsum;                   /* acumulador
para performance online       */
extern int    Plateau;                  /* contador de
evaluaciones para la siguiente salida   */
extern double Rank_min;                 /* minima tasa
de muestreo para seleccion por ranking  */
extern double  Totbest;                 /* mejor de
todos los experimentos                          */
extern double  Totoffline;    /* offline de todos los
experimentos                  */
extern double  Totonline;     /* online  de todos los
experimentos                  */
extern int    Trials;                   /* contador de
evaluaciones */
```

```
extern double  *Window;                 /* cola
circular de los peores de las £ltimas generaciones
        */
extern double  Worst;                   /* peor
performance hasta ahora       */
extern double  Worst_current_perf;/* peor perf en la
generaci¢n actual */
extern int          Spin;               /* cantidad de
generaciones desde la £ltima evaluaci¢n */

/* flags seteados de acuerdo a Options */
extern char   Allflag;                  /*  evaluar
todas las estructuras                   */
extern char   Bestflag;                 /*  imprimir
el mejor valor      final     */
extern char   Collectflag;    /*  llevar estad¡sticas
de performance */
extern char   Convflag;                 /*  llevar
estadisticas de convergencia */
extern char   Displayflag;    /*  mostrar estadisticas
de cada generacion */
extern char   Dumpflag;                 /*  hacer un
vuelco despues de cada evalueacion              */
extern char   Eliteflag;      /*  usar la estrategia
de seleccion elitista         */
extern char   Floatflag;      /*  convertir cadenas a
punto flotante      */
```

```
extern char   Grayflag;                 /*  usar gray
code                          */
extern char   Initflag;                 /*  leer
estructuras iniciales de un archivo             */
extern char   Interflag;      /*  modo interactivo
                              */
extern char   Lastflag;                 /*  vuelco de
la ultima generacion */
extern char   Logflag;                  /*  log de
inicios y reinicios           */
extern char   Maxflag;                  /*  maximizar
en lugar de minimizar         */
extern char   Offlnflag;      /*  imprimir medida de
offline final                 */
extern char   Onlnflag;                 /*  imprimir
medida de online final                  */
extern char   Rankflag;                 /*  usar
seleccion por ranking                   */
extern char   Restartflag;    /*  recomenzar una
corrida                       */
extern char   Schemflag;      /*  seguir la historia
de un esquema       */
extern char   Traceflag;      /*  mostrar las rutinas
que se ejecutan               */

/** fin de archivo **/
```

C.3.3 Módulo FORMAT.H

```
/*
 *  archivo:        format.h
 *
 *  objetivo:       especificar los formatos para
archivos de entrada y salida
 */

/* el archivo in se lee de acuerdo a  IN_FORMAT e
IN_VARS */

#define IN_FORMAT " \
                                        Exp. = %d \
                        Tot. Eval. = %d \
                                        Pob. = %d \
            Long. cromosoma = %d \
                        Prob. cruza = %lf \
             Prob. mutaci¢n = %lf \
             Salto generac. = %lf \
            Ventana scaling = %d \
            Interv. reporte = %d \
           Estruc. guardadas = %d \
   Max gens sin eval. = %d \
Interv. entre vuelcos = %d \
           Vuelcos guardados = %d \
                             Opciones = %s \
           Semilla aleatoria = %lu \
              Ranking m¡nimo = %lf "



#define IN_VARS
&Totalexperiments,&Totaltrials,&Popsize,&Length,\
          &C_rate,&M_rate,&Gapsize,\
          &Windowsize,&Interval,\
          &Savesize,&Maxspin,\
          &Dump_freq,&Num_dumps,\
          Options,\
          &OrigSeed, &Rank_min



/*        LINE_FIN es el formato de cada linea del
archivo out tal como          */
/*        es leida por la rutina report
                              */
#define LINE_FIN "%lf %lf %lf %lf %lf %lf %lf %lf %lf"

#define LINE_VIN
&line[0],&line[1],&line[2],&line[3],&line[4],\
          &line[5],&line[6],&line[7],&line[8]
```

```
/*        formatos de salida.           */

/* OUT_FORMAT es el formato en que se imprimen los
parametros de entrada */

#define OUT_FORMAT " \
                                        Exp. = %d\n\
                        Tot. Eval. = %d\n\
                                        Pob. = %d\n\
            Long. cromosoma = %d\n\
                        Prob. cruza = %lf\n\
             Prob. mutaci¢n = %lf\n\
             Salto generac. = %lf\n\
            Ventana scaling = %d\n\
            Interv. reporte = %d\n\
           Estruc. guardadas = %d\n\
   Max gens sin eval. = %d\n\
Interv. entre vuelcos = %d\n\
           Vuelcos guardados = %d\n\
                             Opciones = %s\n\
           Semilla aleatoria = %lu\n\
              Ranking m¡nimo = %lf\n"

/* OUT_VARS son los par metros que se imprimir n de
acuerdo con OUT_FORMAT */

#define OUT_VARS
Totalexperiments,Totaltrials,Popsize,Length,\
          C_rate,M_rate,Gapsize,\
          Windowsize,Interval,Savesize,Maxspin,\
          Dump_freq,Num_dumps,Options,OrigSeed,Rank_mi
n


/* OUT_F2 es el formato para los datos producidos por
'Measure'.
 * OUT_V2 describe las variables.
 */
#define OUT_F2 "%5d %5d %2d %2d %5.3f %.6e %.6e %.6e
%.6e\n"

#define OUT_V2 Gen,Trials,Lost,Conv,Bias,Online,\
          Offline,Best,Ave_current_perf



/** fin de archivo **/
```

C.3.4 Módulo GLOBAL.H

```
/*
 *  archivo:        global.h
 *
 *  objetivo:       variables globales
 */

#include "define.h"


/* Nombre de archivos */
char      Bestfile[40];       /* archivo con las
mejores estructuras */
char   Ckptfile[40];          /* archivo de check
point                         */
int    Curr_dump;   /* sufijo del ultimo vuelco   */
char   Dumpfile[40];          /* vuelco actual (si hay
mas de uno)         */
char   Initfile[40];          /* archivo de
estructuras iniciales                   */
char   Infile[40];  /* parametros globales
                    */
```

```
char   Logfile[40]; /* logs de inicio y reinicio
                    */
char   Minfile[40]; /* prefijo de Bestfile (se guarda
un bestfile por cada experimento        */
char   Outfile[40]; /* salida "en bruto" de
estad¡sticas                                    */
char   Schemafile[40];        /* archivo para guardar
la historia de un esquema     */
char   Templatefile[40]; /* archivo que describe la
interpretacion de los genes   */

char   *Bitstring;  /* representaci¢n en cadena de
'1's y '0's         */
double *Vector;               /* representacion de
punto flotante */
int Genes;                    /* Cantidad de genes
interpretados                 */
GENESTRUCT *Gene;  /* puntero a unidades de template
                    */
STRUCTURE  *Old;    /* puntero a la generaci¢n
anterior  */
```

```
STRUCTURE  *New;     /* puntero a la nueva generaci¢n
        */
BESTSTRUCT *Bestset;/* conjunto de las mejores
estructuras                    */


/* El archivo in especifica estos par metros */
int    Totalexperiments; /* cantidad de experimentos
                    */
int    Totaltrials; /* evaluaciones por experimentos
                    */
int    Popsize;                /* tama¤o de la
poblaci¢n */
int    Length;                 /* longitud en bits de
la estructura */
double  C_rate;                /* probabilidad de
cruza*/
double  M_rate;                /* probabilidad de
mutacion */
double  Gapsize;    /* fracci¢n de la poblaci¢n que
ser  reemplazada cada generaci¢n */
int    Windowsize;  /* usado para actualizar la peor
performance */
int    Interval;    /* cantidad de evaluaciones entre
estad¡sticas          */
int    Savesize;    /* n£mero de estructuras que se
guardan en minfile  */
int    Maxspin;                /* max gens sin
evaluaciones                   */
int    Dump_freq;   /* gens entre checkpoints
          */
int    Num_dumps;   /* cantidad de archivos de
checkpoint que se guardan      */
char   Options[40]; /* opciones
                    */
unsigned long    Seed;         /* semilla para el
generador aleatorio */
unsigned long    OrigSeed; /* semilla inicial     */

/* datos estad¡sticos y variables auxiliares */
double  Ave_current_perf;/* aptitud promedio de la
generaci¢n actual   */
double  Best;                  /* la mejor aptitud
hasta ahora          */
double  Best_current_perf;/* la mejor aptitud en la
generaci¢n actual   */
int    Best_guy;    /* numero de estructura con
best_current_perf              */
int    Bestsize;    /* cantidad de mejores estructuras
grabadas */
double  Bias;                  /* promedio de la
dominaci¢n de alelos                    */
int    Bytes;                  /* longitud en bytes de
la estructura empaquetada */
int    Conv;                   /* n£mero de genes que
convergieron parcialmente      */
char   Doneflag;               /* se setea cuando se
cumple la condici¢n de parada */
int    Experiment;  /* contador de experimentos
                    */
int    Gen;                    /* contador         de
generaciones                   */
unsigned long    Initseed; /* semilla inicial del
experimento          */
int    Lost;                   /* cantidad de
posiciones que convergieron totalmente */
int    Mu_next;                /* pr¢xima posici¢n a
mutar                */
double  Offline;    /* performance offline
                    */
```

```
double  Offsum;                /* acumulador para la
performance offline            */
double  Online;                /* performance online
                               */
double  Onsum;                 /* acumulador para
performance online  */
int    Plateau;                /* contador de
evaluaciones para la siguiente salida   */
double Rank_min;    /* minima tasa de muestreo para
seleccion por ranking          */
int    Spin;                   /* cantidad de
generaciones desde la £ltima evaluaci¢n */
double  Totbest;    /* mejor de todos los experimentos
                               */
double  Totoffline; /* offline de todos los
experimentos                                */
double  Totonline;  /* online de todos los
experimentos                                */
int    Trials;                 /* contador de
evaluaciones */
double  *Window;    /* cola circular de los peores de
las £ltimas generaciones       */
double  Worst;                 /* peor performance
hasta ahora          */
double  Worst_current_perf;/* peor perf en la
generaci¢n actual */

/* flags seteados de acuerdo a Options */
char   Allflag;                /*  evaluar todas las
estructuras                    */
char   Bestflag;               /*  imprimir el mejor
valor     final     */
char   Collectflag;            /*  llevar estad¡sticas
de performance */
char   Convflag;               /*  llevar estadisticas
de convergencia */
char   Displayflag; /*  mostrar estadisticas de cada
generacion */
char   Dumpflag;               /*  hacer un vuelco
despues de cada evalueacion             */
char   Eliteflag;   /*  usar la estrategia de
seleccion elitista  */
char   Floatflag;   /*  convertir cadenas a punto
flotante  */
char   Grayflag;               /*  usar gray code
                    */
char   Initflag;               /*  leer estructuras
iniciales de un archivo                 */
char   Interflag;   /*  modo interactivo
                    */
char   Lastflag;               /*  vuelco de la ultima
generacion */
char   Logflag;                /*  log de inicios y
reinicios           */
char   Maxflag;                /*  maximizar en lugar
de minimizar        */
char   Offlnflag;   /*  imprimir medida de offline
final               */
char   Onlnflag;               /*  imprimir medida de
online final                   */
char   Rankflag;               /*  usar seleccion por
ranking             */
char   Restartflag; /*  recomenzar una corrida
                    */
char   Schemflag;   /*  seguir la historia de un
esquema   */
char   Traceflag;   /*  mostrar las rutinas que se
ejecutan            */

/** fin de archivo **/
```

C.3.5 Módulo IN_EVAL.H

```
/*
 *  archivo:        in_eval.h
 *
 *  definicion de las variables para los parametros
del problema
 */

 int nt;                /* cantidad de intervalos en
el tiempo */
 int ntmute;                                /* cantidad de
intervalos en t que ignoro la presion escuchada */
 int l;                                     /* cantidad de
intervalos de la malla */
 double dt;                         /* longitud del
intervalito de tiempo */
 double *h;                     /* longitud en metros de
cada intervalo (por capa) */
 double *ronodo;   /* densidad de cada nodo */
 double xsou;           /* posicion de la fuente */
 int jsou;                          /* nodo
inmediatamente superior a la fuente */
 double xjsou;     /*posicion en mts. del nodo
anterior a la fuente */
 double xjsoup1;   /*posicion en mts. del nodo
posterior a la fuente */
 double xrec;             /*posicion en mts. del
receptor  */
 int jrec;                    /*nodo anterior al
receptor  */
 double xjrec;       /*posicion en mts. del nodo
anterior al receptor */
 double xjrecp1;   /*posicion en mts. del nodo
posterior al receptor */
```

```
 int *kx;                       /* cantidad acumulada de
intervalos en x de una capa */
 int keyb;                              /* tipo de
borde:

                                /* 0: superficie libre

                                /* 1: condicion de bordes
absorbente

                                /* 2: neuman

                    */
 int kdel;                              /* tipo de
fuente:

                    1 -> delta de Dirac

                    2 -> derivada de la delta de Dirac

                    3 -> 2 delta de Dirac

                    */
 double *source;    /* funcion fuente muestreada */
 double is;
 double *recmed;   /* presiones observadas en el
receptor */
 /* variables necesarias para la codificacion por nro
de material */
 int cantMateriales;
 double *velMaterial;
 double *roMaterial;
```

```
 /** fin de archivo **/
```

C.3.6 Módulo IN_EVAL2.H

```
/*
 *  archivo:        in_eval2.h
 *
 *  declaraciones externas para los parametros del
problema
 */

extern int nt;                  /* cantidad de
intervalos en el tiempo */
extern int ntmute;                      /* cantidad de
intervalos en t que ignoro la presion escuchada */
extern int l;                                   /*
cantidad de intervalos de la malla */
extern double dt;               /* longitud del
intervalito de tiempo */
extern double *h;             /* longitud en metros de
cada intervalo (por capa) */
extern double *ronodo;   /* densidad de cada nodo */
extern double xsou;     /* posicion de la fuente */
extern int jsou;                /* nodo
inmediatamente superior a la fuente */
extern double xjsou;     /*posicion en mts. del nodo
anterior a la fuente */
extern double xjsoup1;   /*posicion en mts. del nodo
posterior a la fuente */
extern double xrec;      /*posicion en mts. del
receptor  */
extern int jrec;              /*nodo anterior al
receptor  */
extern double xjrec;          /*posicion en mts. del
nodo anterior al receptor */
extern double xjrecp1;   /*posicion en mts. del nodo
posterior al receptor */
extern int *kx;                         /* cantidad
acumulada de intervalos en x de una capa */
```

```
extern int keyb;                        /* tipo de
borde:

                                0: superficie libre

                                1: condici¢n de bordes
absorbente

                                2: neuman

                        */
extern int kdel;                        /* tipo de
fuente:

                        1 -> delta de Dirac

                        2 -> derivada de la delta de Dirac

                        3 -> 2 delta de Dirac

                        */
extern double *source;          /* funcion fuente
muestreada */
extern double is;
extern double *recmed;   /* presiones observadas en el
receptor */
 /* Variables necesarias para la codificacion por nro
de material */
extern int cantMateriales;
extern double *velMaterial;
extern double *roMaterial;

 /** fin de archivo **/
```

C.3.7 Módulo PRESION.H

```
/*
 *  archivo:        presion.h
 *
 *  c lculo de las presiones
 */

#ifndef PRESION_H
#define PRESION_H

double una_presion(double *unm1, double *un, double
*unp1, int paso, double *constk, double *velnodo);
void presiones(double *velocidad, int capas, double
*presion, int nt);
```

```
void vel_en_nodos(double *velocidad, int capas, double
*velnodo);
double borde_sup(double unm1, double unx1, double
unx2, double cte);
double borde_inf(double unm1, double unx1, double
unx2, double cte);
double fuente_en_borde(int paso, int nodo);
double fuente_interior(int paso, int nodo, double cte,
double *velnodo);

#endif

/** fin de archivo **/
```

C.3.8 Módulo MAIN.C

```
/*
 *  archivo:        main.c
 *
 *  objetivo:       programa principal.
 *
 */

#include "global.h"
#include "in_eval.h"

main(argc,argv)
int argc;
char *argv[];
{
          FILE *fp, *fopen();
          long clock;
          long time();
          char *ctime();
          extern void die();  /* manejador de se¤ales
*/
          int i;


          /* ver el uso de parametros en la linea de
comandos en input.c */
          Input(argc,argv);
          In_eval(argc,argv);

          if (Displayflag) {
                    initscr();
                    signal(SIGINT, die);
                    for (i=1;i<=23;i++) {
                              move(i,0);
                              clrtoeol();
                    }
                    refresh();

                    if (Interflag)
                              Interactive(); /* nunca
retorna */

                    /* este punto se alcanza solo si
Interflag esta en OFF */
```

```
                    /* l¡nea superior */
                    move(1,0);
                    for (i=0;i<=79;i++) {
                              printw("-");
                    }
                    /* l¡nea inferior */
                    move(23,0);
                    for (i=0;i<=79;i++) {
                              printw("-");
                    }

                    move(2,0);
                    printw(" Propuesta  II  Modelo %s
Tope de evaluaciones = %d", argv[1], Totaltrials);
                    refresh();
          }

          do                  /* un experimento */
                    {

                    move(2,60);
                    printw("Experimento %d/%d",
Experiment+1,Totalexperiments);
                    /* limpiar de la pantalla los
resultados del experimento anterior */
                    for (i=5;i<=22;i++) {
                              move(i,0);
                              clrtoeol();
                    }
                    refresh();

                    do        /* ver el ciclo
principal de AG en generate.c */
                              {
                              Generate();
                    }
                    while (!Doneflag);

                    if (Traceflag)
                    printf("Online %e   Offline %e
Mejor %e\n",
                              Online, Offline, Best);
```

```
                    /* acumular mediciones de
performance */
                    Totonline += Online;
                    Totoffline += Offline;
                    Totbest += Best;

                    /* preparar el proximo experimento
*/
                    Experiment++;
                    Gen = 0;

          }
          while (Experiment < Totalexperiments);

          /* calcular e imprimir las mediciones
finales de performance */

          free(velMaterial);
          free(roMaterial);

          Totonline /= Totalexperiments;
          Totoffline /= Totalexperiments;
          Totbest /= Totalexperiments;

          if (Displayflag) {
                    move(15,0);
                    printf(" Mejor individuo obtenido
en todos los experimentos:   %e\n", Totbest);
                    refresh();
          }
          if (Logflag)
          {
                    fp = fopen(Logfile, "a");
                    fprintf(fp, "Mejor %e\n",
Totbest);
                    time(&clock);
                    fprintf(fp, "%s\n",
ctime(&clock));
                    fclose(fp);
          }

          if (Displayflag) {
                    move(22,0);
                    printf(" Presione cualquier tecla
para continuar...");
                    getch();
                    die();
          }
}

/** fin de archivo **/
```

C.3.9 Módulo BEST.C

```
/*
 *  archivo:        best.c
 *
 *  objetivo:       entrada, mantenimiento y salida de
las mejores estructuras
 *
 */

#include "extern.h"

static double worst_value;    /* peor valor en
Bestset          */
static int worst;             /* puntero al peor
elemento (elemento con worst_value)     */


Savebest(i)
register int i;               /* indice de estructura
en  Bestset        */
{
          /*  Grabar la i-esima estructura de la
poblacion actual */
          /*  La cantidad de estructuras a grabar es
Savesize */

          register int j;
          register int k;
          int found;

          if (Bestsize < Savesize)
          {
                    /* Bestsize es la cantidad grabada
hasta ahora, entonces */
                    /* todavia hay lugar en Bestset
                    */
                    /* Ver si ya existe una estructura
identica en Bestset */
                    for (j=0, found=0; j<Bestsize &&
(!found); j++)
                              for (k=0, found=1;
(k<Bytes) && (found); k++)
                                        found =
(New[i].Gene[k]
                                                  ==
Bestset[j].Gene[k]);
                    if (found) return;

                    /* insertar la i-esima estructura
*/
                    for (k=0; k<Bytes; k++)
                    {

          Bestset[Bestsize].Gene[k] = New[i].Gene[k];
                    }
                    Bestset[Bestsize].Perf =
New[i].Perf;
                    Bestset[Bestsize].Gen = Gen;
                    Bestset[Bestsize].Trials = Trials;
                    Bestsize++;
                    if (Bestsize == Savesize)
                    {
                              /* buscar el peor
elemento en Bestset */
                              worst_value =
Bestset[0].Perf;
                              worst = 0;
                              for (j=1; j<Savesize;
j++)
                              {
                                        if
(BETTER(worst_value,Bestset[j].Perf))
                                        {

          worst_value = Bestset[j].Perf;

          worst = j;
                                        }
                              }
                    }
          }
          else
          {
                    /* Ya estan ocupados todos los
lugares, ver si hay que desplazar
                       a alguno */
                    if (BETTER(New[i].Perf,
worst_value))       /* hay que grabar New[i] */
                    {
                              /* a menos que ya este
representada */
                              for (j=0, found=0;
j<Bestsize && (!found); j++)
                                        for (k=0,
found=1; (k<Bytes) && (found); k++)

          found = (New[i].Gene[k]

          == Bestset[j].Gene[k]);
                              if (found) return;

                              /* sobreescribir la peor
estructura */
                              for (k=0; k<Bytes; k++)
                              {

          Bestset[worst].Gene[k] = New[i].Gene[k];
                              }
                              Bestset[worst].Perf =
New[i].Perf;
                              Bestset[worst].Gen =
Gen;
                              Bestset[worst].Trials =
Trials;
                              /* buscar el peor
elemento Bestset */
                              worst_value =
Bestset[0].Perf;
                              worst = 0;
                              for (j=1; j<Savesize;
j++)
                              {
                                        if
(BETTER(worst_value, Bestset[j].Perf))
                                        {

          worst_value = Bestset[j].Perf;

          worst = j;
                                        }
                              }
                    }
          }
}


Printbest()
{
          /*        Escribir las mejores estructuras
en Bestfile.        */

          register int i;
          register int j;
          register int k;
          FILE *fp, *fopen();

          fp = fopen(Bestfile, "w");

          for (i=0; i<Bestsize; i++)
          {
                    Unpack(Bestset[i].Gene, Bitstring,
Length);
                    if (Floatflag)
                    {
                              FloatRep(Bitstring,
Vector, Genes);
                              for (j=0; j<Genes; j++)
```

```
                                {
                                        fprintf(fp,
Gene[j].format, Vector[j]);
                                        fprintf(fp, "
");
                                }
                        }
                        else
                        {
                                for (j=0; j<Length; j++)
                                {
                                        fprintf(fp,
"%c", Bitstring[j]);
                                }
                        }
                        fprintf(fp, "  %11.4e ",
Bestset[i].Perf);
                        fprintf(fp, " %4d  %4d\n",
Bestset[i].Gen, Bestset[i].Trials);
                }

                fclose(fp);
}


Readbest()
{
                /*   Leer las mejores estructuras de
Bestfile    */
                /*   durante un Restart
*/

                int i,j,k;
                FILE *fp, *fopen();
                int status;

                if (Savesize)
                {

                        fp = fopen(Bestfile, "r");
                        if (fp == NULL)
                        {
                                Bestsize = 0;
                                return;
                        }

                        Bestsize = 0;
                        status = 1;
                        if (Floatflag)
                        {
                                for (i = 0; i < Genes &&
status != EOF; i++)
                                {
                                        status =
fscanf(fp, "%lf", &Vector[i]);
                                }
                        }
                        else
                                status = fscanf(fp,
"%s", Bitstring);

                        while (status != EOF)
                        {
                                if (Floatflag)

                StringRep(Vector, Bitstring, Genes);

                                Pack(Bitstring,
Bestset[Bestsize].Gene, Length);

                                fscanf(fp, "%lf ",
&Bestset[Bestsize].Perf);
                                fscanf(fp, "%d  %d",
&Bestset[Bestsize].Gen,

                &Bestset[Bestsize].Trials);
                                Bestsize++;

                                /* Tomar la proxima
estructura */
                                if (Floatflag)
                                        for (i = 0; i
< Genes && status != EOF; i++)

                status = fscanf(fp, "%lf", &Vector[i]);
                                else
                                        status =
fscanf(fp, "%s", Bitstring);
                        }
                        fclose(fp);

                        /* buscar el peor elemento en
Bestset */
                        worst_value = Bestset[0].Perf;
                        worst = 0;
                        for (i=1; i<Bestsize; i++)
                        {
                                if (BETTER(worst_value,
Bestset[i].Perf))
                                {
                                        worst_value =
Bestset[i].Perf;
                                        worst = i;
                                }
                        }
                }
}

/**** fin de archivo ****/
```

C.3.10 Módulo CHECKPNT.C

```
/*
 *  archivo:        checkpoint.c
 *
 *  objetivo:       grabar las variables globales para
un eventual reinicio.
 *
 */
#include <dir.h>
#include "extern.h"


Checkpoint(ckptfile)
char ckptfile[];
{
        FILE *fp, *fopen();
        int i,j;
        char CkptfileExp[MAXPATH], drive[MAXDRIVE],
dir[MAXDIR], file[MAXFILE], ext[MAXEXT];

        /* Archivo de checkpoint */
        fnsplit(ckptfile, drive, dir, file, ext);
        sprintf(CkptfileExp, "%s%d%s", file,
Experiment, ext);
        fp = fopen(CkptfileExp, "w");

        fprintf(fp, "Experimentos %d\n",
Experiment);
        fprintf(fp, "On_line total   %12.6e\n",
Totonline);
        fprintf(fp, "Off_line total   %12.6e\n",
Totoffline);
        fprintf(fp, "Generaci¢n %d\n", Gen);
        fprintf(fp, "Performance On_line  %12.6e\n",
Onsum);
        fprintf(fp, "Performance Off_line %12.6e\n",
Offsum);
        fprintf(fp, "Evaluaciones  %d\n", Trials);
        fprintf(fp, "Pr¢xima estad¡stica %d\n",
Plateau);
        fprintf(fp, "Mejor %12.6e\n", Best);
        fprintf(fp, "Peor   %12.6e\n", Worst);
        fprintf(fp, "Cantidad de generaciones desde
la £ltima evaluaci¢n %d\n", Spin);
        fprintf(fp, "Ultimo vuelco %d\n",
Curr_dump);
        fprintf(fp, "Mu_prox %d\n", Mu_next);
        fprintf(fp, "Semilla Aleatoria %lu\n",
Seed);
        fprintf(fp, "Semilla Inicializadora %lu\n",
Initseed);

        fprintf(fp,"\n");
        fprintf(fp, "Window\n");
        for (i=0; i<Windowsize; i++) fprintf(fp,
"%12.6e\n", Window[i]);
        fprintf(fp,"\n");


        for (i=0; i<Popsize; i++)
        {
                Unpack(New[i].Gene, Bitstring,
Length);
                fprintf(fp, "%s", Bitstring);
                fprintf(fp, " %12.8e ",
New[i].Perf);

                fprintf(fp, "%1d   ",
New[i].Needs_evaluation);

                fprintf(fp, "\n");
        }

        if (Floatflag)      /* imprimir en punto
flotante */
        {
                fprintf(fp, "\n");
                for (i=0; i<Popsize; i++)
                {
                        Unpack(New[i].Gene,
Bitstring, Length);
                        FloatRep(Bitstring,
Vector, Genes);
                        for (j=0; j < Genes;
j++)
                        {
                                fprintf(fp,
Gene[j].format, Vector[j]);
                                fprintf(fp, "
");
                        }
                        fprintf(fp, " %10.4f",
New[i].Perf);
```

```
                                fprintf(fp, "\n");
                        }
                }

                fclose(fp);

                /*  guardar las mejores estructuras en
Bestfile */
```

```
        if (Savesize)
                Printbest();

}

/** fin de archivo **/
```

C.3.11 Módulo CONVERT.C

```
/*
 *  archivo:        convert.c
 *
 *  objetivo:       funciones que traducen entre
distinas representaciones
 *
 */

#include "extern.h"

static char BIT[CHARSIZE] ={ '\200', '\100', '\040',
'\020',
                                        '\010',
'\004', '\002', '\001'};


/* Itoc and Ctoi traducen ints a strings y viceversa
*/


unsigned long int Ctoi(instring, length)
        char *instring;                 /* cadena de
'0's y '1's             */
        int length;                     /* longitud de
instring                */
{
        register int i;
        unsigned long n;    /* acumulador para el
valor a retornar */

        n = (unsigned long) 0;
        for (i=0; i<length; i++)
        {
                n <<= 1;
                n += (*instring++ - (int) '0');
        }
        return(n);
}



Itoc(n, outstring, length)
        unsigned long int n;            /* entero a
convertir               */
        char *outstring;    /* cadena de '0's y '1's
resultante */
        int length;                     /* longitud de
outstring               */
{
        register int i;

        for (i=length-1; i>=0; i--)
        {
                outstring[i] = '0' + (n & 1);
                n >>= 1;
        }
}


/* Pack y Unpack traducen entre cadenas de '0's y '1's
y arreglos de bits
   empaquetados */

Pack(instring, outstring, length)
        char *instring;                 /* cadena de
'0's y '1's */
        char *outstring;    /* representacion
empaquetada de instring */
        int length;                     /* longitud de
instring   */
{
        static firstflag = 1 ;
        static full;        /* numero de bytes
completamente usados en outstring       */
        static slop;        /* numbero de bits
usados en el ultimo byte de outstring */
        register i,j;

        if (firstflag)
        {
                full = length / CHARSIZE;
                slop = length % CHARSIZE;
                firstflag = 0;
        }

        for (i=0; i<full; i++, outstring++)
        {
                *outstring = '\0';
                for (j=0; j < CHARSIZE; j++)
                        if (*instring++ == '1')
*outstring |= BIT[j];
        }
        if (slop)
        {
                *outstring = '\0';
                for (j=0; j < slop; j++)
                        if (*instring++ == '1')
*outstring |= BIT[j];
        }
}


Unpack(instring, outstring, length)
        char *instring;                 /*
representacion binaria empaquetada      */
        char *outstring;    /* representacion en
instring en cadena de '0's y '1's */
        int length;                     /* longitud de
outstring                       */
{
        static firstflag = 1 ;
        static full;        /* numbero de bytes
completamente usados en instring */
        static slop;        /* numbero de bits
usados en el ultimo byte de instring */
        register i,j;

        if (firstflag)
        {
                full = length / CHARSIZE;
                slop = length % CHARSIZE;
                firstflag = 0;
        }

        for (i=0; i<full; i++, instring++)
        {
                for (j=0; j < CHARSIZE; j++)
                        if (*instring & BIT[j])
                                *outstring++ =
'1';
                        else
                                *outstring++ =
'0';
        }

        if (slop)
        {
                for (j=0; j < slop; j++)
                        if (*instring & BIT[j])
                                *outstring++ =
'1';
                        else
                                *outstring++ =
'0';
        }
        *outstring = '\0';
}


/* Traducciones entre representaciones binarias
tradicionales de enteros
   (punto fijo) y Gray code */


Gray(instring, outstring, length)
char *instring;                 /* string que representa
un entero en punto fijo */
char *outstring;    /* string que representa el entero
en Gray code        */
register int length;            /* longitud de strings
                                */
{
        register int i;
        register char last;

        last = '0';
        for (i=0; i<length; i++)
        {
                outstring[i] = '0' + (instring[i]
!= last);
                last = instring[i];
        }
}


Degray(instring, outstring, length)
char *instring;                 /* string que representa
el entero en Gray code          */
char *outstring;    /* string que representa el entero
en punto fijo */
register int length;            /* longitud de strings
                                */
{
        register int i;
        register int last;

        last = 0;
        for (i=0; i<length; i++)
        {
                if (instring[i] == '1')
                        outstring[i] = '0' +
(!last);
```

```
                    else
                              outstring[i] = '0' +
last;
                    last = outstring[i] - '0';
          }
}


/* Traducciones entre cadenas de '0's y '1's y
vectores de puntos flotantes
 */


FloatRep(instring, vect, length)
          char instring[];    /* cadena de '0's y '1's
                    */
          double vect[];                /*
representacion en punto flotante */
          int length;                   /* longitud de
vect (arreglo de salida)      */
{
          register int i;
          unsigned long int n;          /* Valor
entero decodificado                     */
          register int pos;   /* posicion para
comenzar a decodificar                  */
          char tmpstring[80]; /* usado para la
interpretacion en Gray code */

          pos = 0;
          for (i=0; i < length; i++)
          {
                    if (Grayflag)
                    {
                              Degray(&instring[pos],
tmpstring, Gene[i].bitlength);
                              n = Ctoi(tmpstring,
Gene[i].bitlength);
                    }
                    else
                    {
                              n = Ctoi(&instring[pos],
Gene[i].bitlength);
                    }
                    vect[i] = Gene[i].min +
n*Gene[i].incr;
                    pos += Gene[i].bitlength;
          }
}
```

```
StringRep(vect, outstring, length)
          double *vect;                 /*
representacion en punto flotante        */
          char *outstring;    /* cadena de '0's y '1's
                              */
          int length;                   /* longitud de
vect                                    */
{
          register int i;
          unsigned long int n;          /* valor
entero (indice) con que se representa vect[i]*/
          register int pos;   /* proxima posicion para
llenar  outstring   */
          char tmpstring[80]; /* usado para la
traduccion a gray code          */

          pos = 0;
          for (i=0; i < length; i++)
          {
                    /* Convertir valor de punto
flotante a un indice */
                    n = (int) ((vect[i] - Gene[i].min)
/ Gene[i].incr + 0.5);

                    /* codificar n como cadena de
caracteres */
                    if (Grayflag)
                    {
                              /* convertir a Gray code
*/
                              Itoc(n, tmpstring,
Gene[i].bitlength);
                              Gray( tmpstring,
&outstring[pos], Gene[i].bitlength);
                    }
                    else
                    {
                              Itoc(n, &outstring[pos],
Gene[i].bitlength);
                    }
                    pos += Gene[i].bitlength;
          }
          outstring[pos] = '\0';
}


/*** fin de archivo ***/
```

C.3.12 Módulo CROSS.C

```
/*
 *  archivo:        cross.c
 *
 *  objetivo:       efectuar la cruza en dos puntos
sobre toda la poblaci¢n
 *              La cruza se realiza a nivel de gen.
 */

#include <alloc.h>
#include "extern.h"

/**** las mascaras estan en octal ****/
char premask[CHARSIZE] = { '\000', '\200', '\300',
'\340',
                                        '\360',
'\370', '\374', '\376' };

char postmask[CHARSIZE] = { '\377', '\177', '\077',
'\037',
                                        '\017',
'\007', '\003', '\001'};


Crossover()
/**** Si los padres no difieren en los segmentos
externos =>
           no cruza & no evalua
      Sino
            cruza, compara los segmentos internos
            Si los segmentos internos son iguales =>
                  no evalua
****/
{
          register int mom, dad;        /*
participantes en la cruza */
          register int xgen1;  /* primer punto de
cruza con respecto al cromosoma */
          register int xgen2;  /* segundo punto de
cruza con respecto al cromosoma */
          register int xpoint1;         /* primer
punto de cruza con respecto a la estructura */
          register int xpoint2;   /* segundo punto de
cruza con respecto a la estructura */
          register int xbyte1;          /* primer byte
a cruzar */
          register int xbit1; /* primer bit a cruzar
en el byte xbyte1 */
          register int xbyte2;          /* ultimo byte
a cruzar */
          register int xbit2; /* ultimo bit a cruzar
en xbyte2 */
          register int i;
```

```
          register char temp; /* usado para
intercambiar alelos */
          static int last;    /* ultimo individuo en
sufrir la cruza */
          int diff;           /* indica si los padres
difieren de los hijos */
          char *kid1;                   /* punteros a
los hijos */
          char *kid2;
          static int firstflag = 1;

          if (firstflag)
            {
                    last = (C_rate*Popsize*Gapsize) -
0.5 ;
               firstflag = 0;
            }

          for (mom=0; mom < last ; mom += 2)
          {
                    dad = mom + 1;

                    /* los hijos comienzan siendo
copias identicas de los padres */
                    kid1 = New[mom].Gene;
                    kid2 = New[dad].Gene;

                    /* se eligen dos puntos de cruza
*/
                    xgen1 = Randint(0,Genes);
                    xgen2 = Randint(0,Genes-1);

                    /* garantizo que xgen1 < xgen2 */
                    if (xgen2 >= xgen1)
                              xgen2++;
                    else
                    {
                              i = xgen1;
                              xgen1 = xgen2;
                              xgen2 = i;
                    }

                    /* Obtengo bits por los que se
hace la cruza */
                    for(i=0, xpoint1=0; i < xgen1;
i++) {
                              xpoint1 +=
Gene[i].bitlength;
                    }

                    for(i=0, xpoint2=0; i < xgen2;
i++) {
```

```
                                xpoint2 +=
Gene[i].bitlength;
                    }

                    xbyte1 = xpoint1 / CHARSIZE;
                    xbit1 = xpoint1 % CHARSIZE;
                    xbyte2 = xpoint2 / CHARSIZE;
                    xbit2 = xpoint2 % CHARSIZE;

                    /* Me fijo si los padres difieren
en los segmentos externos
                       ya que si los mismos son
iguales, la cruza no tiene sentido
                         pues los hijos saldrian iguales
a los padres */
                    diff = 0;
/**** comparo bytes enteros de la izq. ****/
                    for (i=0; i < xbyte1; i++) diff +=
(kid1[i] != kid2[i]);
/**** comparo fraccion de byte de la izq. ****/
                    diff += ( (kid1[xbyte1] &
premask[xbit1]) !=
                                (kid2[xbyte1] &
premask[xbit1]) );
/**** comparo fraccion de byte de la der. ****/
                    diff += ( (kid1[xbyte2] &
postmask[xbit2]) !=
                                (kid2[xbyte2] &
postmask[xbit2]) );
/**** comparo bytes enteros de la der. ****/
                    for (i=xbyte2+1; i < Bytes; i++)
diff += (kid1[i] != kid2[i]);

                    if (diff) /* los padres difieren
en los segmentos externos */
                    {
                                /* cruzarlos */
                                temp = kid1[xbyte1];
/**** cruzo el xbyte1 ****/
                                kid1[xbyte1] =
(kid1[xbyte1] & premask[xbit1]) |

          (kid2[xbyte1] & postmask[xbit1]);

                                kid2[xbyte1] =
(kid2[xbyte1] & premask[xbit1]) |

          (temp & postmask[xbit1]);

                                diff  = ((kid1[xbyte1] &
postmask[xbit1]) !=
                                          (kid2[xbyte1]
& postmask[xbit1]) );

                                for (i=xbyte1 + 1; i <
xbyte2; i++)
/**** cruzo el segmento de bytes enteros, pero no el
xbyte2 ****/
                                {
                                          temp =
kid1[i];
                                          kid1[i] =
kid2[i];
                                          kid2[i] =
temp;
                                          diff +=
(kid1[i] != kid2[i]);
                                }

                                if (xbyte1 < xbyte2)
/**** si la cruza abarca mas de un byte.... ****/
                                {
/**** cruzo el xbyte2 ****/
                                          temp =
kid1[xbyte2];
                                          kid1[xbyte2] =
(kid1[xbyte2] & postmask[xbit2]) |

          (kid2[xbyte2] & premask[xbit2]);

                                          kid2[xbyte2] =
(kid2[xbyte2] & postmask[xbit2]) |

          (temp & premask[xbit2]);

                                          diff  +=
((kid1[xbyte2] & premask[xbit2]) !=

          (kid2[xbyte2] & premask[xbit2]) );
                                }
                                else
                                {
                                          temp =
kid1[xbyte2];
                                          kid1[xbyte2] =
(kid1[xbyte2] & premask[xbit2]) |

          (kid2[xbyte2] & postmask[xbit2]);

                                          kid2[xbyte2] =
(kid2[xbyte2] & premask[xbit2]) |

          (temp & postmask[xbit2]);

                                          diff  =
((kid1[xbyte2] & postmask[xbit1] & premask[xbit2]) !=

          (kid2[xbyte2] & postmask[xbit1] &
premask[xbit2]) );

                                }

                                if (diff) /* los hijos
difieren de los padres */
                                {
                                          /* entonces,
los hijos deben ser evaluados */

          New[mom].Needs_evaluation = 1;

          New[dad].Needs_evaluation = 1;
                                }
                    }
          }
}

/** fin de archivo **/
```

C.3.13 Módulo DISPLAY.C

```
/*
 *  archivo:        display.c
 *
 *  objetivo:       maneja la pantalla
 *
 */

#include "extern.h"

Interactive() {
          char cmd[40];
          char opt[40];
          register int i;
          int ncycles;
          int ok;

          ncycles = 1;
          while (1) {
                    ok = 1;
                    move(22,0);
                    clrtoeol();
                    move(22,35);
                    printw("q (clear & exit), x
(exit), <n> (do n gens)");
                    move(22,0);
                    printw("
");
                    refresh();
                    move(22,0);
                    printw("gens[%d]: ", ncycles);
                    refresh();
                    getstr(cmd);
                    if (strcmp(cmd, "q") == 0) {
                              if (Lastflag)

          Checkpoint(Ckptfile);
                              else
                                        if (Savesize)

          Printbest();
                              clear();
                              die();
                    }
                    if (strcmp(cmd, "x") == 0) {
                              if (Lastflag)

          Checkpoint(Ckptfile);
                              else
                                        if (Savesize)

          Printbest();
                              move(23,0);
                              die();
                    }
                    if (strcmp(cmd, "") != 0) {
                              if (sscanf(cmd, "%d",
&ncycles) !=1)
                              {
                                        move(23,0);
                                        clrtoeol();

          printw("unknown command: %s", cmd);
                                        ok = 0;
                                        refresh();
                              }
                    }
                    if (ok)
                    {
                              move(23,0);
                              clrtoeol();
                              move(1,0);
                              printw("run until Gens =
%d", Gen + ncycles -1);
                              move(1,35);
                              printw("executing: ");
                              refresh();
                              for (i=0; i < ncycles;
i++)
                                        Generate();
                    }
```

```
        }
}

die(sig)
int sig;
{
        sig++;
        signal(SIGINT, SIG_IGN);
        move(23,0);
        clrtoeol();
        refresh();
        endwin();
        exit(0);
}


move(row, col)
        int row, col;
{
        /* mover a las coordenadas fila, columna de
la pantalla */
        /* (0,0) = esquina superior izquierda;
           (23,79) = esquina inferior derecha
        */

        gotoxy(col+1, row+1);
}

clear()
{
        /* limpiar la pantalla */
        clrscr();
}


getstr(s)
     char *s;
{
  getw(s);
}

initscr() {}

endwin() {}

/**** fin de archivo ****/
```

C.3.14 Módulo DONE.C

```
/*
 *  archivo:        done.c
 *
 *  objetivo:       ver si se cumple la condicion de
parada.
 *
 */

#include "extern.h"

int Done()
{
/**** por demasiadas evaluaciones, o toda la poblacion
es igual, o
        demasiadas generaciones sin evaluar ****/
        return ((Trials >= Totaltrials)  || ( Lost
>= Genes)
                || (Spin >= Maxspin));
}

/* fin de archivo */
```

C.3.15 Módulo ELITIST.C

```
/*
 *  archivo:        elitist.c
 *
 *  objetivo:       La politica elitista estipula que
el mejor individuo
 *                  siempre sobrevive en la nueva
generacion. El individuo de elite
 *                  es ubicado en la ultima posicion
de la nueva poblacion
 *            y no es afectado por cruza o mutacion ya
que se pasa a la nueva
 *                  generacion despues de haber
aplicado estos operadores.
 */

#include "extern.h"

Elitist()
{
        register int i;
        register int k;
        register int found; /* se setea si ya existe
el individuo de elite en

            la nueva generacion */


        /* Hay algun elemento en la poblacion actual
                */
        /* que sea identico al mejor de la
generacion anterior?        */

        for (i=0, found=0; i<Popsize && (!found);
i++)
                for (k=0, found=1; (k<Bytes) &&
(found); k++)
                        found = (New[i].Gene[k]
                                ==
Old[Best_guy].Gene[k]);

        if (!found)         /* No se encontro el
individuo de elite */
        {
                /* reemplazar el ultimo individuo
por el individuo de elite */
                for (k=0; k<Bytes; k++)
                        New[Popsize-1].Gene[k] =
Old[Best_guy].Gene[k];
                New[Popsize-1].Perf =
Old[Best_guy].Perf;
                New[Popsize-1].Needs_evaluation =
0;

                if (Traceflag)
                {
                        printf("perf: %e\n",
New[Popsize-1].Perf);
                }
        }

}


/*** fin de archivo ***/
```

C.3.16 Módulo ERROR.C

```
/*
 *  archivo:        error.c
 *
 *  objetivo:       imprimir mensaje de error y
abortar. El error queda tambien
 *                              en un log de
errores.
 *
 */

#include <stdio.h>

Error(s)
char *s;
{
        FILE *fopen(), *fp;
        long clock;
        long time();
        char *ctime();

        fp = fopen("error", "a");
        fprintf(fp, "%s\n", s);
        time(&clock);
        fprintf(fp, "%s\n", ctime(&clock));
        fclose(fp);
        fprintf(stderr, "%s\n", s);

        exit(1);
}

/*** fin de archivo ***/
```

C.3.17 Módulo EVAL.C

```
/*
 * archivo: evalmat.c
 *
 * objetivo: calcular la aptitud del individuo.
 */

#include <alloc.h>
#include "extern.h"
#include "in_eval2.h"
```

```
/* Rutina que hace la evaluacion de un cromosoma cuyos
genes son
   nros de material */

double costo(v1, v2, N)
   double v1[], v2[];
   int N;
/*                N
                                        --
                                1       \              2
                                -       /    ( v1 - v2 )
                                n         --       i
i
                                       i=1
*/
{
          int i;
          double s= 0;

          for (i=1; i <= N; i++)
                    s+= pow(( v1[i] - v2[i] ),2);
          return s/N;
}



double eval(str, length, vect, genes)
char str[];          /* representacion en cadena de
'0's y '1's          */
int length;          /* longitud de la cadena
                     */

/* vect: materiales de cada una de las "genes" capas
*/
```

```
double vect[];       /* representacion en punto
flotante             */

/* genes: cantidad de capas */
int genes;           /* cantidad de elementos en el
vector                        */
{
          double resul;
          /* presiones calculadas a partir de las
velocidades, una por cada
          instante de tiempo */
          double
*recest=malloc((nt+1)*sizeof(double));

          /* convierto materiales a velocidades */
          double *velAux = malloc(genes *
sizeof(double));
          int i,j;
          for (i=0; i < genes; i++)
                    velAux[i] = velMaterial[vect[i]];

          /* lleno ronodo */
          for (i=0; i < genes; i++)
                    for (j=kx[i]+1; j <= kx[i+1]; j++)
                              ronodo[j] =
roMaterial[vect[i]];
          ronodo[1+1]= roMaterial[vect[genes-1]];

          presiones(velAux, genes, recest, nt);
          free(velAux);

          resul =costo(recest, recmed, nt);
          free(recest);
          return resul;
}
```

C.3.18 Módulo EVALUATE.C

```
/*
 *  archivo:        evaluate.c
 *
 *  objetivo:       evaluar la poblacion actual
mediante la funcion "eval"
 *
 */

#include "extern.h"
extern double eval();

Evaluate()
{
          register double performance;
          register int i;

          for (i=0; i<Popsize; i++)
          {
                    if (Displayflag) {
                              move(20,0);
                              clrtoeol();
                              printf(" Evaluando
individuo %d de la poblaci¢n",i);
                              refresh();
                    }

                    if ( New[i].Needs_evaluation )
                    {
                              Unpack(New[i].Gene,
Bitstring, Length);
                              if (Floatflag)

          FloatRep(Bitstring, Vector, Genes);
                              New[i].Perf = eval
(Bitstring, Length,
```

```
                                        Vector,
Genes);
                              performance =
New[i].Perf;
                              New[i].Needs_evaluation
= 0;
                              Trials++;
                              Spin = 0;   /* Hubo una
evaluacion, spin vuelve a 0 */
                              if (Savesize)
Savebest(i);

                              if (Trials == 1)
                                        Best =
performance;
                              if (BETTER(performance,
Best))
                              {
                                        Best =
performance;
                              }

                              Onsum += performance;
                              Offsum += Best;
                              if (Dumpflag)
Checkpoint(Ckptfile);
                    }
          }
          move(20,0);
          clrtoeol();
}


/** fin de archivo **/
```

C.3.19 Módulo GENERATE.C

```
/*
 *  archivo:        generate.c
 *
 *  objetivo:       Una generacion consiste de
 *                              (1) formar una nueva
poblacion de estructuras
 *                              (2) evaluar la poblacion
 *                              (3) recoger estadisticas
de performance
 *
 */

#include "extern.h"
extern int  Done();


Generate()
{
          static int rsflag = 1;        /* flag
reseteado despues de restart          */
```

```
          STRUCTURE *temp;    /* para swapear punteros
de la poblacion     */
          register int i;               /* para marcar
estructuras                   */

          if (Traceflag)
                    printf("                    Gen
%d\n",Gen);

          /* crear una nueva poblacion */

          if (Restartflag && rsflag)
          {
                    /* si es un restart leer archivo
de checkpoint */
                    Restart();
                    rsflag = 0;         /* Ya hicimos
este restart. */
                    Converge();
          }

          else if (Gen == 0)
```

```
                    /* Si recien comienza el
experimento */
          {
                    Initialize();       /* formar una
poblacion inicial */
                    Spin++;
          }

          else
                    /* formar una nueva poblacion a
partir de */
                    /* la vieja via operadores
geneticos */
          {
                    Select();
                    Mutate();
                    Crossover();
                    if (Eliteflag)
                              Elitist();
                    if (Allflag)        /* marcar las
estructuras para evaluacion */
                              for (i=0; i<Popsize;
i++) New[i].Needs_evaluation = 1;
                    Spin++;
          }

          /* evaluar la poblacion recientemente
formada */
          Evaluate();

          /* recoger estadisticas de performance */
          Measure();

          /* chequear la condicion de parada para este
experimento */
          Doneflag = Done();

          /* Hacer vuelco si corresponde */
          if (Num_dumps && Dump_freq && Gen %
Dump_freq == 0)
          {
                    if (Num_dumps > 1)
                    {
                              sprintf(Dumpfile,
"dump.%d", Curr_dump);
                              Curr_dump = (Curr_dump +
1) % Num_dumps;
                              Checkpoint(Dumpfile);
                    }
                    Checkpoint(Ckptfile);
          }
          else
          {
                    if (Doneflag)
                    {
                              if (Lastflag)

          Checkpoint(Ckptfile);
                              else
                                        if (Savesize)

          Printbest();
                    }
          }


          /* Intercambiar punteros para la proxima
generacion */
          temp = Old;
          Old = New;
          New = temp;

          /* actualizar contador de generaciones */
          Gen++;

}

/*  fin de archivo */
```

C.3.20 Módulo IN_EVAL.C

```
/*
 *  archivo:        in_eval.c
 *
 *  objetivo:       Lectura de parametros del modelo a
ser tratado por eval
 *              Este archivo es problema-dependiente
 *                                        Es decir debe
modificarse (o eliminarse) de trabajarse
 *                    con otro problema.
 *
 */
#include <alloc.h>
#include "extern.h"   /* Hay que incluirlo para
obtener Genes */
#include "in_eval2.h" /* parametros de campo */
/*-------------------------------------------------------
------------------*/
char *readFile(FILE *fp, char *line)
{
          char *c;

          do {
                    c= fgets(line, 100, fp);
          } while ( c!= NULL && (line[0]== '#' ||
line[0] == 10));
          return c;
}
/*-------------------------------------------------------
------------------*/
In_eval(argc,argv)
int argc;
char *argv[];
{
          char In_evalfile[40],line[200];
          FILE *fopen(), *fp;
          double *hcapa, dummy;

          int i,j;
          char msg[40];

          /* Determinar los nombres de los archivos */

          if (argc < 2)
          {
                    strcpy(In_evalfile,"campo");
          }
          else
          {
                    sprintf(In_evalfile, "campo.%s",
argv[1]);
          }

          /* leer parametros de campo */

          if ((fp = fopen(In_evalfile, "r")) == NULL)
          {
                    sprintf(msg, "In_eval: No puede
abrirse %s", In_evalfile);
                    Error(msg);
          }

          readFile(fp, line);
          sscanf(line, "%d", &nt);

          readFile(fp, line);
          sscanf(line, "%d", &ntmute);

          readFile(fp, line);
          sscanf(line, "%d", &l);

          readFile(fp, line);
          sscanf(line, "%lf",&dt);

          hcapa = malloc(Genes * sizeof(double) );  /*
cantGenes = cantCapas */
          if (hcapa == NULL) {
                    printf("in_eval: No hay memoria
para h\n");
                    abort();
          }
          for (i=0; i < Genes; i++) {
                    readFile(fp, line);
                    sscanf(line,"%lf", &hcapa[i]);
          }

          /* ro por capa es usado solo por la
propuesta 1. Lo salteo */

          for (i=0; i < Genes; i++) {
                    readFile(fp, line);
                    sscanf(line,"%lf", &dummy);
          }

          readFile(fp, line);
          sscanf(line,"%lf", &xsou);

          readFile(fp, line);
          sscanf(line,"%d", &jsou);

          readFile(fp, line);
          sscanf(line, "%lf", &xjsou);

          readFile(fp, line);
          sscanf(line, "%lf", &xjsoup1);

          readFile(fp, line);
          sscanf(line, "%lf", &xrec);

          readFile(fp, line);
          sscanf(line, "%d", &jrec);

          readFile(fp, line);
          sscanf(line, "%lf", &xjrec);

          readFile(fp, line);
          sscanf(line, "%lf", &xjrecp1);

          kx = malloc((Genes+1) * sizeof(double) );
/* cantGenes = cantCapas */
          if (kx == NULL) {
                    printf("in_eval: No hay memoria
para kx\n");
                    abort();
          }
          for (i=0; i <= Genes; i++) {
                    readFile(fp, line);
                    sscanf(line,"%d", &kx[i]);
          }
```

```
        readFile(fp, line);
        sscanf(line, "%d", &keyb);

        readFile(fp, line);
        sscanf(line, "%d", &kdel);

        source = malloc((nt+1) * sizeof(double));
        if (source == NULL) {
                printf("in_eval: No hay memoria
para source\n");
                abort();
        }
        for (i=1; i <= nt; i++) {
                readFile(fp, line);
                sscanf(line, "%lf", &source[i]);
        }

        readFile(fp, line);
        sscanf(line, "%lf", &is);

        recmed = malloc((nt+1) * sizeof(double));
        if (recmed == NULL) {
                printf("in_eval: No hay memoria
para recmed\n");
                abort();
        }
        for (i=1; i <= nt; i++) {
                readFile(fp, line);
                sscanf(line, "%lf", &recmed[i]);
        }
        /* Levanto la cantidad de materiales y las
velocidades de cada
        material (para propuestas 2 y 3). */
        readFile(fp, line);
        sscanf(line, "%d", &cantMateriales);
        velMaterial = malloc(cantMateriales *
sizeof(double));
```

```
        if (velMaterial == NULL) {
                printf("in_eval:No hay memoria
para velMaterial\n");
                abort();
        }
        for ( i=0; i < cantMateriales; i++) {
                readFile(fp, line);
                sscanf(line, "%lf",
&velMaterial[i]);
        }

        /* Levanto ro de cada material (propuestas 2
y 3)*/
        roMaterial = malloc(cantMateriales *
sizeof(double) );
        for ( i=0; i < cantMateriales; i++) {
                readFile(fp, line);
                sscanf(line, "%lf",
&roMaterial[i]);
        }

        fclose(fp);


        ronodo = malloc((l+2) * sizeof(double));  //
reservo memoria para ronodo

        /* convertimos hcapa en h */
        h = malloc((l+1) * sizeof(double));
        for (i=0; i < Genes; i++)
                for (j=kx[i]+1; j <= kx[i+1]; j++)
                        h[j] = hcapa[i];
        free(hcapa);
}

/** fin de archivo **/
```

C.3.21 Módulo INIT.C

```
/*
 *  archivo:        init.c
 *
 *  objetivo:       crear la poblacion inicial de
estructuras,
 *                  e inicilizar variables de
performance.
 *                  Esta rutina es llamada al inicio
de cada experimento.
 *
 */


#include "extern.h"
#include "dir.h"
#include "in_eval2.h"

Initialize()
{
        FILE *fp, *fopen();
        register int i, j;
        int status;                     /* indicates
fin de archivo en initfile    */
        char drive[MAXDRIVE], dir[MAXDIR],
file[MAXFILE], ext[MAXEXT];

        fnsplit(Minfile, drive, dir, file, ext);
        sprintf(Bestfile, "%s%d%s", file,
Experiment, ext);


        /* preparo para el nuevo experimento */
        Doneflag = 0;
        Curr_dump = 0;
        Bestsize = 0;
        /* setear proximo a mutar */
        if (M_rate < 1.0)
          Mu_next = ceil (log(Rand()) / log(1.0 -
M_rate));
        else
          Mu_next = 1;

        Trials = Gen = 0;
        Lost = Conv = 0;
        Plateau = 0;
        Spin = 0;
        Onsum = Offsum = 0.0;
        for (i=0; i<Windowsize; i++) Window[i] =
0.0;

        /* Inicializar poblacion */

        i = 0;                          /* estructura
actual */

        if (Initflag)                   /* tomar
algunas estructuras de Initfile */
        {
                if ((fp = fopen(Initfile, "r")) ==
NULL)
                {
                        char msg[40];
                        sprintf(msg,
"Initialize: no puede abrirse %s", Initfile);
                        Error(msg);
                }
```

```
                status = 1;
                if (Floatflag)
                {
                        for (j = 0; j < Genes &&
status != EOF; j++)
                        {
                                status =
fscanf(fp, "%lf", &Vector[j]);
                        }
                }
                else
                        status = fscanf(fp,
"%s", Bitstring);

                while (status != EOF && i <
Popsize)
                {
                        if (Floatflag)

        StringRep(Vector, Bitstring, Genes);

                        Pack(Bitstring,
New[i].Gene, Length);
                        New[i].Needs_evaluation
= 1;
                        i++;


                        /* leer la proxima
estructura */
                        if (Floatflag)
                                for (j = 0; j
< Genes && status != EOF; j++)

        status = fscanf(fp, "%lf", &Vector[j]);
                        else
                                status =
fscanf(fp, "%s", Bitstring);
                }
                fclose(fp);
        }

        /*****************************************
************/
        /* La semilla del generador de numeros
aleatorios se       */
        /* graba despues de la inicializacion de la
primera              */
        /* poblacion en cada experimento. El valor
grabado es           */
        /* usado como semilla en los experimentos
siguientes.          */
        /* La razon es permitir correr varios
experimentos con  */
        /* el mismo conjunto de parametros, y
comparar los              */
        /* resultados de corridas de un conjunto de
parametros           */
        /* con corridas realizadas con otro
conjunto.                 */
        /*****************************************
************/
        if ( Experiment > 0 ) Seed = Initseed;

        for (; i < Popsize; i++)    /* inicializar el
resto de la poblacion al azar */
```

```
        {
/* Generar valores enteros validos de acuerdo a la
cant de valores posibles */
                    for(j = 0; j < Genes; j++)
                    {
                              Vector[j] =
(double)Randint(0,Gene[j].values-1);
                    }
                    StringRep(Vector,Bitstring,Genes);
```

```
                    Pack(Bitstring , New[i].Gene ,
Length);
                    New[i].Needs_evaluation = 1;
          }

          Initseed = Seed;

}

/** fin de archivo **/
```

C.3.22 Módulo INPUT.C

```
/*
 *  archivo:        input.c
 *
 *  purpose:        Inicializar nombres de archivos,
leer archivos de entrada
 *                  e inicializar variables para esta
corrida.
 *
 *                  Ver en init.c la inicializacion de
variables para cada experimento
 */

#include <alloc.h>
#include "extern.h"
#include "in_eval2.h"

Input(argc,argv)
int argc;
char *argv[];
{
          FILE *fopen(), *fp;

          int i;
          char msg[40];
          long clock;                    /* fecha
actual                                   */

          long time();
          char *ctime();
          int ilog2();
          int cantbits();

          /* establecer nombres de archivos */

          if (argc < 2)
          {
                    strcpy(Infile,"in");
                    strcpy(Outfile,"out");
                    strcpy(Ckptfile,"ckpt");
                    strcpy(Minfile,"min");
                    strcpy(Logfile, "log");
                    strcpy(Initfile, "init");
                    strcpy(Schemafile, "schema");
                    strcpy(Templatefile, "template");
          }
          else
          {
                    sprintf(Infile, "in.%s", argv[1]);
                    sprintf(Outfile, "out.%s",
argv[1]);
                    sprintf(Ckptfile, "ckpt.%s",
argv[1]);
                    sprintf(Minfile, "min.%s",
argv[1]);
                    sprintf(Logfile, "log.%s",
argv[1]);
                    sprintf(Initfile, "init.%s",
argv[1]);
                    sprintf(Schemafile, "schema.%s",
argv[1]);
                    sprintf(Templatefile,
"template.%s", argv[1]);
          }

          strcpy(Bestfile, Minfile);

          /* leer los parametros de infile */

          if ((fp = fopen(Infile, "r")) == NULL)
          {
                    sprintf(msg, "Input: no puede
abrirse %s", Infile);
                    Error(msg);
          }
          fscanf(fp,IN_FORMAT,IN_VARS);
          Seed = OrigSeed;
          fclose(fp);

          /* activar las opciones */
          for (i=0; Options[i] != '\0'; i++)
                    Setflag(Options[i]);
          if (Displayflag)
                    Traceflag = 0;

          /* Bytes es el tama¤o de cada cromosoma
empaquetado */
          Bytes = Length / CHARSIZE;
          if (Length % CHARSIZE) Bytes++;

          /* leer el archivo template. Solo se usa en
la representacion de punto
             flotante */
          if (Floatflag)
```

```
          {
                    if ((fp = fopen(Templatefile,
"r")) == NULL)
                    {
                              sprintf(msg, "Input: no
puede abrirse %s", Templatefile);
                              Error(msg);
                    }
                    fscanf(fp, "genes: %d ", &Genes);
                    Gene = (GENESTRUCT *)
calloc((unsigned) Genes,
                              sizeof(GENESTRUCT));

                    for (i=0; i<Genes; i++)
                    {
                              fscanf(fp, " gene %*d");
                              fscanf(fp, " min: %lf",
&Gene[i].min);
                              fscanf(fp, " max: %lf",
&Gene[i].max);
                              fscanf(fp, " values:
%lu", &Gene[i].values);
                              fscanf(fp, " format:
%s", Gene[i].format);
                              Gene[i].bitlength =
cantbits(Gene[i].values);
                              Gene[i].incr =
(Gene[i].max - Gene[i].min) /

          (Gene[i].values - 1);
                    }
                    fclose(fp);
          }
          /* reservar memoria para las estructuras de
tama¤o variable */

          /* usado para representacion de punto
flotante del cromosoma */
          Vector = (double *) calloc((unsigned) Genes,
sizeof(double));

          /* usado para la representacion en cadena de
'0's y '1's */
          Bitstring = malloc((unsigned) (Length+1));
          Bitstring[Length] = '\0';

          if (Bitstring == NULL) {
                    printf("input: No hay memoria para
Bitstring\n");
                    abort();
          }
          /* arreglos de la poblacion */
          Old = (STRUCTURE *) calloc((unsigned)
Popsize, sizeof(STRUCTURE));
          New = (STRUCTURE *) calloc((unsigned)
Popsize, sizeof(STRUCTURE));

          for (i=0; i<Popsize; i++)
          {
                    Old[i].Gene = malloc((unsigned)
Bytes);
                    New[i].Gene = malloc((unsigned)
Bytes);

          }

          /* usado para computar Worst.
            (El peor de las ultimas Windowsize
generaciones) */
          if (Windowsize)
          Window = (double *) calloc((unsigned)
Windowsize, sizeof(double));

          /* usado para guardar las mejores
estructuras */
          if (Savesize)
           Bestset = (BESTSTRUCT *) calloc((unsigned)
Savesize, sizeof(BESTSTRUCT));

          for (i=0; i<Savesize; i++)
                    Bestset[i].Gene =
malloc((unsigned) Bytes);

          /* mostrar los parametros de entrada */
          if (Traceflag) printf(OUT_FORMAT, OUT_VARS);

          /* Sobreescribir el archivo de salida a
menos que sea un reinicio */
          if (!Restartflag)
          {
                    if ((fp = fopen(Outfile, "w")) ==
NULL)
                    {
```

```
                                sprintf(msg, "Input: No
puede abrirse %s", Outfile);
                                Error(msg);
                        }
                        fclose(fp);
                }

                /* guardar esta activacion en un log */
                if (Restartflag)
                {
                        if (Logflag)
                        {
                                if ((fp = fopen(Logfile,
"a")) == NULL)
                                {

                sprintf(msg,"Input: No puede abrirse %s",
Logfile);
                                        Error(msg);
                                }
                                fprintf(fp, "%s
Recomenz¢ ", argv[0]);
                                time(&clock);
                                fprintf(fp, "%s",
ctime(&clock));
                                fclose(fp);
                        }
                }
                else
                {
                        if (Logflag)
                        {
                                if ((fp = fopen(Logfile,
"a")) == NULL)
                                {

                sprintf(msg,"Input: No puede abrirse %s",
Logfile);
                                        Error(msg);
                                }
                                fprintf(fp, "%s comenz¢
", argv[0]);
                                time(&clock);
                                fprintf(fp, "%s",
ctime(&clock));
                                fclose(fp);
                        }
                }
}


int ilog2(n)
        unsigned long n;
{
        register int i;

        if (n <= 0)
        {
                printf("La cantidad de valores es
%d, debe ser positiva!\n", n);
                abort();
        }

        i = 0;
        while ((int) (n & 1) == 0)
        {
                n >>= 1;
                i++;
        }
        return(i);
}


Setflag(c)
char c;
{
        switch (c) {
        case 'a' :
                Allflag = 1;
                break;
        case 'b' :
                Bestflag = 1;
                break;
        case 'c' :
                Collectflag = 1;
                Convflag = 1;
                break;
        case 'C' :
                Collectflag = 1;
                break;
        case 'd' :
                Dumpflag = 1;
                break;
        case 'D' :
                Displayflag = 1;
                break;
        case 'e' :
                Eliteflag = 1;
                break;
        case 'f' :
                Floatflag = 1;
                break;
        case 'g' :
                Grayflag = 1;
                break;
        case 'i' :
                Initflag = 1;
                break;
        case 'I' :
                Interflag = 1;
                Displayflag = 1;
                break;
        case 'l' :
                Logflag = 1;
                break;
        case 'L' :
                Lastflag = 1;
                break;
        case 'M' :
                Maxflag = 1;
                break;
        case 'o' :
                Onlnflag = 1;
                break;
        case 'O' :
                Offlnflag = 1;
                break;
        case 'r' :
                Restartflag = 1;
                break;
        case 'R' :
                Rankflag = 1;
                break;
        case 's' :
                Schemflag = 1;
                break;
        case 't' :
                Traceflag = 1;
                break;
        }
}

int cantbits(n)
        unsigned long n;
{
        unsigned long mask;
        int otro1, bits, i, totbits;

        otro1=0;
        n = n - 1;  /* tengo que representar de 0 a
n-1 */
        totbits = sizeof(long) * CHARSIZE;

        /* busca el primer 1 contando de la
izquierda */
        mask = (long)1 << (totbits - 1);
        for (i=0; !(n & mask); i++, mask >>=1);
        bits = totbits -i;

        /* se fija si hay otros unos */
        for (i++, mask>>=1; i < totbits; i++, mask
>>=1) {
                if (n & mask) otro1 = 1;
        }

        if (otro1) bits++;
        return(bits);

}

/** fin de archivo **/
```

C.3.23 Módulo MEASURE.C

```
/*
 *  archivo:        measure.c
 *
 *  objetivo:       calcular mediciones de performance
y adjuntarlas al archivo
 *                  output.
 *
 */

#include "extern.h"

#define DISPMEAS 5

Measure()
{
        double New_worst();
        FILE *fp, *fopen();
        register int i;
        register int w;
        register double performance;
        int j;

        for (i=0; i<Popsize; i++)
        {
                /* estadisticas de la poblacion
actual */
                performance = New[i].Perf;
                if (i>0)
                {
                        Ave_current_perf +=
performance;
                        if (BETTER(performance,
Best_current_perf))
                        {

        Best_current_perf = performance;
```

```
                                        Best_guy = i;
                                }
                                if
(BETTER(Worst_current_perf, performance))

          Worst_current_perf = performance;
                    }
                    else
                    {
                                Best_current_perf =
performance;
                                Worst_current_perf =
performance;
                                Ave_current_perf =
performance;
                                Best_guy = 0;
                    }

          }

          Ave_current_perf /= Popsize;

          /* actualizar Worst */
          if (Windowsize)
          {
                    /* Worst = worst en las ultimas
(Windowsize) generaciones */
                    w = Gen % Windowsize;
                    Window[w] = New_worst();
                    Worst = Window[0];
                    for (i=1; i < Windowsize; i++)
                                if (BETTER(Worst,
Window[i])) Worst = Window[i];
          }
          else
                    if (BETTER(Worst,
Worst_current_perf))
                                Worst = New_worst();

          /* actualizar medidas de performance global
*/
          Online = Onsum / Trials;
          Offline = Offsum / Trials;

          if (Traceflag)
          {
                    printf("     Gen %d     Evals
%d\n",Gen,Trials);
                    if (Onlnflag) printf("     Online
%e\n", Online);
                    if (Offlnflag) printf("
Offline %e\n", Offline);
          }

          if (Displayflag)
          {
                    static firstflag = 1;
                    if (firstflag)
                    {
                                firstflag = 0;
                                move(DISPMEAS - 1,0);
                                printw(" Gens Evals Perd
");
                                printw("Conv  Sesgo
Online      ");
                                printw("Offline
Mejor     Promedio");
                    }

                    move(DISPMEAS,0);
                    clrtoeol();
                    Converge();
                    printw("%4d %6d %4d ",
                                Gen, Trials, Lost);
                    printw("%4d  %5.3f %11.4f  ",
                                Conv, Bias, Online);
                    printw("%11.4f  %11.6f  %11.6f",
                                Offline, Best,
Ave_current_perf);

                    move(DISPMEAS+2,0);
                    clrtoeol();
                    printw(" Mejor estructura actual:
%3d  ", Best_guy);
                    printw("Performance: %0.6f ",
New[Best_guy].Perf);
                    move(DISPMEAS+3,0);
                    clrtoeol();
                    Unpack(New[Best_guy].Gene,
Bitstring, Length);
                    if (Floatflag)
                    {
                                FloatRep(Bitstring,
Vector, Genes);
                                for (j=0; j<Genes; j++)

          printw(Gene[j].format, Vector[j]);
                    }
                    else
                    {
                                printw("%s", Bitstring);
                    }
                    refresh();
          }

          if ( Interval && Collectflag && ((Trials >=
Plateau) || Doneflag))
          {
```

```
                    /* agregar mediciones al archivo
output */
                    Converge();
                    fp = fopen(Outfile, "a");
                    fprintf(fp,OUT_F2, OUT_V2);

                    fclose(fp);
                    Plateau =
(Trials/Interval)*Interval + Interval;
          }

          if (Logflag  && (Spin >= Maxspin))
          {
                    fp = fopen(Logfile, "a");
                    fprintf(fp, "Experimento %ld  ",
Experiment);
                    fprintf(fp, "SPINNING en la Gen
%ld, ",Gen);
                    fprintf(fp, "despu,s de  %ld
Evaluaciones\n", Trials);
                    fclose(fp);
          }

          if ( Interflag && (Spin >= Maxspin))
          {
                    move(22,0);
                    clrtoeol();
                    printw("SPINNING en la Gen %ld,
",Gen);
                    printw("despu,s de %ld
Evaluaciones\n", Trials);
                    refresh();
          }

}


double New_worst()
{
          double delta;

          /* Devolver un valor un poquito peor que
than Worst_current_perf */
/**** un poquito peor porque si comparo contra el peor
la distancia daria 0 ****/

          if (Maxflag)
                    delta = 1.0e-4;
          else
                    delta = -1.0e-4;

          if (Worst_current_perf == 0.0) return (-
delta);

          if (Worst_current_perf > 0.0)
                    return (Worst_current_perf*(1.0 -
delta));

          return (Worst_current_perf*(1.0 + delta));
}



static char BIT[CHARSIZE] ={ '\200', '\100', '\040',
'\020',
                              '\010', '\004', '\002',
'\001'};

Converge()                                  /* medir la
convergencia de la poblacion  */
/* MODIFICADO PARA MEDIR A NIVEL DE MATERIAL */
/* NECESITO CONOCER EL NRO DE MATERIALES Y DEFINIR UN
VECTOR DE ESA DIMENSION */
/* este valor lo tengo en Gene[j].values */

{
          register int i,j;
          register int cantMatGanador; /* cantidad de
representantes del material

                                                ganador en un
gen dado */
          FILE *fp, *fopen();
          int *cantMatEnGen;

          Bias = 0.0;
          Lost = Conv = 0;
          if (!Convflag) return;


          /* en lugar de usar length usar la cantidad
de genes */
          for (j = 0; j < Genes; j++)
          {
                    /* Inicializo contadores en 0 */
                    cantMatEnGen =
malloc(Gene[j].values * sizeof(int) );
                    for (i=0; i < Gene[j].values; i++)
{
                              cantMatEnGen[i]=0;
                    }

                    for (i=0; i < Popsize; i++) {
                              /* ver si se puede
optimizar para que  no haya que desempaquetar
                              en mismo gen popsize
veces
                              */
```

```
                                Unpack(New[i].Gene,
Bitstring, Length);
                                if (Floatflag)
          FloatRep(Bitstring, Vector, Genes);

          cantMatEnGen[Vector[j]]++;
                    }

                    cantMatGanador=0;
                    for (i=0; i < Gene[j].values; i++)
{
                                if (cantMatEnGen[i] >
cantMatGanador) {

          cantMatGanador=cantMatEnGen[i];
                                }
                    }


                    Bias += cantMatGanador;
```

```
                    Lost += (cantMatGanador ==
Popsize);
                    Conv += (cantMatGanador >= Popsize
- FEW);
                    free(cantMatEnGen);
          }

          Bias /= (Popsize*Genes);

          if (Logflag && (Lost==Length))
          {
                    fp = fopen(Logfile, "a");
                    fprintf(fp, "CONVERGIO en Gen %1d,
",Gen);
                    fprintf(fp, "despu,s de %1d
Evaluaciones\n", Trials);
                    fclose(fp);
          }

}

/** fin de archivo **/
```

C.3.24 Módulo MUTATE.C

```
/*
 *  archivo:        mutate.c
 *
 *  purpose:        Mutar la poblacion
 *
 *        La variable global Mu_next indica el proximo
gen a mutar tratando
 *        la poblacion como un arreglo lineal de
genes.
 */

#include "extern.h"


Mutate()
{
          static int genesPop;          /* cantidad de
genes en la poblacion */
          register int i;               /* indice del
individuo a mutar */
          register int j;               /* indice del
gen a mutar */
          register double k;   /* nuevo valor para el
gen */
          register int open;  /* indica si el
cromosoma ya fue desempaquetado */
          static int firstflag = 1;

          if (firstflag)
          {
                    genesPop = Gapsize*Popsize*Genes +
0.5; /**** 0.5 para redondear ****/
                    firstflag = 0;
          }

          if (M_rate > 0.0)
          {
                    open = -1;
/**** Mu_next se inicializa al ppio. del pgm. o en la
generacion anterior ****/
                    while (Mu_next<genesPop)
                    {
                                i = Mu_next/ Genes; /*
estructura a mutar */
                                j = Mu_next % Genes;
          /* gen a mutar dentro de la estructura */
```

```
                                if (open != i)   /* hay
que desempaquetar la estructura i */
                                {
                                          Unpack
(New[i].Gene , Bitstring, Length);
                                          open = i;
                                }

                                FloatRep(Bitstring,
Vector, Genes);
                                k =
Randint(0,Gene[j].values-1);

                                if (k != Vector[j])  /*
es una mutacion verdadera */
                                {
                                          Vector[j] = k;

          New[i].Needs_evaluation = 1;
                                }
                                if
(New[i].Needs_evaluation) {

          StringRep(Vector, Bitstring, Genes);
                                          Pack (
Bitstring , New[i].Gene , Length);
                                }

                                /* calcular proximo gen
a mutar */
                                if (M_rate < 1.0)
                                          Mu_next +=
ceil (log(Rand()) / log(1.0 - M_rate));
                                else
                                          Mu_next += 1;
                    }
                    /* ajustar Mu_next para la proxima
generacion */
                    Mu_next -= genesPop;
          }
}

/** fin de archivo **/
```

C.3.25 Módulo PRESION.C

```
/*
 *  archivo:        presion.c
 *
 *  objetivo:   Calcular las presiones generadas por
un juego de velocidades.
 *
 */
#include <alloc.h>
#include <math.h>
#include <stdio.h>
#include <assert.h>
#include <stdlib.h>
#include "presion.h"
#include "in_eval2.h"
/*-----------------------------------------------------
---------------*/
double una_presion(unm1, un, unp1, paso, constk,
velnodo)
          double *unm1, *un, *unp1, *constk, *velnodo;
          int paso;
{
          double aux1, aux2;
          int k;

          assert(heapcheck()==_HEAPOK);

          /* condicion de bordes */
```

```
          unp1[1]= borde_sup(unm1[1], un[1], un[2],
constk[1]);
          unp1[l+1]= borde_inf( unm1[l+1], un[l],
un[l+1], constk[l+1]);

          /* agregar la contribucion de la fuente a
cada borde */
          /* primero al superior */
          unp1[1]+= fuente_en_borde(paso, 1);
          unp1[1]/= 1 + constk[1];
          /* y ahora al inferior */
          unp1[l+1]+= fuente_en_borde(paso, l+1);
          unp1[l+1]/= 1 + constk[l+1];

          /* calculo de la presion de los nodos
interiores */
          for (k=2; k <= l; k++) {
                    unp1[k]= 2 * un[k] - unm1[k] + 2 *
constk[k] * pow(dt, 2) *
                    ( (un[k+1] - un[k]) / ( h[k] *
ronodo[k] ) -
                    ( un[k] - un[k-1] ) / ( h[k-1] *
ronodo[k-1] ));


                    /* agregar la contribucion de la
fuente */
```

```
                    unp1[k]+= fuente_interior( paso,
k, constk[k], velnodo);
          }

          /* Interpolacion lineal de las presiones en
los dos nodos que rodean al
                    receptor */
          aux1= (xjrecp1 - xrec) / h[jrec] *
unp1[jrec];
          aux2= (xrec - xjrec) / h[jrec] *
unp1[jrec+1];

          assert(heapcheck()==_HEAPOK);
          return (aux1 + aux2);
}

/*-------------------------------------------------
----------------*/
void presiones(velocidad, capas, presion, nt)
          /* velocidades por cada capa, parametro de
entrada */
          double *velocidad;
          int capas;  /* cantidad de capas */

          /* presiones calculadas a partir de las
velocidades */
          double *presion;
          int nt;             /* cantidad de
intervalos en el tiempo */
{

          int i;              /* auxiliar */
          double *aux;  /* puntero auxiliar para hacer
swap */
          double *unm1; /* presiones en el paso n-1 */
          double *un; /* presiones en el paso n   */
          double *unp1; /* presiones en el paso n+1 */
          double *velnodo;  /* velocidades en nodos */
          double *constk; /* constantes auxiliares */

          unm1= malloc((l+2)*sizeof(double)); /*
presiones en el paso n-1 */
          un= malloc((l+2)*sizeof(double));   /*
presiones en el paso n   */
          unp1= malloc((l+2)*sizeof(double)); /*
presiones en el paso n+1 */
          velnodo= malloc((l+2)*sizeof(double));  /*
velocidades en nodos */
          constk= malloc((l+2)*sizeof(double)); /*
constantes auxiliares */
          vel_en_nodos(velocidad, capas, velnodo);

          assert(heapcheck()==_HEAPOK);

          for ( i=1; i <= (l+1); i++) {
                    unm1[i]= 0;
                    un[i]= 0;
          }

          /* Condiciones iniciales */
          presion[1]= 0;
          presion[2]= 0;

          /* Tiempo de espera desde que comienza la
explosion hasta que se comienza
             a escuchar (mute) */
          for ( i= 3; i <= ntmute; i++)
                    presion[i]= 0;

          /* constantes auxiliares */
          constk[1]= velnodo[1] * dt / h[1];
          constk[l+1]= velnodo[l] * dt / h[l];
          for ( i = 2; i <= l; i++)
                    constk[i]= ( pow(velnodo[i], 2) *
pow(velnodo[i-1], 2) * ronodo[i] * ronodo[i-1]) /

          (h[i-1] * pow(velnodo[i], 2) * ronodo[i] +
h[i] * pow(velnodo[i-1], 2) * ronodo[i-1]);

          /* calcula la presion en el paso i */
          for ( i= max(ntmute+1,3); i <= nt; i++) {
                    presion[i]= una_presion(unm1, un,
unp1, i, constk, velnodo);
                    aux= unm1;    /* lo guardo para
despues */
                    unm1= un;     /* avanza un paso en
el tiempo */
                    un= unp1;     /*    "    "   "   "
"    "   */
                    unp1= aux;    /* no interesan los
valores pero si el espacio de memoria */
          }

          free(unm1);
          free(un);
          free(unp1);
          free(velnodo);
          free(constk);
          /* el resultado va en 'presion' */

          assert(heapcheck()==_HEAPOK);
}
/*-------------------------------------------------
----------------*/
void vel_en_nodos(velocidad, capas, velnodo)
double *velocidad;
int capas;
double *velnodo;
{
          int c, nodo;
          for (c=0; c <capas; c++)
                    for (nodo=kx[c]+1; nodo <=
kx[c+1]; nodo++)
                              velnodo[nodo] =
velocidad[c];
          velnodo[l+1]= velocidad[capas-1];
}
/*-------------------------------------------------
----------------*/
double borde_sup(unm1, unx1, unx2, cte)
/* Calcula la presion en el borde superior (sin
fuente) */
double unm1;
double unx1;
double unx2;
double cte;
{
          double presion;
          switch (keyb) {
                    case 0: /* Condicion de bordes de
Dirichlet (superficie libre) */
                                        presion = 0;
                                        break;
                    case 1: /* Condicion de bordes
absorbente */
                                        presion = 2 *
unx1 - unm1 + 2 * pow(cte, 2) * (unx2 - unx1) +

          cte * unm1;
                                        break;
                    case 2: /* Condicion de bordes de
Neumann  */
                                        presion = 2 *
unx1 - unm1 + 2 * pow(cte, 2) * (unx2 - unx1);
                                        break;
          }
          return presion;
}
/*-------------------------------------------------
----------------*/
double borde_inf(unm1, unx1, unx2, cte)
/* Calcula la presion en el borde inferior (sin
fuente) */
double unm1; /* presion en el paso anterior */
double unx1; /* presion en el anteultimo nodo */
double unx2; /* presion en el ultimo nodo */
double cte;  /* constante ad hoc */
{
  double presion;
  presion = 2 * unx2 - unm1 + 2 * pow(cte, 2) * (unx1
- unx2) +

          cte * unm1;
  return presion;
}
/*-------------------------------------------------
----------------*/
/* Las tres rutinas de fuente podrian unificarse */
double fuente_en_borde(paso, nodo)
/* Contribucion de la fuente en el borde indicado por
nodo en el paso 'paso' */
int paso;
int nodo;
{
          double eval, presion;
          switch (kdel) {
                    case 0: /* delta de Dirac en xsou
* source[n] */
                              if (1 == jsou)
                                        eval =
(xjsoup1 - xsou) / h[jsou];
                              else if (1 == jsou+1)
                                                  eval
= (xsou - xjsou) / h[jsou];
                                         else
                                                  eval
= 0;
                              presion = 2 * pow(dt, 2)
* source[paso] * eval / h[nodo];
                              break;

                    case 1: /* derivada del delta de
Dirac en
                                                  .5 * (xsou
+ xsou + h) * source[n] */
                              presion = 2 * pow(dt, 2)
* source[paso] / pow(h[nodo], 2);
                              if (1 == jsou) {
                                        /* no hacemos
nada */
                              }
                              else if (1 == jsou + 1)
{
                                         presion *= -
1;
                                         }
                                         else {

          presion = 0;
                                         }
                              break;

                    case 2: /* 2 delta de Dirac en
xsou * source[n] */
                              presion = 2 * pow(dt, 2)
* source[paso] * h[nodo] / 2.0;
                              if (1 == jsou - is) {
```

```
                                        /* no hacemos
nada */
                                }
                                else if (1 == jsou + is)
{
                                        presion *= -
1;
                                }
                                else {

          presion = 0;
                                }
                                break;
          }
          return presion;
}

/*-------------------------------------------------
-------------------*/
double fuente_interior(paso, nodo, cte, velnodo)
/* Contribucion de la fuente en el un nodo interior */
int paso;
int nodo;
double cte; /* constante auxiliar */
double *velnodo;
{
          double eval, presion;
          switch (kdel) {
                    case 0: /* delta de Dirac en xsou
* source[n] */

                                if (nodo == jsou)
                                        eval =
(xjsoup1 - xsou) / h[jsou];
                                else if (nodo == jsou+1)
                                                eval
= (xsou - xjsou) / h[jsou];
                                        else
                                                eval
= 0;

                                presion = 2 * pow(dt, 2)
* cte * source[paso] * eval /

(pow(velnodo[jsou], 2) * ronodo[jsou]);
                                break;

                    case 1: /* derivada del delta de
Dirac en
                                                .5 * (xsou
+ xsou + h) * source[n] */
```

```
                                if (nodo == jsou) {
                                                presion = 2 *
pow(dt, 2) * cte * source[paso] /

(pow(h[jsou], 2) * pow(velnodo[jsou], 2) *
ronodo[jsou]);
                                }
                                else if (nodo == jsou +
1) {
                                                presion = 2 *
pow(dt, 2) * cte * source[paso] /

(pow(h[jsou], 2) * pow(velnodo[jsou], 2) *
ronodo[jsou]);
                                                presion *= -
1;
                                                }
                                                else {

          presion = 0;
                                                }
                                break;

                    case 2: /* 2 delta de Dirac en
xsou * source[n] */

                                if (nodo == jsou - is) {
                                                presion = 2 *
pow(dt, 2) * cte * source[paso] *

(pow(velnodo[jsou], 2) * ronodo[jsou]) / 2.0;
                                }
                                else if (nodo == jsou +
is) {
                                                presion = 2 *
pow(dt, 2) * cte * source[paso] *

(pow(velnodo[jsou], 2) * ronodo[jsou]) / 2.0;
                                                presion *= -
1;
                                                }
                                                else {

          presion = 0;
                                                }
                                break;
          }
          return presion;
}

/*** fin de archivo ***/
```

C.3.26 Módulo RESTART.C

```
/*
 *  archivo:        restart.c
 *
 *  objetivo:       correr AG a partir de una corrida
anterior
 *
 */

#include <dir.h>
#include "extern.h"

extern void Readbest();


Restart()
{
          FILE *fp, *fopen();
          int i;
          char msg[40];
          char drive[MAXDRIVE], dir[MAXDIR],
file[MAXFILE], ext[MAXEXT];

          fp = fopen(Ckptfile, "r");
          if (fp == NULL)
          {
                    sprintf(msg,"Restart: Ckptfile %s
no se encuentra",
                                Ckptfile);
                    Error(msg);
          }

          fscanf(fp, "Experimentos %d ", &Experiment);
          fscanf(fp, "On_line total   %lf ",
&Totonline);
          fscanf(fp, "Off_line total   %lf ",
&Totoffline);
          fscanf(fp, "Generaci¢n %d ", &Gen);
          fscanf(fp, "Performance On_line  %lf ",
&Onsum);
```

```
          fscanf(fp, "Performance Off_line %lf ",
&Offsum);
          fscanf(fp, "Evaluaciones  %d ", &Trials);
          fscanf(fp, "Pr¢xima estad¡stica %d ",
&Plateau);
          fscanf(fp, "Mejor %lf ", &Best);
          fscanf(fp, "Peor   %lf ", &Worst);
          fscanf(fp, "Cantidad de generaciones desde
la £ltima evaluaci¢n %d ", &Spin);
          fscanf(fp, "Ultimo vuelco %d ", &Curr_dump);
          fscanf(fp, "Mu_prox %d ", &Mu_next);
          fscanf(fp, "Semilla Aleatoria %lu ", &Seed);
          fscanf(fp, "Semilla Inicializadora %lu ",
&Initseed);

          fscanf(fp, " Window ");
          for (i=0; i<Windowsize; i++) fscanf(fp,
"%lf", &Window[i]);

          for (i=0; i<Popsize; i++)
          {
                    fscanf(fp, "%s", Bitstring);
                    fscanf(fp, " %lf ", &New[i].Perf);
                    Pack(Bitstring, New[i].Gene,
Length);
                    fscanf(fp, " %d ",
&New[i].Needs_evaluation);
          }
          fclose(fp);

          fnsplit(Minfile, drive, dir, file, ext);
          sprintf(Bestfile, "%s%d%s", file,
Experiment, ext);

          Readbest();

}

/** fin de archivo **/
```

C.3.27 Módulo SELECT.C

```
/*
 *  archivo:        select.c
 *
 *  objetivo:       seleccionar los individuos para la
nueva generacion
 *
 */
#include <alloc.h>
#include "extern.h"
```

```
Select()
{
          static firstflag = 1;          /* indica
primera ejecucion                */
          static int *sample; /* apunta a las
estructuras seleccionadas        */
          double expected;    /* numero especrado de
hijos                            */
          double factor;                 /*
normalizador para valor esperado */
          double perf;                   /* mejor
performance siguiente (para ranking)                */
          double ptr;                    /* determina
la distribucion de la parte fraccionaria */
          double rank_max;    /* max numero de hijos
(en ranking) */
          double sum;
          int best;           /* indice a la mejor
estructura siguiente             */
          register int i;
          register int j;
          register int k;
          register int temp;  /* usado para
intercambiar punteros            */

          if (firstflag)
          {
                    sample = (int *) calloc((unsigned)
Popsize, sizeof(int));
                    firstflag = 0;
          }

          if (Rankflag)
          {
                    /* Asignar a cada estructura su
nro de orden (rank) dentro de la poblacion. */
                    /* rank = Popsize-1 para el mejor,
rank = 0 para el peor            */
                    /**** Se usa el campo
Needs_evaluation para guardar el valor de rank ****/

                    /* limpiar el campo donde se
guarda el valor de rank */
                    for (i=0; i<Popsize; i++)
                              Old[i].Needs_evaluation
= 0;

                    for (i=0; i < Popsize-1; i++)
                    {
                              /* buscar la mejor i-
esima estructura */
                              best = -1;
                              perf = 0.0;
                              for (j=0; j<Popsize;
j++)
                              {
                                        if
(Old[j].Needs_evaluation == 0 &&

(best == -1 || BETTER(Old[j].Perf,perf)))
                                        {
                                                  perf
= Old[j].Perf;
                                                  best
= j;
                                        }
                              }
                              /* ponerle el orden a la
estructura */

          Old[best].Needs_evaluation = Popsize -1 - i;
                    }
                    /* normalizador para las
probabilidades de seleccion por ranking */
                    rank_max = 2.0 - Rank_min;
                    /**** Calcula el tama¤o del slot
patron ****/
                    factor = (rank_max - Rank_min) /
(double) (Popsize -1);
          }
          else
          {
                    /* normalizador para las
probabilidades de seleccion proporcional */
                    /**** Calcula el tama¤o del slot
patron ****/
                    factor = Maxflag ?
1.0/(Ave_current_perf - Worst) :
                                        1.0/(Worst
- Ave_current_perf);
          }

          /* Algoritmo de muestreo estocastico de
James E. Baker */

          k=0;                /* indice de la proxima
estructura a elegir */

          ptr = Rand();   /* girar la ruleta una vez
*/

          for (sum=i=0; i < Popsize; i++)
          {
                    if (Rankflag)
                    {
                              /**** Mapea los
individuos en forma lineal entre
                                   Rank_min y Rank_max
****/
                              expected = Rank_min +
Old[i].Needs_evaluation * factor;
                    }
                    else
                    {
                              /**** Mapea en forma
proporcional a la distancia con
                                   el peor individuo
****/
                              if (Maxflag) {
                                        if
(Old[i].Perf > Worst)
                                          expected =
(Old[i].Perf - Worst) * factor;
                                        else expected
= 0.0;
                              }
                              else {
                                        if
(Old[i].Perf < Worst)
                                          expected =
(Worst - Old[i].Perf) * factor;
                                        else expected
= 0.0;
                              }
                    }
/**** Asigna entre floor y ceil de la cantidad
esperada de hijos ****/
                    for (sum += expected; sum > ptr;
ptr++){
                      sample[k++] = i;
                    }
          }

          if (k != Popsize) {
            printf("select: Se seleccionaron %d
individuos en lugar de %d\n", k, Popsize);
            abort();
          }

          /* mezclar al azar los punteros de las
nuevas estructuras */
/**** Pensar en que pasa si un individuo recibe mas de
una copia en la
          seleccion. Luego, en la cruza, el padre y
la madre serian muy probablemente
          el mismo. Por eso se mezcla! ****/
          for (i=0; i<Popsize; i++)
          {
                    j = Randint(i,Popsize-1);
                    temp = sample[j];
                    sample[j] = sample[i];
                    sample[i] = temp;
          }

          if (Gapsize<1.0)              /* Salto
generacional */
                    Gap(sample);

          /* finalmente, formar la nueva poblacion */
          for (i=0; i<Popsize; i++)
          {
                    k = sample[i];
                    for (j=0; j<Bytes; j++)
                    {
                              New[i].Gene[j] =
Old[k].Gene[j];
                    }
                    New[i].Perf = Old[k].Perf;
                    New[i].Needs_evaluation = 0;
          }
}

/* Elegir sobrevivientes de la vieja poblacion
uniformemente sin reposicion */

Gap(sample)
int sample[];
/**** (Gap == 0) => no hay seleccion
                              no hay cruza
            Se destruye una porcion de la poblacion
seleccionada y se reemplaza
            por indiv. de la poblacion ORIGINAL sin
tener en cuenta su aptitud ****/
{
          static firstflag = 1;
          static int *survivors;        /* permutacion
al azar de 0 .. Popsize-1 */
          register int i,j;
          int temp;             /* para swapear
                                   */

          if (firstflag)
          {
                    survivors = (int *)
calloc((unsigned) Popsize, sizeof(int));
                    firstflag = 0;
          }

          /* Mezclar uniformemente */
          for (j=0; j<Popsize; j++) survivors[j]=j;
          for (j=0; j<Popsize; j++)
          {
                    i = Randint(j, Popsize-1);
                    temp = survivors[i];
                    survivors[i] = survivors[j];
                    survivors[j] = temp;
          }
```

```
        /* nuevos sobrevivientes elegidos */
        for (i=Gapsize*Popsize; i<Popsize; i++)
                sample[i] = survivors[i];
```

```
}
/*** fin de archivo ***/
```

C.4 Programa AG3

C.4.1 Módulo DEFINE.H

```
/*
 *  archivo:        define.h
 *
 *  objetivo:     definiciones globales
 */

#include <math.h>
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include "format.h"
#include <conio.h>

#define clrtoeol        clreol
#define refresh()       fflush(stdout)
#define printw          printf
#define scanw           scanf
#define getw            gets

/********************** CONSTANTES
*****************************/

#define CHARSIZE 8

/* usados en generador aleatorio de n£meros */
#define MASK 2147483647
#define PRIME 65539
#define SCALE 0.4656612875e-9



/************************ TIPOS
*****************************/

/* tipo para almacenar cada individuo de la poblaci¢n
*/
typedef struct {
          char *Gene;
          double Perf;
          int Needs_evaluation;
          int *BondadAlelo;
} STRUCTURE;

/* tipo para almacenar las mejores estructuras */
typedef struct {
          char *Gene;
          double Perf;
          int Gen;
          int Trials;
} BESTSTRUCT;

/* registro para interpretar el formato empaquetado de
acuerdo con el template definido */
typedef struct {
          double min;
          double max;
          unsigned long values;
          char format[16];
          double incr;
          int bitlength;
} GENESTRUCT;

/************************ MACROS
*****************************/

/* Comparaci¢n de dos valores de performance */
#define BETTER(X,Y)      (Maxflag ? (X) > (Y) : (X) <
(Y))

/* Un alelo converge si todos los individuos - FEW
/* tiene el mismo valor en esa posici¢n.  */
#define FEW              (Popsize/20)

/*****************************************************
***********/
/*       Rand obtiene un n£mero de una serie psuedo
aleatoria */
/*       en el intervalo [0,1).

          */
/*                  Randint devuelve un entero en el
intervalo [low,high]        */
/*****************************************************
***********/

#define Rand()   (( Seed = ( (Seed * PRIME) & MASK) ) *
SCALE )

#define Randint(low,high) ( (int) (low + (high-low+1)
* Rand()))

/** fin de archivo **/
```

C.4.2 Módulo EXTERN.H

```
/*
 *  archivo:        extern.h
 *
 *  objetivo:       declaraciones externas.
 */

#include "define.h"

/* El archivo in especifica estos par metros */

extern int    Totalexperiments;         /* cantidad de
experimentos                    */
extern int    Totaltrials;    /* evaluaciones por
experimentos                    */
extern int    Popsize;                  /* tama¤o de
la poblaci¢n */
extern int    Length;                   /* longitud en
bits de la estructura */
extern double  C_rate;                  /*
probabilidad de cruza*/
extern double  M_rate;                  /*
probabilidad de mutacion */
extern double  Gapsize;                 /* fracci¢n de
la poblaci¢n que ser  reemplazada cada generaci¢n */
extern int    Windowsize;     /* usado para actualizar
la peor performance */
extern int    Interval;                 /* cantidad de
evaluaciones entre estad¡sticas         */
extern int    Savesize;                 /* n£mero de
estructuras que se guardan en minfile   */
extern int    Maxspin;                  /* max gens
sin evaluaciones                */
extern int    Dump_freq;      /* gens entre
checkpoints                     */
extern int    Num_dumps;      /* cantidad de archivos
de checkpoint que se guardan  */
extern char   Options[];      /* opciones
                                */
extern unsigned long Seed;    /* semilla para el
generador aleatorio */
extern unsigned long OrigSeed;          /* semilla
inicial   */
```

```
/*****************************************  de
global.h   ******/

/*        Variables globales. */


/* Nombres de archivos */
extern char   Bestfile[];     /* archivo con las
mejores estructuras */
extern char   Ckptfile[];     /* archivo de check
point                           */
extern int    Curr_dump;      /* sufijo del £ltimo
dumpfile  */
extern char   Dumpfile[];     /* dumpfile actual (si
hay m s de uno)     */
extern char   Initfile[];     /* archivo de
estructuras iniciales                   */
extern char   Infile[];                 /* parametros
globales                        */
extern char   Logfile[];      /* log de inicio y
reinicio            */
extern char   Minfile[];      /* prefijo de Bestfile
(se guarda un bestfile por cada experimento       */
extern char   Outfile[];      /* salida "en bruto" de
estad¡sticas                                    */
extern char   Schemafile[];   /* archivo para guardar
la historia de un esquema     */
extern char   Templatefile[]; /* archivo que describe
la interpretacion de los genes          */

extern char   *Bitstring;     /* representaci¢n en
cadena de '1's y '0's         */
extern double *Vector;                  /*
representaci¢n de punto flotante */
extern int Genes;             /* Cantidad de genes
interpretados                   */
extern GENESTRUCT *Gene;      /* puntero a unidades de
template  */
extern STRUCTURE  *Old;                 /* puntero a
la generaci¢n anterior        */
extern STRUCTURE  *New;                 /* puntero a
la nueva generaci¢n */
extern BESTSTRUCT *Bestset;   /* conjunto de las
mejores estructuras           */
```

```
/* datos estad¡sticos y variables auxiliares */
extern double  Ave_current_perf;/* aptitud promedio de
la generaci¢n actual          */
extern double  Best;                    /* la mejor
aptitud hasta ahora */
extern double  Best_current_perf;/* la mejor aptitud
en la generaci¢n actual       */
extern int     Best_guy;                /* numero de
estructura con best_current_perf                */
extern int     Bestsize;                /* cantidad de
mejores estructuras grabadas */
extern double  Bias;                    /* promedio de
la dominaci¢n de alelos       */
extern int     Bytes;                   /* longitud en
bytes de la estructura empaquetada */
extern int     Conv;            /* n£mero de genes que
convergieron parcialmente     */
extern char    Doneflag;                /* se setea
cuando se cumple la condici¢n de parada */
extern int     Experiment;      /* contador de
experimentos                            */
extern int     Gen;             /* contador         de
generaciones                  */
extern unsigned long    Initseed; /* semilla inicial
del experimento     */
extern int     Lost;            /* cantidad de
posiciones que convergieron totalmente */
extern int     Mu_next;                 /* pr¢xima
posici¢n a mutar              */
extern double  Offline;                 /* performance
offline                       */
extern double  Offsum;                  /* acumulador
para la performance offline   */
extern double  Online;                  /* performance
online                        */
extern double  Onsum;                   /* acumulador
para performance online       */
extern int     Plateau;                 /* contador de
evaluaciones para la siguiente salida   */
extern double Rank_min;                 /* minima tasa
de muestreo para seleccion por ranking  */
extern double  Totbest;                 /* mejor de
todos los experimentos                          */
extern double  Totoffline;     /* offline de todos los
experimentos                            */
extern double  Totonline;      /* online  de todos los
experimentos                            */
extern int     Trials;                  /* contador de
evaluaciones */
extern double  *Window;                 /* cola
circular de los peores de las £ltimas generaciones
          */
```

```
extern double  Worst;                   /* peor
performance hasta ahora       */
extern double  Worst_current_perf;/* peor perf en la
generaci¢n actual */
extern int          Spin;               /* cantidad de
generaciones desde la £ltima evaluaci¢n */

/* flags seteados de acuerdo a Options */
extern char    Allflag;                 /*  evaluar
todas las estructuras                   */
extern char    Bestflag;                /*  imprimir
el mejor valor      final     */
extern char    Collectflag;     /*  llevar estad¡sticas
de performance */
extern char    Convflag;                /*  llevar
estadisticas de convergencia */
extern char    Displayflag;     /*  mostrar estadisticas
de cada generacion */
extern char    Dumpflag;                /*  hacer un
vuelco despues de cada evalueacion                */
extern char    Eliteflag;       /*  usar la estrategia
de seleccion elitista         */
extern char    Floatflag;       /*  convertir cadenas a
punto flotante       */
extern char    Grayflag;                /*  usar gray
code                          */
extern char    Initflag;                /*  leer
estructuras iniciales de un archivo               */
extern char    Interflag;       /*  modo interactivo
                              */
extern char    Lastflag;                /*  vuelco de
la ultima generacion */
extern char    Logflag;                 /*  log de
inicios y reinicios           */
extern char    Maxflag;                 /*  maximizar
en lugar de minimizar         */
extern char    Offlnflag;       /*  imprimir medida de
offline final                 */
extern char    Onlnflag;                /*  imprimir
medida de online final                  */
extern char    Rankflag;                /*  usar
seleccion por ranking                   */
extern char    Restartflag;     /*  recomenzar una
corrida                       */
extern char    Schemflag;       /*  seguir la historia
de un esquema       */
extern char    Traceflag;       /*  mostrar las rutinas
que se ejecutan               */

/** fin de archivo **/
```

C.4.3 Módulo FORMAT.H

```
/*
 *  archivo:        format.h
 *
 *  objetivo:       especificar los formatos para
archivos de entrada y salida
 */

/* el archivo in se lee de acuerdo a  IN_FORMAT e
IN_VARS */

#define IN_FORMAT " \
                                        Exp. = %d \
                        Tot. Eval. = %d \
                                        Pob. = %d \
            Long. cromosoma = %d \
                        Prob. cruza = %lf \
             Prob. mutaci¢n = %lf \
             Salto generac. = %lf \
            Ventana scaling = %d \
            Interv. reporte = %d \
          Estruc. guardadas = %d \
   Max gens sin eval. = %d \
Interv. entre vuelcos = %d \
          Vuelcos guardados = %d \
                             Opciones = %s \
          Semilla aleatoria = %lu \
             Ranking m¡nimo = %lf \
                              Epsilon = %lf"


#define IN_VARS
&Totalexperiments,&Totaltrials,&Popsize,&Length,\
          &C_rate,&M_rate,&Gapsize,\
          &Windowsize,&Interval,\
          &Savesize,&Maxspin,\
          &Dump_freq,&Num_dumps,\
          Options,\
          &OrigSeed, &Rank_min, &errorPresion



/*        LINE_FIN es el formato de cada linea del
archivo out tal como          */
/*        es leida por la rutina report
                              */
#define LINE_FIN "%lf %lf %lf %lf %lf %lf %lf %lf %lf"

#define LINE_VIN
&line[0],&line[1],&line[2],&line[3],&line[4],\
          &line[5],&line[6],&line[7],&line[8]
```

```


/*        formatos de salida.           */

/* OUT_FORMAT es el formato en que se imprimen los
parametros de entrada */

#define OUT_FORMAT " \
                                        Exp. = %d\n\
                        Tot. Eval. = %d\n\
                                        Pob. = %d\n\
            Long. cromosoma = %d\n\
                        Prob. cruza = %lf\n\
             Prob. mutaci¢n = %lf\n\
             Salto generac. = %lf\n\
            Ventana scaling = %d\n\
            Interv. reporte = %d\n\
          Estruc. guardadas = %d\n\
   Max gens sin eval. = %d\n\
Interv. entre vuelcos = %d\n\
          Vuelcos guardados = %d\n\
                             Opciones = %s\n\
          Semilla aleatoria = %lu\n\
             Ranking m¡nimo = %lf\n\
                              Epsilon = %lf\n"

/* OUT_VARS son los par metros que se imprimir n de
acuerdo con OUT_FORMAT */

#define OUT_VARS
Totalexperiments,Totaltrials,Popsize,Length,\
          C_rate,M_rate,Gapsize,\
          Windowsize,Interval,Savesize,Maxspin,\
          Dump_freq,Num_dumps,Options,OrigSeed,Rank_mi
n, errorPresion


/* OUT_F2 es el formato para los datos producidos por
'Measure'.
 * OUT_V2 describe las variables.
 */
#define OUT_F2 "%5d %5d %2d %2d %5.3f %.6e %.6e %.6e
%.6e\n"

#define OUT_V2 Gen,Trials,Lost,Conv,Bias,Online,\
          Offline,Best,Ave_current_perf



/** fin de archivo **/
```

C.4.4 Módulo GLOBAL.H

```
/*
 *  archivo:        global.h
 *
 *  objetivo:       variables globales
 */

#include "define.h"


/* Nombre de archivos */
char      Bestfile[40];       /* archivo con las
mejores estructuras */
char   Ckptfile[40];          /* archivo de check
point                  */
int    Curr_dump;   /* sufijo del ultimo vuelco   */
char   Dumpfile[40];          /* vuelco actual (si hay
mas de uno)            */
char   Initfile[40];          /* archivo de
estructuras iniciales                  */
char   Infile[40];  /* parametros globales
                       */
char   Logfile[40]; /* logs de inicio y reinicio
          */
char   Minfile[40]; /* prefijo de Bestfile (se guarda
un bestfile por cada experimento        */
char   Outfile[40]; /* salida "en bruto" de
estad¡sticas                                    */
char   Schemafile[40];        /* archivo para guardar
la historia de un esquema     */
char   Templatefile[40]; /* archivo que describe la
interpretacion de los genes   */

char   *Bitstring;  /* representaci¢n en cadena de
'1's y '0's           */
double *Vector;               /* representacion de
punto flotante */
int Genes;                    /* Cantidad de genes
interpretados                 */
GENESTRUCT *Gene;  /* puntero a unidades de template
          */
STRUCTURE  *Old;    /* puntero a la generaci¢n
anterior  */
STRUCTURE  *New;    /* puntero a la nueva generaci¢n
          */
BESTSTRUCT *Bestset;/* conjunto de las mejores
estructuras                   */


/* El archivo in especifica estos par metros */
int    Totalexperiments; /* cantidad de experimentos
                       */
int    Totaltrials; /* evaluaciones por experimentos
                       */
int    Popsize;               /* tama¤o de la
poblaci¢n */
int    Length;                /* longitud en bits de
la estructura */
double  C_rate;               /* probabilidad de
cruza*/
double  M_rate;               /* probabilidad de
mutacion */
double  Gapsize;    /* fracci¢n de la poblaci¢n que
ser  reemplazada cada generaci¢n */
int    Windowsize;  /* usado para actualizar la peor
performance */
int    Interval;    /* cantidad de evaluaciones entre
estad¡sticas        */
int    Savesize;    /* n£mero de estructuras que se
guardan en minfile  */
int    Maxspin;               /* max gens sin
evaluaciones                  */
int    Dump_freq;   /* gens entre checkpoints
          */
int    Num_dumps;   /* cantidad de archivos de
checkpoint que se guardan     */
char   Options[40]; /* opciones
                       */
unsigned long    Seed;        /* semilla para el
generador aleatorio */
unsigned long    OrigSeed; /* semilla inicial     */

/* datos estad¡sticos y variables auxiliares */
double  Ave_current_perf;/* aptitud promedio de la
generaci¢n actual   */
double  Best;                 /* la mejor aptitud
hasta ahora         */
double  Best_current_perf;/* la mejor aptitud en la
generaci¢n actual   */
int    Best_guy;    /* numero de estructura con
best_current_perf             */
int    Bestsize;    /* cantidad de mejores estructuras
grabadas */
```

```
double  Bias;                 /* promedio de la
dominaci¢n de alelos                    */
int    Bytes;                 /* longitud en bytes de
la estructura empaquetada */
int    Conv;                  /* n£mero de genes que
convergieron parcialmente     */
char   Doneflag;              /* se setea cuando se
cumple la condici¢n de parada */
int    Experiment;  /* contador de experimentos
                       */
int    Gen;                   /* contador         de
generaciones                  */
unsigned long    Initseed; /* semilla inicial del
experimento         */
int    Lost;                  /* cantidad de
posiciones que convergieron totalmente */
int    Mu_next;               /* pr¢xima posici¢n a
mutar               */
double  Offline;    /* performance offline
                       */
double  Offsum;               /* acumulador para la
performance offline           */
double  Online;               /* performance online
                              */
double  Onsum;                /* acumulador para
performance online  */
int    Plateau;               /* contador de
evaluaciones para la siguiente salida   */
double Rank_min;    /* minima tasa de muestreo para
seleccion por ranking         */
int    Spin;                  /* cantidad de
generaciones desde la £ltima evaluaci¢n */
double  Totbest;    /* mejor de todos los experimentos
                              */
double  Totoffline; /* offline de todos los
experimentos                            */
double  Totonline;  /* online de todos los
experimentos                            */
int    Trials;                /* contador de
evaluaciones */
double  *Window;    /* cola circular de los peores de
las £ltimas generaciones      */
double  Worst;                /* peor performance
hasta ahora         */
double  Worst_current_perf;/* peor perf en la
generaci¢n actual */

/* flags seteados de acuerdo a Options */
char   Allflag;               /*  evaluar todas las
estructuras                   */
char   Bestflag;              /*  imprimir el mejor
valor     final     */
char   Collectflag;           /*  llevar estad¡sticas
de performance */
char   Convflag;              /*  llevar estadisticas
de convergencia */
char   Displayflag; /*  mostrar estadisticas de cada
generacion */
char   Dumpflag;              /*  hacer un vuelco
despues de cada evalueacion             */
char   Eliteflag;   /*  usar la estrategia de
seleccion elitista  */
char   Floatflag;   /*  convertir cadenas a punto
flotante  */
char   Grayflag;              /*  usar gray code
                       */
char   Initflag;              /*  leer estructuras
iniciales de un archivo                 */
char   Interflag;   /*  modo interactivo
                       */
char   Lastflag;              /*  vuelco de la ultima
generacion */
char   Logflag;               /*  log de inicios y
reinicios           */
char   Maxflag;               /*  maximizar en lugar
de minimizar        */
char   Offlnflag;   /*  imprimir medida de offline
final               */
char   Onlnflag;              /*  imprimir medida de
online final                  */
char   Rankflag;              /*  usar seleccion por
ranking             */
char   Restartflag; /*  recomenzar una corrida
                       */
char   Schemflag;   /*  seguir la historia de un
esquema   */
char   Traceflag;   /*  mostrar las rutinas que se
ejecutan            */

/** fin de archivo **/
```

C.4.5 Módulo IN_EVAL.H

```
/*
 *  archivo:        in_eval.h
 *
 *  definicion de las variables para los parametros
del problema
 */
```

```
 int nt;               /* cantidad de intervalos en
el tiempo */
 int ntmute;                          /* cantidad de
intervalos en t que ignoro la presion escuchada */
 int l;                               /* cantidad de
intervalos de la malla */
```

```
 double dt;                          /* longitud del
intervalito de tiempo */
 double *h;                        /* longitud en metros de
cada intervalo (por capa) */
 double *ronodo;    /* densidad de cada nodo */
 double xsou;              /* posicion de la fuente */
 int jsou;                           /* nodo
inmediatamente superior a la fuente */
 double xjsou;      /*posicion en mts. del nodo
anterior a la fuente */
 double xjsoup1;    /*posicion en mts. del nodo
posterior a la fuente */
 double xrec;              /*posicion en mts. del
receptor  */
 int jrec;                      /*nodo anterior al
receptor  */
 double xjrec;        /*posicion en mts. del nodo
anterior al receptor */
 double xjrecp1;    /*posicion en mts. del nodo
posterior al receptor */
 int *kx;                          /* cantidad acumulada de
intervalos en x de una capa */
 int keyb;                                  /* tipo de
borde:

                                   0: superficie libre

                                   1: condicion de bordes
absorbente

                                   2: neuman
                                   */
 int kdel;                                  /* tipo de
fuente:

                                   1 -> delta de Dirac

                                   2 -> derivada de la delta de Dirac

                                   3 -> 2 delta de Dirac

                                   */
 double *source;     /* funcion fuente muestreada */
 double is;
 double *recmed;    /* presiones observadas en el
receptor */
 /* variables necesarias para la codificacion por nro
de material */
 int cantMateriales;
 double *velMaterial;
 double *roMaterial;

 double *hcapa;      /* longitud en metros de cada
intervalo (por intervalo) */

 double errorPresion; /* distancia aceptable entre dos
presiones para la rutina que califica alelos */

 /** fin de archivo **/
```

C.4.6 Módulo IN_EVAL2.H

```
/*
 *  archivo:        in_eval2.h
 *
 *  declaraciones externas para los parametros del
problema
 */

extern int nt;                          /* cantidad de
intervalos en el tiempo */
extern int ntmute;                              /* cantidad de
intervalos en t que ignoro la presion escuchada */
extern int l;                                    /*
cantidad de intervalos de la malla */
extern double dt;                       /* longitud del
intervalito de tiempo */
extern double *h;                       /* longitud en metros de
cada intervalo (por capa) */
extern double *ronodo;    /* densidad de cada nodo */
extern double xsou;      /* posicion de la fuente */
extern int jsou;                        /* nodo
inmediatamente superior a la fuente */
extern double xjsou;      /*posicion en mts. del nodo
anterior a la fuente */
extern double xjsoup1;    /*posicion en mts. del nodo
posterior a la fuente */
extern double xrec;       /*posicion en mts. del
receptor  */
extern int jrec;                /*nodo anterior al
receptor  */
extern double xjrec;           /*posicion en mts. del
nodo anterior al receptor */
extern double xjrecp1;    /*posicion en mts. del nodo
posterior al receptor */
extern int *kx;                         /* cantidad
acumulada de intervalos en x de una capa */
extern int keyb;                        /* tipo de
borde:

                                   0: superficie libre

                                   1: condici¢n de bordes
absorbente

                                   2: neuman
                                   */
extern int kdel;                        /* tipo de
fuente:

                                   1 -> delta de Dirac

                                   2 -> derivada de la delta de Dirac

                                   3 -> 2 delta de Dirac

                                   */
extern double *source;         /* funcion fuente
muestreada */
extern double is;
extern double *recmed;    /* presiones observadas en el
receptor */
 /* Variables necesarias para la codificacion por nro
de material */
extern int cantMateriales;
extern double *velMaterial;
extern double *roMaterial;


extern double *hcapa; /* longitud en metros de cada
intervalo (por intervalo) */
extern double errorPresion; /* distancia aceptable
entre dos presiones para la rutina que califica alelos
*/

 /** fin de archivo **/
```

C.4.7 Módulo PRESION.H

```
/*
 *  archivo:        presion.h
 *
 *  c lculo de las presiones
 */

#ifndef PRESION_H
#define PRESION_H

double una_presion(double *unm1, double *un, double
*unp1, int paso, double *constk, double *velnodo);
void presiones(double *velocidad, int capas, double
*presion, int nt);

void vel_en_nodos(double *velocidad, int capas, double
*velnodo);
double borde_sup(double unm1, double unx1, double
unx2, double cte);
double borde_inf(double unm1, double unx1, double
unx2, double cte);
double fuente_en_borde(int paso, int nodo);
double fuente_interior(int paso, int nodo, double cte,
double *velnodo);

#endif

/** fin de archivo **/
```

C.4.8 Módulo MAIN.C

```
/*
 *  archivo:        main.c
 *
 *  objetivo:       programa principal.
 *
 */

#include "global.h"
#include "in_eval.h"

main(argc,argv)
int argc;
char *argv[];
{
         FILE *fp, *fopen();
         long clock;
         long time();
         char *ctime();
         extern void die();  /* manejador de se¤ales
*/
```

```
        int i;


        /* ver el uso de parametros en la linea de
comandos en input.c */
        Input(argc,argv);
        In_eval(argc,argv);

        if (Displayflag) {
                initscr();
                signal(SIGINT, die);
                for (i=1;i<=23;i++) {
                        move(i,0);
                        clrtoeol();
                }
                refresh();

                if (Interflag)
                        Interactive(); /* nunca
retorna */

                /* este punto se alcanza solo si
Interflag esta en OFF */
                /* l¡nea superior */
                move(1,0);
                for (i=0;i<=79;i++) {
                        printw("-");
                }
                /* l¡nea inferior */
                move(23,0);
                for (i=0;i<=79;i++) {
                        printw("-");
                }

                move(2,0);
                printw(" Propuesta III  Modelo %s
Tope de evaluaciones = %d", argv[1], Totaltrials);
                refresh();
        }

        do                      /* un experimento */
                {

                move(2,60);
                printw("Experimento %d/%d",
Experiment+1,Totalexperiments);
                /* limpiar de la pantalla los
resultados del experimento anterior */
                for (i=5;i<=22;i++) {
                        move(i,0);
                        clrtoeol();
                }
                refresh();

                do      /* ver el ciclo
principal de AG en generate.c */
                        {
                        Generate();
                }
                while (!Doneflag);
```

```
                if (Traceflag)
                printf("Online %e   Offline %e
Mejor %e\n",
                                Online, Offline, Best);

                /* acumular mediciones de
performance */
                Totonline += Online;
                Totoffline += Offline;
                Totbest += Best;

                /* preparar el proximo experimento
*/
                Experiment++;
                Gen = 0;

        }
        while (Experiment < Totalexperiments);

        /* calcular e imprimir las mediciones
finales de performance */

        free(velMaterial);
        free(roMaterial);
        free(hcapa);

        Totonline /= Totalexperiments;
        Totoffline /= Totalexperiments;
        Totbest /= Totalexperiments;

        if (Displayflag) {
                move(15,0);
                printf(" Mejor individuo obtenido
en todos los experimentos:   %e\n", Totbest);
                refresh();
        }
        if (Logflag)
        {
                fp = fopen(Logfile, "a");
                fprintf(fp, "Mejor %e\n",
Totbest);
                time(&clock);
                fprintf(fp, "%s\n",
ctime(&clock));
                fclose(fp);
        }

        if (Displayflag) {
                move(22,0);
                printf(" Presione cualquier tecla
para continuar...");
                getch();
                die();
        }
}

/** fin de archivo **/
```

C.4.9 Módulo BEST.C

```
/*
 *  archivo:        best.c
 *
 *  objetivo:       entrada, mantenimiento y salida de
las mejores estructuras
 *
 */

#include "extern.h"

static double worst_value;    /* peor valor en
Bestset          */
static int worst;             /* puntero al peor
elemento (elemento con worst_value)     */


Savebest(i)
register int i;               /* indice de estructura
en  Bestset        */
{
        /*  Grabar la i-esima estructura de la
poblacion actual */
        /*  La cantidad de estructuras a grabar es
Savesize */

        register int j;
        register int k;
        int found;

        if (Bestsize < Savesize)
        {
                /* Bestsize es la cantidad grabada
hasta ahora, entonces */
                /* todavia hay lugar en Bestset
                */
                /* Ver si ya existe una estructura
identica en Bestset */
                for (j=0, found=0; j<Bestsize &&
(!found); j++)
                        for (k=0, found=1;
(k<Bytes) && (found); k++)
                                found =
(New[i].Gene[k]
```

```
                                                ==
Bestset[j].Gene[k]);
                if (found) return;

                /* insertar la i-esima estructura
*/
                for (k=0; k<Bytes; k++)
                {

        Bestset[Bestsize].Gene[k] = New[i].Gene[k];
                }
                Bestset[Bestsize].Perf =
New[i].Perf;
                Bestset[Bestsize].Gen = Gen;
                Bestset[Bestsize].Trials = Trials;
                Bestsize++;
                if (Bestsize == Savesize)
                {
                        /* buscar el peor
elemento en Bestset */
                        worst_value =
Bestset[0].Perf;
                        worst = 0;
                        for (j=1; j<Savesize;
j++)
                        {
                                if
(BETTER(worst_value,Bestset[j].Perf))
                                {

        worst_value = Bestset[j].Perf;

        worst = j;

                                }
                        }
                }
        }
        else
        {
                /* Ya estan ocupados todos los
lugares, ver si hay que desplazar
                   a alguno */
                if (BETTER(New[i].Perf,
worst_value))       /* hay que grabar New[i] */
```

```
                    {
                                        /* a menos que ya este
representada */
                                        for (j=0, found=0;
j<Bestsize && (!found); j++)
                                                  for (k=0,
found=1; (k<Bytes) && (found); k++)
          found = (New[i].Gene[k]
          == Bestset[j].Gene[k]);
                                        if (found) return;

                                        /* sobreescribir la peor
estructura */
                                        for (k=0; k<Bytes; k++)
                                        {
          Bestset[worst].Gene[k] = New[i].Gene[k];
                                        }
                                        Bestset[worst].Perf =
New[i].Perf;
                                        Bestset[worst].Gen =
Gen;
                                        Bestset[worst].Trials =
Trials;
                                        /* buscar el peor
elemento Bestset */
                                        worst_value =
Bestset[0].Perf;
                                        worst = 0;
                                        for (j=1; j<Savesize;
j++)
                                        {
                                                  if
(BETTER(worst_value, Bestset[j].Perf))
                                                  {
          worst_value = Bestset[j].Perf;

          worst = j;
                                                  }
                                        }
                              }
                    }
}


Printbest()
{
          /*        Escribir las mejores estructuras
en Bestfile.        */

          register int i;
          register int j;
          register int k;
          FILE *fp, *fopen();

          fp = fopen(Bestfile, "w");

          for (i=0; i<Bestsize; i++)
          {
                    Unpack(Bestset[i].Gene, Bitstring,
Length);
                    if (Floatflag)
                    {
                              FloatRep(Bitstring,
Vector, Genes);
                              for (j=0; j<Genes; j++)
                              {
                                        fprintf(fp,
Gene[j].format, Vector[j]);
                                        fprintf(fp, "
");
                              }
                    }
                    else
                    {
                              for (j=0; j<Length; j++)
                              {
                                        fprintf(fp,
"%c", Bitstring[j]);
                              }
                    }
                    fprintf(fp, "  %11.4e ",
Bestset[i].Perf);
                    fprintf(fp, " %4d  %4d\n",
Bestset[i].Gen, Bestset[i].Trials);
          }
          fclose(fp);
}


Readbest()
{
          /*   Leer las mejores estructuras de
Bestfile     */
          /*   durante un Restart
*/

          int i,j,k;
          FILE *fp, *fopen();
          int status;

          if (Savesize)
          {

                    fp = fopen(Bestfile, "r");
                    if (fp == NULL)
                    {
                              Bestsize = 0;
                              return;
                    }

                    Bestsize = 0;
                    status = 1;
                    if (Floatflag)
                    {
                              for (i = 0; i < Genes &&
status != EOF; i++)
                              {
                                        status =
fscanf(fp, "%lf", &Vector[i]);
                              }
                    }
                    else
                              status = fscanf(fp,
"%s", Bitstring);

                    while (status != EOF)
                    {
                              if (Floatflag)

          StringRep(Vector, Bitstring, Genes);

                              Pack(Bitstring,
Bestset[Bestsize].Gene, Length);

                              fscanf(fp, "%lf ",
&Bestset[Bestsize].Perf);
                              fscanf(fp, "%d  %d",
&Bestset[Bestsize].Gen,

          &Bestset[Bestsize].Trials);
                              Bestsize++;

                              /* Tomar la proxima
estructura */
                              if (Floatflag)
                                        for (i = 0; i
< Genes && status != EOF; i++)

          status = fscanf(fp, "%lf", &Vector[i]);
                              else
                                        status =
fscanf(fp, "%s", Bitstring);
                    }
                    fclose(fp);

                    /* buscar el peor elemento en
Bestset */
                    worst_value = Bestset[0].Perf;
                    worst = 0;
                    for (i=1; i<Bestsize; i++)
                    {
                              if (BETTER(worst_value,
Bestset[i].Perf))
                              {
                                        worst_value =
Bestset[i].Perf;
                                        worst = i;
                              }
                    }
          }
}

/**** fin de archivo ****/
```

C.4.10 Módulo CHECKPNT.C

```
/*
 *  archivo:        checkpoint.c
 *
 *  objetivo:       grabar las variables globales para
un eventual reinicio.
 *
 */
#include <dir.h>
#include "extern.h"


Checkpoint(ckptfile)
char ckptfile[];
{
          FILE *fp, *fopen();
          int i,j;
          char CkptfileExp[MAXPATH], drive[MAXDRIVE],
dir[MAXDIR], file[MAXFILE], ext[MAXEXT];

          /* Archivo de checkpoint */
          fnsplit(ckptfile, drive, dir, file, ext);
          sprintf(CkptfileExp, "%s%d%s", file,
Experiment, ext);
          fp = fopen(CkptfileExp, "w");
```

```
          fprintf(fp, "Experimentos %d\n",
Experiment);
          fprintf(fp, "On_line total   %12.6e\n",
Totonline);
          fprintf(fp, "Off_line total   %12.6e\n",
Totoffline);
          fprintf(fp, "Generaci¢n %d\n", Gen);
          fprintf(fp, "Performance On_line  %12.6e\n",
Onsum);
          fprintf(fp, "Performance Off_line %12.6e\n",
Offsum);
          fprintf(fp, "Evaluaciones  %d\n", Trials);
          fprintf(fp, "Pr¢xima estad¡stica %d\n",
Plateau);
          fprintf(fp, "Mejor %12.6e\n", Best);
          fprintf(fp, "Peor   %12.6e\n", Worst);
          fprintf(fp, "Cantidad de generaciones desde
la £ltima evaluaci¢n %d\n", Spin);
          fprintf(fp, "Ultimo vuelco %d\n",
Curr_dump);
          fprintf(fp, "Mu_prox %d\n", Mu_next);
          fprintf(fp, "Semilla Aleatoria %lu\n",
Seed);
          fprintf(fp, "Semilla Inicializadora %lu\n",
Initseed);

          fprintf(fp,"\n");
          fprintf(fp, "Window\n");
          for (i=0; i<Windowsize; i++) fprintf(fp,
"%12.6e\n", Window[i]);
          fprintf(fp,"\n");


          for (i=0; i<Popsize; i++)
          {
                    Unpack(New[i].Gene, Bitstring,
Length);
                    fprintf(fp, "%s", Bitstring);
                    fprintf(fp, " %12.8e ",
New[i].Perf);

                    fprintf(fp, "%1d   ",
New[i].Needs_evaluation);
                    for(j=0; j < Genes; j++)
                              fprintf(fp, " %d",
New[i].BondadAlelo[j]);
                    fprintf(fp, "\n");
          }

          if (Floatflag)      /* imprimir en punto
flotante */
          {
                    fprintf(fp, "\n");
                    for (i=0; i<Popsize; i++)
                    {
                              Unpack(New[i].Gene,
Bitstring, Length);
                              FloatRep(Bitstring,
Vector, Genes);
                              for (j=0; j < Genes;
j++)
                              {
                                        fprintf(fp,
Gene[j].format, Vector[j]);
                                        fprintf(fp, "
");
                              }
                              fprintf(fp, " %10.4f",
New[i].Perf);
                              for(j=0; j < Genes; j++)
                                        fprintf(fp, "
%d", New[i].BondadAlelo[j]);
                              fprintf(fp, "\n");
                    }
          }

          fclose(fp);

          /*  guardar las mejores estructuras en
Bestfile */
          if (Savesize)
                    Printbest();

}

/** fin de archivo **/
```

C.4.11 Módulo CONVERT.C

```
/*
 *  archivo:        convert.c
 *
 *  objetivo:       funciones que traducen entre
distinas representaciones
 *
 */

#include "extern.h"

static char BIT[CHARSIZE] ={ '\200', '\100', '\040',
'\020',
                                        '\010',
'\004', '\002', '\001'};


/* Itoc and Ctoi traducen ints a strings y viceversa
*/


unsigned long int Ctoi(instring, length)
          char *instring;               /* cadena de
'0's y '1's         */
          int length;                   /* longitud de
instring            */
{
          register int i;
          unsigned long n;    /* acumulador para el
valor a retornar */

          n = (unsigned long) 0;
          for (i=0; i<length; i++)
          {
                    n <<= 1;
                    n += (*instring++ - (int) '0');
          }
          return(n);
}



Itoc(n, outstring, length)
          unsigned long int n;          /* entero a
convertir           */
          char *outstring;    /* cadena de '0's y '1's
resultante */
          int length;                   /* longitud de
outstring           */
{
          register int i;

          for (i=length-1; i>=0; i--)
          {
                    outstring[i] = '0' + (n & 1);
                    n >>= 1;
          }
}
```

```
/* Pack y Unpack traducen entre cadenas de '0's y '1's
y arreglos de bits
   empaquetados */

Pack(instring, outstring, length)
          char *instring;               /* cadena de
'0's y '1's */
          char *outstring;    /* representacion
empaquetada de instring */
          int length;                   /* longitud de
instring   */
{
          static firstflag = 1 ;
          static full;        /* numero de bytes
completamente usados en outstring       */
          static slop;        /* numbero de bits
usados en el ultimo byte de outstring */
          register i,j;

          if (firstflag)
          {
                    full = length / CHARSIZE;
                    slop = length % CHARSIZE;
                    firstflag = 0;
          }

          for (i=0; i<full; i++, outstring++)
          {
                    *outstring = '\0';
                    for (j=0; j < CHARSIZE; j++)
                              if (*instring++ == '1')
*outstring |= BIT[j];
          }
          if (slop)
          {
                    *outstring = '\0';
                    for (j=0; j < slop; j++)
                              if (*instring++ == '1')
*outstring |= BIT[j];
          }
}


Unpack(instring, outstring, length)
          char *instring;               /*
representacion binaria empaquetada      */
          char *outstring;    /* representacion en
instring en cadena de '0's y '1's */
          int length;                   /* longitud de
outstring                   */
{
          static firstflag = 1 ;
          static full;        /* numbero de bytes
completamente usados en instring */
          static slop;        /* numbero de bits
usados en el ultimo byte de instring */
          register i,j;

          if (firstflag)
```

```
        {
                full = length / CHARSIZE;
                slop = length % CHARSIZE;
                firstflag = 0;
        }

        for (i=0; i<full; i++, instring++)
        {
                for (j=0; j < CHARSIZE; j++)
                        if (*instring & BIT[j])
                                *outstring++ =
'1';
                        else
                                *outstring++ =
'0';
        }

        if (slop)
        {
                for (j=0; j < slop; j++)
                        if (*instring & BIT[j])
                                *outstring++ =
'1';
                        else
                                *outstring++ =
'0';
        }
        *outstring = '\0';
}


/* Traducciones entre representaciones binarias
tradicionales de enteros
   (punto fijo) y Gray code */


Gray(instring, outstring, length)
char *instring;             /* string que representa
un entero en punto fijo */
char *outstring;    /* string que representa el entero
en Gray code        */
register int length;        /* longitud de strings
                                */
{
        register int i;
        register char last;

        last = '0';
        for (i=0; i<length; i++)
        {
                outstring[i] = '0' + (instring[i]
!= last);
                last = instring[i];
        }
}


Degray(instring, outstring, length)
char *instring;             /* string que representa
el entero en Gray code      */
char *outstring;    /* string que representa el entero
en punto fijo */
register int length;        /* longitud de strings
                                */
{
        register int i;
        register int last;

        last = 0;
        for (i=0; i<length; i++)
        {
                if (instring[i] == '1')
                        outstring[i] = '0' +
(!last);
                else
                        outstring[i] = '0' +
last;
                last = outstring[i] - '0';
        }
}


/* Traducciones entre cadenas de '0's y '1's y
vectores de puntos flotantes
 */
```

```
FloatRep(instring, vect, length)
        char instring[];    /* cadena de '0's y '1's
                            */
        double vect[];              /*
representacion en punto flotante */
        int length;                 /* longitud de
vect (arreglo de salida)    */
{
        register int i;
        unsigned long int n;        /* Valor
entero decodificado                 */
        register int pos;   /* posicion para
comenzar a decodificar              */
        char tmpstring[80]; /* usado para la
interpretacion en Gray code */

        pos = 0;
        for (i=0; i < length; i++)
        {
                if (Grayflag)
                {
                        Degray(&instring[pos],
tmpstring, Gene[i].bitlength);
                        n = Ctoi(tmpstring,
Gene[i].bitlength);
                }
                else
                {
                        n = Ctoi(&instring[pos],
Gene[i].bitlength);
                }
                vect[i] = Gene[i].min +
n*Gene[i].incr;
                pos += Gene[i].bitlength;
        }
}


StringRep(vect, outstring, length)
        double *vect;               /*
representacion en punto flotante    */
        char *outstring;    /* cadena de '0's y '1's
                            */
        int length;                 /* longitud de
vect                        */
{
        register int i;
        unsigned long int n;        /* valor
entero (indice) con que se representa vect[i]*/
        register int pos;   /* proxima posicion para
llenar  outstring   */
        char tmpstring[80]; /* usado para la
traduccion a gray code      */

        pos = 0;
        for (i=0; i < length; i++)
        {
                /* Convertir valor de punto
flotante a un indice */
                n = (int) ((vect[i] - Gene[i].min)
/ Gene[i].incr + 0.5);

                /* codificar n como cadena de
caracteres */
                if (Grayflag)
                {
                        /* convertir a Gray code
*/
                        Itoc(n, tmpstring,
Gene[i].bitlength);
                        Gray( tmpstring,
&outstring[pos], Gene[i].bitlength);
                }
                else
                {
                        Itoc(n, &outstring[pos],
Gene[i].bitlength);
                }
                pos += Gene[i].bitlength;
        }
        outstring[pos] = '\0';
}


/*** fin de archivo ***/
```

C.4.12 Módulo CROSS.C

```
/*
 *  archivo:        crossmp.c
 *
 *  objetivo:       efectuar la cruza multipunto con
heuristica
 *
 */
#include <alloc.h>
#include "extern.h"

/**** las mascaras estan en octal ****/
char premask[CHARSIZE] = { '\000', '\200', '\300',
'\340',
                                        '\360',
'\370', '\374', '\376' };
```

```
char postmask[CHARSIZE] = { '\377', '\177', '\077',
'\037',
                                        '\017',
'\007', '\003', '\001'};


Crossover()
{
        register int mom, dad, parent;          /*
participantes en la cruza */
        register int i;
        register int bit1, bit2, byte;
        static int last;    /* ultimo individuo al
que se aplicara la cruza */
```

```
          char *kid1;                    /* punteros a
los hijos */
          char *kid2;
          static int firstflag = 1;
          STRUCTURE aux;

          if (firstflag)
            {
                    last = (C_rate*Popsize*Gapsize) -
0.5 ;
                    firstflag = 0;
            }

          for (mom=0; mom < last ; mom += 2)
          {
            dad = mom + 1;
            if
(memcmp(New[dad].Gene,New[mom].Gene,Bytes)) {
                    // mom y dad son distintos
                    kid1 = calloc (Bytes,
sizeof(char)); // calloc lo inicializa en 0
                    kid2 = calloc (Bytes,
sizeof(char)); // calloc lo inicializa en 0
                    /* copio en kid1 los genes
"buenos" de mom y los "malos" de
                    mom los reemplazo con genes de dad
*/
                    bit1=0;
                    for (i=0; i < Genes; i++) {
                              bit2= bit1 +
Gene[i].bitlength;
                              if
(New[mom].BondadAlelo[i])
                                        parent=mom;
                              else
                                        parent=dad;
                              if (bit1/8 == bit2/8) {
// si bit1 y bit2 estan en el mismo byte
                                        kid1[bit1/8]
|= (New[parent].Gene[bit1/8] &

                                         premask[bit2
% 8]  &

                                         postmask[bit1
% 8]);
                              }
                              else { // bit1 y bit2 en
bytes distintos

                                        // copiar de
bit1 al fin del byte
                                        kid1[bit1/8]
|= New[parent].Gene[bit1/8] &

                                        postmask[bit1
% 8];
                                        // copiar
bytes
                                        for (byte=
bit1/8 + 1; byte < bit2/8; byte++)

          kid1[byte] = New[parent].Gene[byte];

                                        // copiar del
ppio del byte al bit2
                                        kid1[bit2/8]
|= New[parent].Gene[bit2/8] & premask[bit2 % 8];
                              }
                              bit1=bit2;
                    }

                    /* copio en kid2 los genes
"buenos" de dad y los "malos" de dad
                    los reemplazo con genes de mom */
                    bit1=0;
                    for (i=0; i < Genes; i++) {
                              bit2= bit1 +
Gene[i].bitlength;
                              if
(New[dad].BondadAlelo[i])
                                        parent=dad;
                              else
                                        parent=mom;
                              if (bit1/8 == bit2/8) {
// si bit1 y bit2 estan en el mismo byte
                                        kid2[bit1/8]
|= (New[parent].Gene[bit1/8] &

                                         premask[bit2
% 8]  &
                                         postmask[bit1
% 8]);
                              }
                              else { // bit1 y bit2 en
bytes distintos

                                        // copiar de
bit1 al fin del byte
                                        kid2[bit1/8]
|= New[parent].Gene[bit1/8] &

                                        postmask[bit1
% 8];
                                        // copiar
bytes
                                        for (byte=
bit1/8 + 1; byte < bit2/8; byte++)

          kid2[byte] = New[parent].Gene[byte];

                                        // copiar del
ppio del byte al bit2
                                        kid2[bit2/8]
|= New[parent].Gene[bit2/8] & premask[bit2 % 8];
                              }
                              bit1=bit2;
                    }

                    /* hago mom = kid1 */
                    if
(memcmp(kid1,New[mom].Gene,Bytes)) { //kid1 != mom
                              if
(memcmp(kid1,New[dad].Gene,Bytes)== 0) {
                                        // si kid1 =
dad => basta con hacer swap(mom,dad)
                                        // y ahorramos
evaluar nuevamente la estructura

          memcpy(&aux,&New[mom],sizeof(STRUCTURE));

          memcpy(&New[mom],&New[dad],sizeof(STRUCTURE)
);

          memcpy(&New[dad],&aux,sizeof(STRUCTURE));
                              }
                              else {

          memcpy(New[mom].Gene,kid1,Bytes);

          New[mom].Needs_evaluation=1;
                              }
                    }
                    // si kid1 == mom no hago nada. Ya
esta hecho.

                    /* hago dad = kid2 */
                    if
(memcmp(kid2,New[dad].Gene,Bytes)) { //kid2 != dad
                              if
(memcmp(kid2,New[mom].Gene,Bytes)== 0) {
                                        // si kid2 =
mom => dad <- mom

          memcpy(New[dad].Gene,New[mom].Gene,Bytes);
                                        New[dad].Perf
= New[mom].Perf;

          New[dad].Needs_evaluation =
New[mom].Needs_evaluation;

          memcpy(New[dad].BondadAlelo,New[mom].BondadA
lelo,Genes);
                              }
                              else {

          memcpy(New[dad].Gene,kid2,Bytes);

          New[dad].Needs_evaluation=1;
                              }
                    }
                    free(kid1);
                    free(kid2);
            }
          }
}

/** fin de archivo **/
```

C.4.13 Módulo CROSSMP.C

```
/*
 *  archivo:        crossmp.c
 *
 *  objetivo:       efectuar la cruza multipunto con
heuristica
 *
 */
#include <alloc.h>
#include "extern.h"

/**** las mascaras estan en octal ****/
char premask[CHARSIZE] = { '\000', '\200', '\300',
'\340',
                                        '\360',
'\370', '\374', '\376' };

char postmask[CHARSIZE] = { '\377', '\177', '\077',
'\037',
                                        '\017',
'\007', '\003', '\001'};


Crossover()
{
```

```
        register int mom, dad, parent;          /*
participantes en la cruza */
        register int i;
        register int bit1, bit2, byte;
        static int last;    /* ultimo individuo al
que se aplicara la cruza */
        char *kid1;                     /* punteros a
los hijos */
        char *kid2;
        static int firstflag = 1;
        STRUCTURE aux;

        if (firstflag)
          {
                last = (C_rate*Popsize*Gapsize) -
0.5 ;
                firstflag = 0;
          }

        for (mom=0; mom < last ; mom += 2)
        {
          dad = mom + 1;
          if
(memcmp(New[dad].Gene,New[mom].Gene,Bytes)) {
                // mom y dad son distintos
                kid1 = calloc (Bytes,
sizeof(char)); // calloc lo inicializa en 0
                kid2 = calloc (Bytes,
sizeof(char)); // calloc lo inicializa en 0
                /* copio en kid1 los genes
"buenos" de mom y los "malos" de
                mom los reemplazo con genes de dad
*/
                bit1=0;
                for (i=0; i < Genes; i++) {
                        bit2= bit1 +
Gene[i].bitlength;
                        if
(New[mom].BondadAlelo[i])
                                parent=mom;
                        else
                                parent=dad;
                        if (bit1/8 == bit2/8) {
// si bit1 y bit2 estan en el mismo byte
                                kid1[bit1/8]
|= (New[parent].Gene[bit1/8] &

                                 premask[bit2
% 8]  &

                                 postmask[bit1
% 8]);
                        }
                        else { // bit1 y bit2 en
bytes distintos

                                // copiar de
bit1 al fin del byte
                                kid1[bit1/8]
|= New[parent].Gene[bit1/8] &

                                postmask[bit1
% 8];
                                // copiar
bytes
                                for (byte=
bit1/8 + 1; byte < bit2/8; byte++)

          kid1[byte] = New[parent].Gene[byte];

                                // copiar del
ppio del byte al bit2
                                kid1[bit2/8]
|= New[parent].Gene[bit2/8] & premask[bit2 % 8];
                                }
                                bit1=bit2;
                }

                /* copio en kid2 los genes
"buenos" de dad y los "malos" de dad
                los reemplazo con genes de mom */
                bit1=0;
                for (i=0; i < Genes; i++) {
                        bit2= bit1 +
Gene[i].bitlength;
                        if
(New[dad].BondadAlelo[i])
                                parent=dad;
                        else
                                parent=mom;
                        if (bit1/8 == bit2/8) {
// si bit1 y bit2 estan en el mismo byte
                                kid2[bit1/8]
|= (New[parent].Gene[bit1/8] &

                                 premask[bit2
% 8]  &

                                 postmask[bit1
% 8]);
                        }
                        else { // bit1 y bit2 en
bytes distintos

                                // copiar de
bit1 al fin del byte
                                kid2[bit1/8]
|= New[parent].Gene[bit1/8] &

                                postmask[bit1
% 8];
                                // copiar
bytes
                                for (byte=
bit1/8 + 1; byte < bit2/8; byte++)

          kid2[byte] = New[parent].Gene[byte];

                                // copiar del
ppio del byte al bit2
                                kid2[bit2/8]
|= New[parent].Gene[bit2/8] & premask[bit2 % 8];
                                }
                                bit1=bit2;
                }

                /* hago mom = kid1 */
                if
(memcmp(kid1,New[mom].Gene,Bytes)) { //kid1 != mom
                        if
(memcmp(kid1,New[dad].Gene,Bytes)== 0) {
                                // si kid1 =
dad => basta con hacer swap(mom,dad)
                                // y ahorramos
evaluar nuevamente la estructura

          memcpy(&aux,&New[mom],sizeof(STRUCTURE));

          memcpy(&New[mom],&New[dad],sizeof(STRUCTURE)
);

          memcpy(&New[dad],&aux,sizeof(STRUCTURE));
                        }
                        else {

          memcpy(New[mom].Gene,kid1,Bytes);

          New[mom].Needs_evaluation=1;
                        }
                }
                // si kid1 == mom no hago nada. Ya
esta hecho.

                /* hago dad = kid2 */
                if
(memcmp(kid2,New[dad].Gene,Bytes)) { //kid2 != dad
                        if
(memcmp(kid2,New[mom].Gene,Bytes)== 0) {
                                // si kid2 =
mom => dad <- mom

          memcpy(New[dad].Gene,New[mom].Gene,Bytes);
                                New[dad].Perf
= New[mom].Perf;

          New[dad].Needs_evaluation =
New[mom].Needs_evaluation;

          memcpy(New[dad].BondadAlelo,New[mom].BondadA
lelo,Genes);
                        }
                        else {

          memcpy(New[dad].Gene,kid2,Bytes);

          New[dad].Needs_evaluation=1;
                        }
                }
                free(kid1);
                free(kid2);
          }
        }
}

/** fin de archivo **/
```

C.4.14 Módulo DISPLAY.C

```
/*
 *  archivo:        display.c
 *
 *  objetivo:       maneja la pantalla
 *
 */

#include "extern.h"

Interactive() {
        char cmd[40];
        char opt[40];
        register int i;
        int ncycles;
        int ok;

        ncycles = 1;
        while (1) {
                ok = 1;
                move(22,0);
```

```
                    clrtoeol();
                    move(22,35);
                    printw("q (clear & exit), x
(exit), <n> (do n gens)");
                    move(22,0);
                    printw("
");
                    refresh();
                    move(22,0);
                    printw("gens[%d]: ", ncycles);
                    refresh();
                    getstr(cmd);
                    if (strcmp(cmd, "q") == 0) {
                              if (Lastflag)

          Checkpoint(Ckptfile);
                              else
                                        if (Savesize)

          Printbest();
                              clear();
                              die();
                    }
                    if (strcmp(cmd, "x") == 0) {
                              if (Lastflag)

          Checkpoint(Ckptfile);
                              else
                                        if (Savesize)

          Printbest();
                              move(23,0);
                              die();
                    }
                    if (strcmp(cmd, "") != 0) {
                              if (sscanf(cmd, "%d",
&ncycles) !=1)
                              {
                                        move(23,0);
                                        clrtoeol();

          printw("unknown command: %s", cmd);
                                        ok = 0;
                                        refresh();
                              }
                    }
                    if (ok)
                    {
                              move(23,0);
                              clrtoeol();
                              move(1,0);
                              printw("run until Gens =
%d", Gen + ncycles -1);
                              move(1,35);
                              printw("executing: ");
                              refresh();
                              for (i=0; i < ncycles;
i++)
                                        Generate();
                    }
          }
}

die(sig)
int sig;
{
          sig++;
          signal(SIGINT, SIG_IGN);
          move(23,0);
          clrtoeol();
          refresh();
          endwin();
          exit(0);
}


move(row, col)
          int row, col;
{
          /* mover a las coordenadas fila, columna de
la pantalla */
          /* (0,0) = esquina superior izquierda;
             (23,79) = esquina inferior derecha
          */

          gotoxy(col+1, row+1);
}

clear()
{
          /* limpiar la pantalla */
          clrscr();
}


getstr(s)
     char *s;
{
  getw(s);
}

initscr() {}

endwin() {}

/**** fin de archivo ****/
```

C.4.15 Módulo DONE.C

```
/*
 *  archivo:        done.c
 *
 *  objetivo:       ver si se cumple la condicion de
parada.
 *
 */

#include "extern.h"

int Done()
{

/**** por demasiadas evaluaciones, o toda la poblacion
es igual, o
          demasiadas generaciones sin evaluar ****/
          return ((Trials >= Totaltrials)  || ( Lost
>= Genes)
                    || (Spin >= Maxspin));
}

/* fin de archivo */
```

C.4.16 Módulo ELITIST.C

```
/*
 *  archivo:        elitist.c
 *
 *  objetivo:       La politica elitista estipula que
el mejor individuo
 *                  siempre sobrevive en la nueva
generacion. El individuo de elite
 *                  es ubicado en la ultima posicion
de la nueva poblacion
 *            y no es afectado por cruza o mutacion ya
que se pasa a la nueva
 *                  generacion despues de haber
aplicado estos operadores.
 */

#include "extern.h"

Elitist()
{
          register int i;
          register int k;
          register int found; /* se setea si ya existe
el individuo de elite en

             la nueva generacion */


          /* Hay algun elemento en la poblacion actual
                    */
          /* que sea identico al mejor de la
generacion anterior?      */
          for (i=0, found=0; i<Popsize && (!found);
i++)
                    for (k=0, found=1; (k<Bytes) &&
(found); k++)
                              found = (New[i].Gene[k]
                                        ==
Old[Best_guy].Gene[k]);

          if (!found)         /* No se encontro el
individuo de elite */
          {
                    /* reemplazar el ultimo individuo
por el individuo de elite */
                    for (k=0; k<Bytes; k++)
                              New[Popsize-1].Gene[k] =
Old[Best_guy].Gene[k];
                    New[Popsize-1].Perf =
Old[Best_guy].Perf;
                    New[Popsize-1].Needs_evaluation =
0;

                    for (k=0; k<Genes; k++)
                              New[Popsize-
1].BondadAlelo[k] = Old[Best_guy].BondadAlelo[k];

                    if (Traceflag)
                    {
                              printf("perf: %e\n",
New[Popsize-1].Perf);
                    }
          }

}
```

```
/*** fin de archivo ***/
```

C.4.17 Módulo ERROR.C

```
/*
 *  archivo:        error.c
 *
 *  objetivo:       imprimir mensaje de error y
abortar. El error queda tambien
 *                              en un log de
errores.
 *
 */

#include <stdio.h>

Error(s)
char *s;
{
          FILE *fopen(), *fp;
          long clock;
          long time();
          char *ctime();

          fp = fopen("error", "a");
          fprintf(fp, "%s\n", s);
          time(&clock);
          fprintf(fp, "%s\n", ctime(&clock));
          fclose(fp);
          fprintf(stderr, "%s\n", s);

          exit(1);
}

/*** fin de archivo ***/
```

C.4.18 Módulo EVAL.C

```
/*
 * archivo: evalgen2.c
 *
 * objetivo: calcular la aptitud del individuo.
Calificar los alelos de
 * acuerdo a la heuristica.
 */
#include <alloc.h>
#include <stdlib.h>
#include "extern.h"
#include "in_eval2.h"

#define frandom()      ( random(32767) / 32766.0 )
/* Rutina que hace la evaluacion de un cromosoma cuyos
genes son
   nros de material calificando alelo por alelo*/
/*-------------------------------------------------
-----------------*/
double costo(v1, v2, N)
   double v1[], v2[];
   int N;
/*                      N
                                    --
                              1     \              2
                              -     /    ( v1 - v2 )
                              n           --       i
i
                                    i=1
*/
{
          int i;
          double s= 0;

          for (i=1; i <= N; i++)
                    s+= pow(( v1[i] - v2[i] ),2);
          return s/N;
}

/*-------------------------------------------------
-----------------*/
void calificaAlelos(presion1, presion2, nt, vect,
bondadAlelo)

double presion1[];
double presion2[];
int nt;
double vect[];       /* material por c/capa        */
int bondadAlelo[];
{
          int t; // tiempo t. Indice para recorrer
presiones
          int alelo; // alelo a clasificar
          int taux; // tiempo de la capa anterior
          int dtAlelo; // cantidad de instantes que
corresponden a alelos del cromosoma
          int distancia;
          int ok;  // cantidad de intervalos en el
tiempo en que las presiones coinciden
          int nok; // cantidad de intervalos en el
tiempo en que las presiones NO coinciden


taux=1;
for (alelo=0; alelo < Genes; alelo++) {
          // calculo dtAlelo. Si es la primera capa
descuento xsou y xrec
          distancia = 2 * (kx[alelo+1]-kx[alelo]) *
hcapa[alelo];
          if (alelo == 0) distancia = distancia - xsou
- xrec;
          dtAlelo =
distancia/velMaterial[vect[alelo]]/dt;
          ok=0;
          nok=0;
          for (t=taux;t <= taux+dtAlelo && t <= nt;
t++) {
                    if( fabs(presion1[t] -
presion2[t]) <= errorPresion )
                              ok++;
                    else
                              nok++;
          }

          if (nok == 0 || (nok > 0 && ok/nok >
frandom()))
                    bondadAlelo[alelo] =1;
          else
                    bondadAlelo[alelo] =0;
          taux = t;
}
}
/*-------------------------------------------------
-----------------*/

double eval(str, length, vect, genes, bondadAlelo)
char str[];          /* representacion en cadena de
'0's y '1's          */
int length;          /* longitud de la cadena
                     */

/* vect: materiales de cada una de las "genes" capas
*/
double vect[];       /* representacion en punto
flotante             */

/* genes: cantidad de capas */
int genes;           /* cantidad de elementos en el
vector                    */
int bondadAlelo[];
{
          double resul;
          /* presiones calculadas a partir de las
velocidades, una por cada
          instante de tiempo */
          double *recest;

          /* convierto materiales a velocidades */
          double *velAux;
          int i,j;

          recest =malloc((nt+1)*sizeof(double));
          velAux = malloc(genes * sizeof(double));
          for (i=0; i < genes; i++)
                    velAux[i] = velMaterial[vect[i]];

          /* lleno ronodo */
          for (i=0; i < genes; i++)
                    for (j=kx[i]+1; j <= kx[i+1]; j++)
                              ronodo[j] =
roMaterial[vect[i]];
          ronodo[1+1]= roMaterial[vect[genes-1]];

          presiones(velAux, genes, recest, nt);



          free(velAux);
          calificaAlelos(recest, recmed, nt, vect,
bondadAlelo);

          resul =costo(recest, recmed, nt);
          free(recest);
          return resul;

}

/**** fin de archivo ****/
```

C.4.19 Módulo EVALGEN2.C

```
/*
 * archivo: evalgen2.c
 *
 * objetivo: calcular la aptitud del individuo.
Calificar los alelos de
 * acuerdo a la heuristica.
 */
#include <alloc.h>
#include <stdlib.h>
#include "extern.h"
#include "in_eval2.h"

#define frandom()      ( random(32767) / 32766.0 )
/* Rutina que hace la evaluacion de un cromosoma cuyos
genes son
   nros de material calificando alelo por alelo*/
/*-------------------------------------------------
-----------------*/
double costo(v1, v2, N)
   double v1[], v2[];
   int N;
/*                      N
                                  --
                            1     \          2
                            -     /   ( v1 - v2 )
                            n           --      i
i
                                        i=1
*/
{
          int i;
          double s= 0;

          for (i=1; i <= N; i++)
                    s+= pow(( v1[i] - v2[i] ),2);
          return s/N;
}

/*-------------------------------------------------
-----------------*/
void calificaAlelos(presion1, presion2, nt, vect,
bondadAlelo)

double presion1[];
double presion2[];
int nt;
double vect[];       /* material por c/capa        */
int bondadAlelo[];
{
          int t; // tiempo t. Indice para recorrer
presiones
          int alelo; // alelo a clasificar
          int taux; // tiempo de la capa anterior
          int dtAlelo; // cantidad de instantes que
corresponden a alelos del cromosoma
          int distancia;
          int ok;  // cantidad de intervalos en el
tiempo en que las presiones coinciden
          int nok; // cantidad de intervalos en el
tiempo en que las presiones NO coinciden


taux=1;
for (alelo=0; alelo < Genes; alelo++) {
          // calculo dtAlelo. Si es la primera capa
descuento xsou y xrec
          distancia = 2 * (kx[alelo+1]-kx[alelo]) *
hcapa[alelo];
          if (alelo == 0) distancia = distancia - xsou
- xrec;
          dtAlelo =
distancia/velMaterial[vect[alelo]]/dt;
          ok=0;
          nok=0;
          for (t=taux;t <= taux+dtAlelo && t <= nt;
t++) {
                    if( fabs(presion1[t] -
presion2[t]) <= errorPresion )
                              ok++;
                    else
                              nok++;
          }

          if (nok == 0 || (nok > 0 && ok/nok >
frandom()))
                    bondadAlelo[alelo] =1;
          else
                    bondadAlelo[alelo] =0;
          taux = t;
}
}
/*-------------------------------------------------
-----------------*/

double eval(str, length, vect, genes, bondadAlelo)
char str[];          /* representacion en cadena de
'0's y '1's          */
int length;          /* longitud de la cadena
                     */

/* vect: materiales de cada una de las "genes" capas
*/
double vect[];       /* representacion en punto
flotante             */

/* genes: cantidad de capas */
int genes;           /* cantidad de elementos en el
vector                    */
int bondadAlelo[];
{
          double resul;
          /* presiones calculadas a partir de las
velocidades, una por cada
          instante de tiempo */
          double *recest;

          /* convierto materiales a velocidades */
          double *velAux;
          int i,j;

          recest =malloc((nt+1)*sizeof(double));
          velAux = malloc(genes * sizeof(double));
          for (i=0; i < genes; i++)
                    velAux[i] = velMaterial[vect[i]];

          /* lleno ronodo */
          for (i=0; i < genes; i++)
                    for (j=kx[i]+1; j <= kx[i+1]; j++)
                              ronodo[j] =
roMaterial[vect[i]];
          ronodo[1+1]= roMaterial[vect[genes-1]];

          presiones(velAux, genes, recest, nt);



          free(velAux);
          calificaAlelos(recest, recmed, nt, vect,
bondadAlelo);

          resul =costo(recest, recmed, nt);
          free(recest);
          return resul;

}

/**** fin de archivo ****/
```

C.4.20 Módulo EVALUATE.C

```
/*
 *  archivo:        evaluate.c
 *
 *  objetivo:       evaluar la poblacion actual
mediante la funcion "eval"
 *
 */

#include "extern.h"
extern double eval();

Evaluate()
{
          register double performance;
          register int i;

          for (i=0; i<Popsize; i++)
          {
                    if (Displayflag) {
                              move(20,0);
                              clrtoeol();
                              printf(" Evaluando
individuo %d de la poblaci¢n",i);
                              refresh();
                    }

                    if ( New[i].Needs_evaluation )
                    {
                              Unpack(New[i].Gene,
Bitstring, Length);
                              if (Floatflag)

                    FloatRep(Bitstring, Vector, Genes);
                              New[i].Perf = eval
(Bitstring, Length,
                                        Vector, Genes,
New[i].BondadAlelo);
                              performance =
New[i].Perf;
                              New[i].Needs_evaluation
= 0;
                              Trials++;
                              Spin = 0;   /* Hubo una
evaluacion, spin vuelve a 0 */
                              if (Savesize)
Savebest(i);

                              if (Trials == 1)
                                        Best =
performance;
                              if (BETTER(performance,
Best))
                              {
                                        Best =
performance;
```

```
                        }

                        Onsum += performance;
                        Offsum += Best;
                        if (Dumpflag)
Checkpoint(Ckptfile);
                        }
```

```
        }
        move(20,0);
        clrtoeol();
}


/** fin de archivo **/
```

C.4.21 Módulo GENERATE.C

```
/*
 *  archivo:        generate.c
 *
 *  objetivo:       Una generacion consiste de
 *                  (1) formar una nueva
poblacion de estructuras
 *                  (2) evaluar la poblacion
 *                  (3) recoger estadisticas
de performance
 *
 */

#include "extern.h"
extern int  Done();


Generate()
{
        static int rsflag = 1;          /* flag
reseteado despues de restart            */

        STRUCTURE *temp;    /* para swapear punteros
de la poblacion     */
        register int i;                 /* para marcar
estructuras             */

        if (Traceflag)
                printf("                    Gen
%d\n",Gen);

        /* crear una nueva poblacion */

        if (Restartflag && rsflag)
        {
                /* si es un restart leer archivo
de checkpoint */
                Restart();
                rsflag = 0;             /* Ya hicimos
este restart. */
                Converge();
        }

        else if (Gen == 0)
                /* Si recien comienza el
experimento */
        {
                Initialize();           /* formar una
poblacion inicial */
                Spin++;
        }

        else
                /* formar una nueva poblacion a
partir de */
                /* la vieja via operadores
geneticos */
        {
                Select();
                Mutate();
                Crossover();
                if (Eliteflag)
                        Elitist();
                if (Allflag)            /* marcar las
estructuras para evaluacion */
                        for (i=0; i<Popsize;
i++) New[i].Needs_evaluation = 1;
                Spin++;
        }

        /* evaluar la poblacion recientemente
formada */
        Evaluate();

        /* recoger estadisticas de performance */
        Measure();

        /* chequear la condicion de parada para este
experimento */
        Doneflag = Done();

        /* Hacer vuelco si corresponde */
        if (Num_dumps && Dump_freq && Gen %
Dump_freq == 0)
        {
                if (Num_dumps > 1)
                {
                        sprintf(Dumpfile,
"dump.%d", Curr_dump);
                        Curr_dump = (Curr_dump +
1) % Num_dumps;
                        Checkpoint(Dumpfile);
                }
                Checkpoint(Ckptfile);
        }
        else
        {
                if (Doneflag)
                {
                        if (Lastflag)

        Checkpoint(Ckptfile);
                        else
                                if (Savesize)

        Printbest();
                }
        }


        /* Intercambiar punteros para la proxima
generacion */
        temp = Old;
        Old = New;
        New = temp;

        /* actualizar contador de generaciones */
        Gen++;

}

/*  fin de archivo */
```

C.4.22 Módulo INIT.C

```
/*
 *  archivo:        init.c
 *
 *  objetivo:       crear la poblacion inicial de
estructuras,
 *                  e inicilizar variables de
performance.
 *                  Esta rutina es llamada al inicio
de cada experimento.
 *
 */


#include "extern.h"
#include "dir.h"
#include "in_eval2.h"

Initialize()
{
        FILE *fp, *fopen();
        register int i, j;
        int status;                     /* indicates
fin de archivo en initfile    */
        char drive[MAXDRIVE], dir[MAXDIR],
file[MAXFILE], ext[MAXEXT];

        fnsplit(Minfile, drive, dir, file, ext);
        sprintf(Bestfile, "%s%d%s", file,
Experiment, ext);
```

```
        /* preparo para el nuevo experimento */
        Doneflag = 0;
        Curr_dump = 0;
        Bestsize = 0;
        /* setear proximo a mutar */
        if (M_rate < 1.0)
          Mu_next = ceil (log(Rand()) / log(1.0 -
M_rate));
        else
          Mu_next = 1;

        Trials = Gen = 0;
        Lost = Conv = 0;
        Plateau = 0;
        Spin = 0;
        Onsum = Offsum = 0.0;
        for (i=0; i<Windowsize; i++) Window[i] =
0.0;

        /* Inicializar poblacion */

        i = 0;                          /* estructura
actual */

        if (Initflag)                   /* tomar
algunas estructuras de Initfile */
        {
                if ((fp = fopen(Initfile, "r")) ==
NULL)
```

```
                    {
                              char msg[40];
                              sprintf(msg,
"Initialize: no puede abrirse %s", Initfile);
                              Error(msg);
                    }

                    status = 1;
                    if (Floatflag)
                    {
                              for (j = 0; j < Genes &&
status != EOF; j++)
                              {
                                        status =
fscanf(fp, "%lf", &Vector[j]);
                              }
                    }
                    else
                              status = fscanf(fp,
"%s", Bitstring);

                    while (status != EOF && i <
Popsize)
                    {
                              if (Floatflag)

          StringRep(Vector, Bitstring, Genes);

                              Pack(Bitstring,
New[i].Gene, Length);
                              New[i].Needs_evaluation
= 1;
                              i++;

                              /* leer la proxima
estructura */
                              if (Floatflag)
                                        for (j = 0; j
< Genes && status != EOF; j++)

          status = fscanf(fp, "%lf", &Vector[j]);
                              else
                                        status =
fscanf(fp, "%s", Bitstring);
                    }
                    fclose(fp);
          }

          /*****************************************
************/
          /* La semilla del generador de numeros
aleatorios se       */
          /* graba despues de la inicializacion de la
primera             */
          /* poblacion en cada experimento. El valor
grabado es          */
          /* usado como semilla en los experimentos
siguientes.         */
          /* La razon es permitir correr varios
experimentos con  */
          /* el mismo conjunto de parametros, y
comparar los                */
          /* resultados de corridas de un conjunto de
          parametros          */
          /* con corridas realizadas con otro
conjunto.                   */
          /*****************************************
************/
          if ( Experiment > 0 ) Seed = Initseed;

          for (; i < Popsize; i++)   /* inicializar el
resto de la poblacion al azar */
                    {
/* Generar valores enteros validos de acuerdo a la
cant de valores posibles */
                    for(j = 0; j < Genes; j++)
                    {
                              Vector[j] =
(double)Randint(0,Gene[j].values-1);
                    }
                    StringRep(Vector,Bitstring,Genes);

                    Pack(Bitstring , New[i].Gene ,
Length);
                    New[i].Needs_evaluation = 1;
          }

          Initseed = Seed;

}

/** fin de archivo **/
```

C.4.23 Módulo INPUT.C

```
/*
 *  archivo:        input.c
 *
 *  purpose:        Inicializar nombres de archivos,
leer archivos de entrada
 *                  e inicializar variables para esta
corrida.
 *
 *                  Ver en init.c la inicializacion de
variables para cada experimento
 */

#include <alloc.h>
#include "extern.h"
#include "in_eval2.h"

Input(argc,argv)
int argc;
char *argv[];
{
          FILE *fopen(), *fp;

          int i;
          char msg[40];
          long clock;                   /* fecha
actual                                  */

          long time();
          char *ctime();
          int ilog2();
          int cantbits();


          /* establecer nombres de archivos */

          if (argc < 2)
          {
                    strcpy(Infile,"in");
                    strcpy(Outfile,"out");
                    strcpy(Ckptfile,"ckpt");
                    strcpy(Minfile,"min");
                    strcpy(Logfile, "log");
                    strcpy(Initfile, "init");
                    strcpy(Schemafile, "schema");
                    strcpy(Templatefile, "template");
          }
          else
          {
                    sprintf(Infile, "in.%s", argv[1]);
                    sprintf(Outfile, "out.%s",
argv[1]);
                    sprintf(Ckptfile, "ckpt.%s",
argv[1]);
                    sprintf(Minfile, "min.%s",
argv[1]);
                    sprintf(Logfile, "log.%s",
argv[1]);
                    sprintf(Initfile, "init.%s",
argv[1]);
                    sprintf(Schemafile, "schema.%s",
argv[1]);
                    sprintf(Templatefile,
"template.%s", argv[1]);
                    }

          strcpy(Bestfile, Minfile);

          /* leer los parametros de infile */

          if ((fp = fopen(Infile, "r")) == NULL)
          {
                    sprintf(msg, "Input: no puede
abrirse %s", Infile);
                    Error(msg);
          }
          fscanf(fp,IN_FORMAT,IN_VARS);
          Seed = OrigSeed;
          fclose(fp);

          /* activar las opciones */
          for (i=0; Options[i] != '\0'; i++)
                    Setflag(Options[i]);
          if (Displayflag)
                    Traceflag = 0;

          /* Bytes es el tama¤o de cada cromosoma
empaquetado */
          Bytes = Length / CHARSIZE;
          if (Length % CHARSIZE) Bytes++;

          /* leer el archivo template. Solo se usa en
la representacion de punto
             flotante */
          if (Floatflag)
          {
                    if ((fp = fopen(Templatefile,
"r")) == NULL)
                    {
                              sprintf(msg, "Input: no
puede abrirse %s", Templatefile);
                              Error(msg);
                    }
                    fscanf(fp, "genes: %d ", &Genes);
                    Gene = (GENESTRUCT *)
calloc((unsigned) Genes,
                              sizeof(GENESTRUCT));

                    for (i=0; i<Genes; i++)
                    {
                              fscanf(fp, " gene %*d");
                              fscanf(fp, " min: %lf",
&Gene[i].min);
                              fscanf(fp, " max: %lf",
&Gene[i].max);
```

```
                                fscanf(fp, " values:
%lu", &Gene[i].values);
                                fscanf(fp, " format:
%s", Gene[i].format);
                                Gene[i].bitlength =
cantbits(Gene[i].values);
                                Gene[i].incr =
(Gene[i].max - Gene[i].min) /

          (Gene[i].values - 1);
                    }
                    fclose(fp);
          }
          /* reservar memoria para las estructuras de
tama¤o variable */

          /* usado para representacion de punto
flotante del cromosoma */
          Vector = (double *) calloc((unsigned) Genes,
sizeof(double));

          /* usado para la representacion en cadena de
'0's y '1's */
          Bitstring = malloc((unsigned) (Length+1));
          Bitstring[Length] = '\0';

          if (Bitstring == NULL) {
                    printf("input: No hay memoria para
Bitstring\n");
                    abort();
          }
          /* arreglos de la poblacion */
          Old = (STRUCTURE *) calloc((unsigned)
Popsize, sizeof(STRUCTURE));
          New = (STRUCTURE *) calloc((unsigned)
Popsize, sizeof(STRUCTURE));

          for (i=0; i<Popsize; i++)
          {
                    Old[i].Gene = malloc((unsigned)
Bytes);
                    New[i].Gene = malloc((unsigned)
Bytes);

                    Old[i].BondadAlelo =
malloc(sizeof(int) * Genes);
                    New[i].BondadAlelo =
malloc(sizeof(int) * Genes);

          }

          /* usado para computar Worst.
            (El peor de las ultimas Windowsize
generaciones) */
          if (Windowsize)
          Window = (double *) calloc((unsigned)
Windowsize, sizeof(double));

          /* usado para guardar las mejores
estructuras */
          if (Savesize)
           Bestset = (BESTSTRUCT *) calloc((unsigned)
Savesize, sizeof(BESTSTRUCT));

          for (i=0; i<Savesize; i++)
                    Bestset[i].Gene =
malloc((unsigned) Bytes);

          /* mostrar los parametros de entrada */
          if (Traceflag) printf(OUT_FORMAT, OUT_VARS);

          /* Sobreescribir el archivo de salida a
menos que sea un reinicio */
          if (!Restartflag)
          {
                    if ((fp = fopen(Outfile, "w")) ==
NULL)
                    {
                              sprintf(msg, "Input: No
puede abrirse %s", Outfile);
                              Error(msg);
                    }
                    fclose(fp);
          }

          /* guardar esta activacion en un log */
          if (Restartflag)
          {
                    if (Logflag)
                    {
                              if ((fp = fopen(Logfile,
"a")) == NULL)
                              {

          sprintf(msg,"Input: No puede abrirse %s",
Logfile);

                                        Error(msg);
                              }
                              fprintf(fp, "%s
Recomenz¢ ", argv[0]);
                              time(&clock);
                              fprintf(fp, "%s",
ctime(&clock));
                              fclose(fp);
                    }
          }
          else
          {
                    if (Logflag)
                    {
                              if ((fp = fopen(Logfile,
"a")) == NULL)
                              {

          sprintf(msg,"Input: No puede abrirse %s",
Logfile);

                                        Error(msg);
                              }
                              fprintf(fp, "%s comenz¢
", argv[0]);
                              time(&clock);
                              fprintf(fp, "%s",
ctime(&clock));
                              fclose(fp);
                    }
          }
}


int ilog2(n)
          unsigned long n;
{
          register int i;

          if (n <= 0)
          {
                    printf("La cantidad de valores es
%d, debe ser positiva!\n", n);
                    abort();
          }

          i = 0;
          while ((int) (n & 1) == 0)
          {
                    n >>= 1;
                    i++;
          }
          return(i);
}


Setflag(c)
char c;
{
          switch (c) {
          case 'a' :
                    Allflag = 1;
                    break;
          case 'b' :
                    Bestflag = 1;
                    break;
          case 'c' :
                    Collectflag = 1;
                    Convflag = 1;
                    break;
          case 'C' :
                    Collectflag = 1;
                    break;
          case 'd' :
                    Dumpflag = 1;
                    break;
          case 'D' :
                    Displayflag = 1;
                    break;
          case 'e' :
                    Eliteflag = 1;
                    break;
          case 'f' :
                    Floatflag = 1;
                    break;
          case 'g' :
                    Grayflag = 1;
                    break;
          case 'i' :
                    Initflag = 1;
                    break;
          case 'I' :
                    Interflag = 1;
                    Displayflag = 1;
                    break;
          case 'l' :
                    Logflag = 1;
                    break;
          case 'L' :
                    Lastflag = 1;
                    break;
          case 'M' :
                    Maxflag = 1;
                    break;
          case 'o' :
                    Onlnflag = 1;
                    break;
          case 'O' :
                    Offlnflag = 1;
                    break;
          case 'r' :
                    Restartflag = 1;
                    break;
          case 'R' :
                    Rankflag = 1;
                    break;
          case 's' :
                    Schemflag = 1;
                    break;
          case 't' :
                    Traceflag = 1;
                    break;
          }
}
```

```
int cantbits(n)
          unsigned long n;
{
          unsigned long mask;
          int otro1, bits, i, totbits;

          otro1=0;
          n = n - 1;  /* tengo que representar de 0 a
n-1 */
          totbits = sizeof(long) * CHARSIZE;

          /* busca el primer 1 contando de la
izquierda */
          mask = (long)1 << (totbits - 1);
          for (i=0; !(n & mask); i++, mask >>=1);
          bits = totbits -i;

          /* se fija si hay otros unos */
          for (i++, mask>>=1; i < totbits; i++, mask
>>=1) {
                    if (n & mask) otro1 = 1;
          }

          if (otro1) bits++;
          return(bits);

}

/** fin de archivo **/
```

C.4.24 Módulo IN_EVAL.C

```
/*
 *  archivo:        in_eval.c
 *
 *  objetivo:       Lectura de parametros del modelo a
ser tratado por eval
 *              Este archivo es problema-dependiente
 *                                        Es decir debe
modificarse (o eliminarse) de trabajarse
 *                    con otro problema.
 *
 */
#include <alloc.h>
#include "extern.h"   /* Hay que incluirlo para
obtener Genes */
#include "in_eval2.h" /* parametros de campo */
/*-----------------------------------------------------
------------------*/
char *readFile(FILE *fp, char *line)
{
          char *c;

          do {
                    c= fgets(line, 100, fp);
          } while ( c!= NULL && (line[0]== '#' ||
line[0] == 10));
          return c;
}
/*-----------------------------------------------------
------------------*/
In_eval(argc,argv)
int argc;
char *argv[];
{
          char In_evalfile[40],line[200];
          FILE *fopen(), *fp;
          double dummy;

          int i,j;
          char msg[40];

          /* Determinar los nombres de los archivos */

          if (argc < 2)
          {
                    strcpy(In_evalfile,"campo");
          }
          else
          {
                    sprintf(In_evalfile, "campo.%s",
argv[1]);
          }

          /* leer parametros de campo */

          if ((fp = fopen(In_evalfile, "r")) == NULL)
          {
                    sprintf(msg, "In_eval: No puede
abrirse %s", In_evalfile);
                    Error(msg);
          }

          readFile(fp, line);
          sscanf(line, "%d", &nt);

          readFile(fp, line);
          sscanf(line, "%d", &ntmute);

          readFile(fp, line);
          sscanf(line, "%d", &l);

          readFile(fp, line);
          sscanf(line, "%lf",&dt);

          hcapa = malloc(Genes * sizeof(double) );  /*
cantGenes = cantCapas */
          if (hcapa == NULL) {
                    printf("in_eval: No hay memoria
para h\n");
                    abort();
          }
          for (i=0; i < Genes; i++) {
                    readFile(fp, line);
                    sscanf(line,"%lf", &hcapa[i]);
          }

          /* ro por capa es usado solo por la
propuesta 1. Lo salteo */
          for (i=0; i < Genes; i++) {
                    readFile(fp, line);
                    sscanf(line,"%lf", &dummy);
          }

          readFile(fp, line);
          sscanf(line,"%lf", &xsou);

          readFile(fp, line);
          sscanf(line,"%d", &jsou);

          readFile(fp, line);
          sscanf(line, "%lf", &xjsou);

          readFile(fp, line);
          sscanf(line, "%lf", &xjsoup1);

          readFile(fp, line);
          sscanf(line, "%lf", &xrec);

          readFile(fp, line);
          sscanf(line, "%d", &jrec);

          readFile(fp, line);
          sscanf(line, "%lf", &xjrec);

          readFile(fp, line);
          sscanf(line, "%lf", &xjrecp1);

          kx = malloc((Genes+1) * sizeof(double) );
/* cantGenes = cantCapas */
          if (kx == NULL) {
                    printf("in_eval: No hay memoria
para kx\n");
                    abort();
          }
          for (i=0; i <= Genes; i++) {
                    readFile(fp, line);
                    sscanf(line,"%d", &kx[i]);
          }

          readFile(fp, line);
          sscanf(line, "%d", &keyb);

          readFile(fp, line);
          sscanf(line, "%d", &kdel);

          source = malloc((nt+1) * sizeof(double));
          if (source == NULL) {
                    printf("in_eval: No hay memoria
para source\n");
                    abort();
          }
          for (i=1; i <= nt; i++) {
                    readFile(fp, line);
                    sscanf(line, "%lf", &source[i]);
          }

          readFile(fp, line);
          sscanf(line, "%lf", &is);

          recmed = malloc((nt+1) * sizeof(double));
          if (recmed == NULL) {
                    printf("in_eval: No hay memoria
para recmed\n");
                    abort();
          }
          for (i=1; i <= nt; i++) {
                    readFile(fp, line);
                    sscanf(line, "%lf", &recmed[i]);
          }
          /* Levanto la cantidad de materiales y las
velocidades de cada
          material (para propuestas 2 y 3). */
          readFile(fp, line);
          sscanf(line, "%d", &cantMateriales);
          velMaterial = malloc(cantMateriales *
sizeof(double));
          if (velMaterial == NULL) {
                    printf("in_eval:No hay memoria
para velMaterial\n");
                    abort();
          }
          for ( i=0; i < cantMateriales; i++) {
                    readFile(fp, line);
                    sscanf(line, "%lf",
&velMaterial[i]);
          }
```

```
        /* Levanto ro de cada material (propuestas 2
y 3)*/
        roMaterial = malloc(cantMateriales *
sizeof(double) );
        for ( i=0; i < cantMateriales; i++) {
                readFile(fp, line);
                sscanf(line, "%lf",
&roMaterial[i]);
        }

        fclose(fp);
```

```
        ronodo = malloc((l+2) * sizeof(double));  //
reservo memoria para ronodo

        /* convertimos hcapa en h */
        h = malloc((l+1) * sizeof(double));
        for (i=0; i < Genes; i++)
                for (j=kx[i]+1; j <= kx[i+1]; j++)
                        h[j] = hcapa[i];

}

/** fin de archivo **/
```

C.4.25 Módulo MEASURE.C

```
/*
 *  archivo:        measure.c
 *
 *  objetivo:       calcular mediciones de performance
y adjuntarlas al archivo
 *                  output.
 *
 */

#include "extern.h"

#define DISPMEAS 5

Measure()
{
        double New_worst();
        FILE *fp, *fopen();
        register int i;
        register int w;
        register double performance;
        int j;

        for (i=0; i<Popsize; i++)
        {
                /* estadisticas de la poblacion
actual */
                performance = New[i].Perf;
                if (i>0)
                {
                        Ave_current_perf +=
performance;
                        if (BETTER(performance,
Best_current_perf))
                        {

        Best_current_perf = performance;
                                        Best_guy = i;
                        }
                        if
(BETTER(Worst_current_perf, performance))

        Worst_current_perf = performance;
                }
                else
                {
                        Best_current_perf =
performance;
                        Worst_current_perf =
performance;
                        Ave_current_perf =
performance;
                        Best_guy = 0;
                }

        }

        Ave_current_perf /= Popsize;

        /* actualizar Worst */
        if (Windowsize)
        {
                /* Worst = worst en las ultimas
(Windowsize) generaciones */
                w = Gen % Windowsize;
                Window[w] = New_worst();
                Worst = Window[0];
                for (i=1; i < Windowsize; i++)
                        if (BETTER(Worst,
Window[i])) Worst = Window[i];
        }
        else
                if (BETTER(Worst,
Worst_current_perf))
                        Worst = New_worst();

        /* actualizar medidas de performance global
*/
        Online = Onsum / Trials;
        Offline = Offsum / Trials;

        if (Traceflag)
        {
                printf("     Gen %d     Evals
%d\n",Gen,Trials);
                if (Onlnflag) printf("     Online
%e\n", Online);
                if (Offlnflag) printf("
Offline %e\n", Offline);
        }

        if (Displayflag)
```

```
        {
                static firstflag = 1;
                if (firstflag)
                {
                        firstflag = 0;
                        move(DISPMEAS - 1,0);
                        printw(" Gens Evals Perd
");
                        printw("Conv  Sesgo
Online      ");
                        printw("Offline
Mejor     Promedio");
                }

                move(DISPMEAS,0);
                clrtoeol();
                Converge();
                printw("%4d %6d %4d ",
                        Gen, Trials, Lost);
                printw("%4d  %5.3f %11.4f  ",
                        Conv, Bias, Online);
                printw("%11.4f  %11.6f  %11.6f",
                        Offline, Best,
Ave_current_perf);

                move(DISPMEAS+2,0);
                clrtoeol();
                printw(" Mejor estructura actual:
%3d  ", Best_guy);
                printw("Performance: %0.6f ",
New[Best_guy].Perf);
                move(DISPMEAS+3,0);
                clrtoeol();
                Unpack(New[Best_guy].Gene,
Bitstring, Length);
                if (Floatflag)
                {
                        FloatRep(Bitstring,
Vector, Genes);
                        for (j=0; j<Genes; j++)

        printw(Gene[j].format, Vector[j]);
                }
                else
                {
                        printw("%s", Bitstring);
                }
                move(DISPMEAS+5,0);
                printw(" Calificaci¢n de los
alelos de este individuo.");
                move(DISPMEAS+6,0);
                clrtoeol();
                for (j=0; j<Genes; j++)
                        printw(" %d",
New[Best_guy].BondadAlelo[j]);
                refresh();
        }

        if ( Interval && Collectflag && ((Trials >=
Plateau) || Doneflag))
        {
                /* agregar mediciones al archivo
output */
                Converge();
                fp = fopen(Outfile, "a");
                fprintf(fp,OUT_F2, OUT_V2);

                fclose(fp);
                Plateau =
(Trials/Interval)*Interval + Interval;
        }

        if (Logflag  && (Spin >= Maxspin))
        {
                fp = fopen(Logfile, "a");
                fprintf(fp, "Experimento %1d  ",
Experiment);
                fprintf(fp, "SPINNING en la Gen
%1d, ",Gen);
                fprintf(fp, "despu,s de  %1d
Evaluaciones\n", Trials);
                fclose(fp);
        }

        if ( Interflag && (Spin >= Maxspin))
        {
                move(22,0);
                clrtoeol();
                printw("SPINNING en la Gen %1d,
",Gen);
```

```
                    printw("despu,s de %1d
Evaluaciones\n", Trials);
                    refresh();
          }

}


double New_worst()
{
          double delta;

          /* Devolver un valor un poquito peor que
than Worst_current_perf */
/**** un poquito peor porque si comparo contra el peor
la distancia daria 0 ****/

          if (Maxflag)
                    delta = 1.0e-4;
          else
                    delta = -1.0e-4;

          if (Worst_current_perf == 0.0) return (-
delta);

          if (Worst_current_perf > 0.0)
                    return (Worst_current_perf*(1.0 -
delta));

          return (Worst_current_perf*(1.0 + delta));
}



static char BIT[CHARSIZE] ={ '\200', '\100', '\040',
'\020',
                              '\010', '\004', '\002',
'\001'};

Converge()                                /* medir la
convergencia de la poblacion  */
/* MODIFICADO PARA MEDIR A NIVEL DE MATERIAL */
/* NECESITO CONOCER EL NRO DE MATERIALES Y DEFINIR UN
VECTOR DE ESA DIMENSION */
/* este valor lo tengo en Gene[j].values */

{
          register int i,j;
          register int cantMatGanador; /* cantidad de
representantes del material

                                                ganador en un
gen dado */
          FILE *fp, *fopen();
          int *cantMatEnGen;

          Bias = 0.0;
          Lost = Conv = 0;
          if (!Convflag) return;
```

```
          /* en lugar de usar length usar la cantidad
de genes */
          for (j = 0; j < Genes; j++)
          {
                    /* Inicializo contadores en 0 */
                    cantMatEnGen =
malloc(Gene[j].values * sizeof(int) );
                    for (i=0; i < Gene[j].values; i++)
{
                              cantMatEnGen[i]=0;
                    }

                    for (i=0; i < Popsize; i++) {
                              /* ver si se puede
optimizar para que  no haya que desempaquetar
                              en mismo gen popsize
veces
                              */
                              Unpack(New[i].Gene,
Bitstring, Length);
                              if (Floatflag)
          FloatRep(Bitstring, Vector, Genes);

          cantMatEnGen[Vector[j]]++;
                    }

                    cantMatGanador=0;
                    for (i=0; i < Gene[j].values; i++)
{
                              if (cantMatEnGen[i] >
cantMatGanador) {

          cantMatGanador=cantMatEnGen[i];
                              }
                    }


                    Bias += cantMatGanador;

                    Lost += (cantMatGanador ==
Popsize);
                    Conv += (cantMatGanador >= Popsize
- FEW);
                    free(cantMatEnGen);
          }

          Bias /= (Popsize*Genes);

          if (Logflag && (Lost==Length))
          {
                    fp = fopen(Logfile, "a");
                    fprintf(fp, "CONVERGIO en Gen %1d,
",Gen);
                    fprintf(fp, "despu,s de %1d
Evaluaciones\n", Trials);
                    fclose(fp);
          }

}
/** fin de archivo **/
```

C.4.26 Módulo MUTATE.C

```
/*
 *  archivo:        mutatemp.c
 *
 *  purpose:        Mutar la poblacion usando
heuristica
 *
 *                  Se mutan todos los individuos de
acuerdo a la heuristica.
 */

#include "extern.h"


Mutate()
{

          static int last;    /* ultimo individuo en
sufrir Mutacion */

          register int i;               /* indice del
individuo a mutar */
          register int j;               /* indice del
gen a mutar */
          register int h;               /* si vale 0
puedo mutar un gen cualquiera del cromosoma, */

          /* sino puedo mutar un gen a partir del 1er
gen malo */
          register double k;   /* nuevo valor para el
gen */

          register int open;  /* Indica si el
cromosoma ya fue desempaquetado */
          static int firstflag = 1;

          if (firstflag)
          {

                    last = (M_rate*Popsize*Gapsize) -
0.5; /**** 0.5 no pasarme ***/
```

```
                    firstflag = 0;
          }
          open = -1;
          for (i=0; i < last; i++)
          {
                    // busco primer alelo malo.
                    for(j=0; j < Genes &&
New[i].BondadAlelo[j]; j++);
                    if (j < Genes) {
                              // => mutar desde el
primer malo en adelante con prob 0.9
                              // con prob 0.1 mutar
uno cualquiera
                              // (asumo que los que
restan son malos)
                              // es una suposicion
atinada dados los resultados empiricos
                              h = Randint(0,9);
                              if (h == 0)
                                        // muto uno
cualquiera
                                        j =
Randint(0,Genes-1);
                              else
                                        // muto desde
el primer gen malo.
                                        j =
Randint(j,Genes-1);
                              if (open != i)   /* hay
que desempaquetar la estructura i */
                              {
                                        Unpack
(New[i].Gene , Bitstring, Length);
                                        open = i;
                              }

                              FloatRep(Bitstring,
Vector, Genes);
                              k =
Randint(0,Gene[j].values-1);
```

```
                                if (k != Vector[j])  /*
es una mutacion verdadera */
                                {
                                        Vector[j] = k;

          New[i].Needs_evaluation = 1;
                                }
                                if
(New[i].Needs_evaluation) {

          StringRep(Vector, Bitstring, Genes);
                                        Pack (
Bitstring , New[i].Gene , Length);
                                }

                    }
          }
}

/** fin de archivo **/
```

C.4.27 Módulo MUTATEMP.C

```
/*
 *  archivo:        mutatemp.c
 *
 *  purpose:        Mutar la poblacion usando
heuristica
 *
 *                  Se mutan todos los individuos de
acuerdo a la heuristica.
 */

#include "extern.h"


Mutate()
{

          static int last;    /* ultimo individuo en
sufrir Mutacion */

          register int i;               /* indice del
individuo a mutar */
          register int j;               /* indice del
gen a mutar */
          register int h;               /* si vale 0
puedo mutar un gen cualquiera del cromosoma, */

          /* sino puedo mutar un gen a partir del 1er
gen malo */
          register double k;   /* nuevo valor para el
gen */

          register int open;  /* Indica si el gen ya
fue desempaquetado */
          static int firstflag = 1;

          if (firstflag)
          {

                    last = (M_rate*Popsize*Gapsize) -
0.5; /**** 0.5 no pasarme ****/
                    firstflag = 0;
          }
          open = -1;
          for (i=0; i < last; i++)
          {
                    // busco primer alelo malo.
                    for(j=0; j < Genes &&
New[i].BondadAlelo[j]; j++);
                    if (j < Genes) {
                              // => mutar desde el
primer malo en adelante con prob 0.9
                              // con prob 0.1 mutar
uno cualquiera
                              // (asumo que los que
restan son malos)
                              // es una suposicion
atinada dados los resultados empiricos
                              h = Randint(0,9);
                              if (h == 0)
                                        // muto uno
cualquiera
                                        j =
Randint(0,Genes-1);
                              else
                                        // muto desde
el primer gen malo.
                                        j =
Randint(j,Genes-1);
                              if (open != i)   /* hay
que desempaquetar la estructura i */
                              {
                                        Unpack
(New[i].Gene , Bitstring, Length);
                                        open = i;
                              }

                              FloatRep(Bitstring,
Vector, Genes);
                              k =
Randint(0,Gene[j].values-1);

                              if (k != Vector[j])  /*
es una mutacion verdadera */
                              {
                                        Vector[j] = k;

          New[i].Needs_evaluation = 1;
                              }
                              if
(New[i].Needs_evaluation) {

          StringRep(Vector, Bitstring, Genes);
                                        Pack (
Bitstring , New[i].Gene , Length);
                              }

                    }
          }
}

/** fin de archivo **/
```

C.4.28 Módulo PRESION.C

```
/*
 *  archivo:        presion.c
 *
 *  objetivo:   Calcular las presiones generadas por
un juego de velocidades.
 *
 */
#include <alloc.h>
#include <math.h>
#include <stdio.h>
#include <assert.h>
#include <stdlib.h>
#include "presion.h"
#include "in_eval2.h"
/*-------------------------------------------------------
----------------*/
double una_presion(unm1, un, unp1, paso, constk,
velnodo)
          double *unm1, *un, *unp1, *constk, *velnodo;
          int paso;
{
          double aux1, aux2;
          int k;

          assert(heapcheck()==_HEAPOK);

          /* condicion de bordes */
          unp1[1]= borde_sup(unm1[1], un[1], un[2],
constk[1]);
          unp1[l+1]= borde_inf( unm1[l+1], un[l],
un[l+1], constk[l+1]);

          /* agregar la contribucion de la fuente a
cada borde */
          /* primero al superior */
          unp1[1]+= fuente_en_borde(paso, 1);
          unp1[1]/= 1 + constk[1];
          /* y ahora al inferior */
          unp1[l+1]+= fuente_en_borde(paso, l+1);
          unp1[l+1]/= 1 + constk[l+1];

          /* calculo de la presion de los nodos
interiores */
          for (k=2; k <= l; k++) {
                    unp1[k]= 2 * un[k] - unm1[k] + 2 *
constk[k] * pow(dt, 2) *
                    ( (un[k+1] - un[k]) / ( h[k] *
ronodo[k] ) -
                    ( un[k] - un[k-1] ) / ( h[k-1] *
ronodo[k-1] ));


                    /* agregar la contribucion de la
fuente */
                    unp1[k]+= fuente_interior( paso,
k, constk[k], velnodo);
          }

          /* Interpolacion lineal de las presiones en
los dos nodos que rodean al
                    receptor */
          aux1= (xjrecp1 - xrec) / h[jrec] *
unp1[jrec];
          aux2= (xrec - xjrec) / h[jrec] *
unp1[jrec+1];

          assert(heapcheck()==_HEAPOK);
          return (aux1 + aux2);
```

```
}

/*-------------------------------------------------
----------------*/
void presiones(velocidad, capas, presion, nt)
          /* velocidades por cada capa, parametro de
entrada */
          double *velocidad;
          int capas;  /* cantidad de capas */

          /* presiones calculadas a partir de las
velocidades */
          double *presion;
          int nt;             /* cantidad de
intervalos en el tiempo */
{

          int i;              /* auxiliar */
          double *aux;  /* puntero auxiliar para hacer
swap */
          double *unm1; /* presiones en el paso n-1 */
          double *un; /* presiones en el paso n   */
          double *unp1; /* presiones en el paso n+1 */
          double *velnodo;  /* velocidades en nodos */
          double *constk; /* constantes auxiliares */

          unm1= malloc((l+2)*sizeof(double)); /*
presiones en el paso n-1 */
          un= malloc((l+2)*sizeof(double));   /*
presiones en el paso n   */
          unp1= malloc((l+2)*sizeof(double)); /*
presiones en el paso n+1 */
          velnodo= malloc((l+2)*sizeof(double));  /*
velocidades en nodos */
          constk= malloc((l+2)*sizeof(double)); /*
constantes auxiliares */
          vel_en_nodos(velocidad, capas, velnodo);

          assert(heapcheck()==_HEAPOK);

          for ( i=1; i <= (l+1); i++) {
                    unm1[i]= 0;
                    un[i]= 0;
          }

          /* Condiciones iniciales */
          presion[1]= 0;
          presion[2]= 0;

          /* Tiempo de espera desde que comienza la
explosion hasta que se comienza
             a escuchar (mute) */
          for ( i= 3; i <= ntmute; i++)
                    presion[i]= 0;

          /* constantes auxiliares */
          constk[1]= velnodo[1] * dt / h[1];
          constk[l+1]= velnodo[l] * dt / h[l];
          for ( i = 2; i <= l; i++)
                    constk[i]= ( pow(velnodo[i], 2) *
pow(velnodo[i-1], 2) * ronodo[i] * ronodo[i-1]) /

          (h[i-1] * pow(velnodo[i], 2) * ronodo[i] +
h[i] * pow(velnodo[i-1], 2) * ronodo[i-1]);

          /* calcula la presion en el paso i */
          for ( i= max(ntmute+1,3); i <= nt; i++) {
                    presion[i]= una_presion(unm1, un,
unp1, i, constk, velnodo);
                    aux= unm1;    /* lo guardo para
despues */
                    unm1= un;     /* avanza un paso en
el tiempo */
                    un= unp1;     /*    "    "   "   "
"    "   */
                    unp1= aux;    /* no interesan los
valores pero si el espacio de memoria */
          }

          free(unm1);
          free(un);
          free(unp1);
          free(velnodo);
          free(constk);
          /* el resultado va en 'presion' */

          assert(heapcheck()==_HEAPOK);
}
/*-------------------------------------------------
----------------*/
void vel_en_nodos(velocidad, capas, velnodo)
double *velocidad;
int capas;
double *velnodo;
{
          int c, nodo;
          for (c=0; c <capas; c++)
                    for (nodo=kx[c]+1; nodo <=
kx[c+1]; nodo++)
                              velnodo[nodo] =
velocidad[c];
          velnodo[l+1]= velocidad[capas-1];
}
/*-------------------------------------------------
----------------*/
double borde_sup(unm1, unx1, unx2, cte)
/* Calcula la presion en el borde superior (sin
fuente) */
double unm1;
double unx1;
double unx2;
double cte;
{
          double presion;
          switch (keyb) {
                    case 0: /* Condicion de bordes de
Dirichlet (superficie libre) */
                              presion = 0;
                              break;
                    case 1: /* Condicion de bordes
absorbente */
                              presion = 2 *
unx1 - unm1 + 2 * pow(cte, 2) * (unx2 - unx1) +

          cte * unm1;
                              break;
                    case 2: /* Condicion de bordes de
Neumann  */
                              presion = 2 *
unx1 - unm1 + 2 * pow(cte, 2) * (unx2 - unx1);
                              break;
          }
          return presion;
}
/*-------------------------------------------------
----------------*/
double borde_inf(unm1, unx1, unx2, cte)
/* Calcula la presion en el borde inferior (sin
fuente) */
double unm1; /* presion en el paso anterior */
double unx1; /* presion en el anteultimo nodo */
double unx2; /* presion en el ultimo nodo */
double cte;  /* constante ad hoc */
{
  double presion;
  presion = 2 * unx2 - unm1 + 2 * pow(cte, 2) * (unx1
- unx2) +

          cte * unm1;
  return presion;
}
/*-------------------------------------------------
----------------*/
/* Las tres rutinas de fuente podrian unificarse */
double fuente_en_borde(paso, nodo)
/* Contribucion de la fuente en el borde indicado por
nodo en el paso 'paso' */
int paso;
int nodo;
{
          double eval, presion;
          switch (kdel) {
                    case 0: /* delta de Dirac en xsou
* source[n] */
                              if (1 == jsou)
                                        eval =
(xjsoup1 - xsou) / h[jsou];
                              else if (1 == jsou+1)
                                                  eval
= (xsou - xjsou) / h[jsou];
                                        else
                                                  eval
= 0;
                              presion = 2 * pow(dt, 2)
* source[paso] * eval / h[nodo];
                              break;

                    case 1: /* derivada del delta de
Dirac en
                                                  .5 * (xsou
+ xsou + h) * source[n] */
                              presion = 2 * pow(dt, 2)
* source[paso] / pow(h[nodo], 2);
                              if (1 == jsou) {
                                        /* no hacemos
nada */
                              }
                              else if (1 == jsou + 1)
{
                                        presion *= -
1;
                                        }
                                        else {

          presion = 0;
                                        }
                              break;

                    case 2: /* 2 delta de Dirac en
xsou * source[n] */
                              presion = 2 * pow(dt, 2)
* source[paso] * h[nodo] / 2.0;
                              if (1 == jsou - is) {
                                        /* no hacemos
nada */
                              }
                              else if (1 == jsou + is)
{
                                        presion *= -
1;
                                        }
                                        else {

          presion = 0;
                                        }
                              break;
          }
          return presion;
```

```
}

/*-------------------------------------------------------
--------------------*/
double fuente_interior(paso, nodo, cte, velnodo)
/* Contribucion de la fuente en el un nodo interior */
int paso;
int nodo;
double cte; /* constante auxiliar */
double *velnodo;
{
          double eval, presion;
          switch (kdel) {
                    case 0: /* delta de Dirac en xsou
* source[n] */

                              if (nodo == jsou)
                                        eval =
(xjsoup1 - xsou) / h[jsou];
                              else if (nodo == jsou+1)
                                                  eval
= (xsou - xjsou) / h[jsou];
                                         else
                                                  eval
= 0;

                              presion = 2 * pow(dt, 2)
* cte * source[paso] * eval /

(pow(velnodo[jsou], 2) * ronodo[jsou]);
                              break;

                    case 1: /* derivada del delta de
Dirac en
                                                  .5 * (xsou
+ xsou + h) * source[n] */

                              if (nodo == jsou) {
                                        presion = 2 *
pow(dt, 2) * cte * source[paso] /

(pow(h[jsou], 2) * pow(velnodo[jsou], 2) *
ronodo[jsou]);
                              }
```

```
                              else if (nodo == jsou +
1) {
                                                  presion = 2 *
pow(dt, 2) * cte * source[paso] /

(pow(h[jsou], 2) * pow(velnodo[jsou], 2) *
ronodo[jsou]);
                                                  presion *= -
1;
                                                  }
                                                  else {

                    presion = 0;
                                                  }
                              break;

                    case 2: /* 2 delta de Dirac en
xsou * source[n] */

                              if (nodo == jsou - is) {
                                                   presion = 2 *
pow(dt, 2) * cte * source[paso] *

(pow(velnodo[jsou], 2) * ronodo[jsou]) / 2.0;
                                                  }
                              else if (nodo == jsou +
is) {
                                                  presion = 2 *
pow(dt, 2) * cte * source[paso] *

(pow(velnodo[jsou], 2) * ronodo[jsou]) / 2.0;
                                                  presion *= -
1;
                                                  }
                                                  else {

                    presion = 0;
                                                  }
                              break;
          }
          return presion;
}

/*** fin de archivo ***/
```

C.4.29 Módulo RESTART.C

```
/*
 *  archivo:        restart.c
 *
 *  objetivo:       correr AG a partir de una corrida
anterior
 *
 */

#include <dir.h>
#include "extern.h"

extern void Readbest();


Restart()
{
          FILE *fp, *fopen();
          int i,j;
          char msg[40];
          char drive[MAXDRIVE], dir[MAXDIR],
file[MAXFILE], ext[MAXEXT];

          fp = fopen(Ckptfile, "r");
          if (fp == NULL)
          {
                    sprintf(msg,"Restart: Ckptfile %s
no se encuentra",
                              Ckptfile);
                    Error(msg);
          }

          fscanf(fp, "Experimentos %d ", &Experiment);
          fscanf(fp, "On_line total   %lf ",
&Totonline);
          fscanf(fp, "Off_line total   %lf ",
&Totoffline);
          fscanf(fp, "Generaci¢n %d ", &Gen);
          fscanf(fp, "Performance On_line  %lf ",
&Onsum);
          fscanf(fp, "Performance Off_line %lf ",
&Offsum);
          fscanf(fp, "Evaluaciones  %d ", &Trials);
```

```
          fscanf(fp, "Pr¢xima estad¡stica %d ",
&Plateau);
          fscanf(fp, "Mejor %lf ", &Best);
          fscanf(fp, "Peor   %lf ", &Worst);
          fscanf(fp, "Cantidad de generaciones desde
la £ltima evaluaci¢n %d ", &Spin);
          fscanf(fp, "Ultimo vuelco %d ", &Curr_dump);
          fscanf(fp, "Mu_prox %d ", &Mu_next);
          fscanf(fp, "Semilla Aleatoria %lu ", &Seed);
          fscanf(fp, "Semilla Inicializadora %lu ",
&Initseed);

          fscanf(fp, " Window ");
          for (i=0; i<Windowsize; i++) fscanf(fp,
"%lf", &Window[i]);

          for (i=0; i<Popsize; i++)
          {
                    fscanf(fp, "%s", Bitstring);
                    fscanf(fp, " %lf ", &New[i].Perf);
                    Pack(Bitstring, New[i].Gene,
Length);
                    fscanf(fp, " %d ",
&New[i].Needs_evaluation);

                    for (j=0; j < Genes; j++)
                              fscanf(fp, "%d",
&New[i].BondadAlelo[j]);

          }
          fclose(fp);

          fnsplit(Minfile, drive, dir, file, ext);
          sprintf(Bestfile, "%s%d%s", file,
Experiment, ext);

          Readbest();

}

/** fin de archivo **/
```

C.4.30 Módulo SCHEMA.C

```
/*
 *  archivo:        schema.c
 *
 *  objetivo:       medir la cantidad de individuos
pertenecientes a un esquema
 *                                        y guardar los
resultados en el archivo Schema
 *
 */
```

```
#include <alloc.h>
#include "extern.h"

#define DISPSCH 9


Schema()
{
```

```
        int i;
        double expected;
        double perf;
        int count;
        static int lastcount = 1;
        char msg[40];
        int ok;
        static int firstflag = 1;
        static int firstcount = 1;

        FILE *fopen();
        static FILE *fp;
        int j;
        char tmp;
        static char *S;


        if (firstflag)
        {
        /*  inicializar el esquema S a partir del
schemafile */
                S = malloc((unsigned) (Length +
1));
                if ((fp = fopen(Schemafile, "r"))
== NULL)
                {
                        sprintf(msg, "Schema: no
puede abrirse %s", Schemafile);
                        Error(msg);
                }
                for (i=0; i<Length; i++)
                {
                        fscanf(fp, "%c", &S[i]);
                }
                fclose(fp);
                S[Length] = '\0';
                if ((fp = fopen(Schemafile, "w"))
== NULL)
                {
                        sprintf(msg, "Schema: no
puede abrirse %s", Schemafile);
                        Error(msg);
                }
                fprintf(fp, "%s\n", S);
                fprintf(fp, " Gen  Cant   Incr
Esper  ");
                fprintf(fp, "Esquem Prom   Prom
Pob\n");

                if (Displayflag)
                {
                        move(DISPSCH,0);
                        clrtoeol();
                        printw("Esquema:");
                        move(DISPSCH+1,0);
                        clrtoeol();
                        printw("%s", S);
                        move(DISPSCH+3,0);
                        clrtoeol();
                        printw(" Gen  Cant
Incr  Esper  ");
                        printw("Esquem Prom
Prom Pob\n");
                        refresh();
                }

                firstflag = 0;
        }

        /* guardar cantidad de individuos
pertenecientes al esquema S
        y la cantidad esperada de hijos que
pertenezcan a S en la proxima
        generacion          */
        expected = 0.0;
        perf = 0.0;
        count = 0;
        for (i=0; i<Popsize; i++)
        {
                Unpack(New[i].Gene, Bitstring,
Length);
                for (ok = 1, j = 0; ok &&
(j<Length); j++)
                {
                        ok = (S[j] == '#') ||
(S[j] == Bitstring[j]);
                }
                if (ok)
                {
                        count++;
                        expected += (New[i].Perf
- Worst) /
(Ave_current_perf - Worst);
                        perf += New[i].Perf;
                }
        }

        if (firstcount && count)
        {
                lastcount = count;
                firstcount = 0;
        }

        if (Displayflag)
        {
                move(DISPSCH+4,0);
                clrtoeol();
                printw("%4d  %4d ", Gen, count);
                printw(" %5.3f ",
(count*1.0)/lastcount);
                if (count)
                {
                        printw(" %5.3f ",
expected/count);
                        printw(" %10.3e ",
perf/count);
                }
                else
                {
                        printw(" %5.3f ", 0.0);
                        printw(" %10.3e ", 0.0);
                }
                printw(" %10.3e ",
Ave_current_perf);
                refresh();
        }

        fprintf(fp, "%4d  %4d ", Gen, count);
        fprintf(fp, " %5.3f ",
(count*1.0)/lastcount);
        if (count)
        {
                fprintf(fp, " %5.3f ",
expected/count);
                fprintf(fp, " %10.3e ",
perf/count);
                lastcount = count;
        }
        else
        {
                fprintf(fp, " %5.3f ", 0.0);
                fprintf(fp, " %10.3e ", 0.0);
        }
        fprintf(fp, " %10.3e ", Ave_current_perf);
        fprintf(fp, "\n");

}

/***  fin de archivo ***/
```

C.4.31 Módulo SELECT.C

```
/*
 *  archivo:        select.c
 *
 *  objetivo:       seleccionar los individuos para la
nueva generacion
 *
 */
#include <alloc.h>
#include "extern.h"

Select()
{
        static firstflag = 1;         /* indica
primera ejecucion                */
        static int *sample; /* apunta a las
estructuras seleccionadas        */
        double expected;    /* numero especrado de
hijos                */
        double factor;                /*
normalizador para valor esperado        */
        double perf;                  /* mejor
performance siguiente (para ranking)                */
        double ptr;                   /* determina
la distribucion de la parte fraccionaria */
        double rank_max;    /* max numero de hijos
(en ranking) */
        double sum;
        int best;           /* indice a la mejor
estructura siguiente                    */
        register int i;
        register int j;
        register int k;
        register int temp;  /* usado para
intercambiar punteros                    */

        if (firstflag)
        {
                sample = (int *) calloc((unsigned)
Popsize, sizeof(int));
                firstflag = 0;
        }

        if (Rankflag)
        {
                /* Asignar a cada estructura su
nro de orden (rank) dentro de la poblacion. */
                /* rank = Popsize-1 para el mejor,
rank = 0 para el peor        */
                /**** Se usa el campo
Needs_evaluation para guardar el valor de rank ****/

                /* limpiar el campo donde se
guarda el valor de rank */
                for (i=0; i<Popsize; i++)
                        Old[i].Needs_evaluation
= 0;

                for (i=0; i < Popsize-1; i++)
```

```
                    {
                              /* buscar la mejor i-
esima estructura */
                              best = -1;
                              perf = 0.0;
                              for (j=0; j<Popsize;
j++)
                              {
                                        if
(Old[j].Needs_evaluation == 0 &&

(best == -1 || BETTER(Old[j].Perf,perf)))
                                        {
                                                  perf
= Old[j].Perf;
                                                  best
= j;
                                        }
                              }
                              /* ponerle el orden a la
estructura */

          Old[best].Needs_evaluation = Popsize -1 - i;
                    }
                    /* normalizador para las
probabilidades de seleccion por ranking */
                    rank_max = 2.0 - Rank_min;
                    /**** Calcula el tama¤o del slot
patron ****/
                    factor = (rank_max - Rank_min) /
(double) (Popsize -1);
                    }
                    else
                    {
                    /* normalizador para las
probabilidades de seleccion proporcional */
                    /**** Calcula el tama¤o del slot
patron ****/
                    factor = Maxflag ?
1.0/(Ave_current_perf - Worst) :
                                                  1.0/(Worst
- Ave_current_perf);
          }

          /* Algoritmo de muestreo estocastico de
James E. Baker */

          k=0;                    /* indice de la proxima
estructura a elegir */

          ptr = Rand();   /* girar la ruleta una vez
*/

          for (sum=i=0; i < Popsize; i++)
          {
                    if (Rankflag)
                    {
                              /**** Mapea los
individuos en forma lineal entre
                                        Rank_min y Rank_max
****/
                              expected = Rank_min +
Old[i].Needs_evaluation * factor;
                    }
                    else
                    {
                              /**** Mapea en forma
proporcional a la distancia con
                                        el peor individuo
****/
                              if (Maxflag) {
                                        if
(Old[i].Perf > Worst)
                                                  expected =
(Old[i].Perf - Worst) * factor;
                                        else expected
= 0.0;
                              }
                              else {
                                        if
(Old[i].Perf < Worst)
                                                  expected =
(Worst - Old[i].Perf) * factor;
                                        else expected
= 0.0;
                              }
                    }
/**** Asigna entre floor y ceil de la cantidad
esperada de hijos ****/
                    for (sum += expected; sum > ptr;
ptr++){
                      sample[k++] = i;
                    }
          }
```

```
          if (k != Popsize) {
            printf("select: Se seleccionaron %d
individuos en lugar de %d\n", k, Popsize);
            abort();
          }

          /* mezclar al azar los punteros de las
nuevas estructuras */
/**** Pensar en que pasa si un individuo recibe mas de
una copia en la
            seleccion. Luego, en la cruza, el padre y
la madre serian muy probablemente
            el mismo. Por eso se mezcla! ****/
          for (i=0; i<Popsize; i++)
          {
                    j = Randint(i,Popsize-1);
                    temp = sample[j];
                    sample[j] = sample[i];
                    sample[i] = temp;
          }

          if (Gapsize<1.0)              /* Salto
generacional */
                    Gap(sample);

          /* finalmente, formar la nueva poblacion */
          for (i=0; i<Popsize; i++)
          {
                    k = sample[i];
                    for (j=0; j<Bytes; j++)
                    {
                              New[i].Gene[j] =
Old[k].Gene[j];
                    }
                    New[i].Perf = Old[k].Perf;
                    New[i].Needs_evaluation = 0;

                    for (j=0; j<Genes; j++) {
                              New[i].BondadAlelo[j] =
Old[k].BondadAlelo[j];
                    }


          }

}



/* Elegir sobrevivientes de la vieja poblacion
uniformemente sin reposicion */

Gap(sample)
int sample[];
/**** (Gap == 0) => no hay seleccion
                              no hay cruza
            Se destruye una porcion de la poblacion
seleccionada y se reemplaza
            por indiv. de la poblacion ORIGINAL sin
tener en cuenta su aptitud ****/
{
          static firstflag = 1;
          static int *survivors;        /* permutacion
al azar de 0 .. Popsize-1 */
          register int i,j;
          int temp;             /* para swapear
                                   */

          if (firstflag)
          {
                    survivors = (int *)
calloc((unsigned) Popsize, sizeof(int));
                    firstflag = 0;
          }

          /* Mezclar uniformemente */
          for (j=0; j<Popsize; j++) survivors[j]=j;
          for (j=0; j<Popsize; j++)
          {
                    i = Randint(j, Popsize-1);
                    temp = survivors[i];
                    survivors[i] = survivors[j];
                    survivors[j] = temp;
          }

          /* nuevos sobrevivientes elegidos */
          for (i=Gapsize*Popsize; i<Popsize; i++)
                    sample[i] = survivors[i];
}

/*** fin de archivo ***/
```