

Desarrollo de Algoritmos Genéticos en Lenguaje C
aplicados a la resolución de problemas de
optimización

Luis Marco Giménez

UNED - 2005

Genetic Algorithms
Implementation in C Language
applied to the resolution of
optimization's problems

Resumen

Los Algoritmos Genéticos, como paradigma principal de la computación evolutiva, presentan una alternativa a los procedimientos tradicionales de búsqueda y optimización como métodos sistemáticos para la resolución de estos problemas. En concreto, y en el ámbito de este trabajo, los algoritmos genéticos se han aplicado con buenos resultados en problemas de optimización en los que se desea localizar los máximos o mínimos de una función matemática determinada. La herramienta desarrollada para este proyecto fin de carrera, ***SIGENOF*** -*Sistema Genético para la Optimización de Funciones*-, consiste en la implementación de un algoritmo genético para la optimización de una amplia variedad de funciones matemáticas de las que se desea localizar sus máximos globales. La principal aportación de esta herramienta frente a otras implementaciones genéticas es la incorporación de un sencillo e intuitivo interfaz gráfico de usuario que permite, entre otras funcionalidades, la introducción de la función a optimizar junto con los parámetros que guiarán la ejecución del algoritmo genético, representación gráfica de las mejores soluciones localizadas por la herramienta, así como la generación de ficheros con los resultados de la ejecución que permiten realizar un análisis posterior de las soluciones encontradas.

Abstract

Genetic Algorithms, as a major main paradigm of evolutionary computation, present a good alternative to traditional procedures of searching and optimizing as systematic methods for the resolution of this kind of problems. Genetic algorithms have been applied with good results in optimizing problems where the objective is to obtain the maximus or minimus about a mathematical function. The implemented tool in this degree work, ***SIGENOF*** -*Sistema Genético para la Optimización de Funciones*- (*Genetic System for the Optimization Functions*), consist in the implementation of a genetic algorithm for the optimization of a wide variety of mathematical functions of which we want to obtain their local maximums. The main contribution of this tool compared to other genetic implementations is its easy and intuitive graphical user interface allowing, among others functions, the input of the function to be optimized, the parameters of the genetic algorithm that will guide its execution, a graphical representation of the best solutions founded by the tool, and a generation of files within the execution's results that will allow a further analysis of the founded solutions.

Lista de palabras clave

Algoritmo Genético, Aptitud, Búsqueda ciega, Búsqueda estocástica, Computación Evolutiva, Cromosoma, Cruce, Elitismo, Evolución biológica, Gen, Individuo, Maximización, Mutación, Optimización, Población, Reproducción, Selección, Solución.

Keywords

Genetic Algorithm, Fitness, Blind search, Stochastic search, Evolutionary Computation, Chromosome, Crossover, Elitism, Biological evolution, Gen, Individual, Maximization, Mutation, Optimization, Population, Reproduction, Selection, Solution.

Siglas, Abreviaturas y Acrónimos

AG: Algoritmo Genético

ASCII: American Standard Code for Information Interchange

GIMP: GNU Image Manipulation Program

GNU: GNU's Not Unix

GPL: General Public License

GTK: GIMP Toolkit

GUI: Interfaz Gráfico de Usuario

SIGENOF: Sistema Genético para la Optimización de Funciones

Índice general

| | |
|--------------------------------------------------------------------------------------------------|-----------|
| Lista de palabras clave | 1 |
| Siglas, Abreviaturas y Acrónimos | 5 |
| 1. Introducción | 15 |
| 1.1. Preliminares | 15 |
| 1.2. Computación Evolutiva | 16 |
| 1.3. Algoritmos Genéticos | 17 |
| 1.4. Características de los AG como métodos de búsqueda | 18 |
| 1.5. Métodos de Optimización y Búsqueda tradicionales | 19 |
| 1.6. Diferencias de los AG con los métodos de búsqueda y optimización tradicionales | 20 |
| 1.7. Funcionamiento de un AG básico | 20 |
| 1.8. Herramientas software actuales que implementan AG | 23 |
| 1.9. Herramienta SIGENOF | 24 |
| 1.10. Problemas de optimización que resuelve SIGENOF | 25 |
| 1.11. Estructura de la memoria | 26 |
| 2. Algoritmo Genético implementado en SIGENOF | 29 |
| 2.1. Lenguaje de implementación utilizado | 29 |
| 2.2. Estructuras de datos | 30 |
| 2.3. Sintaxis para la definición de funciones J | 33 |
| 2.3.1. Variables y Constantes: | 34 |
| 2.3.2. Operadores disponibles: | 34 |
| 2.3.3. Funciones predeterminadas: | 35 |

| | |
|-------------------------------------------------------------------------|-----------|
| 2.3.4. Reglas de precedencia: | 36 |
| 2.4. Algoritmo genético implementado | 37 |
| 2.4.1. Paso 1. Inicialización | 37 |
| 2.4.2. Paso 2. Evaluación | 39 |
| 2.4.3. Paso 3. Mantener al mejor individuo | 39 |
| 2.4.4. Paso 4. Bucle de ejecución del AG hasta MAXGENS | 40 |
| 2.4.5. Paso 5. Terminación del AG | 45 |
| 3. Interfaz Gráfico de Usuario de SIGENOF | 47 |
| 3.1. Librería gráfica utilizada | 47 |
| 3.2. Visión general | 48 |
| 3.3. Funcionalidades del GUI | 48 |
| 3.3.1. Panel 1: Parámetros Poblacionales | 50 |
| 3.3.2. Panel 2: Parámetros Naturales | 50 |
| 3.3.3. Panel 3: Otros Parámetros del AG | 50 |
| 3.3.4. Panel 4: Ventana de Resultados | 51 |
| 3.3.5. Panel 5: Función J a Optimizar | 52 |
| 3.3.6. Panel 6: Gráfico de Soluciones generacionales | 55 |
| 3.3.7. Panel 7: Datos estadísticos y Tiempo de ejecución | 55 |
| 3.3.8. Panel 8: Progreso y Control del proceso | 55 |
| 3.4. Ejemplo de uso del GUI | 56 |
| 3.4.1. Paso 1. Iniciar SIGENOF | 57 |
| 3.4.2. Paso 2. Definir la función J a optimizar | 58 |
| 3.4.3. Paso 3. Establecer el espacio S | 59 |
| 3.4.4. Paso 4. Grabar la función J y su espacio S definido. | 60 |
| 3.4.5. Paso 5. Establecer los parámetros poblacionales | 61 |
| 3.4.6. Paso 6. Establecer los parámetros naturales del AG | 62 |
| 3.4.7. Paso 7. Establecer otros parámetros del AG | 63 |
| 3.4.8. Paso 8. Seleccionar el tipo de visualización gráfica | 63 |
| 3.4.9. Paso 9. Comenzar con la ejecución | 64 |
| 3.4.10. Paso 10. Evaluación de las soluciones obtenidas | 64 |

| | |
|------------------------------------------------------------------------------------------------------------------------|-----------|
| 4. Ejemplos de aplicación de SIGENOF | 67 |
| 4.1. Caso particular 1 | 67 |
| 4.2. Caso particular 2 | 68 |
| 4.3. Caso particular 3. Función de Ackley | 69 |
| 4.4. Caso particular 4. Función de Shaffer | 70 |
| 4.5. Caso particular 5 | 71 |
| 4.6. Caso particular 6 | 73 |
| 4.7. Ejecución con variantes | 74 |
| 4.7.1. Sin variación de parámetros (Ejecución 1) | 75 |
| 4.7.2. Variación de la probabilidad de mutación (Ejecución 2) | 75 |
| 4.7.3. Variación de las probabilidades de cruce y mutación (Ejecución 3) | 77 |
| 4.7.4. Variación del tamaño de la población a un valor pequeño (Ejecución 4) | 78 |
| 4.7.5. Variación del n° de generaciones a un tamaño pequeño (Ejecución 5) | 80 |
| 4.7.6. Variación del n° de generaciones a un tamaño grande (Ejecución 6) | 80 |
| 4.7.7. Variación del tamaño de la población a un valor grande (Ejecución 7) | 82 |
| 4.7.8. Variación de la probabilidad de mutación y del tamaño de la población a un valor grande (Ejecución 8) | 83 |
| 5. Conclusiones | 85 |
| A. Gramática para Funciones J en SIGENOF | 87 |
| B. Función <i>ran2</i> | 89 |
| B.1. Fichero aleatorio.h | 89 |
| B.2. Fichero aleatorio.c | 90 |
| C. Código del AG de SIGENOF | 93 |
| C.1. Fichero ag.h | 93 |
| C.2. Fichero ag.c | 95 |

| | |
|-------------------------------------------|------------|
| D. Ficheros de salida de SIGENOF | 111 |
| D.1. Descripción de las salidas | 111 |
| D.2. Fichero salida_ag | 111 |
| D.3. Fichero soluciones_ag | 112 |
| Referencias Bibliográficas | 115 |

Índice de figuras

| | |
|----------------------------------------------------------------------------|----|
| 1.1. Codificación de Soluciones en los Cromosomas | 18 |
| 1.2. Esquema de funcionamiento de un AG básico | 21 |
| 1.3. Estructura genérica del bucle básico de ejecución de un AG | 22 |
| 2.1. Esquema de un algoritmo genético básico | 37 |
| 2.2. Generación de 1000 números reales aleatorios | 38 |
| 2.3. Generación de 5000 números reales aleatorios | 39 |
| 2.4. Método de la Ruleta | 41 |
| 2.5. Cruce en un solo punto | 43 |
| 2.6. Cruce en un solo punto en SIGENOF | 43 |
| 3.1. GUI de SIGENOF | 49 |
| 3.2. Función de ejemplo | 56 |
| 3.3. Inicio de SIGENOF | 58 |
| 3.4. Definición de la función J y del espacio S | 60 |
| 3.5. Grabación de la función J y espacio S | 61 |
| 3.6. Parámetros poblacionales | 62 |
| 3.7. Parámetros naturales | 62 |
| 3.8. Otros parámetros | 63 |
| 3.9. Tipo de visualización gráfica | 64 |
| 3.10. Soluciones obtenidas por SIGENOF para el ejemplo propuesto | 64 |
| 3.11. Resultados estadísticos para el ejemplo propuesto | 64 |
| 3.12. Máximo para el ejemplo propuesto | 65 |
| 4.1. Caso Particular 1 | 68 |

| | |
|------------------------------------------------------------------------------------|----|
| 4.2. <i>Caso Particular 3</i> | 70 |
| 4.3. <i>Caso Particular 4</i> | 71 |
| 4.4. <i>Caso Particular 5 (vista 1)</i> | 72 |
| 4.5. <i>Caso Particular 5 (vista 2)</i> | 72 |
| 4.6. <i>Caso Particular 6</i> | 73 |
| 4.7. <i>Caso inicial - Generación 1</i> | 76 |
| 4.8. <i>Variación probabilidad de mutación</i> | 77 |
| 4.9. <i>Variación probabilidades de cruce y mutación</i> | 78 |
| 4.10. <i>Variación a un tamaño de la población pequeño</i> | 79 |
| 4.11. <i>Variación a un n° de generaciones pequeño</i> | 81 |
| 4.12. <i>Variación a un n° de generaciones grande</i> | 82 |
| 4.13. <i>Variación a un tamaño de la población grande</i> | 83 |
| 4.14. <i>Variación tamaño población y probabilidad mutación a valores grandes</i> | 84 |

Índice de cuadros

| | |
|-------------------------------------------------------------------------------|----|
| 2.1. <i>Definición del cromosoma</i> | 31 |
| 2.2. <i>Definición de la función J a optimizar</i> | 32 |
| 2.3. <i>Sintaxis para la definición de funciones J</i> | 33 |
| 2.4. <i>Definición de una función J</i> | 37 |
| 4.1. <i>Parámetros del AG utilizados en los ejemplos</i> | 67 |

Capítulo 1

Introducción

1.1. Preliminares

Para la correcta comprensión de esta memoria a continuación se van a definir una serie de conceptos básicos necesarios para la lectura del mismo y que se enmarcan dentro del contexto de los Algoritmos Genéticos.

- **Algoritmo Genético:** Técnica de programación que imita la evolución biológica como estrategia para resolver problemas.
- **Población:** Conjunto de candidatos que representan soluciones potenciales con las que el Algoritmo Genético trabajará para resolver un determinado problema.
- **Cromosoma:** Codificación de las soluciones potenciales del problema de forma que una computadora pueda procesarlas. Un enfoque común es codificar estos candidatos, o soluciones, como cadenas binarias donde cada valor de bit representa el valor de algún aspecto de la solución. Un método alternativo consiste en codificar las soluciones como números enteros o reales, donde cada posición representa algún aspecto particular de la solución.
- **Gen:** Elemento de información atómica que compone una parte de la posible solución que es codificada en un cromosoma.
- **Aptitud:** Métrica que permite evaluar cuantitativamente a cada candidato

con respecto al problema que se desea resolver.

- **Selección:** Proceso por el cual se eligen a los candidatos que representan una mayor aptitud al problema a resolver para su posterior cruce y reproducción.
- **Cruce:** Procedimiento por el que los candidatos seleccionados para su reproducción intercambian segmentos de información genética, para producir posteriormente una descendencia artificial cuyos individuos sean combinaciones de sus padres. Este proceso pretende simular el proceso análogo a la recombinación que se da en los cromosomas durante la reproducción sexual.
- **Reproducción:** Proceso por el cual se producen copias de los candidatos previamente seleccionados y que pasarán a formar parte de una nueva población de candidatos.
- **Mutación:** Cambios aleatorios que se producen durante el proceso de copia en algunos cromosomas para producir diversidad en la población.
- **Generación:** Acervo de soluciones candidatas producidas por la aplicación sucesiva de los mecanismos de selección, cruce, reproducción y mutación entre los candidatos de una población, y que produce una nueva población de candidatos.
- **Solución del Algoritmo Genético:** Mejor solución obtenida por el algoritmo, de entre todas las soluciones candidatas que componen la población, en el momento de haber iterado un número determinado de generaciones.
- **Era:** Sucesivas ejecuciones del Algoritmo Genético que proporcionan diferentes mejores soluciones.

1.2. Computación Evolutiva

La Computación Evolutiva presenta un enfoque alternativo a los algoritmos tradicionales para abordar problemas complejos de búsqueda y aprendizaje a través de modelos computacionales de procesos evolutivos, cuyo principal objetivo consiste en guiar una búsqueda estocástica haciendo evolucionar a un conjunto de estructuras, y seleccionando, de modo iterativo, las más adecuadas.

Son cuatro los paradigmas fundamentales de la computación evolutiva:

- **Los Algoritmos Genéticos** [Holl92]. Hacen evolucionar a una población de enteros binarios sometiéndolos a transformaciones unitarias y binarias genéticas, junto a un proceso de selección.
- **Los Programas Evolutivos** [Mich99]. Hacen evolucionar a una población de estructuras de datos sometiéndolas a una serie de transformaciones específicas y a un proceso de selección.
- **Las Estrategias Evolutivas**. Hacen evolucionar a una población de números reales que codifican las posibles soluciones de un problema numérico y los tamaños de salto. La selección es implícita.
- **La Programación Evolutiva**. Hacen evolucionar a una población de máquinas de estados finitos sometiéndolas a transformaciones unitarias.

1.3. Algoritmos Genéticos

Desarrollados por John Holland, junto a su equipo de investigación, en la Universidad de Michigan en la década de 1970, los algoritmos genéticos representan el paradigma principal de la computación evolutiva. Son métodos sistemáticos para la resolución de problemas de búsqueda y optimización que aplican a estos los mismos métodos de la evolución biológica: selección basada en la población, reproducción sexual y mutación, combinando las nociones de supervivencia del individuo más apto con un intercambio estructurado y aleatorio de características entre individuos de una población de posibles soluciones, conformando un algoritmo de búsqueda que puede aplicarse para resolver problemas de optimización en diversos campos.

Estos algoritmos de optimización tratan de obtener el vector de parámetros (x_1, x_2, \dots, x_n) que genera el máximo o el mínimo global de una cierta función $F(x_1, x_2, \dots, x_n)$.

En un algoritmo genético el problema se parametriza en un conjunto de variables (x_1, x_2, \dots, x_n) que a su vez se codifican en cromosomas, formando el conjunto de estas últimas poblaciones, y, a diferencia de otros métodos de búsqueda, en los algoritmos genéticos el método de búsqueda está implícito en él, por tanto se puede

afirmar que los algoritmos genéticos son independientes del problema que se desea resolver, lo cual por un lado los hacen robustos por su utilidad ante cualquier problema, pero por otro lado los hacen débiles al no estar especializados en ninguno.

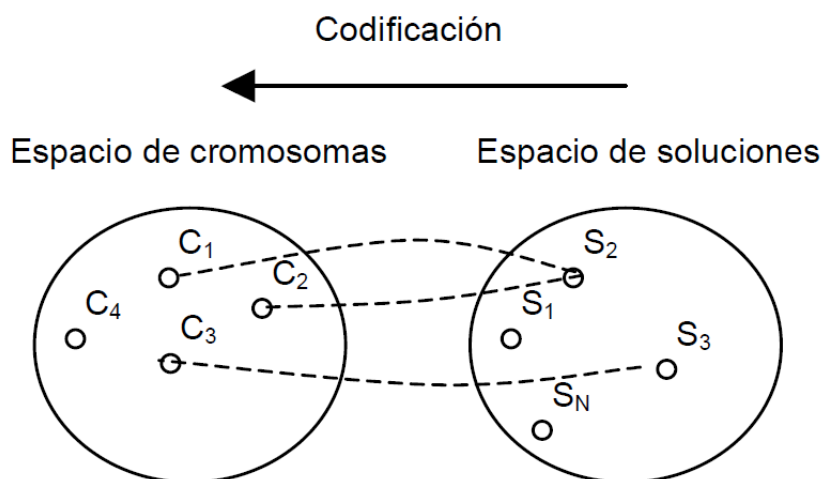


Figura 1.1: *Codificación de Soluciones en los Cromosomas*

Como se puede apreciar en la figura 1.1, las soluciones, codificadas en los cromosomas ($C1$, $C2$, ..., Cn), compiten para ver cuál de ellos constituye la mejor solución al problema ($S1$, $S2$, ..., Sn), de forma que únicamente los cromosomas mejor adaptados sobrevivirán, dando lugar, en las siguientes generaciones, a cromosomas más fuertes, y por tanto a mejores soluciones, las cuales legarán su material genético a las siguientes generaciones. Este escenario de competición y legado es análogo al de la naturaleza, en el que la presión de la selección natural provoca que sean los mejores individuos aquellos que sobrevivan frente a los más débiles, siendo estos los que se reproducirán y crearán nuevos individuos. La diversidad genética, de forma análoga al escenario natural, se introducirá mediante mutaciones y cruces.

1.4. Características de los AG como métodos de búsqueda

Además de la característica de selección natural, se pueden citar otras características inherentes a los algoritmos genéticos como métodos de búsqueda [Perez96]:

- **Ciega**, es decir, no se dispone de ningún conocimiento específico del problema, de manera que la búsqueda se basa exclusivamente en los valores de la función objetivo.
- **Codificada**, puesto que no se trabaja directamente sobre el dominio del problema, sino sobre representaciones de sus elementos.
- **Múltiple**, ya que se busca simultáneamente entre un conjunto de candidatos.
- **Estocástica**, referida tanto a las fases de selección como a las de transformación. Ello proporciona control sobre el factor de penetración de la búsqueda.

Todas las características enumeradas se introducen deliberadamente para proporcionar más robustez a la búsqueda.

1.5. Métodos de Optimización y Búsqueda tradicionales

Los métodos de optimización y búsqueda tradicionales se pueden clasificar en tres tipos principales: basados en el cálculo infinitesimal, de enumeración y aleatorios [Gold89].

1. **Métodos de cálculo basados en el cálculo infinitesimal:** Estos métodos se dividen a su vez en dos tipos: Directos e Indirectos
 - *Métodos Indirectos:* Buscan un extremo local mediante la resolución de un conjunto de ecuaciones no lineales que aparecen tras igualar el gradiente de la función objetivo a cero. Dada una función sin restricciones y suave, buscando un posible pico empezando por restringir la búsqueda a aquellos puntos que poseen pendiente cero en todas las direcciones.
 - *Métodos Directos:* Buscan puntos locales óptimos moviéndose en una dirección relativa al gradiente local.
2. **Técnicas de enumeración:** Consisten en ir probando uno a uno todos los puntos de un espacio de búsqueda restringido, ya sea un espacio finito o uno

infinito discretizado. Presentan el problema de ser poco eficientes ya que requieren un tiempo excesivo de cálculo cuando el espacio de búsqueda es grande.

3. **Algoritmos de búsqueda aleatoria:** Consisten en probar con distintos valores de manera aleatoria. Se puede afirmar que no actúan peor que las técnicas de enumeración.

Estas técnicas se caracterizan por no ser robustas, aunque esto no significa que no sean útiles.

1.6. Diferencias de los AG con los métodos de búsqueda y optimización tradicionales

Las diferencias de los AG con los algoritmos de búsqueda y optimización tradicionales se pueden resumir en las siguientes [Gold89]:

1. Los AG trabajan con una codificación de un conjunto de parámetros no con los parámetros directamente.
2. Los AG no se limitan a buscar en las cercanías de un punto, sino que utilizan una población de puntos.
3. Los AG utilizan únicamente la información que les proporciona la función de coste, sin necesidad de ninguna otra información. Por lo tanto no requieren calcular derivadas.
4. Los AG utilizan reglas de transición probabilísticas para guiar su búsqueda en lugar de las reglas deterministas que otros métodos tradicionales suelen utilizar.

Estas cuatro propiedades contribuyen a que los AG sean más robustos que los métodos tradicionalmente usados.

1.7. Funcionamiento de un AG básico

La figura 1.2 representa el esquema de funcionamiento de un algoritmo genético básico. En este algoritmo básico, el proceso comienza seleccionando un número

de cromosomas que conformarán la población inicial. A continuación se evalúa la función de adaptación para estos individuos. Esta función de adaptación proporciona una medida de la aptitud de cada cromosoma para sobrevivir en su entorno, y debe estar definida de tal forma que los cromosomas que representen las mejores soluciones obtengan valores más altos de adaptación; de este modo, los individuos más aptos se seleccionan en parejas para reproducirse. La reproducción genera nuevos cromosomas que combinarán características de ambos padres. Estos nuevos cromosomas reemplazarán a los individuos con menores valores de adaptación. A continuación algunos cromosomas son seleccionados al azar para ser mutados. La mutación consiste en aplicar un cambio aleatorio en su estructura. Posteriormente, los nuevos cromosomas se incorporarán a la población reemplazando a cromosomas ya existentes. Para realizar esta sustitución existen varios criterios que pueden utilizarse para elegir a los cromosomas que serán reemplazados.

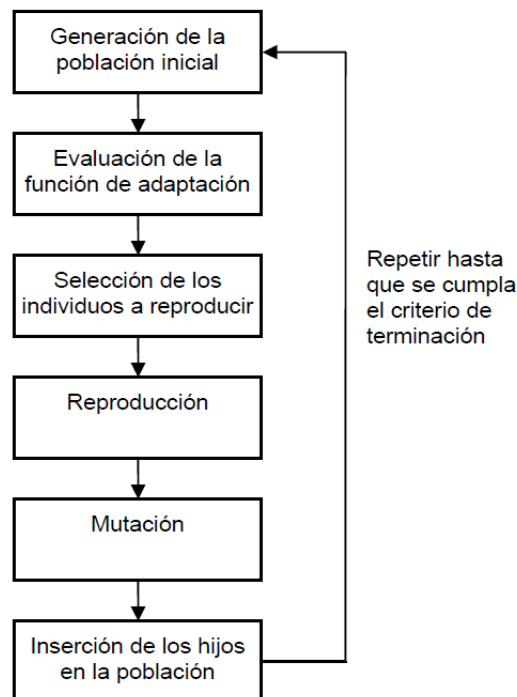


Figura 1.2: *Esquema de funcionamiento de un AG básico*

El ciclo de selección, reproducción y mutación se repetirá hasta que se cumpla el criterio de terminación del algoritmo genético, momento en el cual el cromosoma mejor adaptado se devolverá como la mejor solución.

Los criterios de terminación empleados en implementaciones de AGs son básicamente dos [Mich99]:

1. La condición de terminación más simple es aquella que va comprobando el número actual de generación g ; la búsqueda se termina si el número total de generaciones excede un valor constante predefinido, es decir si $g \leq MAXGENS$.
2. Como la condición de terminación anterior supone el conocimiento del usuario de las características de la función, lo cual influye en la longitud de la búsqueda, parece más conveniente que el algoritmo finalice la búsqueda cuando la probabilidad de una mejora significativa entre sucesivas generaciones sea muy pequeña, es decir que el valor de la función de coste entre generaciones no mejore por encima de un cierto umbral ε .

En el algoritmo genético programado, SIGENOF¹, se han implementado ambas condiciones de terminación, siendo la primera el criterio por defecto, mientras que el segundo criterio se puede utilizar de forma discrecional por el usuario de la herramienta.

La estructura genérica del bucle básico de ejecución de un algoritmo genético se puede sintetizar en el diagrama mostrado en la figura 1.3 [Perez96]:

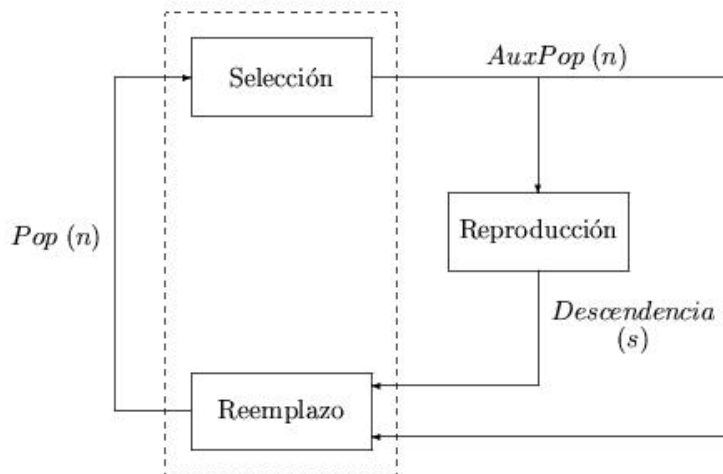


Figura 1.3: Estructura genérica del bucle básico de ejecución de un AG

Como puede observarse en la figura anterior, una población Pop , que consta

¹Sistema Genético para la Optimización de Funciones

de n miembros, se somete a un proceso de selección para constituir una población intermedia *AuxPop* de n miembros. De dicha población intermedia se extraerá un grupo reducido de individuos llamados progenitores, que son los que efectivamente se van a reproducir. Sirviéndose de los operadores genéticos, los progenitores son sometidos a ciertas transformaciones de alteración y recombinación en la fase de reproducción, en virtud de las cuales se generan s nuevos individuos que constituyen la descendencia. Para formar la nueva población $Pop[t + 1]$, se deben seleccionar n supervivientes de entre los $n + s$ de la población auxiliar y la descendencia; esto se realizará en la fase de reemplazo.

Todo el procedimiento de búsqueda estará guiado exclusivamente por una función de aptitud $u(x)$, la cual se obtiene directamente a partir de la función de evaluación $f(x)$ del problema.

1.8. Herramientas software actuales que implementan AG

Se pueden encontrar una gran variedad de herramientas que implementan AG y que son usadas para diferentes propósitos en diversidad de campos, entre los que se encuentran la acústica, ingeniería aeroespacial, astronomía y astrofísica, química, ingeniería eléctrica, los mercados financieros, juegos, geofísica, ingeniería de materiales, matemática y algoritmia, diseño de rutas y horarios, biología molecular, reconocimiento de patrones y robótica, entre otros.

Muchas de estas herramientas presentan unas limitaciones que se pueden resumir en las siguientes:

- La mayoría no implementan ningún GUI que permita la interacción con la herramienta por parte del usuario de manera sencilla e intuitiva. Sus salidas se suelen realizar mediante consola o a través de ficheros de texto.
- No son fácilmente parametrizables, es decir, suele ser difícil cambiar el objetivo del problema que resuelven. Por ejemplo, muchas herramientas que optimizan funciones buscando máximo o mínimos necesitan ser compiladas cada vez que la función objetivo, o algunos de sus parámetros, varía.

- En muchas ocasiones el cambio de los parámetros del AG, tales como el tamaño de la población, la probabilidad de cruce, etc., resulta complejo al tener que realizar cambios en el programa o, en el mejor de los casos, en algunos ficheros de configuración de la herramienta. Frecuentemente el programa necesita volver a compilarse para hacer efectivos dichos cambios.
- Los lenguajes utilizados en su desarrollo son bastante heterogéneos, encontrándose herramientas programadas en C/C++, Perl, Java, LISP, Fortran, etc. En muchos casos estas herramientas son portables, es decir se pueden ejecutar en distintas computadoras, como por ejemplo las que se desarrollan en Java (Java application o Java applet), sin embargo en otros casos, sobretodo cuando incorporan un GUI, son dependientes del sistema operativo para el cual se han desarrollado.

1.9. Herramienta SIGENOF

SIGENOF, se ha desarrollado con el objetivo de suplir algunas de las limitaciones que otras herramientas presentan, sobretodo en lo referente a la capacidad de interactividad del usuario con ésta.

Las características más destacables que aporta SIGENOF con respecto a otras herramientas se pueden resumir en las siguientes:

1. Proporciona un entorno gráfico sencillo e intuitivo que permite:
 - Introducir fácilmente diferentes parámetros al AG como tamaño de la población, número de generaciones, número de eras, probabilidad de cruce y de mutación, etc.
 - Definir dinámicamente, sin necesidad de volver a compilar el programa, la expresión de la función matemática que se desea optimizar, así como los valores máximos y mínimos que sus variables de incertidumbre pueden tomar.
 - Interactuar en tiempo de ejecución con la herramienta pudiendo iniciar o cancelar la ejecución a petición del usuario, o definir el tipo de visualización gráfica que mejor convenga.

- Representación gráfica de las mejores soluciones obtenidas por el AG en cada generación / era.
 - Posibilidad de pausar la ejecución tras finalizar una era para poder observar con detenimiento el gráfico de las mejores soluciones obtenidas antes de pasar a la siguiente era.
2. Portabilidad frente a distintas plataformas hardware y software. La herramienta se ha desarrollado en un lenguaje portable, en lenguaje C, que permite la compilación en cualquier sistema operativo que disponga de un compilador de ANSI C. Para la implementación del GUI se han utilizado las librerías gráficas GTK+ distribuidas bajo licencia GNU GPL. Estas librerías requieren la instalación del framework GTK+, distribuido también bajo licencia GNU GPL y que puede descargarse gratuitamente desde Internet.
 3. Fácil instalación y distribución ya que únicamente se requiere la distribución del framework GTK+ y del fichero ejecutable de la herramienta.
 4. Generación de ficheros log con los resultados de cada ejecución.
 5. La herramienta ha sido desarrollada bajo licencia GNU GPL, lo que permite su libre distribución.

1.10. Problemas de optimización que resuelve SIGENOF

SIGENOF, se utiliza de forma eficiente para resolver problemas de optimización como el que se describe a continuación.

Se define el *vector de parámetros* θ :

$$\theta = [x_1, \dots, x_n]$$

Donde los elementos x_i son números reales acotados dentro de un cierto espacio S .

$$S = \{x_1 \in [x_1^{\min}, x_1^{\max}], x_2 \in [x_2^{\min}, x_2^{\max}], \dots, x_n \in [x_n^{\min}, x_n^{\max}]\}$$

Es decir, el valor que puede tomar cada elemento x_i se encuentra acotado dentro de un cierto valor mínimo x_i^{min} y de un cierto valor máximo x_i^{max} .

Sea la función de coste $J : R^n \rightarrow R$

$$J = f(\theta)$$

Se verifica la condición de que $J > 0$ para cualquier θ considerado.

El problema que resuelve el algoritmo genético, por tanto, es el de obtener el vector de parámetros óptimo $\theta_{opt} \in S$ que genera el valor máximo J_{opt} de la función de coste J .

$$J_{opt} = J(\theta_{opt}) = \max_{\theta \in S}(J(\theta))$$

1.11. Estructura de la memoria

Esta memoria se ha dividido en cinco capítulos. En el capítulo 1 se exponen los conceptos básicos que envuelven a los AG como uno de los paradigmas de la Computación Evolutiva, presentándose estos como un enfoque alternativo a los algoritmos tradicionales usados en problemas de búsqueda y optimización de soluciones. Asimismo se enumeran las características de los AG como métodos de búsqueda y se comparan con los métodos tradicionales, se expone el funcionamiento de un AG básico y se hace un breve estudio de las herramientas software actuales que implementan AG. Por último se proporciona una visión general de la herramienta implementada para este proyecto, SIGENOF, así como los problemas de optimización que ésta resuelve.

En el capítulo 2 se describen los detalles de implementación de SIGENOF, tales como el lenguaje utilizado para su escritura, las estructuras de datos relevantes para el AG programado (Cromosomas, Funciones a optimizar, Parámetros de ejecución, etc.), la sintaxis para la definición de las funciones J a optimizar, y el detalle procedimental de los principales pasos que realiza la herramienta en la implementación de su AG.

En el capítulo 3 se justifica, en primer lugar, la librería gráfica con la que se ha implementado el GUI de SIGENOF, las GTK+. A continuación se realiza un análisis de todas y cada una de las funciones que la herramienta proporciona a

través de su GUI, describiendo, paso a paso y a modo de manual de usuario, la ejecución de la herramienta con un ejemplo.

El capítulo 4 muestra algunos de los ejemplos utilizados para verificar el funcionamiento de SIGENOF como herramienta para la optimización de funciones matemáticas. En cada uno de estos ejemplos se describe el problema a resolver y se muestra la solución gráfica en la que se puede observar el valor óptimo de la función; posteriormente se compara la solución encontrada por SIGENOF frente a la solución real de forma que se pueda ponderar la calidad de la solución proporcionada por la herramienta. Asimismo se explica cómo se comporta la herramienta en la optimización de un caso particular con diferentes variaciones de los parámetros del AG, tales como el tamaño de la población, la probabilidad de cruce y/o mutación, entre otros.

En el capítulo 5 se describen algunas de las conclusiones a las que he llegado tras la implementación de SIGENOF.

Al final de la memoria se incluyen, como apéndices, información relativa a diversos aspectos clave en la implementación del AG de SIGENOF, así como de los ficheros de salida (*ficheros LOG*) generados por la herramienta.

Capítulo 2

Algoritmo Genético implementado en SIGENOF

2.1. Lenguaje de implementación utilizado

El lenguaje de programación de alto nivel utilizado para implementar el AG que usa SIGENOF ha sido el lenguaje C, puesto que presenta las siguientes ventajas:

- El lenguaje C es adecuado para la programación a bajo nivel cubriendo así el vacío entre el lenguaje máquina y los lenguajes de alto nivel más convencionales.
- Permite la redacción de programas fuentes muy concisos debido, en parte, al gran número de operadores que incluye en lenguaje.
- Tiene un repertorio de instrucciones básicas relativamente pequeño, aunque incluye numerosas funciones de biblioteca que mejoran las instrucciones básicas. Además los usuarios pueden escribir bibliotecas adicionales para su propio uso.
- Los compiladores de C son, frecuentemente, compactos y generan programas objeto pequeños y eficientes.
- Los programas escritos en C son muy portables. La razón de esto es que C deja en manos de las funciones de biblioteca la mayoría de las características

dependientes de la computadora. De esta forma, la mayoría de los programas escritos en C se pueden compilar y ejecutar en muchas computadoras diferentes, con muy pocas o ninguna modificación.

2.2. Estructuras de datos

Los cromosomas poblacionales se han definido como estructuras que contienen los siguientes datos:

- Un vector de parámetros o variables x de la función J que se desea optimizar, particular para el cromosoma. Este vector de parámetros representa una solución particular al problema. Estos parámetros, también denominados *genes*, están acotados al espacio S de posibles valores que pueden tomar, y que han sido definidos para la función J .
- Un valor que representa el *fitness* o grado de adaptación del cromosoma a la función J que se desea optimizar.
- Un valor p que indica la probabilidad de selección del cromosoma para su reproducción.
- Un valor q que representa la probabilidad acumulada del cromosoma con respecto a los demás cromosomas de la población, decisivo para su selección para el cruce con otro cromosoma.
- Un valor booleano que indica que el cromosoma ha sido seleccionado para su cruce y reproducción.

La definición del cromosoma se ha realizado como se muestra en el cuadro 2.1.

| Estructura CROMOSOMA |
|-----------------------------------------------------------------------------------------------------------------------------------------|
| <pre>typedef struct CROMOSOMA { long *x; float fitness; float p; float q; int seleccionado; } Tipo_cromosoma;</pre> |

Cuadro 2.1: Definición del cromosoma

De esta forma, la población se ha establecido como un conjunto de *pop_size* cromosomas definidos como un vector de *pop_size* elementos de tipo *Tipo_cromosoma*.

La función J que se desea optimizar se ha definido como una estructura que contiene los siguientes datos:

- Una cadena de caracteres que contiene la descripción aritmética de la función J a optimizar, ajustada a la sintaxis definida en el cuadro 2.3. Esta representación proporciona una gran flexibilidad ya que permite optimizar diferentes funciones J sin necesidad de volver a compilar el programa cada vez que ésta varíe. La gramática definida para las expresiones aritméticas de las funciones J se describe en el Apéndice A.
- Un valor entero que indica el número de parámetros o variables x_i que contiene la función, con $0 \leq i \leq \text{numero_parametros} - 1$.
- Un vector de valores de coma flotante que establece el valor mínimo que cada variable x_i puede tomar como extremo inferior x^{min} del espacio S de la función J .
- Un vector de valores de coma flotante que establece el valor máximo que cada variable x_i puede tomar como extremo superior x^{max} del espacio S de la función J .

La definición de la función J se ha realizado como se muestra en el cuadro 2.2.

| Estructura FUNCIÓN |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>typedef struct DEF_FUNCION { char *funcion; int numero_parametros; float *valor_min_parametro; float *valor_max_parametro; } Tipo_funcion;</pre> |

Cuadro 2.2: Definición de la función J a optimizar

Como se puede observar en el cuadro 2.1, se ha definido el tipo del vector de parámetros x como entero largo (`long *x`), cuyo rango de posibles valores que puede representar se encuentra en el intervalo $[-2.147.483.648 \dots 2.147.483.647]$, sin embargo los posibles valores que dichos parámetros pueden tomar se han definido en la tabla 2.2 como valores de coma flotante (`float *valor_min_parametro` y `float *valor_max_parametro`). Este cambio en el tipo de datos para el vector de parámetros se ha realizado con el fin de optimizar el tiempo de ejecución del AG así como su tamaño en memoria, ya que los cálculos con enteros son sensiblemente más rápidos que con números en coma flotante. Estos parámetros, tratados como valores en coma flotante, son redondeados, por defecto, a tres cifras decimales¹ y almacenados posteriormente en el cromosoma como valores enteros largos. Cuando se desea operar con estas variables, como sucede cuando se realiza la evaluación del cromosoma según la función de coste J para determinar su adaptación o *fitness* a dicha función, se realiza el proceso inverso convirtiendo los valores enteros a su correspondiente valor en coma flotante.

Otras variables que se han considerado en la implementación del AG son:

- **Tamaño de la población** (`POP_SIZE`), es decir, el número de cromosomas de que consta la población.
- **Número máximo de generaciones** (`MAXGENS`) en las que se ejecutará el bucle básico del AG para dar por finalizada una era.

¹Aunque el número de cifras decimales es un valor que, como se verá posteriormente, puede ser definido por el usuario de la herramienta.

- **Número de eras (ERAS)**; establece el número de ejecuciones completas e independientes entre sí del AG.
- **Probabilidad de cruce** de los cromosomas (**pc**). Es un número real comprendido en el intervalo $[0, 1]$ que define la probabilidad que cada cromosoma tendrá de ser seleccionado para su cruce y reproducción con otro cromosoma.
- **Probabilidad de mutación** de los cromosomas (**pm**). Número real comprendido en el intervalo $[0, 1]$ que define la probabilidad que cada cromosoma tendrá de sufrir una mutación en sus genes.

2.3. Sintaxis para la definición de funciones J

La sintaxis que permite definir diferentes funciones J se muestra en el cuadro 2.3.

| SINTAXIS PARA LA DEFINICIÓN DE FUNCIONES |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>1. VARIABLES Y CONSTANTES:</p> <p><i>Variables de función $x[0], x[1], \dots, x[j]$</i></p> <p><i>Constantes π, e</i></p> <p><i>Constantes Numéricas enteras, Punto Flotante o Expresiones Exponenciales</i></p> <p>2. OPERADORES DISPONIBLES:</p> <p><i>suma(+) resta/negación(-) producto(*) div(/) potencia(/)</i></p> <p>3. FUNCIONES PREDEFINIDAS:</p> <p><i>sin, cos, tan, asin, acos, atan, hsin, hcos, htan, sqrt, exp, ln, log, abs</i></p> <p>4. REGLAS DE PRECEDENCIA (de mayor a menor):</p> <p><i>1° Paréntesis</i></p> <p><i>2° Negación unaria</i></p> <p><i>3° Potencia, Producto y División</i></p> <p><i>4° Suma y Resta</i></p> |

Cuadro 2.3: Sintaxis para la definición de funciones J

A continuación se detallan, de manera individual, los diferentes elementos que componen dicha sintaxis.

2.3.1. Variables y Constantes:

- **Variables de incertidumbre de la función ($x[j]$):** Dada una determinada función f con n variables de incertidumbre, es decir $f(x_1, x_2, \dots, x_n)$, las variables x_i de la función se definirán como variables $x[j]$, donde $j = i - 1$ y $0 \leq j \leq n - 1$. Con esta sintaxis las variables x_1, x_2, \dots, x_n , se definirán como $x[0], x[1], \dots, x[n - 1]$. Así en la expresión de una función $f(x_1, x_2)$ aparecerían únicamente las variables $x[0]$ y $x[1]$, junto con los operadores necesarios, funciones predefinidas y las constantes numéricas necesarias para su definición completa.
- **Constantes predefinidas:**
 1. **Constante π (pi):** Establece el valor del número π el cual se define de manera aproximada como 3,14159265358979. Una expresión en la que se haga uso de la constante π como 2π se redefine como $2 * pi$.
 2. **Constante e de Euler (e):** Establece el valor del número e el cual se define de manera aproximada como 2,718282. Una expresión en la que se haga uso de la constante e como ocurre en $2e$ se redefine como $2 * e$.
- **Constantes numéricas:**
 1. **Constantes Enteras:** Como por ejemplo las constantes 2, -5 ó 125. Se redefinen en la expresión de la misma forma.
 2. **Constantes de Punto Flotante:** Como por ejemplo las constantes 2.25 ó -48,002. En este caso se redefinen en la expresión de la misma forma pudiéndose sustituir el punto decimal por una coma decimal o viceversa, como mejor convenga, quedando redefinidos como 2,25 ó 2.25 y -48,002 ó -48.002.
 3. **Expresiones Exponenciales:** Expresiones como $2 \cdot 10^3$ ó $-2.05 \cdot 10^{-5}$ se pueden redefinir en notación exponencial como $2E + 3$, $2E3$ y $-2.05E - 5$.

2.3.2. Operadores disponibles:

- **Operador suma (+):** Suma dos términos a y b . Una expresión como $x + y$ se redefine como $x[0] + x[1]$.
- **Operador resta (-):** Obtiene la diferencia de dos términos a y b . Una expresión como $x - (y - z)$ se redefine como $x[0] - (x[1] - x[2])$.

- **Operador producto (*)**: Multiplica dos términos a y b . Una expresión como $2 * x$ se redefine como $2 * x[0]$.
- **Operador división (/)**: Divide dos términos a y b . Una expresión como y/x se redefine como $x[1]/x[0]$.
- **Operador potencia (^)**: Realiza el cálculo a elevado al exponente b . Una expresión como x^3 se redefine como $x[0]^3$.
- **Operador negación unaria (-)**: Realiza el complemento de signo del término al que antecede. Una expresión como $-5y$ se redefine como $-5 * x[1]$.

2.3.3. Funciones predeterminadas:

- **Funciones trigonométricas:**
 1. **seno, coseno y tangente** ($\sin(a), \cos(a), \tan(a)$): Obtiene el valor de expresiones trigonométricas como $\text{seno}(2.5)$, $\text{coseno}(x)$ y $\text{tangente}(1 - z)$ redefiniéndose éstas como $\sin(2, 5)$, $\cos(x[0])$ y $\tan(1 - x[2])$.
 2. **arcoseno, arcocoseno y arcotangente** ($\text{asin}(a), \text{acos}(a), \text{atan}(a)$): Obtiene el valor de expresiones trigonométricas como $\text{arcoseno}(2.5)$, $\text{arcocoseno}(x)$ y $\text{arcotangente}(1 - z)$, redefiniéndose éstas como $\text{asin}(2, 5)$, $\text{acos}(x[0])$ y $\text{atan}(1 - x[2])$.
 3. **seno, coseno y tangente hiperbólica** ($\text{hsin}(a), \text{hcos}(a), \text{htan}(a)$): Obtiene el valor de expresiones trigonométricas hiperbólicas como $\text{senoHiperbolico}(2.5)$, $\text{cosenoHiperbolico}(x)$ y $\text{tangenteHiperbolica}(1 - z)$, redefiniéndose éstas como $\text{hsin}(2, 5)$, $\text{hcos}(x[0])$ y $\text{htan}(1 - x[2])$.
- **Función raíz cuadrada** ($\text{sqrt}(a)$): Realiza cálculos del tipo $\sqrt{2}$, redefiniéndose mediante esta sintaxis como $\text{sqrt}(2)$.
- **Función exponencial** ($\text{exp}(a)$): Realiza cálculos del tipo e^x , donde e es la constante de Euler y x es el argumento a . Estas expresiones se redefinen como $\text{exp}(x[0])$. Alternativamente esta expresión también se podría redefinir como $e^{x[0]}$.
- **Funciones logarítmicas:**
 1. **Logaritmo en base 10** ($\log(a)$): Obtiene el logaritmo en base 10 del

argumento a . Una expresión como $\log(y - x)$ se redefine como $\log(x[1] - x[0])$.

2. **Logaritmo natural** ($\ln(a)$): Obtiene el logaritmo natural del argumento

a . Una expresión como $\ln e$ se redefine como $\ln(e)$.

- **Valor absoluto** ($\text{abs}(a)$): Obtiene el valor absoluto del argumento a . Una expresión como $|x|$ se redefine como $\text{abs}(x[0])$.

2.3.4. Reglas de precedencia:

Una expresión en la que existan diversos operadores juntos puede resultar ambigua. Para resolver estas situaciones se han definido unas sencillas reglas de precedencia, las cuales aplican los diferentes operadores en el orden definido a continuación, considerando en este caso al factor *Paréntesis* como un elemento de mayor precedencia que los operadores *Suma* y *Resta*. Los operadores que aparecen agrupados en esta enumeración, como *Potencia*, *Producto* y *División*, tienen la misma precedencia cuando aparecen en una expresión; en este caso se resolverá la expresión mediante un análisis sintáctico de izquierda a derecha.

- 1º *Paréntesis*.
- 2º *Negación unaria*.
- 3º *Potencia, Producto y División*.
- 4º *Suma y Resta*.

Así, una función J definida como,

$$f(x_1, x_2) = 21.5 + x_1 \cdot \sin(4\pi x_1) + x_2 \cdot \sin(4\pi x_2)$$

$$S = \{x_1 \in [-3.0, 12.1], x_2 \in [4.1, 5.8]\}$$

según la sintaxis establecida para el AG, en el cuadro 2.3 y la gramática descrita en el Apéndice A, se define como se muestra en el cuadro 2.4.

| |
|------------------------------------------------------------------------------|
| <i>Función:</i> $21.5 + x[0] * \sin(4\pi * x[0]) + x[1] * \sin(4\pi * x[1])$ |
| <i>Número de parámetros:</i> 2 |
| <i>Valor mínimo parámetro</i> $x[0]$: -3.0, $x[1]$: 4.1 |
| <i>Valor máximo parámetros</i> $x[0]$: 12.1, $x[1]$: 5.8 |

Cuadro 2.4: Definición de una función *J*

2.4. Algoritmo genético implementado

El AG implementado se ha realizado siguiendo el esquema básico de algoritmo genético propuesto por Michalewicz [Mich99], y mostrado en la figura 2.1.

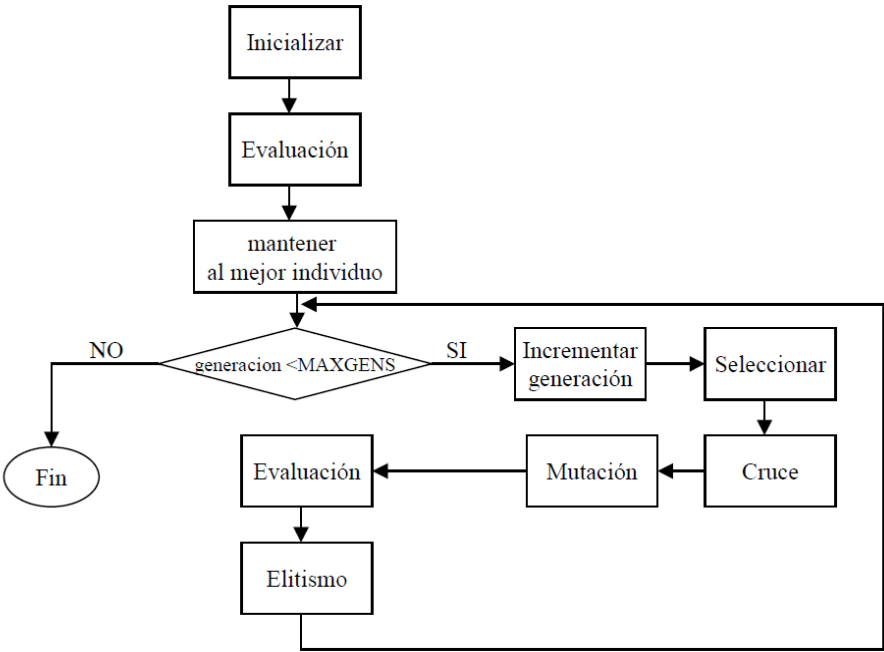


Figura 2.1: Esquema de un algoritmo genético básico

Se detalla a continuación la implementación de los diferentes pasos del AG:

2.4.1. Paso 1. Inicialización

Este primer paso es uno de los más críticos, en lo que a resultados del AG se refiere, ya que es el responsable de generar *aleatoriamente* la población inicial

de individuos con la que el AG empezará a operar. Es especialmente importante que el generador de números aleatorios, o más concretamente *pseudoaleatorios*² sea lo suficientemente bueno y capaz para generar individuos de forma verdaderamente aleatoria. Para ello se ha utilizado la implementación que se da en el texto [Recip88] como `ran2`, una función que produce números aleatorios de manera *perfecta*, considerando la definición de función perfecta la que los autores describen en el texto: "... *pagaremos \$1000 al primer lector que nos convenza de lo contrario*". En las figuras 2.2 y 2.3 se muestra la distribución de varios números aleatorios generados con dicha implementación; en el primer gráfico se generaron 1000 números reales aleatorios comprendidos en el intervalo $[0, 100]$, mientras que en la segunda se generaron 5000 números reales aleatorios en el intervalo $[-1000, 1000]$. Se puede observar en ellas cómo los distintos números generados se distribuyen de forma aleatoria a lo largo del espacio de posibles valores que pueden tomar. El código fuente de la función `ran2` se muestra en el Apéndice B.

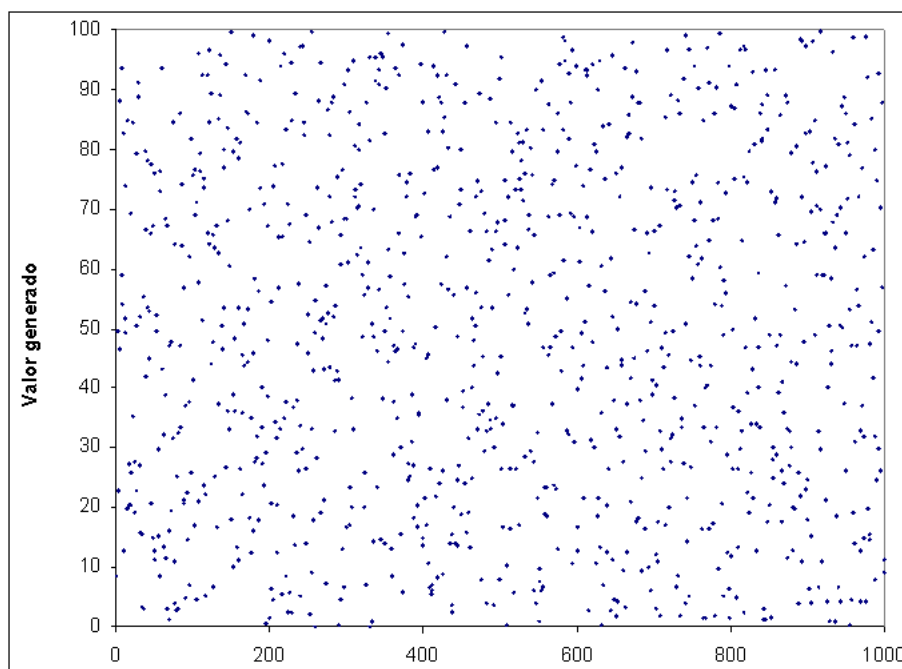


Figura 2.2: *Generación de 1000 números reales aleatorios*

²El concepto de aleatorio se suele reservar a la generación mediante fenómenos físicos, tales como el tiempo transcurrido entre sucesivos *clicks* de un contador *Geiger* situado cerca de una muestra radiactiva.

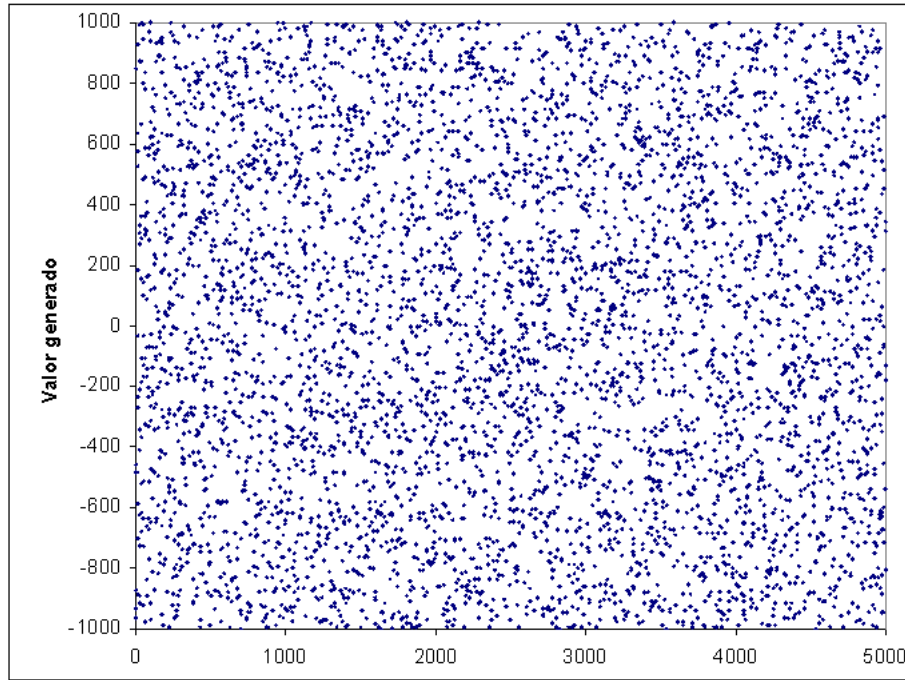


Figura 2.3: *Generación de 5000 números reales aleatorios*

2.4.2. Paso 2. Evaluación

En este paso se evalúa cada individuo o cromosoma i de la población inicial de tamaño POP_SIZE sobre la función de coste J del problema a resolver, obteniendo su *fitness* o adaptación al problema.

2.4.3. Paso 3. Mantener al mejor individuo

Se identifica y se selecciona el mejor individuo de la generación actual para conservar su información genética para la siguiente generación. Esta selección se realiza en función del grado de adaptación del individuo a la función de coste J que se está optimizando. En este caso su adaptación vendrá determinada por el valor que hace máximo la función de coste J en la generación actual. El AG guarda al mejor individuo de la generación en una variable de tipo *Tipo_cromosoma* llamada *mejor_cromosoma*.

2.4.4. Paso 4. Bucle de ejecución del AG hasta MAXGENS

Este cuarto paso se subdivide a su vez en los siguientes seis subpasos:

Paso 4.1. Incremento del contador de generaciones

En este paso únicamente se actualiza un contador de generaciones para controlar el bucle de ejecución del AG.

Paso 4.2. Aplicación del operador de Selección

El propósito de la selección de padres es dar más oportunidades de reproducción a aquellos individuos de una población que son los que mejor se ajustan tras haber sido evaluados por la función de evaluación. Con este propósito, el operador de Selección aplica, en este AG, el *Método de la Ruleta* para establecer la probabilidad de selección de cada cromosoma de la población actual.

El algoritmo que describe el Método de la Ruleta se detalla como:

1. Se suman los *fitness* de todos los individuos de la población J_i . A esta suma se le denominará F , definiéndose como:

$$F = \sum_{i=1}^{POP_SIZE} J_i$$

2. Se calcula la probabilidad de selección p_i para cada cromosoma de la población como:

$$p_i = \frac{J_i}{F}$$

3. Se calcula la probabilidad acumulada q_i para cada cromosoma de la población como:

$$q_i = \sum_{k=1}^{POP_SIZE} p_k$$

4. Se gira la ruleta POP_SIZE veces. Cada vez que se gira la ruleta se realiza lo siguiente:

- Se genera un número real aleatorio r dentro del rango $[0, 1]$.
- Si $r < q_1$ se selecciona el cromosoma C_1 , en caso contrario se selecciona el cromosoma i -ésimo C_i ($2 \leq i \leq POP_SIZE$), tal que $q_{i-1} < r \leq q_i$ para una nueva población.

Tras haber girado la ruleta POP_SIZE veces, se habrán seleccionado POP_SIZE cromosomas para formar parte de la nueva población.

Se muestra gráficamente el proceso de selección de cromosomas con el método de la ruleta en la figura 2.4.

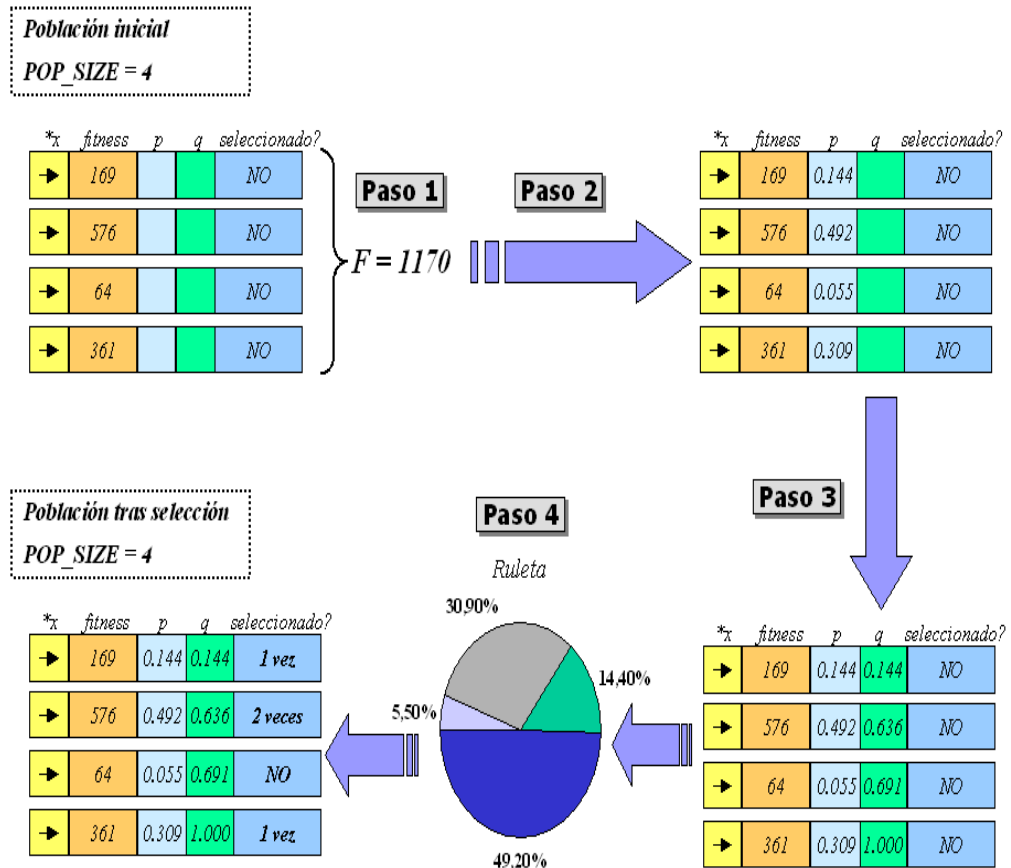


Figura 2.4: Método de la Ruleta

Paso 4.3. Aplicación del operador de Cruce en un solo punto

La probabilidad de cruce vendrá determinada por el valor de probabilidad $pc \in [0, 1]$, siendo éste un parámetro configurable por el usuario que ejecuta el AG.

Teniendo en cuenta este parámetro pc , el operador de cruce en un solo punto se ha implementado de la siguiente forma:

1. En primer lugar se seleccionan los individuos de la nueva población de `POP_SIZE` individuos para reproducirse. Para ello para cada individuo se genera un número real aleatorio r comprendido en el rango $[0, 1]$. Si $r < pc$, entonces el cromosoma i es seleccionado para reproducirse.
Como el operador de cruce en un solo punto opera con parejas de individuos, si tras finalizar la selección de cromosomas para su reproducción se observa que este número es impar, se procede a la selección de un cromosoma adicional de forma aleatoria, de forma que el conjunto final de cromosomas seleccionados sea un número par.
2. Se realiza la operación de cruce en un solo punto entre parejas de individuos seleccionados para reproducción. El punto de cruce para el intercambio de información entre genes se selecciona, en este caso, de forma aleatoria. Se puede observar esta operación gráficamente en la figura 2.5.

Tras la selección de parejas de cromosomas y cruce en un solo punto, se obtendrán dos nuevos cromosomas hijos que reemplazarán a los cromosomas padre en la población de individuos.

En el AG implementado, al contener el cromosoma un número de variables de genes determinado por la variable $nGenes$, y ésta a su vez condicionada por el número n de variables de la función J que se desea optimizar (x_1, \dots, x_n) , el intercambio de información genética se realiza seleccionando un número entero aleatorio comprendido en el intervalo $[0, n - 1]$ que indicará la posición del parámetro x_i (gen i) a partir del cual se intercambiará la información genética, codificada como valores enteros que representan valores reales, entre la pareja de cromosomas. Este funcionamiento puede apreciarse en la figura 2.6.

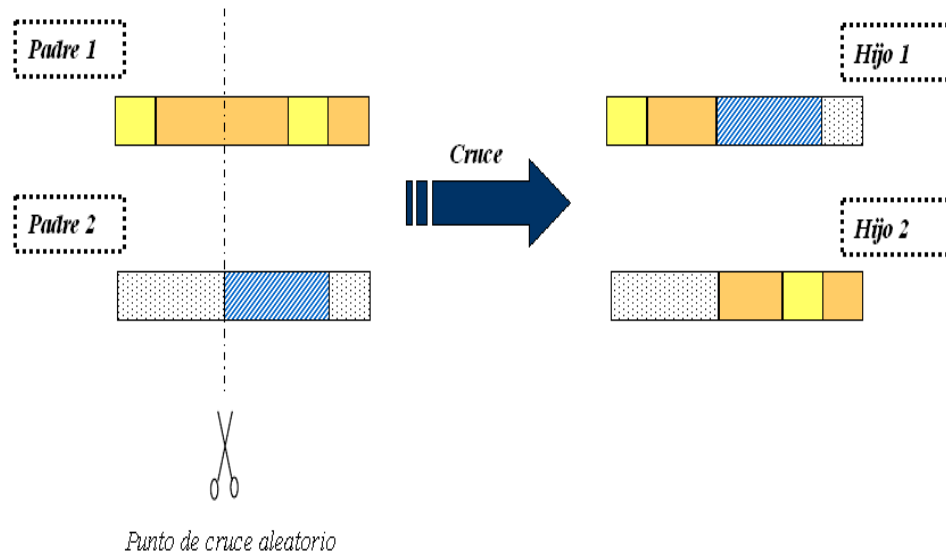


Figura 2.5: Cruce en un solo punto

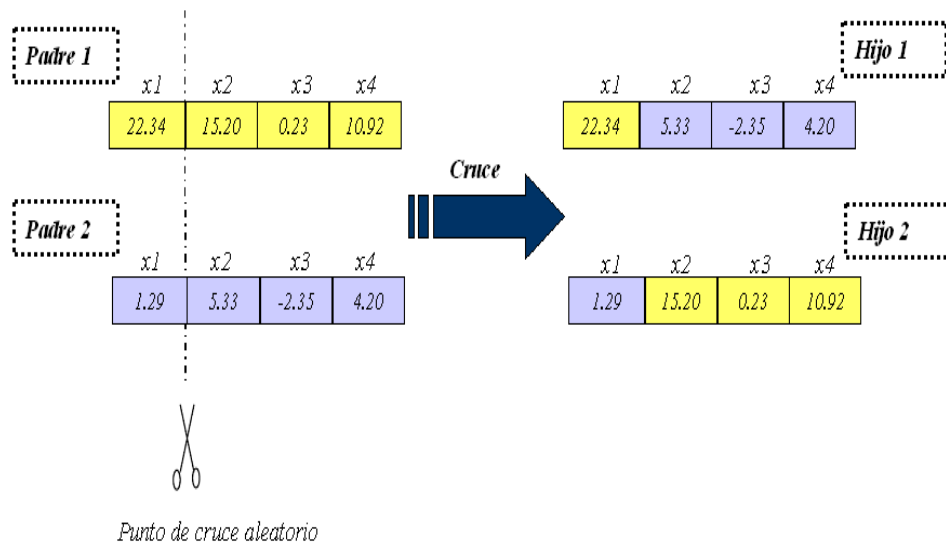


Figura 2.6: Cruce en un solo punto en SIGENOF

Paso 4.4. Aplicación del operador de Mutación Uniforme

El operador de mutación se aplica a todos los individuos de la población, aunque su efecto estará condicionado por el valor de probabilidad $pm \in [0, 1]$ que el usuario

que ejecuta el AG haya establecido para una ejecución concreta.

Este operador se ha implementado de forma que para cada gen x_k de cada uno de los POP_SIZE cromosomas de la población se genera un número aleatorio r comprendido entre $[0, 1]$, de forma que si $r < pm$ el cromosoma x_k será mutado.

La mutación consiste, en este caso, en el reemplazo del valor x_k por un nuevo valor real aleatorio comprendido dentro del rango de posibles valores definidos para dicho gen entre $[x_k^{min}, x_k^{max}]$.

Paso 4.5. Evaluación de la nueva población actual

Este paso se realiza sobre la nueva población de individuos tal y como se describió en el Paso 2.

Paso 4.6. Mecanismos de Elitismo

Puede suceder que el mejor miembro de una población falle en la reproducción de descendientes para la siguiente generación. Por este motivo se ha optado por la inclusión de una estrategia elitista en el AG de forma que se resuelva esta posible debilidad copiando los mejores individuos de una generación en la siguiente generación, de forma incondicional.

Se ha implementado la estrategia de elitismo de la siguiente forma:

1. Se busca el mejor cromosoma de la población actual de POP_SIZE individuos y se almacena en la variable *mejor_actual*.
 2. Se busca el peor cromosoma de la población actual de POP_SIZE individuos y se almacena en la variable *peor_actual*.
 3. Se compara el valor del *fitness* del mejor cromosoma de la población actual (*mejor_actual*) con el mejor cromosoma de las generaciones anteriores (*mejor_cromosoma*).
- Si el mejor cromosoma de la población actual (*mejor_actual*) es mejor que el mejor cromosoma de las generaciones anteriores, es decir, tiene un *fitness* mayor, se guarda como mejor cromosoma el de la población actual. Es decir, `mejor_cromosoma = mejor_actual`.

- En caso contrario, se sustituye el peor cromosoma de la población actual (*peor_actual*) por el mejor cromosoma de las generaciones anteriores. Es decir, `peor_actual = mejor_cromosoma`.

2.4.5. Paso 5. Terminación del AG

El AG implementado finaliza, de forma normal, su bucle de ejecución cuando el contador *generación* alcanza el valor definido en la variable `MAXGENS` por el usuario que ejecuta el AG. Otra forma alternativa de finalizar el AG es a través de la interacción del usuario con el Interfaz Gráfico de forma explícita. Este algoritmo se ejecuta tantas veces como ERAS han sido definidas por el usuario de la herramienta.

Tras la finalización del AG se habrán generado dos fichero *log* de salida con los nombres `./salida_ag_DD.MM.AAAA_HH.MM.SS.txt` y `./soluciones_ag_DD.MM.AAAA_HH.MM.SS.txt` respectivamente, donde DD, MM y AAAA corresponden al día, mes y año de la ejecución respectivamente, mientras que HH, MM y SS a la hora, minutos y segundos.

El contenido del primer fichero de salida (`salida_ag`) se organiza en cuatro secciones precedidas por un encabezamiento:

1. Encabezamiento que muestra el nombre de la herramienta, SIGENOF, junto con la fecha y la hora de ejecución.
2. Sección que describe la función J que se ha optimizado. En ella se muestra la descripción aritmética de la función, así como los valores $[x_k^{min}, x_k^{max}]$ definidos para cada variable x_k de la función.
3. Sección que muestra los parámetros introducidos para la ejecución del AG: número de eras ejecutadas (`ERAS`), número de generaciones para cada una de las eras (`MAXGENS`), número de individuos de la población (`POP_SIZE`), probabilidad de cruce (`pc`) y probabilidad de mutación (`pm`).
4. Sección de soluciones, en las que se muestran las mejores soluciones encontradas por el AG en cada era.
5. Sección resumen del proceso, en la que se detallan los resultados de la ejecución. Se muestran datos como el tiempo de ejecución empleado por el AG para

llegar a las soluciones, así como el valor máximo, medio y desviación estándar de las soluciones encontradas en todas las eras de la ejecución del AG.

El contenido del segundo fichero de salida (`soluciones_ag`) es más sencillo y se organiza en una única sección precedida, al igual que el fichero anterior, por un encabezamiento:

1. Encabezamiento que muestra el nombre de la herramienta, SIGENOF, junto con la fecha y la hora de ejecución.
2. Sección con los mejores valores de la función de adaptación para cada una de las generaciones de las que consta cada era.

Se incluye en el Apéndice D un listado de los dos fichero *log* obtenidos por SIGENOF para una determinada ejecución.

Capítulo 3

Interfaz Gráfico de Usuario de SIGENOF

3.1. Librería gráfica utilizada

Para la implementación del Interfaz Gráfico de Usuario, GUI en lo sucesivo, se ha empleado la librería gráfica GTK+¹, la cual se integra perfectamente con el lenguaje de programación elegido para la implementación del AG, el lenguaje C. Esta librería, básicamente, proporciona un conjunto de componentes útiles en el diseño de GUIs que permiten el desarrollo de aplicaciones que podrán ejecutarse en distintas plataformas software. Los aspectos decisivos que han condicionado la elección de esta librería para la implementación del GUI han sido:

- GTK+ es software libre y parte del proyecto GNU².
- Permite el desarrollo de aplicaciones gráficas multiplataforma.
- Proporciona un abanico amplio de componentes gráficos.
- Existen editores gráficos, como GLADE³, que también es software libre, que facilitan la labor de diseño del GUI.

¹www.gtk.org

²www.gnu.org

³glade.gnome.org y gladewin32.sourceforge.net (para MS-Windows)

3.2. Visión general

La herramienta implementada, SIGENOF, dispone de un sencillo, cómodo e intuitivo GUI, que sirve como *front-end* al AG para proporcionar interactividad con el usuario que lo ejecuta. Este GUI permite, entre otras funcionalidades, introducir los parámetros poblacionales básicos del AG como el número de eras en las que se va a ejecutar el algoritmo, el número de generaciones para cada una de estas eras y el número de individuos de que constará la población de cromosomas; parámetros naturales tales como la probabilidad de cruce entre cromosomas y la probabilidad de mutación natural, así como el detalle de la función matemática que el algoritmo deberá optimizar. Esta función se podrá introducir de forma manual o desde un fichero almacenado con anterioridad por la propia herramienta en formato texto. Además se facilitan una serie de controles y visualizaciones que permitirán al usuario pausar la ejecución entre eras o cancelar el proceso en una era determinada, así como observar gráficamente cómo las sucesivas generaciones de soluciones varían. El GUI también mostrará resultados de la ejecución en términos de tiempo empleado, soluciones encontradas en cada una de las eras y valor máximo, medio y desviación estándar de las soluciones. Por último, para que el usuario que ejecuta la herramienta tenga una visión estimada del tiempo restante de ejecución del AG, se visualiza en la parte inferior del GUI una barra de progreso que avanza conforme la ejecución del algoritmo también lo hace.

3.3. Funcionalidades del GUI

Para describir todas las funcionalidades que proporciona el GUI, se ha optado por clasificar en paneles las diferentes partes que éste aporta tal y como se puede observar en la figura 3.1.

En total se identifican los ocho paneles siguientes:

1. Panel de parámetros poblacionales.
2. Panel de parámetros naturales.
3. Panel de otros parámetros de aplicación para el funcionamiento del AG.
4. Panel de visualización textual de las mejores soluciones entre eras.

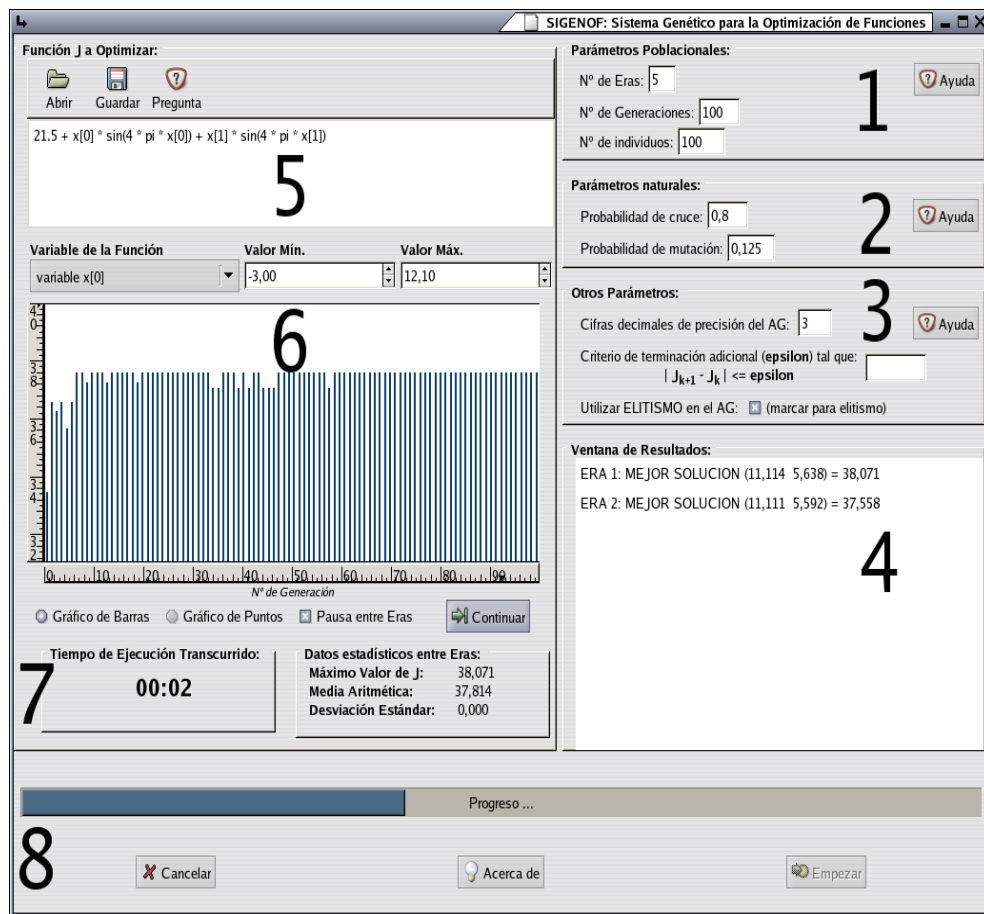


Figura 3.1: GUI de SIGENOF

5. Panel para definir, guardar o recuperar la función J a optimizar por la herramienta.
6. Panel de visualización gráfica de las mejores soluciones para la función de adaptación en cada generación para cada era.
7. Panel estadístico y de tiempo de ejecución.
8. Panel de control y visualización del avance del proceso.

A continuación se detallarán las funcionalidades aportadas por cada uno de los ocho paneles anteriormente enumerados.

3.3.1. Panel 1: Parámetros Poblacionales

Este panel permite introducir tres parámetros que afectan a la población con la que el AG trabajará. En concreto el número de Eras que correrá el algoritmo, el número de Generaciones que cada era contendrá, y el número de individuos de la población.

Se proporciona un botón AYUDA al usuario que informa del significado de estos parámetros.

Se realiza una comprobación de que los valores introducidos son los correctos, es decir que son números positivos mayores que cero dentro del rango de posibles valores permitidos para cada uno de los parámetros:

- Número de Eras $\in [1, 999]$
- Número de Generaciones $\in [1, 99999]$
- Número de Individuos $\in [1, 999999]$

3.3.2. Panel 2: Parámetros Naturales

Permite la especificación de dos parámetros que afectan al proceso de selección de cromosomas, reproducción y mutación genética. Estos parámetros son la Probabilidad de Cruce y la Probabilidad de Mutación.

Se proporciona un botón AYUDA al usuario que informa del significado de estos parámetros.

De la misma manera que en el panel anterior, se realiza una comprobación de que los valores introducidos son los correctos, es decir que son números positivos mayores que cero dentro del rango de posibles valores permitidos para cada uno de los parámetros:

- Probabilidad de Cruce $\in [0.001, 0.999]$
- Probabilidad de Mutación $\in [0.00001, 0.99999]$

3.3.3. Panel 3: Otros Parámetros del AG

En este panel se pueden introducir tres parámetros adicionales que condicionarán el comportamiento del AG de diferentes formas. Estos parámetros son: el número

de cifras decimales con las que el AG trabajará internamente para la representación de las soluciones, un criterio adicional de terminación (criterio épsilon) y la opción de selección del mecanismo de elitismo.

- Cifras decimales de precisión del AG: Número de cifras decimales con las que el AG trabajará para representar las soluciones.
- Criterio adicional de terminación (épsilon): Se define un umbral ε a partir del cual el AG finalizará cuando la diferencia entre el valor J de la generación $(k+1)$ y el de la generación (k) sea menor o igual a dicho ε , aunque no se haya alcanzado el número máximo de generaciones indicado, es decir cuando $|J_{k+1} - J_k| \leq \varepsilon$. Cuando no se desee aplicar dicho criterio adicional de terminación, este parámetro se deberá dejar sin contenido.
- Elitismo: Esta opción se marcará si se desea aplicar en el AG el mecanismo de elitismo. En caso contrario se deberá dejar sin marcar.

Se proporciona un botón AYUDA al usuario que informa del significado de estos parámetros.

De la misma manera que en los paneles anteriores, se realiza una comprobación de que los valores introducidos son los correctos, es decir que son números positivos mayores que cero dentro del rango de posibles valores permitidos para cada uno de los parámetros:

- Cifras decimales de precisión del AG $\in [1, 6]$
- Epsilon $\in [0, 000001, 99999999]$

3.3.4. Panel 4: Ventana de Resultados

Este panel muestra, para cada una de las eras, la mejor solución encontrada. La información que se visualiza es la siguiente:

- Información del número de era.
- Valor máximo obtenido para la función J , junto con los valores x_k que la maximizan.

- Si se ha establecido un valor ε como criterio de terminación adicional, se muestra la información de la generación en la que se ha alcanzado dicho umbral para cada una de las eras.
- Si el proceso ha sido cancelado por el usuario antes de la finalización del AG, se muestra dicha información en este panel.

A continuación se muestra la salida del AG en la ventana de resultados para una ejecución determinada:

```
ERA 1: MEJOR SOLUCION (11,180 5,627) = 35,739 :: (epsilon en generacion 4)
ERA 2: MEJOR SOLUCION (9,598 4,609) = 35,067 :: (epsilon en generacion 4)
ERA 3: MEJOR SOLUCION (10,633 5,609) = 37,575 :: (epsilon en generacion 7)
ERA 4: MEJOR SOLUCION (11,572 5,641) = 36,126 :: (epsilon en generacion 6)
*** Proceso cancelado por el usuario ***
```

3.3.5. Panel 5: Función J a Optimizar

Es en este panel donde se deberá expresar aritméticamente la función que el AG deberá maximizar, mediante una sintaxis previamente definida y que se puede consultar a través del botón PREGUNTA. Este panel se divide, a su vez, en tres subpaneles:

Subpanel 5.1: Barra de herramientas

Muestra tres botones, ABRIR, GUARDAR y PREGUNTA.

- Botón ABRIR. Recupera desde un archivo una función previamente almacenada por la herramienta. Este botón despliega un diálogo que permite navegar a través del sistema de ficheros local o remoto para localizar de forma sencilla el fichero buscado. El diálogo de selección de fichero permite, incluso, añadir accesos a carpetas *favoritas* para localizar posteriores ficheros en esa ruta con rapidez.
- Botón GUARDAR. Permite guardar a disco la definición de una función definida en la herramienta, así como los valores $[x_k^{min}, x_k^{max}]$ establecidos para cada variable x_k de la función. El diálogo que se muestra para definir el nombre del fichero que se desea almacenar, así como su ubicación que puede ser local

o remota, permite añadir también carpetas *favoritas* de la misma manera a como lo realiza el diálogo ABRIR. Estas ubicaciones *favoritas* son compartidas entre ambos diálogos.

- Botón PREGUNTA. Muestra una ventana con los operadores disponibles en la definición de la función a optimizar (*suma*, *resta*, *producto...*), las funciones (*sin*, *cos*, ...) y constantes que se pueden usar (*e*, *pi*, *números enteros*, ...), la forma de escribir las diferentes variables x_k que componen la función ($x[0]$, ..., $x[n]$), así como las reglas de precedencia que regirán la interpretación de la función.

El nombre y extensión de los ficheros que almacenan las funciones que la herramienta guarda y recupera se deja a elección del usuario, aunque su formato interno debe ajustarse al definido por la herramienta según el siguiente esquema:

- Línea de definición sintáctica de la función: #FUN: , ajustada a la sintaxis definida en la sección 2.3.
- Línea del valor mínimo que puede tomar el parámetro x_1 de la función: #X0_MIN: , y así sucesivamente hasta el parámetro x_{10} .
- Línea del valor máximo que puede tomar el parámetro x_1 de la función: #X0_MAX: , y así sucesivamente hasta el parámetro x_{10} .
- Cuando en la definición de la función únicamente se requieren n parámetros x , de forma que $n \leq 10$, no es necesario *declarar* las diez variables #Xi_MIN y #Xi_MAX, aunque sí es aconsejable hacerlo con valores cero para definir, de manera explícita, que estas variables no tienen contenido y que no son necesarias en la función definida en la línea #FUN:. Este es el criterio seguido por SIGENOF.
- El orden de estas líneas es irrelevante, aunque para facilitar la lectura y edición de estos ficheros la herramienta los almacena en el mismo orden en el que aquí se han descrito.
- El contenido íntegro del fichero se almacena en formato ASCII, de manera que puede ser editado externamente desde cualquier editor de textos.

Un ejemplo del contenido de un fichero de este tipo es el siguiente:

```
#FUN:21.5 + x[0] * sin(4 * pi * x[0]) + x[1] * sin(4 * pi * x[1])

#X0_MIN:-3,00
#X0_MAX:12,10
#X1_MIN: 4,10
#X1_MAX: 5,80
#X2_MIN: 0,00
#X2_MAX: 0,00
#X3_MIN: 0,00
#X3_MAX: 0,00
#X4_MIN: 0,00
#X4_MAX: 0,00
#X5_MIN: 0,00
#X5_MAX: 0,00
#X6_MIN: 0,00
#X6_MAX: 0,00
#X7_MIN: 0,00
#X7_MAX: 0,00
#X8_MIN: 0,00
#X8_MAX: 0,00
#X9_MIN: 0,00
#X9_MAX: 0,00
```

Subpanel 5.2: Área de definición de la función

Se compone de un área de texto en el que se puede escribir la expresión que definirá la función a optimizar, o donde se mostrará la función tras recuperarla desde un archivo.

Subpanel 5.3: Área de definición de las variables x_k

Permitirá, a través de la lista desplegable VARIABLE DE LA FUNCIÓN, así como de los campos VALOR MÍN. y VALOR MÁX., definir el rango de valores que las diferentes variables x_k ($x[k]$) podrán tomar. Se permite especificar los valores de hasta diez variables x_k .

3.3.6. Panel 6: Gráfico de Soluciones generacionales

Las diferentes soluciones encontradas para la función J en cada una de las sucesivas generaciones de una era particular, son visualizadas gráficamente en este panel. En el eje horizontal del gráfico se muestran las generaciones, mientras que en el eje vertical se muestran las mejores soluciones obtenidas para cada generación. Los datos se muestran, por defecto, como barras. Esta visualización puede cambiarse de forma que las soluciones se muestren como puntos seleccionando al pie del gráfico el botón de radio GRÁFICO DE PUNTOS. Para volver de nuevo a la visualización de barras se puede seleccionar el botón de radio GRÁFICO DE BARRAS.

Como este gráfico muestra las mejores soluciones obtenidas para la función de adaptación en cada generación de una cierta era, cuando el AG cambia de era el gráfico también cambia los valores mostrados. Debido a que estos cambios, en muchas ocasiones, se realizarán rápidamente, el usuario dispone de la posibilidad de realizar pausas entre eras para que, de este modo, se pueda observar con detenimiento el gráfico de mejores soluciones antes del cambio de era, acción que se realiza pulsando el botón CONTINUAR.

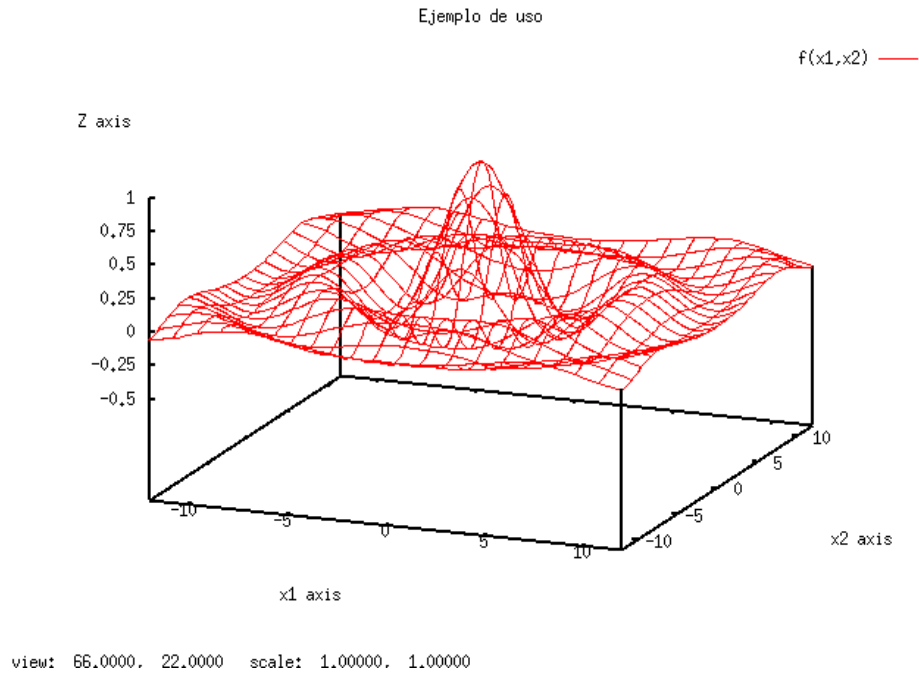
3.3.7. Panel 7: Datos estadísticos y Tiempo de ejecución

En este panel se muestra el tiempo de ejecución consumido por el AG en formato MM:SS, donde MM y SS representan los minutos y segundos respectivamente, junto con algunos datos estadísticos del resultado de la ejecución entre las sucesivas eras. En concreto se muestra el Máximo Valor encontrado para J , su Media Aritmética y, al finalizar el algoritmo, la Desviación Estándar entre soluciones.

3.3.8. Panel 8: Progreso y Control del proceso

Este último panel proporciona una barra de progreso que muestra de forma visual el tiempo de ejecución del proceso, junto con tres botones adicionales, de los cuales dos de ellos permiten el control del proceso. Estos botones son:

- Botón EMPEZAR. Valida los datos introducidos en el GUI y, en caso de estar completos y correctos, empieza la ejecución del AG.

Figura 3.2: *Función de ejemplo*

- Botón CANCELAR. Permite terminar el AG tras finalizar la ejecución de la era en curso.
- Botón ACERCA DE. Proporciona información de la herramienta y del autor.

3.4. Ejemplo de uso del GUI

A continuación se va a describir el uso del GUI siguiendo un caso práctico. Para ello se ha definido la función de ejemplo $f(x_1, x_2)$ en el espacio S tal y como se muestra seguidamente.

$$f(x_1, x_2) = \frac{\sin(\sqrt{x_1^2 + x_2^2})}{\sqrt{x_1^2 + x_2^2}}$$

$$S = \{x_1 \in [-12, 12.01], x_2 \in [-12, 12.01]\}$$

Se presenta gráficamente la función $f(x_1, x_2)$ en S propuesta como ejemplo en la figura 3.2.

En dicha representación gráfica se puede observar que el máximo de esta función para el espacio S definido se encuentra en el punto $(0, 0)$ con un valor máximo para

$f(x_1, x_2) = 1$. Comprobaremos al finalizar el ejemplo cómo SIGENOF se aproxima a la solución alrededor de dicho punto.

Se va a describir la realización del caso práctico en diferentes pasos, los cuales se enumeran como:

- Paso 1. Iniciar SIGENOF
- Paso 2. Definir la función J a optimizar
- Paso 3. Establecer el espacio S
- Paso 4. Grabar la función J y su espacio S definido
- Paso 5. Establecer los parámetros poblacionales
- Paso 6. Establecer los parámetros naturales del AG
- Paso 7. Establecer otros parámetros del AG
- Paso 8. Seleccionar el tipo de visualización gráfica
- Paso 9. Comenzar con la ejecución
- Paso 10. Evaluación de las soluciones obtenidas

Es importante destacar que los pasos 1 a 8 se pueden realizar en cualquier orden sin afectar al resultado final del proceso.

A continuación se describen de forma detallada las acciones a realizar para completar cada uno de los pasos anteriormente enumerados.

3.4.1. Paso 1. Iniciar SIGENOF

Se procede a ejecutar el archivo binario ***sigenof***. En plataformas *MS-Windows* el archivo binario será ***sigenof.exe***.

La apariencia del GUI será como la mostrada en la figura 3.3. Los parámetros del AG mostrarán los valores por defecto que se pueden observar en la figura, mientras que la función a optimizar se encontrará en este paso sin contenido alguno a la espera de que el usuario que ejecuta la herramienta decida qué función desea optimizar, así como qué parámetros desea establecer para la ejecución del AG.

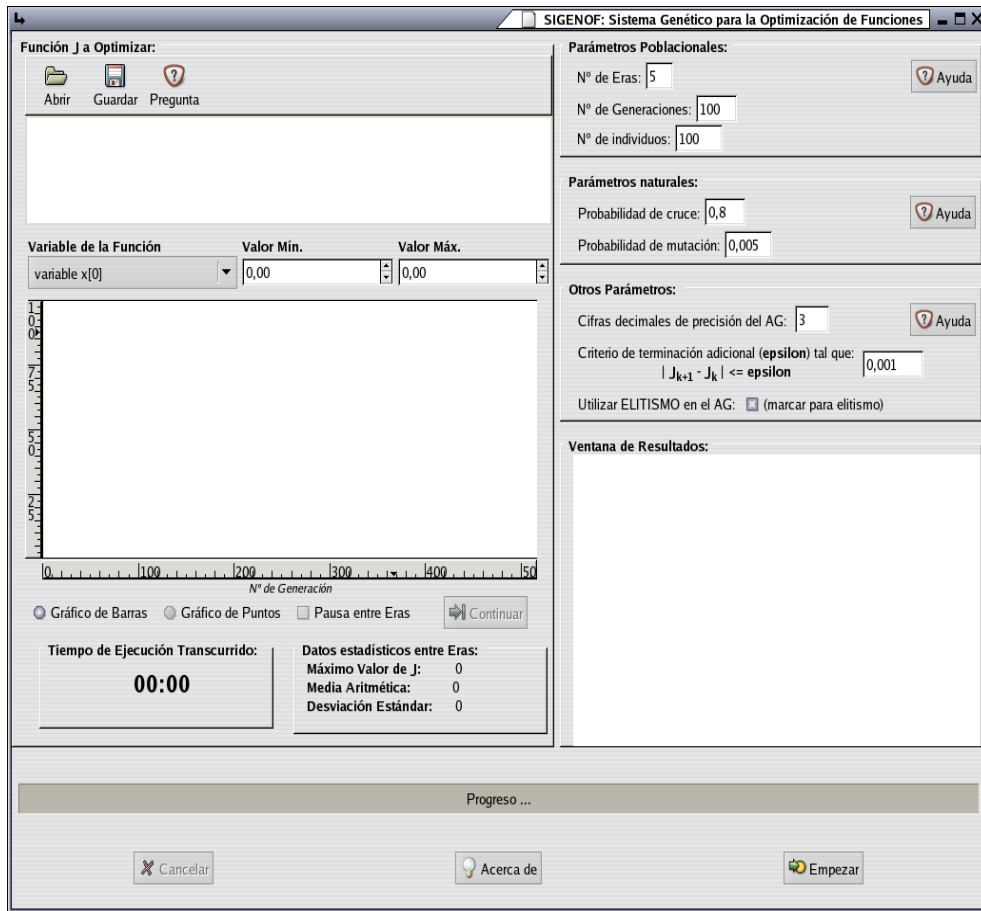


Figura 3.3: Inicio de SIGENOF

3.4.2. Paso 2. Definir la función J a optimizar

La función J propuesta anteriormente para seguir este ejemplo fue:

$$f(x_1, x_2) = \frac{\sin(\sqrt{x_1^2 + x_2^2})}{\sqrt{x_1^2 + x_2^2}}$$

Esta función J se escribirá en el área de texto del área "**Función J a optimizar**" del GUI de SIGENOF, según la sintaxis descrita en el cuadro 2.3, con la siguiente expresión:

```
sin( sqrt( x[0]^2 + x[1]^2 ) ) / sqrt( x[0]^2 + x[1]^2 )
```

Se puede observar esta acción en la figura 3.4.

3.4.3. Paso 3. Establecer el espacio S

El espacio S para la función $f(x_1, x_2)$ se define como:

$$S = \{x_1 \in [-12, 12.01], x_2 \in [-12, 12.01]\}$$

Este espacio de posibles valores para las variables de incertidumbre de la función J que se desea optimizar se definirá en el área "**Variable de la Función**".

En este área existen tres elementos para la definición de los distintos valores que componen el espacio S para cada uno de las variables x_k de la función:

- *Lista desplegable para establecer las distintas variables x_k .* A través de esta lista se permite definir valores para hasta diez variables x_k (desde $x[0]$ hasta $x[9]$).
- *Valor Mínimo de S para cada x_k .* Para cada $x[k]$ se indicará el valor mínimo que la variable puede tomar.
- *Valor Máximo de S para cada x_k .* Para cada $x[k]$ se indicará el valor máximo que la variable puede tomar.

En nuestro caso se definirán los siguientes valores para las variable x_1 y x_2 de la función:

- Variable x_1
 - Variable $x[0]$ en la lista desplegable.
 - Valor Mín: -12,00
 - Valor Máx: 12,01
- Variable x_2
 - Variable $x[1]$ en la lista desplegable.
 - Valor Mín: -12,00
 - Valor Máx: 12,01

Se puede observar el procedimiento de definición de S en la figura 3.4.

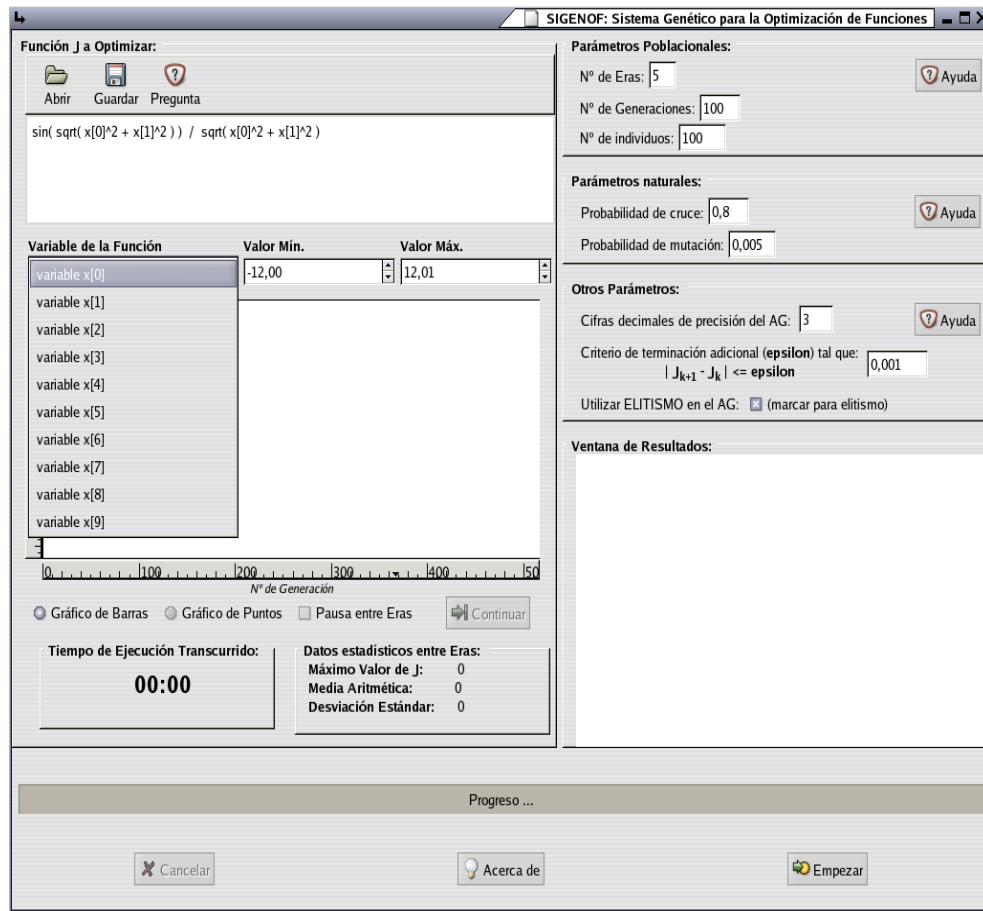
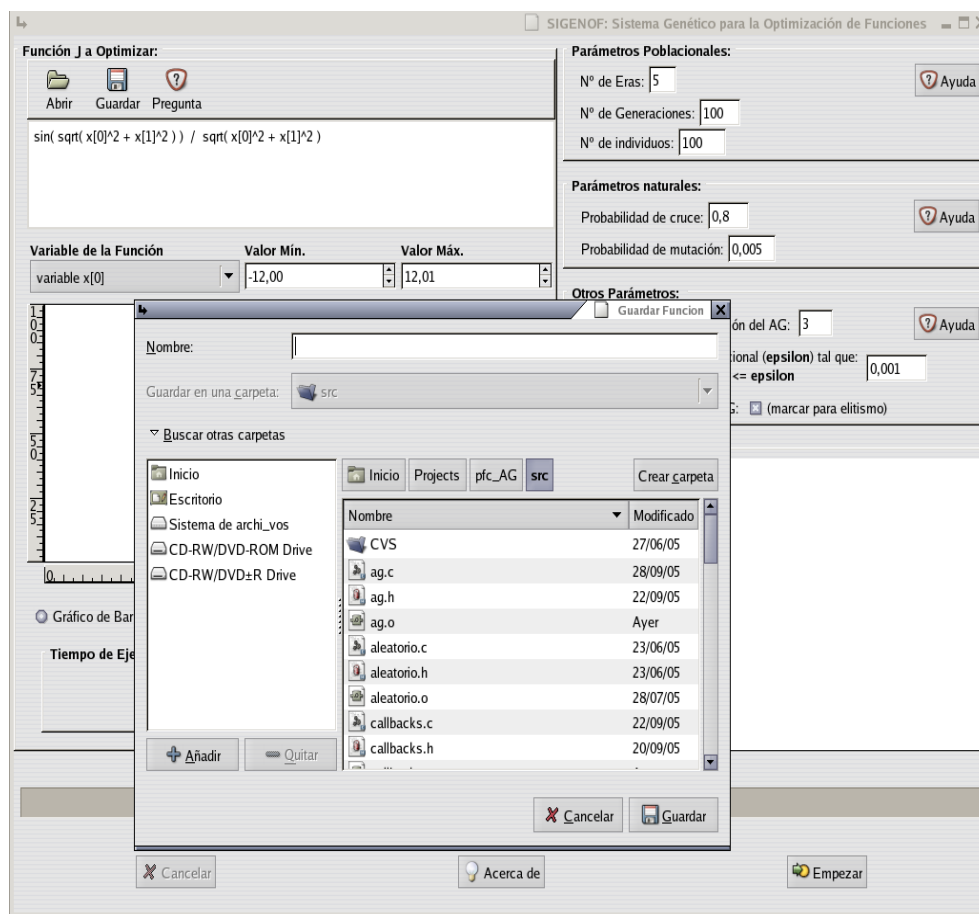


Figura 3.4: Definición de la función J y del espacio S

3.4.4. Paso 4. Grabar la función J y su espacio S definido.

Llegados a este punto, la función J y su espacio S se encuentran completamente definidos. En este momento es conveniente guardar dichas definiciones en un fichero para poder recuperarlas en cualquier otro momento y poder utilizarlas con la herramienta. Para ello pulsaremos el icono "**Guardar**" situado en la barra de herramientas superior del área "**Función J a optimizar**", tal y como se muestra en la figura 3.5.

En el diálogo para guardar la función, que aparece en la parte central del GUI de la figura como diálogo "**Guardar Función**", se deberá introducir el nombre del fichero que almacenará los datos introducidos en pantalla, así como su ubicación en el sistema de archivos, que puede ser local o remoto. Para buscar carpetas en ubicaciones remotas o en distintos lugares del sistema de archivos local, se podrá

Figura 3.5: Grabación de la función J y espacio S

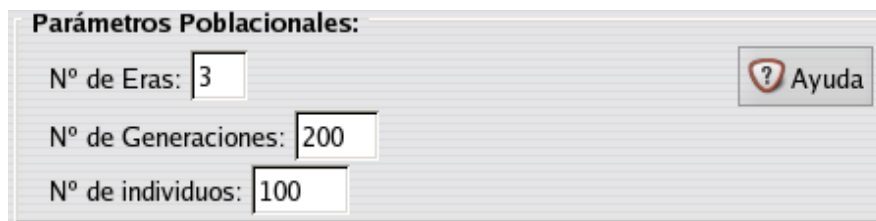
desplegar la opción "**Buscar otras carpetas**" de este diálogo para ampliar, de esta forma, su funcionalidad. Este diálogo extendido es el que se muestra en la figura 3.5.

Mediante el botón "+ AÑADIR" del diálogo "**Guardar Función**", se pueden definir "**rutas favoritas**", es decir rutas a las que se acceden con frecuencia en operaciones de guardado o recuperación de funciones.

Una función guardada se podrá recuperar en cualquier momento pulsando el icono "**Abrir**" de la barra de herramientas y seleccionando su ubicación de manera similar a como se ha realizado con el diálogo "**Guardar Función**".

3.4.5. Paso 5. Establecer los parámetros poblacionales

Para este ejemplo se definieron los siguientes parámetros:



Parámetros Poblacionales:

Nº de Eras: 3

Nº de Generaciones: 200

Nº de individuos: 100

Ayuda

Figura 3.6: *Parámetros poblacionales*

- **Número de Eras:** Se establecieron tres eras.
- **Número de Generaciones:** Para el ejemplo se establecieron doscientas generaciones para cada una de las tres eras definidas en el parámetro anterior.
- **Número de Individuos:** Se definieron cien individuos para la población del ejemplo.

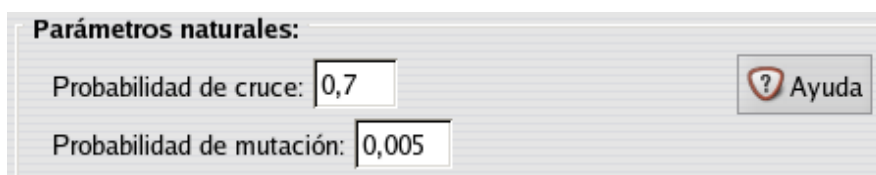
Se puede observar la introducción de estos parámetros en la figura 3.6.

3.4.6. Paso 6. Establecer los parámetros naturales del AG

En el ejemplo se definieron estos parámetros como:

- **Probabilidad de cruce:** Se estableció una probabilidad de cruce entre parejas de individuos equivalente a un 70 %, es decir con un valor de 0,7.
- **Probabilidad de mutación:** Se definió para la ejecución una probabilidad de mutación genética para cada individuo equivalente a un 0,5 %, es decir con un valor de 0,005.

Se pueden observar estos parámetros en la figura 3.7.



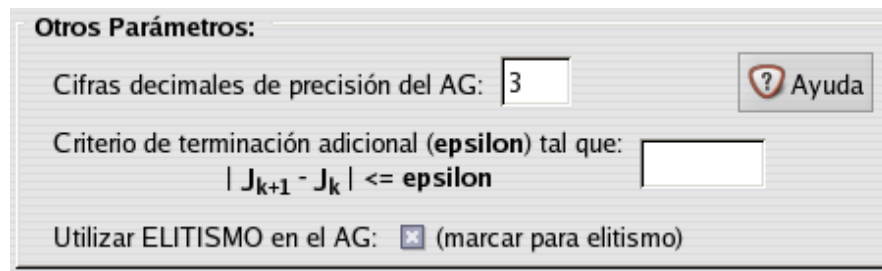
Parámetros naturales:

Probabilidad de cruce: 0,7

Probabilidad de mutación: 0,005

Ayuda

Figura 3.7: *Parámetros naturales*

Figura 3.8: *Otros parámetros*

3.4.7. Paso 7. Establecer otros parámetros del AG

Otros parámetros que se definieron para esta ejecución fueron:

- **Cifras decimales de precisión del AG:** Se optó por dejar el valor de este parámetro al que por defecto la herramienta propone, tres decimales.
- **Criterio de terminación adicional (*epsilon*):** Para este ejemplo se decidió no aplicar el criterio de terminación adicional *epsilon*, así que se dejó sin contenido el campo.
- **Utilizar *ELITISMO* en el AG:** En este caso sí se deseaba aplicar el mecanismo de elitismo y, por tanto, se marcó dicha casilla.

Los parámetros introducidos en este caso se pueden observar en la figura 3.8.

3.4.8. Paso 8. Seleccionar el tipo de visualización gráfica

Se seleccionó como tipo de gráfico "**Gráfico de Barras**". Esta selección se realiza pulsando el botón de radio que aparece al pie del espacio reservado para la visualización gráfica, tal y como se puede observar en la figura 3.9.

Para observar con detenimiento la gráfica de las mejores soluciones encontradas por la herramienta, se decidió marcar la casilla "**Pausa entre Eras**", de forma que al finalizar una era su ejecución, quede ésta suspendida hasta que el usuario pulse el botón "CONTINUAR", momento en el cual comenzará la siguiente era, y así sucesivamente hasta la finalización del proceso.

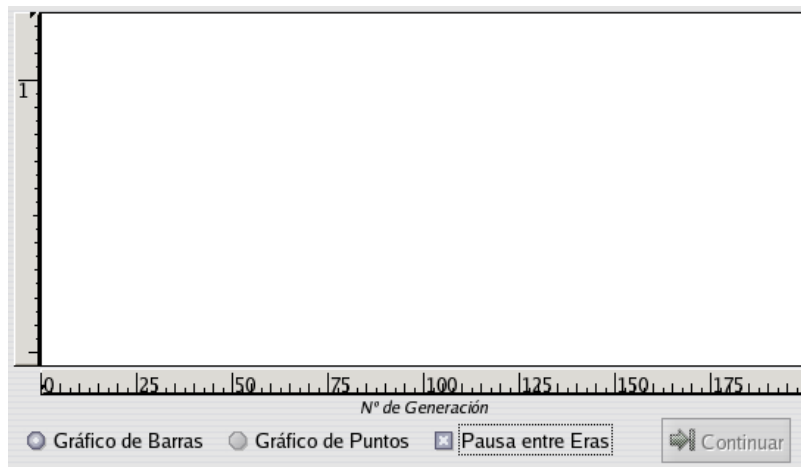


Figura 3.9: Tipo de visualización gráfica

3.4.9. Paso 9. Comenzar con la ejecución

En este último paso únicamente resta pulsar el botón "EMPEZAR" para que SIGENOF comience su ejecución y obtenga las soluciones para el problema propuesto.

3.4.10. Paso 10. Evaluación de las soluciones obtenidas

Tras finalizar la ejecución de todas las eras, el AG implementado en SIGENOF obtuvo las soluciones que se observan en la figura 3.10, así como los valores estadísticos de la figura 3.11.

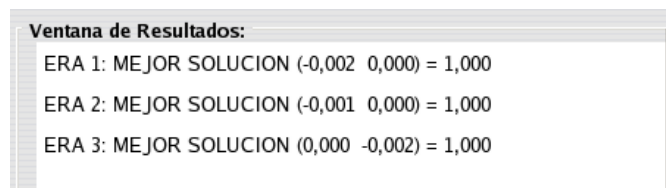


Figura 3.10: Soluciones obtenidas por SIGENOF para el ejemplo propuesto

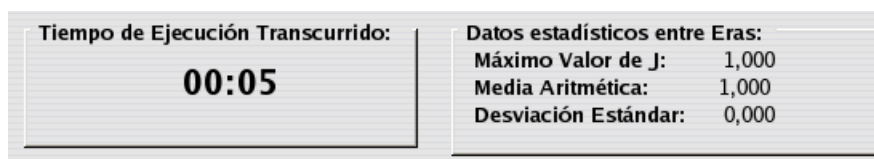
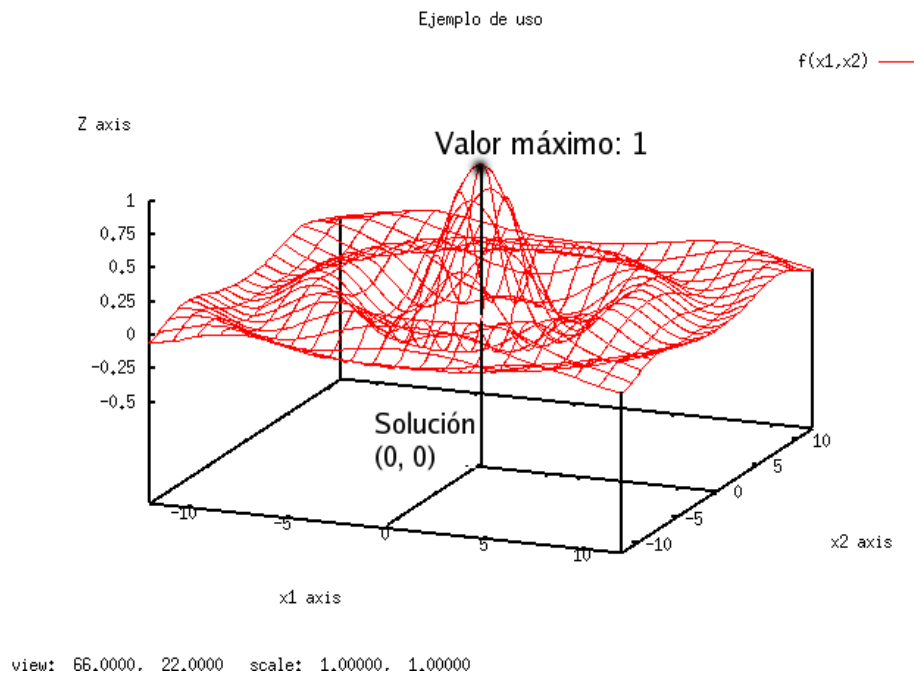


Figura 3.11: Resultados estadísticos para el ejemplo propuesto

Figura 3.12: *Máximo para el ejemplo propuesto*

Se observa que la herramienta encontró en las tres eras similares soluciones en las cercanías de $(0, 0)$, con un valor máximo para $J = 1, 0$.

Si ahora examinamos de nuevo la forma gráfica de la función J definida para este ejemplo, observaremos claramente cómo el valor máximo efectivamente se encuentra en el punto $(0, 0)$ con una cota de 1 para el espacio S definido en nuestro ejemplo. Se muestra este máximo en la figura 3.12.

Esta solución, a la que podríamos haber llegado fácilmente mediante técnicas analíticas convencionales, se ha alcanzado satisfactoriamente mediante una implementación de AG. En este caso la solución obtenida tiene una precisión muy alta con respecto a la solución real.

Capítulo 4

Ejemplos de aplicación de SIGENOF

Se detallan a continuación algunos casos particulares que describen diversas funciones matemáticas con las que se ha verificado el buen funcionamiento del algoritmo genético implementado.

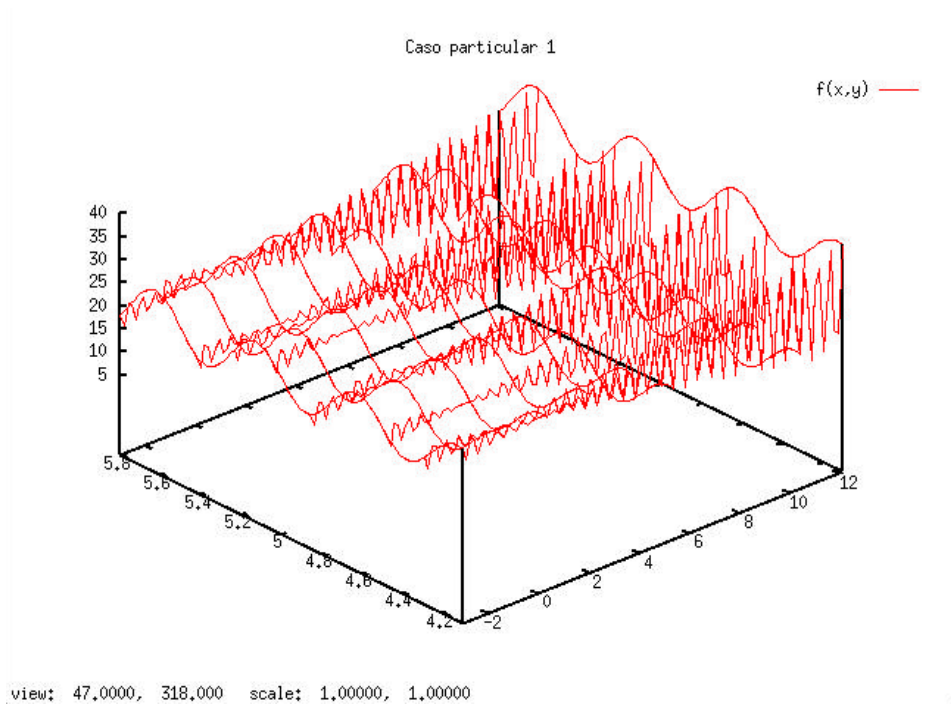
En la ejecución de todos los casos particulares que se describen, se han empleado los parámetros que se muestran en el cuadro 4.1:

| <i>Parámetro</i> | <i>Valor establecido</i> |
|----------------------------------|--------------------------|
| NÚMERO DE ERAS | 5 |
| NÚMERO DE GENERACIONES | 100 |
| TAMAÑO DE LA POBLACIÓN | 100 |
| NÚMERO DE DECIMALES DE PRECISIÓN | 3 |
| CRITERIO DE TERMINACIÓN EPSILON | NO |
| USO DE MECANISMO DE ELITISMO | SÍ |
| PROBABILIDAD DE CRUCE | 80 % (0,8) |
| PROBABILIDAD DE MUTACIÓN | 0,5 % (0,005) |

Cuadro 4.1: *Parámetros del AG utilizados en los ejemplos*

4.1. Caso particular 1

La función objetivo a optimizar se describe como:

Figura 4.1: *Caso Particular 1*

$$f(x_1, x_2) = 21.5 + x_1 \cdot \sin(4\pi x_1) + x_2 \cdot \sin(4\pi x_2)$$

$$S = \{x_1 \in [-3.0, 12.1], x_2 \in [4.1, 5.8]\}$$

Se representa la función $f(x_1, x_2)$ en S en la figura 4.1.

En ella se puede observar que el valor máximo para la función se encuentra entorno al punto $x_1 = 12$ y $x_2 = 5.6$, con un valor para f próximo a 39.

SIGENOF obtuvo como mejores soluciones las cercanas al punto $x_1 = 11.6$ y $x_2 = 5.629$, con un valor para f de 38.682.

4.2. Caso particular 2

La función objetivo a optimizar se describe como:

$$f(x_1, x_2, x_3, x_4, x_5, x_6) = 100 - (x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2)$$

$$S = \left\{ \begin{array}{l} x_1 \in [-3.0, 5.1], x_2 \in [2.1, 7.8], x_3 \in [-10.1, 20.3], \\ x_4 \in [-3.3, 4.2], x_5 \in [-15.3, 70.1], x_6 \in [-0.25, 0.35] \end{array} \right\}$$

Fácilmente se puede comprobar que el máximo de esta función se encuentra en el punto $x_i = 0$, con $1 \leq i \leq 6$, con una cota para f de 100.

Para este caso particular, la probabilidad de mutación se varió hasta un valor de 2,5 % (0,025) para que el algoritmo pudiera converger a la solución con mayor precisión. De esta forma SIGENOF coincidió con la solución real obteniendo puntos próximos a cero ($x_i = 0$), con un valor para f de 100.

4.3. Caso particular 3. Función de Ackley

La función objetivo a optimizar se describe como:

$$f(x_1, x_2) = -c_1 \cdot e^{\left(-c_2 \sqrt{\frac{1}{2}(x_1^2 + x_2^2)}\right)} - e^{\frac{1}{2}(\cos(c_3 x_1) + \cos(c_3 x_2))} + c_1 + e$$

$$c_1 = 20 \quad c_2 = 0.2 \quad c_3 = 2\pi \quad e = 2.71282$$

$$S = \{x_1, x_2 \in [-5, 5]\}$$

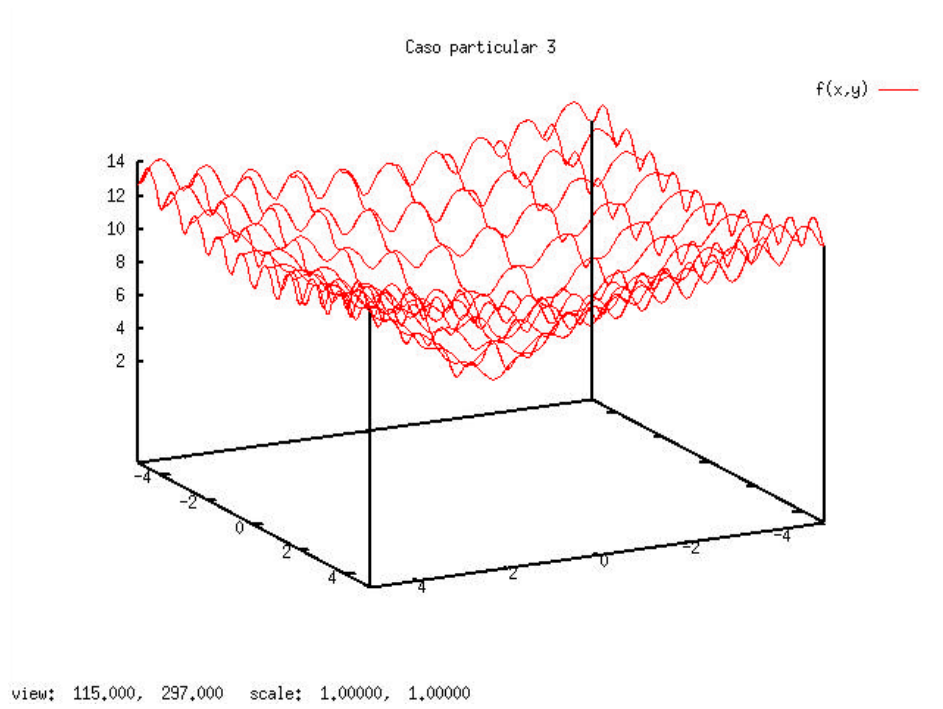
Se representa la función de Ackley $f(x_1, x_2)$ en S en la figura 4.2.

En ella se puede observar que existen cuatro máximos para la función. Los puntos que hacen máxima la función con un valor cercano a 14.30, se encuentran entorno a los cuatro siguientes puntos:

- Solución 1: $x_1 = -4.60$ y $x_2 = 4.60$
- Solución 2: $x_1 = 4.60$ y $x_2 = -4.60$
- Solución 3: $x_1 = -4.60$ y $x_2 = -4.60$
- Solución 4: $x_1 = 4.60$ y $x_2 = 4.60$

SIGENOF obtuvo como mejores soluciones las cercanas a los cuatro mismos puntos anteriormente enumerados tras la observación de la gráfica, todos ellos con valores para f próximos a 14.300.

- Solución 1: $x_1 = -4.599$ y $x_2 = 4.633$ con $f = 14.297$.
- Solución 2: $x_1 = 4.608$ y $x_2 = -4.572$ con $f = 14.300$.
- Solución 3: $x_1 = -4.640$ y $x_2 = -4.656$ con $f = 14.274$.
- Solución 4: $x_1 = 4.638$ y $x_2 = 4.596$ con $f = 14.296$.

Figura 4.2: *Caso Particular 3*

4.4. Caso particular 4. Función de Shaffer

La función objetivo a optimizar se describe como:

$$f(x_1, x_2) = 100 - [(x_1^2 + x_2^2)^{0.25} \cdot [\sin^2(50 \cdot (x_1^2 + x_2^2)^{0.1}) + 1.0]]$$

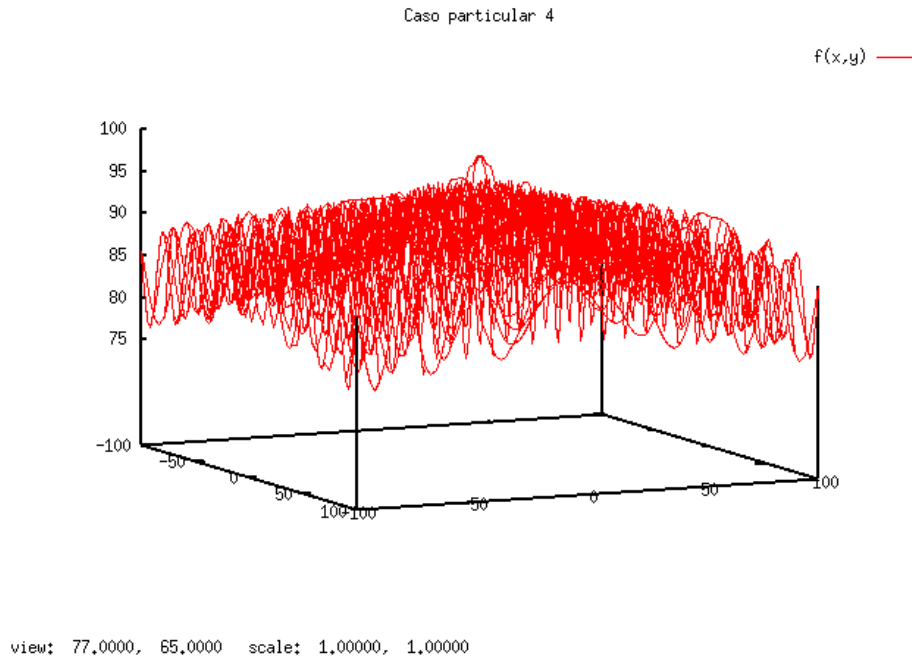
$$S = \{x_1, x_2 \in [-100, 100]\}$$

Se representa la función de Shaffer $f(x_1, x_2)$ en S en la figura 4.3.

Observando la gráfica se comprueba que el máximo de esta función se encuentra en el punto $x_1 = x_2 = 0$ con un valor para f de 100.

SIGENOF obtuvo la misma solución que la observada gráficamente, es decir valores para x_1 y x_2 cercanos a 0 con valores para f entorno a 99.9. Ejemplos de soluciones obtenidas por la herramienta son las siguientes:

- $x_1 = -0.002$ y $x_2 = -0.002$ con $f = 99.944$.
- $x_1 = 0.001$ y $x_2 = 0.000$ con $f = 99.968$.

Figura 4.3: *Caso Particular 4*

4.5. Caso particular 5

La función objetivo a optimizar se describe como:

$$f(x_1, x_2) = 500 - [a \cdot (x_2 - b \cdot x_1^2 + c \cdot x_1 - d)^2 + e \cdot (1 - f) \cdot \cos(x_1) + e]$$

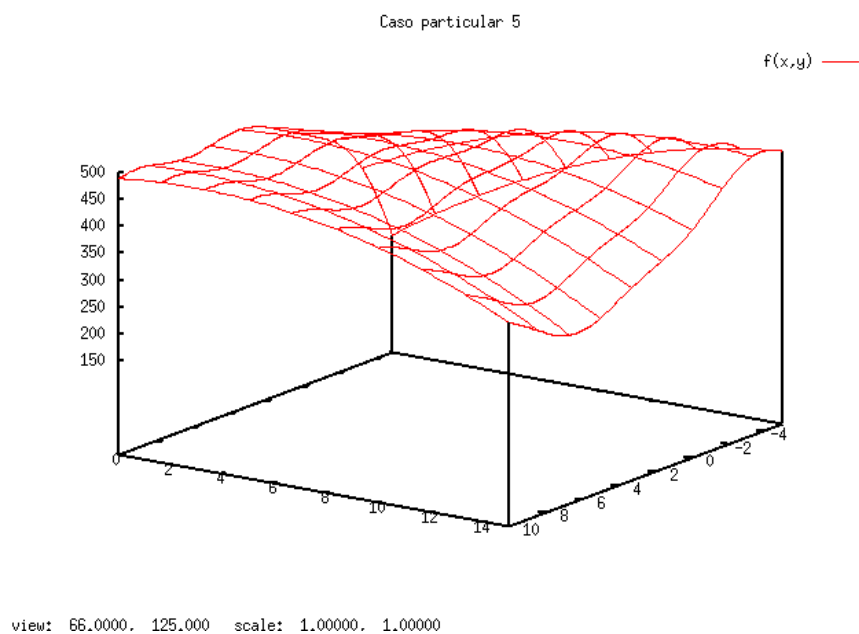
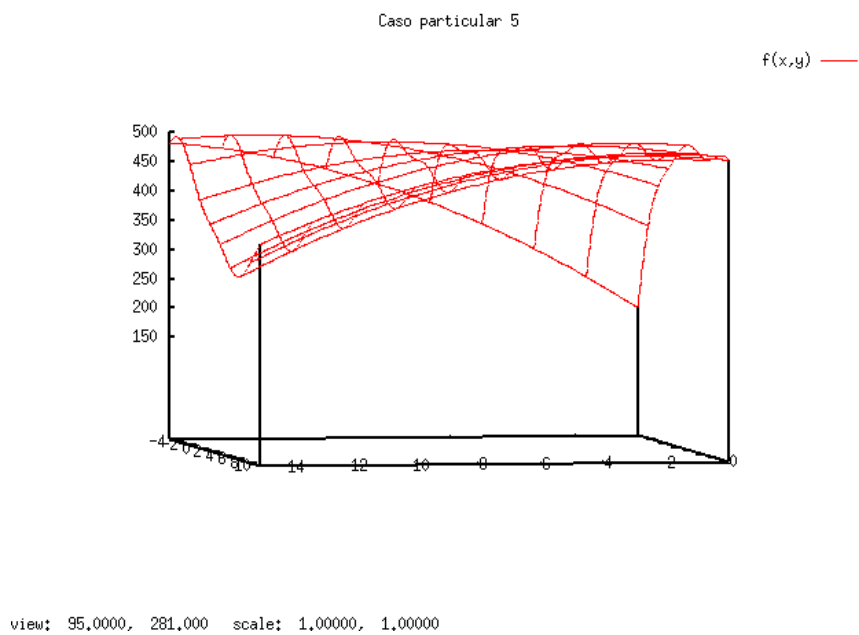
$$a = 1 \quad b = \frac{5.1}{4\pi^2} \quad c = \frac{5}{\pi} \quad d = 6 \quad e = 10 \quad f = \frac{1}{8\pi}$$

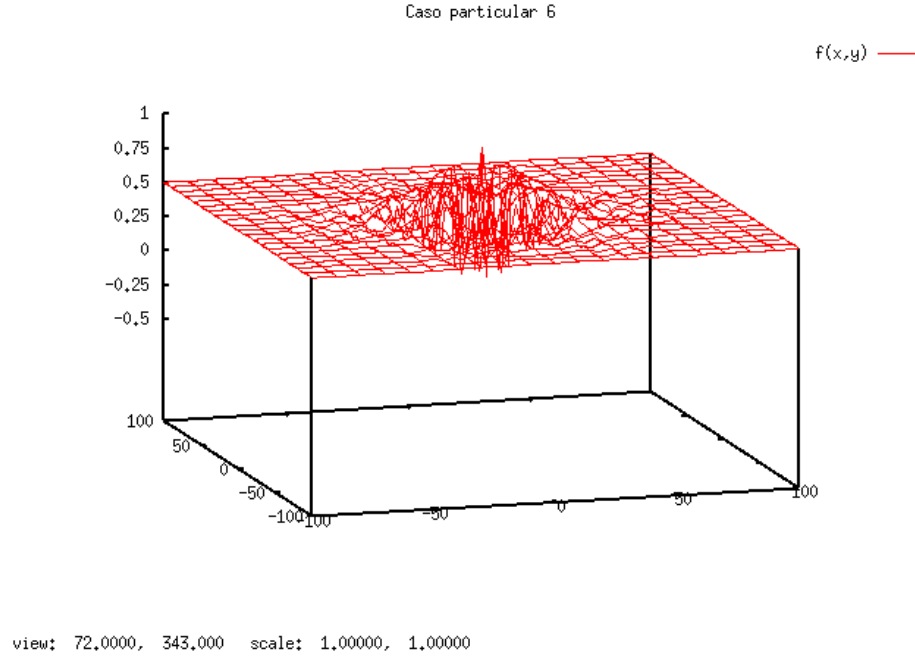
$$S = \{-5 \leq x_1 \leq 10, \quad 0 \leq x_2 \leq 15\}$$

Se representa la función $f(x_1, x_2)$ en S en las figuras 4.4 y 4.5, desde diferentes puntos de vista.

En estas dos gráficas se puede observar que existen diversos puntos que hacen máxima la función. Estos puntos hacen que la función tome valores cercanos a 500. Se pueden identificar éstos fácilmente en las gráficas, como por ejemplo los siguientes:

- $x_1 = -3.00$ y $x_2 = 11.00$
- $x_1 = 3.00$ y $x_2 = 1.15$
- $x_1 = 3.00$ y $x_2 = 0.85$

Figura 4.4: *Caso Particular 5 (vista 1)*Figura 4.5: *Caso Particular 5 (vista 2)*

Figura 4.6: *Caso Particular 6*

La herramienta SIGENOF, tras una ejecución, obtuvo las siguientes soluciones:

- $x_1 = -3.089$ y $x_2 = 10.940$ con $f = 499.58$.
- $x_1 = 3.026$ y $x_2 = 1.148$ con $f = 499.536$.
- $x_1 = 2.889$ y $x_2 = 1.152$ con $f = 499.230$.
- $x_1 = 2.975$ y $x_2 = 0.832$ con $f = 499.273$.

4.6. Caso particular 6

La función objetivo a optimizar se describe como:

$$f(x_1, x_2) = 0.5 - \frac{\sin^2 \sqrt{x_1^2 + x_2^2} - 0.5}{(1 + 0.001 \cdot (x_1^2 + x_2^2))^2}$$

$$S = \{x_1, x_2 \in [-100, 100]\}$$

Se representa la función $f(x_1, x_2)$ en S en la figura 4.6.

Visualmente se puede comprobar que el máximo de esta función se encuentra en el punto $x_1 = x_2 = 0$, con un valor para f de 1.0 .

En este caso SIGENOF también coincidió con la solución real obteniendo puntos próximos a cero con un valor para f igual o muy cercano a 1.0 , como se puede observar en las siguientes soluciones obtenidas por la herramienta:

- $x_1 = -0.009$ y $x_2 = -0.002$ con $f = 1.000$.
- $x_1 = 0.006$ y $x_2 = -0.002$ con $f = 1.000$.
- $x_1 = 0.054$ y $x_2 = 0.003$ con $f = 0.997$.

4.7. Ejecución con variantes

El caso particular 2 expuesto anteriormente representa un ejemplo de maximización de una función multivariable:

$$f(x_1, x_2, x_3, x_4, x_5, x_6) = 100 - (x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2)$$

$$S = \left\{ \begin{array}{l} x_1 \in [-3.0, 5.1], x_2 \in [2.1, 7.8], x_3 \in [-10.1, 20.3], \\ x_4 \in [-3.3, 4.2], x_5 \in [-15.3, 70.1], x_6 \in [-0.25, 0.35] \end{array} \right\}$$

Aunque la solución que la hace máxima se obtiene con facilidad con una simple observación de la definición de la función, localizada en el punto $(0, 0, 0, 0, 0, 0)$, con un valor para f equivalente a 100 , en SIGENOF se necesitó realizar una ligera variación de los parámetros que se aplicaban al resto de los casos particulares para que alcanzara de forma eficiente la solución deseada. Por ello, tal y como se expuso anteriormente, en este caso particular la probabilidad de mutación se varió desde un valor de hasta un valor $0,5\%$ ($0,005$) hasta un valor de $2,5\%$ ($0,025$). Esto no quiere decir que la herramienta no fuera capaz de encontrar el máximo para este caso particular con los parámetros establecidos por defecto en el cuadro 4.1, ya que en la mayoría de las ejecuciones siempre había eras en las que se localizaba éste con bastante precisión, aunque en algunas otras eras no se acercaba lo suficiente a la solución real. Lo que ocurría era que las variaciones entre sucesivas generaciones eran extremadamente grandes, obteniendo divergencias tales como valores para $f = -4791.560$ así como para $f = 97.900$, es decir valores con una desviación estándar enorme, a pesar de que la desviación estándar entre sucesivas eras para una ejecución determinada no resultaban ser tan marcadas.

Debido a estas observaciones se realizó en primer lugar una ejecución con los parámetros definidos en el cuadro 4.1 para posteriormente realizar variaciones en algunos de los parámetros de ejecución del AG, y de esta forma observar cómo se comportaba SIGENOF para este caso particular. Se detalla a continuación el caso inicial y siete variaciones realizadas a éste, así como los resultados obtenidos para cada una ellas en la primera era.

4.7.1. Sin variación de parámetros (Ejecución 1)

En este caso se obtuvo en la primera era un valor máximo para $f = 97.900$ en el punto $(-1.313 \ 0.005 \ 0.612 \ -0.002 \ 0.007 \ 0.037)$. Los parámetros empleados en esta ejecución fueron los definidos en el cuadro 4.1, es decir:

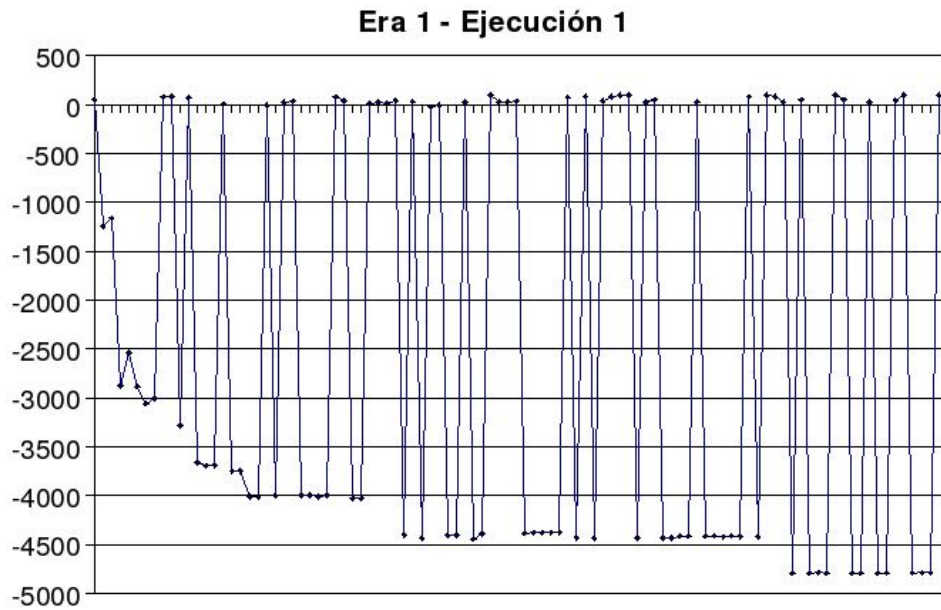
- *Nº de Generaciones: 100*
- *Nº de Individuos de la Población: 100*
- *Probabilidad de Cruce: 0,8*
- *Probabilidad de Mutación: 0,005*
- *Elitismo: Sí*

Se puede observar la divergencia entre las distintas soluciones obtenidas por SIGENOF para la primera era en la figura 4.7. En este y en todos los siguientes gráficos, el eje horizontal representa las n generaciones de la ejecución, mientras que el eje vertical representa los valores de f de cada mejor solución en cada generación.

4.7.2. Variación de la probabilidad de mutación (Ejecución 2)

Se varió únicamente en este caso la probabilidad de mutación a un valor de $0,015$. De esta manera la ejecución se realizó con los siguientes parámetros:

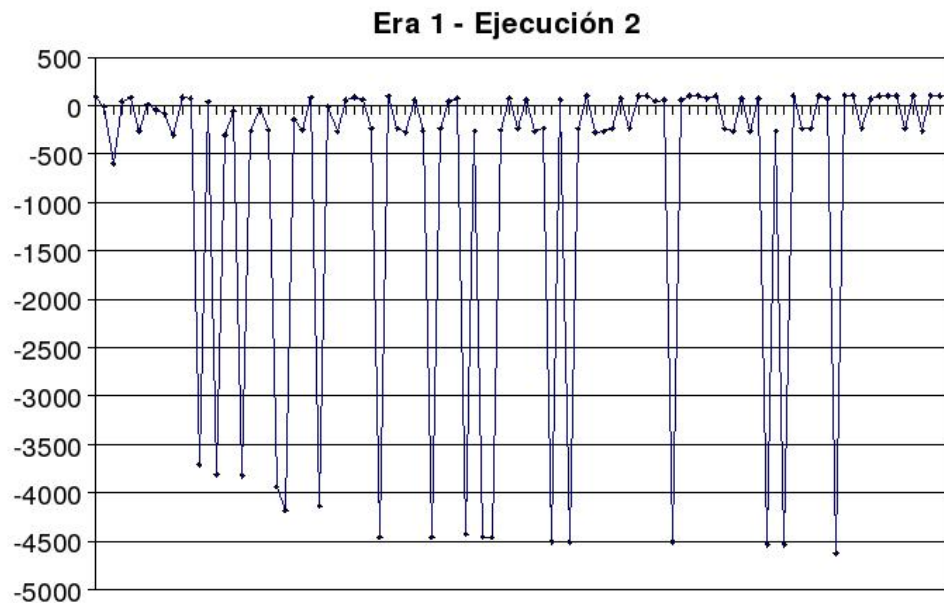
- *Nº de Generaciones: 100*
- *Nº de Individuos de la Población: 100*
- *Probabilidad de Cruce: 0,8*

Figura 4.7: *Caso inicial - Generación 1*

- *Probabilidad de Mutación: 0,015*
- *Elitismo: SÍ*

El resultado obtenido por SIGENOF en la primera era tras esta variación fue $f = 100$ en el punto $(0.004 \ 0.003 \ 0.003 \ -0.001 \ 0.000 \ 0.000)$.

Se puede observar en la figura 4.8 cómo las soluciones obtenidas por SIGENOF para la primera era ya no divergían tanto como ocurría en el caso inicial, a pesar de que había unas pocas soluciones con valores negativos para f muy alejados, la mayoría de las soluciones se encontraban en una región cercana a la solución real. Por tanto se puede afirmar que una probabilidad de mutación más elevada producía, para este problema, una mejora en el rendimiento del AG ya que esta mayor probabilidad permitía realizar cambios en un número mayor de soluciones que en el caso inicial, dando lugar a que algunas de las que se encontraban alejadas de la solución real tuvieran una mayor probabilidad de ser mutadas, y con más posibilidades de convertirse en soluciones adaptadas al problema.

Figura 4.8: *Variación probabilidad de mutación*

4.7.3. Variación de las probabilidades de cruce y mutación (Ejecución 3)

Se variaron en este caso las probabilidades de cruce y de mutación a valores de $0,2$ y $0,015$ respectivamente, es decir se disminuyó la probabilidad de cruce hasta un 20% , y la probabilidad de mutación se estableció, al igual que en el caso anterior, a un valor de $1,5\%$. De esta manera la ejecución se realizó con los siguientes parámetros:

- *Nº de Generaciones: 100*
- *Nº de Individuos de la Población: 100*
- *Probabilidad de Cruce: 0,2*
- *Probabilidad de Mutación: 0,015*
- *Elitismo: Sí*

El resultado obtenido por SIGENOF en la primera era tras esta variación fue similar al obtenido en la ejecución anterior, $f = 100$ en el punto $(0.000\ 0.003\ -0.001\ -0.001\ 0.000\ 0.000)$.



Figura 4.9: *Variación probabilidades de cruce y mutación*

Se puede observar en la figura 4.9 cómo las soluciones obtenidas por SIGENOF para la primera era tampoco divergieron tanto como ocurrió en el caso inicial, a pesar de que todavía existían soluciones con valores para f muy alejados, la mayoría de las soluciones se encontraban en una región mucho más cercana a la solución real que en la ejecución anterior. A la vista del gráfico, se deduce que aunque la probabilidad de mutación en este caso era acertada, la probabilidad de cruce más adecuada para obtener una convergencia rápida de las soluciones era la anterior, es decir valores próximos a un 80% , ya que esta probabilidad fomentaba el cruce entre distintas soluciones que pudieran estar muy alejadas de la solución real y que podían dar lugar a soluciones más adaptadas al problema.

4.7.4. Variación del tamaño de la población a un valor pequeño (Ejecución 4)

La única variación que se realizó en este caso fue la del tamaño de la población, variándose desde un valor inicial establecido en 100 individuos a una población pequeña de 25 individuos. El resto de los parámetros se mantuvieron tal y como se definieron inicialmente en el cuadro 4.1, es decir:

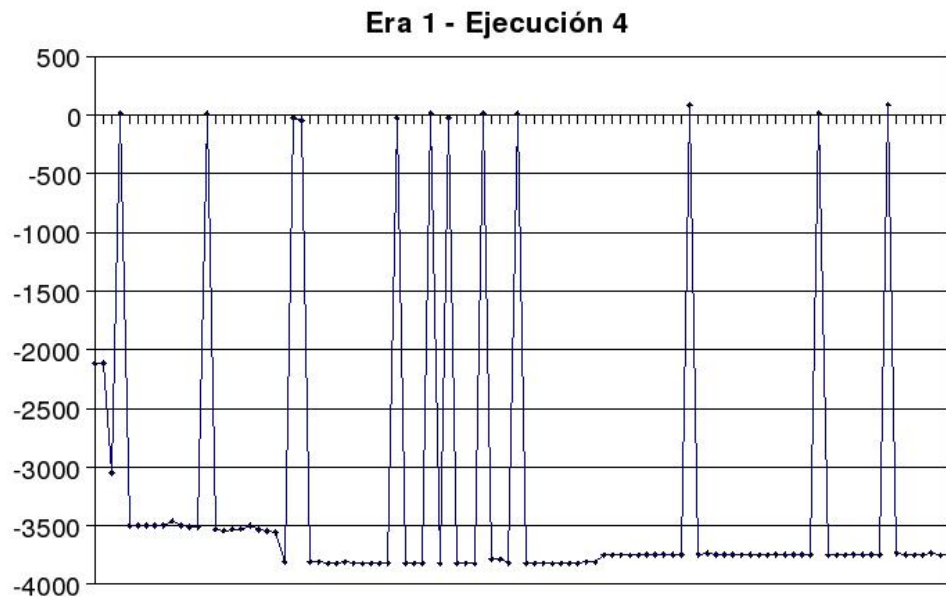


Figura 4.10: Variación a un tamaño de la población pequeño

- *Nº de Generaciones: 100*
- *Nº de Individuos de la Población: 25*
- *Probabilidad de Cruce: 0,8*
- *Probabilidad de Mutación: 0,005*
- *Elitismo: SÍ*

El resultado obtenido por SIGENOF en la primera era tras esta variación fue $f = 85.779$ en el punto $(1.255 \ 0.003 \ 0.017 \ 3.556 \ 0.020 \ 0.000)$.

Se puede observar en la figura 4.10 cómo en este caso la mayoría de las soluciones obtenidas por SIGENOF para la primera era se encontraban alejadas de la solución real. A pesar de ello algunas pocas soluciones consiguieron alcanzar un valor aceptable para el problema, en este caso un valor para $f = 85.779$, aunque en diversas ejecuciones los resultados obtenidos no fueron tan buenos, es decir tan cercanos a la solución real. Se deduce, por tanto, en este caso que el tamaño de la población era demasiado pequeño para que el AG consiguiera soluciones adaptadas a este problema concreto.

4.7.5. Variación del n° de generaciones a un tamaño pequeño (Ejecución 5)

En este caso se varió únicamente el número de generaciones para cada era estableciéndose un valor de pequeño para este parámetro, 25 generaciones. El resto de los valores se dejaron como inicialmente se establecieron. De esta manera la ejecución se realizó con los siguientes parámetros:

- *N° de Generaciones: 25*
- *N° de Individuos de la Población: 100*
- *Probabilidad de Cruce: 0,8*
- *Probabilidad de Mutación: 0,005*
- *Elitismo: SÍ*

El resultado obtenido por SIGENOF en la primera era tras esta variación no fue muy alto con un valor de $f = 71.935$ en el punto $(-1.775 \ 4.639 \ 0.832 \ -1.608 \ 0.037 \ 0.339)$.

Se puede observar en la figura 4.11 cómo la mayoría de las soluciones obtenidas por SIGENOF en la primera era se alejaron bastante de la solución real, encontrándose tan sólo unas pocas que ligeramente se adaptaban al problema. Esto fue debido a que el número de generaciones era muy pequeño y, por tanto, si las soluciones generadas inicialmente aleatoriamente no eran buenas el AG no tenía la posibilidad de mejorarlas en un número tan reducido de generaciones, es decir en sucesivas etapas de *selección-cruce-reproducción-mutación*.

4.7.6. Variación del n° de generaciones a un tamaño grande (Ejecución 6)

En este caso se varió número de generaciones para cada era estableciéndose un valor grande para este parámetro, 400 generaciones. El resto de los valores se dejaron como inicialmente se establecieron. De esta manera la ejecución se realizó con los siguientes parámetros:

- *N° de Generaciones: 400*

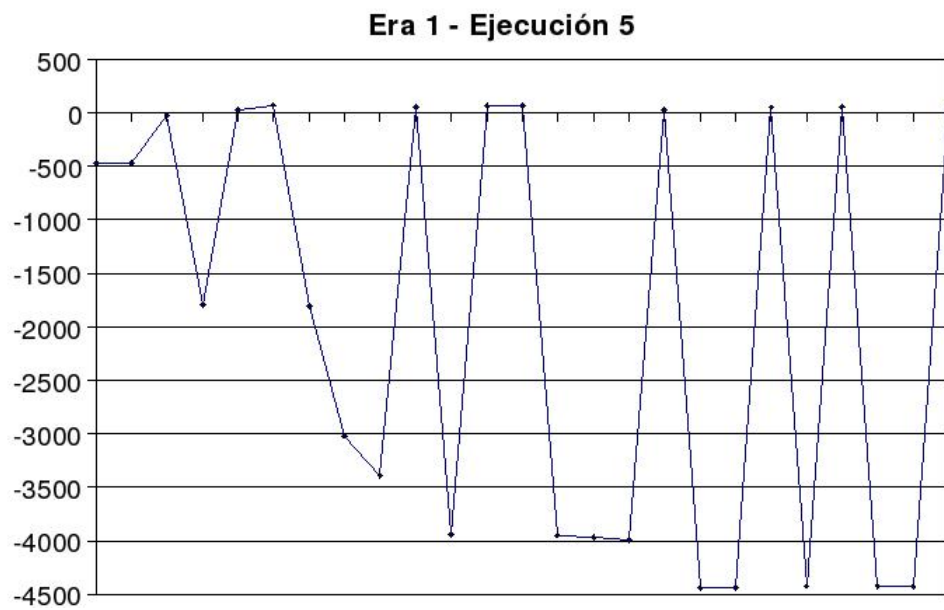


Figura 4.11: Variación a un n° de generaciones pequeño

- N° de Individuos de la Población: 100
- Probabilidad de Cruce: 0,8
- Probabilidad de Mutación: 0,005
- Elitismo: Sí

El resultado obtenido por SIGENOF en la primera era tras esta variación coincidió con la solución real con un valor para $f = 100$ en el punto $(0.003 \ 0.002 \ 0.001 \ 0.000 \ 0.004 \ 0.000)$.

Se puede observar en la figura 4.12 cómo la mayoría de las soluciones obtenidas por SIGENOF en la primera era se alejaban bastante de la solución real, encontrándose tan sólo unas pocas que se adaptaban al problema, y aún así el AG fue capaz de encontrar la solución real. Esto fue debido a que el número de generaciones era bastante grande y, por tanto, como existieron muchas etapas de *selección-cruce-reproducción-mutación* también existieron muchas posibilidades de alcanzar la solución correcta.

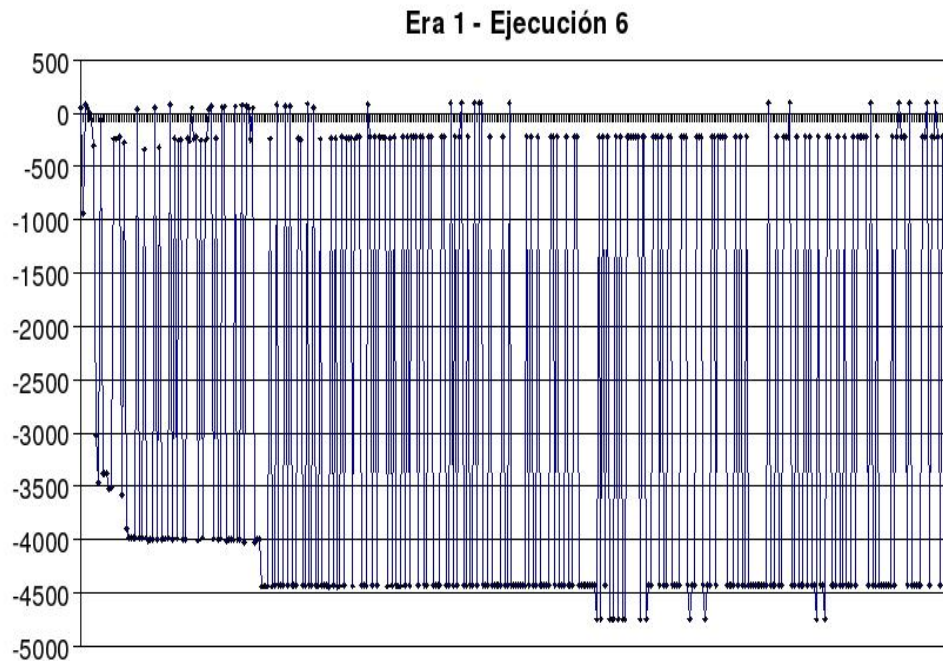


Figura 4.12: Variación a un n° de generaciones grande

4.7.7. Variación del tamaño de la población a un valor grande (Ejecución 7)

En este caso se varió únicamente el número de individuos de la población, estableciéndose un tamaño grande para este parámetro, 400 individuos. El resto de los valores se dejaron como inicialmente se establecieron. De esta manera la ejecución se realizó con los siguientes parámetros:

- *Nº de Generaciones: 100*
- *Nº de Individuos de la Población: 400*
- *Probabilidad de Cruce: 0,8*
- *Probabilidad de Mutación: 0,005*
- *Elitismo: SÍ*

El resultado obtenido por SIGENOF en la primera era tras esta variación se acercó bastante a la solución real con un valor para $f = 98.144$ en el punto $(0.001 \ 0.002 \ 0.012 \ 1.361 \ 0.030 \ -0.049)$.

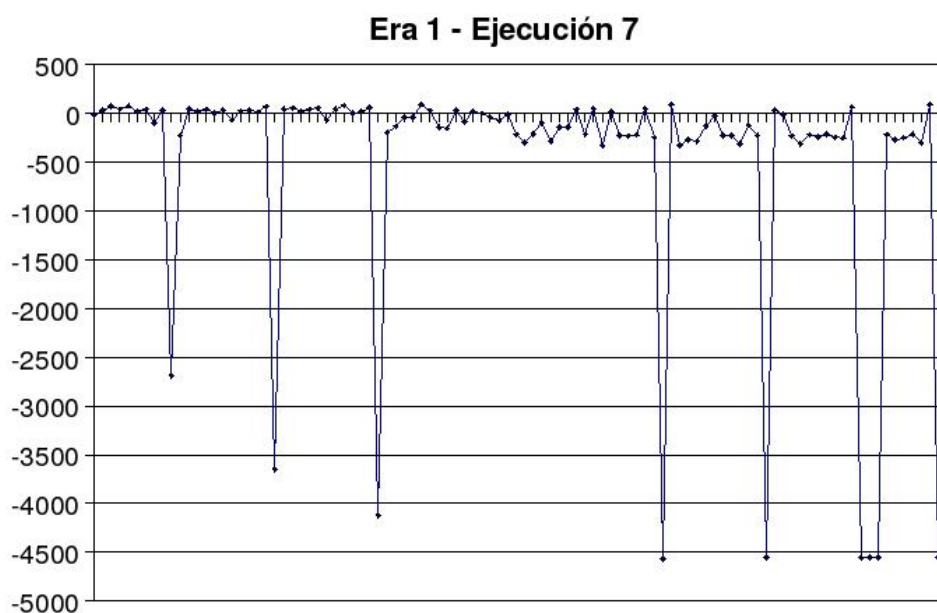


Figura 4.13: Variación a un tamaño de la población grande

Se puede observar en la figura 4.13 cómo la mayoría de las soluciones obtenidas por SIGENOF en la primera era se encontraban en un rango de valores más o menos cercano a la solución real, encontrándose tan sólo unas pocas que se alejaban mucho de la solución al problema. Esto fue debido a que como el número de individuos de que constaba la población era bastante grande, y por tanto existía una gran variación genética, existía más probabilidad de intercambio de información entre los individuos más adaptados al problema en las etapas de *selección-cruce-reproducción*.

4.7.8. Variación de la probabilidad de mutación y del tamaño de la población a un valor grande (Ejecución 8)

En este caso se variaron dos parámetros, el número de individuos de la población, estableciéndose de la misma forma que en el caso anterior un tamaño grande para este parámetro con 400 individuos, y la probabilidad de mutación, con una probabilidad de 1,5 % (0,015). Se eligieron los dos parámetros y valores mencionados porque en las anteriores variaciones se observó que la herramienta se comportaba relativamente bien con estos valores. El resto se dejaron como inicialmente fueron establecidos. De esta manera la ejecución se realizó con los siguientes parámetros:

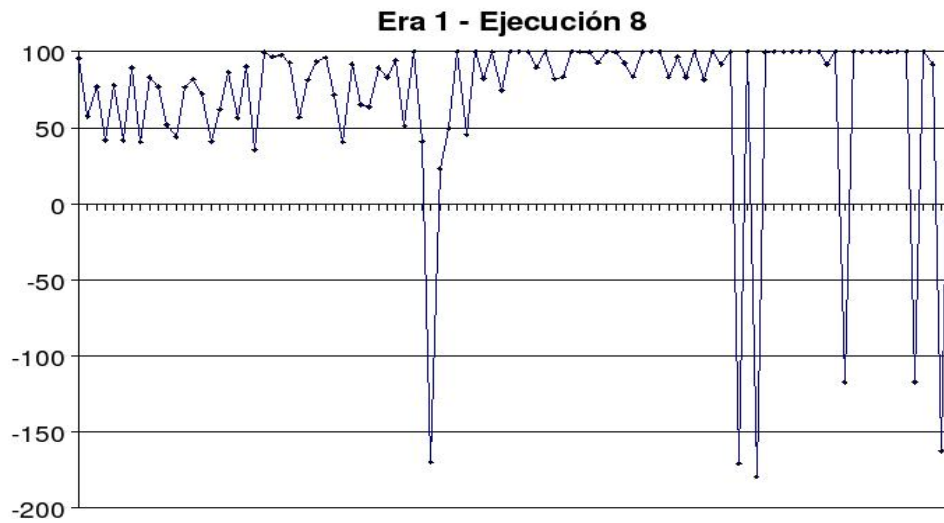


Figura 4.14: Variación tamaño población y probabilidad mutación a valores grandes

- *Nº de Generaciones: 400*
- *Nº de Individuos de la Población: 400*
- *Probabilidad de Cruce: 0,8*
- *Probabilidad de Mutación: 0,015*
- *Elitismo: SÍ*

El resultado obtenido por SIGENOF en la primera era tras esta variación coincidió con la solución real con un valor para $f = 100$ en el punto $(-0.002 \ 0.005 \ 0.000 \ -0.002 \ -0.002 \ 0.000)$. Otras ejecuciones también dieron resultados muy similares a esta primera era.

Se puede observar en la figura 4.14 cómo la mayoría de las soluciones obtenidas por SIGENOF en la primera era se acercaban a la solución real, empezando con valores algo más alejados a ésta pero que según iban avanzando las sucesivas generaciones se acercaban cada vez más al valor 100 . Unas pocas soluciones se alejaron bastante de la solución debido probablemente a mutaciones aleatorias. No obstante, se comprobó que el AG convergía con facilidad a la solución con estas variaciones.

Capítulo 5

Conclusiones

La herramienta implementada en este proyecto fin de carrera, SIGENOF, no pretende ser en ningún momento una herramienta universal para la optimización de funciones matemáticas, sino más bien una aplicación práctica de una de las ramas de la Computación Evolutiva, los Algoritmos Genéticos. A través de SIGENOF se ha demostrado cómo un sencillo algoritmo que simula los mecanismos básicos de la evolución biológica es capaz, en la mayoría de los casos, de obtener soluciones aceptables a un problema de optimización dado a través de una sencilla formulación matemática. Es cierto que en ocasiones, y dependiendo del problema que se esté resolviendo, se deberán ajustar algunos parámetros del AG, tales como el tamaño de la población, el número de generaciones o las probabilidades de selección y mutación, pero tras estos pequeños ajustes el AG probablemente será capaz de encontrar una *buena solución* al problema a resolver.

Cuando se habla de una *buena solución* nos referimos a una solución muy cercana a la solución real que se busca, es decir es posible que el AG no localice la solución exacta al 100 %, sin embargo probablemente sí será capaz de localizarla en un entorno muy cercano, y en algunos casos incluso se llegará al 100 % de precisión. Es evidente por tanto que la aplicación de esta herramienta, y en general la de los AG, se circunscribe a aquellos problemas en los que una solución cercana puede ser tan válida como la solución real. Al mismo tiempo estas soluciones, generalmente muy próximas al máximo global, servirán como magníficas soluciones iniciales en los métodos tradicionales de optimización.

SIGENOF resuelve problemas de optimización basados en funciones matemáticas descritas con una sencilla sintaxis definida por la herramienta, donde el espacio de búsqueda puede ser muy amplio. Aparentemente no es mucho. Sin embargo el verdadero potencial de la herramienta es su interfaz gráfico de usuario (GUI), que permite realizar multitud de pruebas con diferentes funciones matemáticas y diferentes parámetros para el AG, de forma que el usuario puede, con un tiempo de aprendizaje muy pequeño, comprobar cómo se comporta un AG frente a diversas situaciones. Además, ya que los AG son una herramienta genérica para la resolución de problemas de optimización, SIGENOF podría utilizarse para resolver cualquier problema de optimización que pudiera formularse como una función matemática acotada en un espacio S .

Un aspecto que se podría mejorar en esta herramienta es el relativo al módulo que realiza el análisis léxico y sintáctico de la expresión matemática que se desea optimizar, es decir de la función J , de forma que este analizador fuera más robusto y completo, con posibilidad de ayudar al usuario a detectar errores de parentización, de introducción de operadores incorrectos, etc.

En lo relativo a futuras ampliaciones que la herramienta podría disponer, destacar las siguientes:

- Posibilidad de representar en tiempo real la función J a optimizar en su espacio S .
- Distintos tipos de representaciones gráficas de las mejores soluciones encontradas en cualquier generación/era una vez finalizado el proceso haciendo uso de alguna herramienta externa a la aplicación.
- Posibilidad de introducir restricciones más complejas a la función J .
- Paralelización del AG.

Para finalizar, destacar que el contenido íntegro de este Proyecto de Fin de Carrera, es decir la herramienta y la memoria, se publican bajo licencia GNU GPL (www.gnu.org), lo que permite la libre copia, modificación y distribución de éstos siempre y cuando se realice bajo los términos de esta misma licencia y no se vulnere ningún otro derecho o norma.

Apéndice A

Gramática para Funciones J en SIGENOF

La gramática que define la construcción de expresiones aritméticas válidas para la definición de las funciones J a optimizar se describe con la siguientes reglas:

```
<expr>:= <term><expr_prime>
```

```
<expr_prime>:= + <term><expr_prime>  
              | - <term><expr_prime>  
              | do nothing
```

```
<term>:= <factor><term_prime>
```

```
<term_prime>:= ^ <factor><term_prime>  
              | * <factor><term_prime>  
              | / <factor><term_prime>  
              | do nothing
```

```
<factor>:= ( <expr>)  
          | -( <expr>)
```

| <value>
| - <value>

<value>:= <functVal>
| <constnt>

<functVal>:= <functName>(<expr>)

<functName>:= sin | cos | tan | sqrt
| asin | acos | atan | exp
| ln | abs | pi | e
| log | hsin | hcos | htan

<constnt>:= integer
| floating point
| exponential number
| variables x[i]

Apéndice B

Función *ran2*

En este apéndice se muestra el listado fuente de la función *ran2*, extraída del texto "*Numerical Recipes in C*" [Recip88] y ligeramente modificada para incluir los extremos del rango de valores posibles que la función puede devolver. Esta función es usada durante la generación de números aleatorios para resolver el problema de la creación de individuos de la población inicial de forma verdaderamente aleatoria, como se describe en el paso de Inicialización del AG, así como para la obtención de cualquier número aleatorio empleado por el AG, tal y como se realiza en los pasos de Selección, Cruce en un solo punto y Mutación uniforme.

B.1. Fichero aleatorio.h

```
/* GENERACIÓN DE NÚMEROS ALEATORIOS -----  
----- */  
  
#define IM1 2147483563  
#define IM2 2147483399  
#define AM (1.0/IM1)  
#define IMM1 (IM1-1)  
#define IA1 40014  
#define IA2 40692  
#define IQ1 53668  
#define IQ2 52774  
#define IR1 12211  
#define IR2 3791  
#define NTAB 32  
#define NDIV (1+IMM1/NTAB)  
#define EPS 0  
#define RNMX (1.0-EPS)
```

```

/*
Prototipos de las funciones que generan números pseudoaleatorios
*/
float ran2(long *);
float aleatorio(float, float);

```

B.2. Fichero aleatorio.c

```

/* GENERACIÓN DE NÚMEROS ALEATORIOS -----
----- */

#include <stdlib.h>
#include <time.h>
#include "aleatorio.h"

static long idum = -1;

/*
Código extraído del texto "Numerical Recipes in C".
Long period (>2 x 1018) random number generator of L'Ecuyer with Bays-Durham shuffle
and added safeguards. Returns a uniform random deviate between 0.0 and 1.0.
Call with idum a negative integer to initialize; thereafter, do not alter
idum between successive deviates in a sequence. RNMX should approximate the largest floating
value that is less than 1.
*/
float ran2(long *idum)
{
    int j;
    long k;
    static long idum2 = 123456789;
    static long iy = 0;
    static long iv[NTAB];
    float temp;

    if (*idum <= 0)
    {
        // Initialize.
        if ((*idum) <= 0) *idum = time(NULL) % 32000; // use clock: Generación de semilla
        else *idum = -(*idum);

        idum2 = (*idum);

        for (j = NTAB+7; j >= 0; j--)
        {
            // Load the shuffle table (after 8 warm-ups).
            k=(*idum) / IQ1;
            *idum = IA1 * (*idum - k * IQ1) - k * IR1;

            if (*idum < 0) *idum += IM1;
            if (j < NTAB) iv[j] = *idum;
        }
    }
}

```

```
    }

    iy = iv[0];
}

k = (*idum) / IQ1;                                // Start here when not initializing.

*idum = IA1 * (*idum - k*IQ1) - k * IR1; // Compute idum=(IA1*idum)% IM1 without
                                         // overflows by Schrage's method.

if (*idum < 0) *idum += IM1;
k = idum2 / IQ2;
idum2 = IA2 * (idum2 - k * IQ2) - k * IR2; // Compute idum2=(IA2*idum)% IM2 likewise.

if (idum2 < 0) idum2 += IM2;
j = iy / NDIV;                                // Will be in the range 0..NTAB-1.
iy = iv[j] - idum2;                            // Here idum is shuffled, idum and idum2 are
                                         // combined to generate output. iv[j] = *idum;

if (iy < 1) iy += IMM1;
if ((temp = AM * iy) > RNMX) return RNMX; // Because users don't expect endpoint values.
else return temp;
}

/*
Generación de números pseudoaleatorios entre min y max
*/
float aleatorio(float min, float max)
{
    return min + ran2(&idum) * (max - min);
}
```


Apéndice C

Código del AG de SIGENOF

C.1. Fichero ag.h

```
/*
*****
* Copyright (C) 2005 by Luis Marco Giménez - UNED 2004/2005
* luigimaloni@hotmail.com
*
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program; if not, write to the Free Software Foundation,
* Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*****
*/

/*
Definición de la función J a optimizar
*/
typedef struct DEF_FUNCION
{
    char *funcion;
    int num_parametros;
    float *valor_min_parametro;
    float *valor_max_parametro;
}
```

```

Tipo_funcion;

/*
El cromosoma se define como un vector de parámetros xi de la funcion J a
optimizar. Se realiza una sustitución en los parámetros de la función,
de forma que las diferentes variables x, y, z, ... toman los valores
x[0], x[1], x[2], ...
*/
typedef struct CROMOSOMA
{
    long *x; // Vector de parámetros de la función J
    float fitness; // Fitness del resultado de la evaluación de la función J
    float p; // Probabilidad de selección p para cada cromosoma
    float q; // Probabilidad acumulada q para cada cromosoma
    int seleccionado; // Cromosoma seleccionado para cruce
}
Tipo_cromosoma;

/*
Era en la que comenzará a correr el AG y que será útil para continuar tras pulsar_grafico
*/
static int era_inicial = 1;

/*
Prototipos de funciones usadas por el AG
*/
void copiar_cromosoma (Tipo_cromosoma *, Tipo_cromosoma *);
void establecer_J (void);
void inicializar_cromosomas (void);
void generar_poblacion_aleatoria (void);
void seleccion_cromosomas (void);
void cruce_cromosomas (void);
void mutacion_uniforme (void);
void evaluacion_poblacion (void);
void elitismo (void);
void copiar_genes (Tipo_cromosoma *, int, Tipo_cromosoma *, int, int);
void init_ag (int, int, int, float, float, int, float, int);
void cancel_ag (void);
float *precision_decimal (long *);
void ver_cromosomas (Tipo_cromosoma *);
float *reservar_memoria_parametros (int);
void gui_mejor_solucion (int);
void trata_linea_fun (char *);
float calculo_desviacion_estandar(void);
void bucle_ag(void);
int condicion_epsilon(void);

```

C.2. Fichero ag.c

```

/*****

* Copyright (C) 2005 by Luis Marco Giménez - UNED 2004/2005
* luigimaloni@hotmail.com
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program; if not, write to the Free Software Foundation,
* Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
*****/

/* PROGRAMA PRINCIPAL: Algoritmo Genético -----
#####
S I G E N O F: Sistema de Implementación Genética para la Optimización de Funciones
#####
----- */

#ifdef HAVE_CONFIG_H
    #include <config.h>
#endif

/*
Directivas include estándar
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <float.h>
#include <string.h>

/*
Directivas include específicas del problema
*/

#include "aleatorio.h"
#include "pfc_ag.h"
#include "ag.h"
#include "eval.h"
#include "fichero.h"

/*
Parámetros del algoritmo genético
*/

```

```

int MAXGENS; // Nm. de Generaciones
int POP_SIZE; // Tamaño de la población
int ERAS; // Nm. de Eras
float pc; // Probabilidad de cruce [0,1]
float pm; // Probabilidad de mutación [0,1]
float *x_dec = NULL; // Codificación de los genes en formato decimal
int PRECISION = 3; // Precisión en decimales para la codificación de soluciones
float EPSILON; // Criterio de terminación adicional tal que:  $|J(k+1) - J(k)| \leq \text{EPSILON}$ 
int ELITISMO; // Aplicación de mecanismos de Elitismo en el AG

/*
Cromosomas
*/
Tipo_cromosoma *cromosoma = NULL;
Tipo_cromosoma *nueva_poblacion = NULL;
Tipo_cromosoma mejor_cromosoma;
Tipo_cromosoma cromosoma_aux;

/*
Variables y contadores del algoritmo genético
*/
int generacion;
int era;
int nGenes;
char *str_gui = NULL;
char *str_aux = NULL;
int cancelado = 0;
float *lista_mejores_soluciones_eras = NULL;
float mejor_solucion_gen = -FLT_MAX;
float mejor_solucion_gen_ant = -FLT_MAX;
float mejor_solucion_eras = -FLT_MAX;
float suma_mejor_solucion_eras = 0;
int epsilon_alcanzado = FALSE;

/*
Datos estadísticos del AG
*/
float maximo_J = 0;
float media_J = 0;
float desviacion_J = 0;

/*
Definición de la función J y de los cromosomas poblacionales necesarios
*/
Tipo_funcion J;

/*
Establece la función J a optimizar
*/
void establecer_J()
{
    // Define la función J definida por el usuario

```



```

    if (J.funcion) free(J.funcion);
    J.funcion = malloc( sizeof(char) * strlen(get_user_function()) );
    strcpy(J.funcion, get_user_function());
    J.num_parametros = contar_variables_x(J.funcion);
    if (J.valor_min_parametro) free(J.valor_min_parametro);
    if (J.valor_max_parametro) free(J.valor_max_parametro);
    J.valor_min_parametro = reservar_memoria_parametros(J.num_parametros);
    J.valor_max_parametro = reservar_memoria_parametros(J.num_parametros);
    /*
    Salida del AG a fichero
    */
    char str_fich[350];
    abrir_fichero("./salida_ag", FICHERO_SALIDA);
    abrir_fichero("./soluciones_ag", FICHERO_SOLUCIONES);
    escribir_fichero("1. FUNCIÓN J A OPTIMIZAR:\n\n", FICHERO_SALIDA);
    sprintf(str_fich, "\tJ = %s\n", J.funcion);
    escribir_fichero(str_fich, FICHERO_SALIDA);
    escribir_fichero("\tcon,\n", FICHERO_SALIDA);
    int i;
    for (i = 0; i < J.num_parametros; i++)
    {
        J.valor_min_parametro[i] = get_max_min_x(i, 0);
        J.valor_max_parametro[i] = get_max_min_x(i, 1);
        // Salida del AG a fichero
        sprintf(str_fich, "\t\tx[%i] entre [%5.2f,%5.2f]\n", i, J.valor_max_parametro[i],
            J.valor_min_parametro[i]);
        escribir_fichero(str_fich, FICHERO_SALIDA);
    }
    // Salida del AG a fichero
    escribir_fichero("\n\n", FICHERO_SALIDA);
    escribir_fichero("2. PARÁMETROS DEL AG:\n\n", FICHERO_SALIDA);
    sprintf(str_fich, "\t* N° de ERAS:%i\n", ERAS);
    escribir_fichero(str_fich, FICHERO_SALIDA);
    sprintf(str_fich, "\t* N° de GENERACIONES:%i\n", MAXGENS);
    escribir_fichero(str_fich, FICHERO_SALIDA);
    sprintf(str_fich, "\t* INDIVIDUOS de la Población:%i\n", POP_SIZE);
    escribir_fichero(str_fich, FICHERO_SALIDA);
    sprintf(str_fich, "\t* Probabilidad de Cruce:%0.4f\n", pc);
    escribir_fichero(str_fich, FICHERO_SALIDA);
    sprintf(str_fich, "\t* Probabilidad de Mutación:%0.4f\n", pm);
    escribir_fichero(str_fich, FICHERO_SALIDA);
    sprintf(str_fich, "\t* Precisión decimal:%i\n", PRECISION);
    escribir_fichero(str_fich, FICHERO_SALIDA);
    sprintf(str_fich, "\t* Condición adicional (epsilon):%f\n", EPSILON);
    escribir_fichero(str_fich, FICHERO_SALIDA);
    if (ELITISMO)
    {
        sprintf(str_fich, "\t* ELITISMO: SI\n\n\n");
    }

```

```

    }
    else
    {
        sprintf(str_fich, "\t* ELITISMO: NO\n\n");
    }
    escribir_fichero(str_fich, FICHERO_SALIDA);
    escribir_fichero("3. MEJORES SOLUCIONES EN CADA ERA:\n\n", FICHERO_SALIDA);
    // Número de genes por cromosoma
    nGenes = J.num_parametros;
}

/*
Se reserva memoria para los rangos de valores de cada uno de los parámetros de la función J
*/
float *reservar_memoria_parametros(int n)
{
    return (float *) calloc(n, sizeof(float));
}

/*
Inicializa la población aleatoriamente
*/
void inicializar_cromosomas()
{
    // Se reserva memoria para la población de cromosomas y las sucesivas nuevas poblaciones
    if (cromosoma) free(cromosoma);
    if (nueva_poblacion) free(nueva_poblacion);
    cromosoma = malloc(sizeof(Tipo_cromosoma) * POP_SIZE);
    nueva_poblacion = malloc(sizeof(Tipo_cromosoma) * POP_SIZE);
    // Se reserva memoria para guardar el MEJOR cromosoma de la población futura y se inicializa
    if (mejor_cromosoma.x) free(mejor_cromosoma.x);
    mejor_cromosoma.x = (long *) calloc(nGenes, sizeof(long));
    mejor_cromosoma.fitness = -FLT_MAX;
    mejor_cromosoma.p = mejor_cromosoma.q = mejor_cromosoma.seleccionado = 0;
    int i;
    for (i = 0; i < POP_SIZE; i++)
    {
        // Se reserva memoria para cada cromosoma poblacional
        cromosoma[i].x = (long *) calloc(nGenes, sizeof(long));
        nueva_poblacion[i].fitness = -FLT_MAX;
        cromosoma[i].p = cromosoma[i].q = cromosoma[i].seleccionado = 0;
        // Se reserva memoria para cada cromosoma de la nueva población futura y se inicializan
        nueva_poblacion[i].x = (long *) calloc(nGenes, sizeof(long));
        nueva_poblacion[i].fitness = -FLT_MAX;
        nueva_poblacion[i].p = nueva_poblacion[i].q = nueva_poblacion[i].seleccionado = 0;
    }
    // Se reserva memoria para un cromosoma auxiliar utilizado en algunas operaciones genéticas
    cromosoma_aux.x = (long *) calloc(nGenes, sizeof(long));

```

```

// Se reserva memoria para la cadena de texto de las salidas impresas del AG
if (str_gui) free(str_gui);
if (str_aux) free(str_aux);
str_gui = malloc(sizeof(char) * 350);
str_aux = malloc(sizeof(char) * 350);
// Se reserva memoria para la codificación decimal de los genes de cada cromosoma;
if (x_dec) free(x_dec);
x_dec = (float *) calloc(nGenes, sizeof(float));
// Se inicializa la mejor solución entre eras, sus contadores y estadísticos
mejor_solucion_eras = -FLT_MAX;
suma_mejor_solucion_eras = 0;
maximo_J = media_J = desviacion_J = 0;
// Se reserva memoria para la lista de las mejores soluciones de las eras
if (lista_mejores_soluciones_eras) free(lista_mejores_soluciones_eras);
lista_mejores_soluciones_eras = malloc(sizeof(float) * ERAS);
// Se inicializa la mejor solución de la generación actual y anterior
mejor_solucion_gen = -FLT_MAX;
mejor_solucion_gen_ant = -FLT_MAX;
}

/*
Genera POP_SIZES cromosomas poblacionales aleatoriamente, realiza la evaluación en
función de J para cada cromosoma y obtiene el mejor cromosoma poblacional
*/
void generar_poblacion_aleatoria()
{
    int i, j;
    float v;

    for (i = 0; i < POP_SIZE; i++)
    {
        // PASO 1: GENERACIÓN ALEATORIA de los Genes de cada cromosoma
        for (j = 0; j < nGenes; j++)
        {
            v = aleatorio(J.valor_min_parametro[j], J.valor_max_parametro[j]);
            // Redondeo a PRECISION cifras decimales
            cromosoma[i].x[j] = floor(v * pow(10, PRECISION));
        }
        // PASO 2: EVALUACIÓN del cromosoma
        cromosoma[i].fitness = eval(J.funcion, precision_decimal(cromosoma[i].x));
    }
}

/*
Selección de cromosomas mediante el método de la RULETA
*/
void seleccion_cromosomas()
{

```

```

int i, j;
float r, F = 0;
// 1) Se calcula la suma total de los F valores de la función de coste J
// asociada a cada cromosoma i
for (i = 0; i < POP_SIZE; i++)
{
    F += cromosoma[i].fitness;
}
// 2) Se calcula la probabilidad de selección p para cada cromosoma i
for (i = 0; i < POP_SIZE; i++)
{
    cromosoma[i].p = cromosoma[i].fitness / F;
}
// 3) Se calcula la probabilidad acumulada q para cada cromosoma i como la suma de
// las probabilidades q de los cromosomas anteriores.
for (i = 0; i < POP_SIZE; i++)
{
    for (j = 0; j <= i; j++)
    {
        cromosoma[i].q += cromosoma[j].p;
    }
}
// 4) Se gira la ruleta POP_SIZE veces.
for (j = 0; j < POP_SIZE; j++)
{
    // 4.1) Se genera un número real aleatorio r dentro del rango [0,1]
    r = aleatorio(0, 1);
    // 4.2) Recorremos los cromosomas de forma que si r < q(1) se selecciona
    // el cromosoma 1 (índice 0), en caso contrario se selecciona el cromosoma
    // i-ésimo (2 <= i <= POP_SIZE) tal que q(i-1) < r <= q(i)
    for (i = 0; i < POP_SIZE; i++)
    {
        if (r < cromosoma[i].q)
        {
            copiar_cromosoma(&nueva_poblacion[j], &cromosoma[i]);
            i = POP_SIZE + 1;
        }
    }
}
// Se copia la nueva_población al array cromosoma, como población actual
for (i = 0; i < POP_SIZE; i++)
{
    copiar_cromosoma(&cromosoma[i], &nueva_poblacion[i]);
}
}

/*
Cruce de dos cromosomas en un solo punto

```

```
*/
void cruce_cromosomas()
{
    int i, pos, seleccionados = 0;
    for (i = 0; i < POP_SIZE; i++)
    {
        // 1) Se genera un número aleatorio [0,1] para la selección aleatoria
        // de cromosomas para el cruce. Si r < pc se selecciona el cromosoma
        if (aleatorio(0,1) < pc)
        {
            cromosoma[i].seleccionado = 1;
            seleccionados++;
        }
    }
    // Si el número de cromosomas seleccionados es impar se opta por seleccionar,
    // aleatoriamente, otro cromosoma de la población
    if ((seleccionados % 2) != 0 && seleccionados < POP_SIZE)
    {
        do
        {
            pos = (int) floor(aleatorio(0, POP_SIZE));
        }
        while (cromosoma[pos].seleccionado);
        cromosoma[pos].seleccionado = 1;
        seleccionados++;
    }
    // 2) Se realiza la operación de CRUCE EN UN SOLO PUNTO entre parejas de
    // individuos seleccionados.
    int k;
    int padre1;
    int padre2 = -1;
    for (k = 0; k < (seleccionados / 2); k++)
    {
        for (padre1 = padre2 + 1; !(cromosoma[padre1].seleccionado); padre1++)
            ; // 1er. cromosoma padre de la pareja
        for (padre2 = padre1 + 1; !(cromosoma[padre2].seleccionado); padre2++)
            ; // 2 cromosoma padre de la pareja
        // Cruce: Se realiza la operación cruce en un solo punto elegido al azar
        pos = (int) floor(aleatorio(1, nGenes));
        // Se copia el cromosoma padre a un cromosoma auxiliar de trabajo para
        // no perder información durante el cruce
        copiar_cromosoma(&cromosoma_aux, &cromosoma[padre1]);
        // Se realiza el cruce mediante el intercambio de información genética
        copiar_genes(&cromosoma[padre1], pos, &cromosoma[padre2], pos, nGenes);
        copiar_genes(&cromosoma[padre2], pos, &cromosoma_aux, pos, nGenes);
    }
}
```

```

/*
Mutación uniforme
*/
void mutacion_uniforme()
{
    int i, j;
    for (i = 0; i < POP_SIZE; i++)
    {
        for (j = 0; j < nGenes; j++)
        {
            if (aleatorio(0,1) < pm)
            {
                cromosoma[i].x[j] = aleatorio(J.valor_min_parametro[j],
                                                J.valor_max_parametro[j]);
            }
        }
    }
}

/*
Elitismo
*/
void elitismo()
{
    int i;
    int mejor_actual = 0;
    int peor_actual = 0;
    for (i = 1; i < POP_SIZE; i++)
    {
        mejor_actual = (cromosoma[mejor_actual].fitness < cromosoma[i].fitness) ?
                        i :
                        mejor_actual;
        peor_actual = (cromosoma[peor_actual].fitness > cromosoma[i].fitness) ?
                      i :
                      peor_actual;
    }

    // Se guarda la mejor solución de la generación anterior (i-1)
    mejor_solucion_gen_ant = mejor_solucion_gen;
    // Se guarda la mejor solución de la generación
    mejor_solucion_gen = cromosoma[mejor_actual].fitness;
    // Mecanismo de Elitismo si el usuario así lo ha seleccionado en el GUI
    // Se compara el valor del mejor cromosoma de la población actual
    // con el mejor de la anterior. Si el mejor_actual >= mejor_cromosoma,
    // se considera como mejor_cromosoma al mejor_actual. En caso contrario
    // (ELITISMO), se sustituye el peor_actual por el mejor_cromosoma.
    if (cromosoma[mejor_actual].fitness >= mejor_cromosoma.fitness)
    {
        copiar_cromosoma(&mejor_cromosoma, &cromosoma[mejor_actual]);
    }
}

```

```

    }
    else
    {
        // ELITISMO: Se sustituye el peor_actual por el mejor_cromosoma
        if (ELITISMO)
        {
            copiar_cromosoma(&cromosoma[peor_actual], &mejor_cromosoma);
        }
    }
}

/*
Evaluación de la población en función de J (fitness de cada cromosoma)
*/
void evaluacion_poblacion()
{
    int i;
    for (i = 0; i < POP_SIZE; i++)
    {
        // Evaluación
        cromosoma[i].fitness = eval(J.funcion, precision_decimal(cromosoma[i].x));
        // Se inicializan los atributos de los cromosomas p, q y seleccionado
        // que ya no son de utilidad en la siguiente vuelta
        cromosoma[i].p = cromosoma[i].q = cromosoma[i].seleccionado = 0;
    }
}

/*
Intercambio de información genética del cromosoma origen al cromosoma destino
en la posición apuntada por pos1.
Los genes copiados del origen son los apuntados desde la pos2 hasta la pos3.
*/
void copiar_genes(Tipo_cromosoma *destino, int pos1,
                  Tipo_cromosoma *origen, int pos2, int pos3)
{
    int i;
    // Copia los valores primitivos x (genes) referenciado por puntero
    for (i = pos2; i < pos3; i++, pos1++)
    {
        *(destino->x + pos1) = *(origen->x + i);
    }
}

/*
Copia el valor de un cromosoma a otro.
Estructura del cromosoma:
    long *x; // Vector de parámetros de la función J
    float fitness; // Fitness del resultado de la evaluación de J

```

```

float p; // Probabilidad de selección p para cada cromosoma
float q; // Probabilidad acumulada q para cada cromosoma
int seleccionado; // Cromosoma seleccionado para cruce

*/
void copiar_cromosoma(Tipo_cromosoma *destino, Tipo_cromosoma *origen)
{
    int i;
    // Copia valores primitivos
    destino->fitness = origen->fitness;
    destino->p = origen->p;
    destino->q = origen->q;
    destino->seleccionado = origen->seleccionado;
    // Copia los valores primitivos x (genes) referenciado por puntero
    for (i = 0; i < nGenes; i++)
    {
        *(destino->x + i) = *(origen->x + i);
    }
}

/*
Convierte cromosomas de LONG ->FLOAT, para evaluar la función de coste J
*/
float *precision_decimal(long *x)
{
    int i;
    for (i = 0; i < nGenes; i++)
    {
        x_dec[i] = x[i] / pow(10, PRECISION);
    }
    return x_dec;
}

/*
Cancela la ejecución del AG, a petición del usuario
*/
void cancel_ag()
{
    cancelado = 1;
}

/*
Dibujar la mejor solución en el GUI
*/
void gui_mejor_solucion(int tiempo)
{
    // Datos estadísticos entre eras
    if (tiempo == 2)
    {

```



```

        maximo_J = mejor_solucion_eras;
        suma_mejor_solucion_eras += mejor_cromosoma.fitness;
        media_J = suma_mejor_solucion_eras / era;
        lista_mejores_soluciones_eras[era - 1] = mejor_cromosoma.fitness;
        // Inicializar mejor_solucion_gen generacional
        mejor_solucion_gen = -FLT_MAX;
        mejor_solucion_gen_ant = -FLT_MAX;
    }
    else
    {
        // Graficar mejor solución de cada generación
        dibujar_solucion(mejor_solucion_gen);
    }
    // Datos estadísticos
    escribir_datos_estadisticos(maximo_J, media_J, -FLT_MAX,
                                str_tiempo_transcurrido());
}

/*
Calcula la desviación estándar del proceso una vez finalizado
*/
float calculo_desviacion_estandar()
{
    int i;
    float numerador = 0;
    for (i = 0; i < (era-1); i++)
    {
        numerador += pow( (lista_mejores_soluciones_eras[i] - media_J), 2 );
    }
    desviacion_J = sqrt( numerador / (era-1) );
    return desviacion_J;
}

/*
Verifica la condición epsilon: J(k+1) - J(k) <= EPSILON
*/
int condicion_epsilon()
{
    if (EPSILON > 0)
    {
        if ( fabs(mejor_solucion_gen - mejor_solucion_gen_ant) <= EPSILON )
        {
            return TRUE;
        }
    }
    return FALSE;
}

```

```

/*
=====
Función principal del AG
=====
*/
void init_ag(int arg_ERAS, int arg_MAXGENS, int arg_POP_SIZE,
             float arg_pc, float arg_pm,
             int precision, float epsilon, int elitismo)
{
    /*
    Parámetros del AG
    */
    ERAS = arg_ERAS;
    MAXGENS = arg_MAXGENS;
    POP_SIZE = arg_POP_SIZE;
    pc = arg_pc;
    pm = arg_pm;
    cancelado = 0;
    era_inicial = 1;
    PRECISION = precision;
    EPSILON = epsilon;
    ELITISMO = elitismo;
    /* Paso INICIAL -----
    Se establece la función J a optimizar
    -----
    */
    establecer_J();
    /* Inicialización de cromosomas y Reserva de memoria -----
    Necesaria para la nueva_poblacion y el mejor_cromosoma
    -----
    */
    inicializar_cromosomas();
    // Se ejecuta el bucle principal del AG
    bucle_ag();
}

/*
Bucle principal del AG
*/
void bucle_ag()
{
    // Se ejecuta el algoritmo ERAS veces
    for (era = era_inicial; era <= ERAS; era++)
    {
        // Si se ha pulsado el botón cancelado se sale del bucle de ERAS
        if (cancelado)
        {
            break;
        }
    }
}

```

```

}

// Inicializa el gráfico de soluciones en cada era
inicia_graph("drawingareal");

/* PASO 1, PASO 2 y PASO 3 -----
PASO 1: Se genera la población inicial de forma aleatoria
PASO 2: Se evala cada cromosoma de la función inicial sobre J
PASO 3: Se obtiene el mejor individuo de la población inicial
-----

*/

generar_poblacion_aleatoria();

/* PASO 4 -----
Bucle de ejecución hasta MAXGENS (Pasos 1..6)

    PASO 4.1: Incrementar el contador de generaciones
    PASO 4.2: Aplicar el operador de selección
    PASO 4.3: Aplicar el operador de cruce en un solo punto
    PASO 4.4: Aplicar el operador de mutación uniforme
    PASO 4.5: Evaluación de la población actual
    PASO 4.6: Elitismo
-----

*/

// Bucle generacional de ejecución hasta MAXGENS veces
for (generacion = 1; generacion <= MAXGENS; generacion++)
{
    // PASO 4.1: Incrementar el contador de generaciones (en bucle)
    actualiza_barra_progreso();
    // PASO 4.2: Aplicar el operador de selección (MÉTODO DE LA RULETA)
    seleccion_cromosomas();
    // PASO 4.3: Aplicación del operador de CRUCE EN UN SOLO PUNTO
    cruce_cromosomas();
    // PASO 4.4: Se aplica el operador de mutación UNIFORME
    // Se genera, PARA CADA GEN, un número aleatorio [0,1]
    // para la selección aleatoria de cromosomas para la mutación.
    // Si r < pm se muta el cromosoma, es decir, se reemplaza EL GEN por
    // un número real aleatorio
    mutacion_uniforme();
    // PASO 4.5: EVALUACIÓN de la población actual
    evaluacion_poblacion();
    // PASO 4.6: ELITISMO
    // 1) Se localizan el mejor y peor cromosoma de la población actual
    elitismo();
    // Escribir en fichero de salida la mejor solución de la generación
    escribir_fichero("\t* ", FICHERO_SOLUCIONES);
    sprintf(str_gui, "ERA%i - Generación%i: Mejor Solución%f\n", era,
            generacion, mejor_solucion_gen);
    escribir_fichero(str_gui, FICHERO_SOLUCIONES);
    // Graficar en el GUI la mejor solución de la GENERACIÓN
    gui_mejor_solucion(1);
    // Condición adicional (epsilon)

```

```

    if (generacion >1 && condicion_epsilon())
    {
        epsilon_alcanzado = TRUE;
        int pr;
        for (pr = generacion; pr <MAXGENS; pr++)
        {
            actualiza_barra_progreso();
        }
        break;
    }
    else
    {
        epsilon_alcanzado = FALSE;
    }
} // ##### Fin bucle generacional #####
/*
Salida a GUI
*/
strcpy(str_aux, "ERA%i: MEJOR SOLUCION (%.");
sprintf(str_gui, "%if", PRECISION);
strcat(str_aux, str_gui);
sprintf(str_gui, str_aux, era, mejor_cromosoma.x[0]/pow(10, PRECISION));
int i;
char *str_aux2 = malloc(sizeof(char)*100);
for (i = 1; i <nGenes; i++)
{
    strcpy(str_aux, "%.");
    sprintf(str_aux2, "%if", PRECISION);
    strcat(str_aux, str_aux2);
    sprintf(str_aux2, str_aux, mejor_cromosoma.x[i]/pow(10, PRECISION));
    strcat(str_gui, str_aux2);
}
strcpy(str_aux, ") =%5.");
sprintf(str_aux2, "%if", PRECISION);
if (epsilon_alcanzado)
{
    if (generacion >MAXGENS) generacion = MAXGENS;
    char *str_aux3 = malloc(sizeof(char) * 50);
    sprintf(str_aux3, " :: (epsilon en generacion%i)\n", generacion);
    strcat(str_aux2, str_aux3);
}
else
{
    strcat(str_aux2, "\n");
}
strcat(str_aux, str_aux2);
sprintf(str_aux2, str_aux, mejor_cromosoma.fitness);
strcat(str_gui, str_aux2);

```

```

/*
Salida a Fichero
*/
escribir_fichero("\t* ", FICHERO_SALIDA);
escribir_fichero(str_gui, FICHERO_SALIDA);
escribir_fichero("\n", FICHERO_SOLUCIONES);
// Se guarda el mejor cromosoma de la ERA
mejor_solucion_eras = (mejor_solucion_eras < mejor_cromosoma.fitness) ?
                        mejor_cromosoma.fitness :
                        mejor_solucion_eras;

// Imprimir salida en GUI
salida_gui(str_gui);
// Graficar en el GUI la mejor solución de la ERA
gui_mejor_solucion(2);
// Se inicializa el mejor cromosoma
mejor_cromosoma.fitness = -FLT_MAX;
mejor_cromosoma.p = mejor_cromosoma.q = mejor_cromosoma.seleccionado = 0;
// Siguiente era
era_inicial++;
// Controla la pausa entre eras
if (getPausar_grafico())
{
    habilita("continuar_btn", TRUE);
    break;
}
} // ##### Fin bucle eras #####
// Fin del AG cuando se alcanza la última era o es cancelado por usuario
if (era >= ERAS || cancelado)
{
    if (cancelado)
    {
        sprintf(str_aux, "*** Proceso cancelado por el usuario ***");
        salida_gui(str_aux);
        sprintf(str_aux, "\n\t*** Proceso cancelado por el usuario ***\n");
        escribir_fichero(str_aux, FICHERO_SALIDA);
        escribir_fichero(str_aux, FICHERO_SOLUCIONES);
    }
    // Se escriben datos estadísticos en el GUI y en el fichero de salida
    escribir_datos_estadisticos(-FLT_MAX, -FLT_MAX,
                                calculo_desviacion_estandar(), NULL);
    escribir_resultados_en_salida();
    // Notificación de finalización del AG al GUI
    fin_gui();
}
}

```


Apéndice D

Ficheros de salida de SIGENOF

D.1. Descripción de las salidas

En este apéndice se muestra el formato de los dos ficheros de salida del AG tras una ejecución. En el primero de ellos, `./salida_ag_DD.MM.AAAA_HH.MM.SS`, se guarda información de la función J que ha sido optimizada, los parámetros introducidos por el usuario para dicha ejecución, el detalle de las mejores soluciones de cada era, y un resumen con los resultados del proceso en términos de tiempo de ejecución, valor máximo, medio y desviación estándar de las soluciones entre eras, mientras que en el segundo, `./soluciones_ag_DD.MM.AAAA_HH.MM.SS`, se muestran las mejores soluciones obtenidas por el AG en cada generación y era.

D.2. Fichero salida__ag

```
-----
                                S I G E N O F
Mon Sep 26 23:14:55 2005
-----

1. FUNCIÓN J A OPTIMIZAR:
    J = 21,5 + x[0] * sin(4 * pi * x[0]) + x[1] * sin(4 * pi * x[1])
con,
    x[0] entre [12,10, -3,00]
    x[1] entre [ 5,80, 4,10]

2. PARÁMETROS DEL AG:
```

```
* N° de ERAS: 5
* N° de GENERACIONES: 100
* INDIVIDUOS de la Población: 100
* Probabilidad de Cruce: 0,8000
* Probabilidad de Mutación: 0,1050
* Precisión decimal: 3
* Condición adicional (epsilon): 0,000000
* ELITISMO: SI
```

3. MEJORES SOLUCIONES EN CADA ERA:

```
* ERA 1: MEJOR SOLUCION (9,598 5,615) = 36,122
* ERA 2: MEJOR SOLUCION (11,134 5,631) = 38,178
* ERA 3: MEJOR SOLUCION (11,639 5,583) = 37,783
* ERA 4: MEJOR SOLUCION (11,603 5,093) = 37,349
* ERA 5: MEJOR SOLUCION (10,137 5,609) = 37,018
```

4. RESULTADOS DE LA EJECUCIÓN:

```
* Máximo Valor: 38,178
* Media Aritmética 37,290
* Desviación Estándar: 0,703
* Tiempo de Ejecución: 00:02 seg.
```

D.3. Fichero soluciones_ag

S I G E N O F

Mon Sep 26 23:14:55 2005

```
* ERA 1 - Generación 1: Mejor Solución 36,078178
* ERA 1 - Generación 2: Mejor Solución 35,150215
* ERA 1 - Generación 3: Mejor Solución 35,358189
* ERA 1 - Generación 4: Mejor Solución 36,121540
* ERA 1 - Generación 5: Mejor Solución 36,121540
* ERA 1 - Generación 6: Mejor Solución 36,078178
* ERA 1 - Generación 7: Mejor Solución 36,078178
* ERA 1 - Generación 8: Mejor Solución 36,078178
* ERA 1 - Generación 9: Mejor Solución 36,121540
* ERA 1 - Generación 10: Mejor Solución 36,121540
```


...

* ERA 2 - Generación 1: Mejor Solución 35,296928
* ERA 2 - Generación 2: Mejor Solución 35,781898
* ERA 2 - Generación 3: Mejor Solución 38,177864
* ERA 2 - Generación 4: Mejor Solución 38,177864
* ERA 2 - Generación 5: Mejor Solución 37,936325
* ERA 2 - Generación 6: Mejor Solución 38,177864
* ERA 2 - Generación 7: Mejor Solución 37,155708
* ERA 2 - Generación 8: Mejor Solución 38,177864
* ERA 2 - Generación 9: Mejor Solución 38,177864
* ERA 2 - Generación 10: Mejor Solución 38,177864

...

Referencias Bibliográficas

- [Chen89] Mitsuo Gen and Runwei Cheng - *Genetic Algorithms & Engineering Design, 1989*
- [Davis91] Lawrence Davis - *Handbook of Genetic Algorithms, 1991*
- [Gold89] David E. Goldberg - *Genetic Algorithms in Search, Optimization and Machine Learning, 1989*
- [Mich99] Zbigniew Michalewicz - *Genetic Algorithms + Data Structures = Evolution Programs, 1999*
- [Holl92] John H. Holland - *Algoritmos Genéticos. Investigación y Ciencia, 1992.*
- [Perez96] Anselmo Pérez Serrada - *Una Introducción a la Computación Evolutiva, 1996*
- [Kern91] Brian W. Kernighan & Dennis M. Ritchie - *El lenguaje de programación C, 2ª Edición 1991*
- [UNav98] Universidad de Navarra - *Aprenda el lenguaje ANSI C como si estuviera en primero, 1998*
- [Dom03] Francisco Domínguez-Adame - UCM - *Desarrollo de aplicaciones científicas con Glade, 2003*
- [Recip88] Cambridge University Press - *Numerical Recipes in C: The art of Scientific Computing, 1988*