



UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Proyecto de Fin de Carrera de Ingeniero Informático

## **DESARROLLO DE UNA HERRAMIENTA SOFTWARE PARA LA RESOLUCIÓN DE PROBLEMAS DE OPTIMIZACIÓN CON ALGORITMOS GENÉTICOS**

FRANCISCO JAVIER GARCÍA PAREDERO

Dirigido por: DICTINO CHAOS GARCÍA

Curso: 2014/2015





DESARROLLO DE UNA HERRAMIENTA SOFTWARE PARA LA RESOLUCIÓN DE  
PROBLEMAS DE OPTIMIZACIÓN CON ALGORITMOS GENÉTICOS

Proyecto de Fin de Carrera de modalidad oferta general

Realizado por: FRANCISCO JAVIER GARCÍA PAREDERO

Dirigido por: DICTINO CHAOS GARCÍA

Supervisado por: DICTINO CHAOS GARCÍA

Tribunal calificador:

Presidente: D. /Da. ....

Secretario: D./Da. ....

Vocal: D./Da. ....

Fecha de lectura y defensa: .....

Calificación: .....



## **Resumen**

Los algoritmos genéticos son una rama de la computación evolutiva que se caracteriza por obtener de forma rápida buenas soluciones para problemas complejos. En este trabajo se han analizado estos algoritmos para desarrollar una herramienta capaz de buscar óptimos para funciones matemáticas.

El trabajo comienza con una introducción teórica a los algoritmos genéticos, presentando los fundamentos teóricos sobre los que se desarrolla.

A continuación se realiza el desarrollo de la aplicación objeto del trabajo. El proceso de desarrollo se ha documentado de forma extensa, comenzando por la formalización de los requisitos, análisis, diseño, pruebas y documentación de usuario.

Finalmente se muestra la planificación que se ha seguido para el desarrollo de la aplicación.

Este texto viene acompañado por el producto desarrollado. Se ha construido la aplicación OptimizadorGA, capaz de obtener soluciones óptimas para funciones matemáticas. Esta aplicación conforma la parte principal del trabajo. Puede encontrarse en los medios digitales adjuntos, bajo el fichero OptimizadorGA.jar.



## **Palabras clave**

Algoritmo genético, población, individuo, cromosoma, gen, calidad, coste, selección, cruce, ruleta, torneo, mutación, elitismo.





## **Overview**

Genetic algorithms are a branch in the evolutionary computing that is characterized by obtaining quickly good solutions to complex problems. During this work, this kind of algorithms has been analyzed in order to create a tool that is able to find optimal solutions for mathematic functions.

The work starts with a theoretical introduction to genetic algorithms, where the foundations of the developed product are presented.

The development process of the application is then documented in depth. This process has been extensively detailed, starting with the requirement formalization, the analysis and design process, testing and user documentation.

Finally the planning that guided the development process is shown.

Alongside with this text, comes the developed product. An application named OptimizadorGA has been built, this application is able to find optimal solutions to mathematic function. This application is named OptimizadorGA and it is the main part of the Project. It can be found in the digital media that comes with this text as the file OptimizadorGA.jar.

## **Keywords**

Genetic algorithm, population, individual, chromosome, gene, fitness, cost, selection, crossover, roulette, tournament, mutation, elitism.



# Índice

1.	Introducción .....	1
2.	Algoritmos genéticos .....	3
2.1.	Representación de individuos.....	4
2.2.	Algoritmo general.....	4
2.3.	Operador de selección .....	5
2.3.1.	Método de la ruleta.....	5
2.3.2.	Selección por torneo.....	8
2.4.	Operador de cruce.....	10
2.4.1.	Cruce en un punto .....	10
2.4.2.	Cruce en dos puntos .....	11
2.5.	Operador de mutación .....	12
2.6.	Limitaciones de los algoritmos genéticos .....	12
3.	Planteamiento del problema.....	15
4.	Diseño .....	17
4.1.	Requisitos .....	17
4.1.1.	Introducción de la función de coste .....	17
4.1.2.	Codificación del vector de parámetros o cromosoma.....	18
4.1.3.	Introducción del tamaño de la población .....	18

4.1.4.	Establecimiento del número de generaciones .....	18
4.1.5.	Establecimiento del número de eras.....	19
4.1.6.	Selección de elitismo .....	19
4.1.7.	Introducción de la probabilidad de cruce .....	19
4.1.8.	Introducción de la probabilidad de mutación.....	19
4.1.9.	Selección de un operador de selección .....	20
4.1.10.	Validación de la configuración.....	20
4.1.11.	Inicialización de la población.....	20
4.1.12.	Evaluación de la función de coste .....	20
4.1.13.	Operador de selección .....	21
4.1.14.	Operador de cruce .....	21
4.1.15.	Operador de mutación .....	21
4.1.16.	Ejecución del algoritmo.....	21
4.1.17.	Cancelación de la ejecución .....	21
4.1.18.	Resultados parciales .....	22
4.1.19.	Resultados finales.....	22
4.1.20.	Guardado de la ejecución .....	23
4.1.21.	Carga de la configuración guardada .....	24
4.2.	Casos de uso .....	24
4.3.	Componentes del sistema .....	37

4.4.	Diagrama de clases .....	39
4.4.1.	Modelo de datos .....	42
4.4.2.	Módulo de ejecución del algoritmo.....	44
4.4.3.	Módulo de presentación de resultados .....	46
4.4.4.	Módulo de interfaz gráfica de usuario .....	50
4.4.5.	Módulo de guardado de datos .....	51
4.5.	Flujo de ejecución .....	53
4.5.1.	Flujo de ejecución general .....	53
4.5.2.	Flujo de ejecución del algoritmo genético .....	54
4.5.3.	Cancelación de la ejecución .....	55
4.5.4.	Presentación de resultados parciales .....	56
4.6.	Interfaz gráfica de usuario .....	57
4.6.1.	Ventana principal de la aplicación .....	58
4.6.2.	Ventana de progreso del cálculo .....	59
4.6.3.	Ventana de resultados de una era .....	61
5.	Funcionamiento de la aplicación.....	63
5.1.	Requisitos previos .....	63
5.2.	Acceso a la aplicación .....	63
5.3.	Introducción de la función de coste.....	64
5.4.	Codificación del vector de parámetros .....	66

5.5.	Configuración de la ejecución .....	68
5.6.	Ejecución .....	69
5.7.	Cancelación de la ejecución .....	71
5.8.	Resultados de la ejecución .....	72
5.9.	Detalles de la evolución de una era .....	74
5.10.	Guardado de datos .....	75
5.11.	Carga de datos guardados .....	76
5.12.	Ayuda de la aplicación.....	77
6.	Ejemplos de ejecución .....	79
6.1.	Caso de prueba 1 .....	79
6.2.	Caso de prueba 2 .....	80
6.3.	Caso de prueba 3: Función de Ackley .....	81
6.4.	Caso de prueba 4: Función de Schaffer.....	83
6.1.	Caso de prueba 5 .....	86
6.2.	Caso de prueba 6 .....	87
7.	Planificación y presupuesto .....	89
7.1.	Fase de planificación y documentación del proyecto.....	90
7.1.1.	Búsqueda de referencias bibliográficas.....	90
7.1.2.	Búsqueda de referencias técnicas.....	91
7.1.3.	Análisis preliminar .....	91

7.2.	Fase de desarrollo de motor de ejecución .....	91
7.2.1.	Análisis.....	91
7.2.2.	Desarrollo del modelo de datos.....	92
7.2.3.	Desarrollo del módulo de ejecución.....	92
7.2.4.	Pruebas y correcciones .....	92
7.3.	Desarrollo de la herramienta gráfica .....	93
7.3.1.	Diseño de la interfaz gráfica .....	93
7.3.2.	Implementación de la interfaz gráfica.....	93
7.3.3.	Integración con el motor de ejecución .....	93
7.3.4.	Introducción de la función de coste .....	94
7.3.5.	Codificación de cromosomas .....	94
7.3.6.	Procesamiento de resultados .....	94
7.3.7.	Pruebas y correcciones .....	95
7.4.	Mejoras en la herramienta gráfica .....	95
7.4.1.	Guardado de datos en ficheros .....	95
7.4.2.	Carga de ficheros.....	95
7.4.3.	Presentación de resultados gráficos .....	96
7.5.	Documentación.....	96
7.6.	Presupuesto.....	97
8.	Conclusiones .....	99

9. Referencias y bibliografía ..... 101



## Índice de figuras

Figura 1 Selección por torneo .....	9
Figura 2 Cruce en un punto .....	11
Figura 3 Cruce en dos puntos .....	12
Figura 4 Componentes del sistema .....	38
Figura 5 Diagrama de clases .....	40
Figura 6 Modelo de datos.....	43
Figura 7 Módulo de ejecución.....	44
Figura 8 El patrón Observer .....	47
Figura 9 Módulo de presentación de resultados.....	48
Figura 10 Módulo de interfaz gráfica.....	51
Figura 11 Módulo de guardado de datos.....	52
Figura 12 Flujo de la ejecución general .....	54
Figura 13 Flujo de ejecución del algoritmo genético.....	55
Figura 14 Cancelación de la ejecución.....	56
Figura 15 Presentación de resultados parciales.....	57
Figura 16 Ventana principal de la aplicación.....	59
Figura 17 Ventana de progreso .....	60
Figura 18 Ventana de resultados de una era.....	61

Figura 19 Ventana principal de la aplicación.....	64
Figura 20 Área de la función de coste.....	66
Figura 21 Área de introducción de nuevos parámetros.....	66
Figura 22 Listado de parámetros declarados.....	67
Figura 23 Área de configuración de la aplicación .....	68
Figura 24 Ventana de progreso .....	70
Figura 25 Sección gráfica de los resultados .....	73
Figura 26 Mejores cromosomas de la ejecución .....	74
Figura 27 Evolución del cálculo de una era .....	75
Figura 28 Diálogo de guardado de datos .....	76
Figura 29 Diálogo de apertura de datos .....	77
Figura 30 Acceso a la ayuda de la aplicación .....	78
Figura 31 Caso de prueba 1.....	80
Figura 32 Caso de prueba 3. Función de Ackley .....	82
Figura 33 Caso de prueba 4. Función de Schaffer .....	83
Figura 34 Resultados con la configuración por defecto.....	84
Figura 35 Evolución de la mejor era.....	85
Figura 36 Resultados con 1000 generaciones .....	86
Figura 37 Caso de prueba 5.....	87
Figura 38 Caso de prueba 6.....	88

Figura 39 Fases y tareas del proyecto ..... 90



## 1. Introducción

En este proyecto se ha desarrollado una herramienta destinada a la optimización de funciones reales mediante el uso de algoritmos genéticos. La aplicación de esta herramienta es fundamentalmente experimental, sirviendo de ilustración acerca del funcionamiento de este tipo de algoritmos.

El desarrollo de la herramienta se aborda tratando de mantener dos capas diferenciadas. Por una parte, se trata de crear una capa que implemente un motor de optimización que utilice las técnicas deseadas. Por otra parte se trata de crear una interfaz gráfica que permita extraer los datos y métricas convenientes.

Existen diversas herramientas en la industria destinadas a la construcción de algoritmos genéticos. Fundamentalmente se trata de librerías que proporcionan las herramientas necesarias para construir algoritmos concretos. Algunas de las más conocidas son las siguientes:

- GAlib (<http://lancet.mit.edu/ga/>). Una librería escrita en C++ para construir algoritmos genéticos.
- OpenBeagle (<https://github.com/chgagne/beagle>). Librería escrita en C++ destinada a la construcción de algoritmos evolutivos. Dispone de licencia GPLv3.
- Watchmaker framework (<http://watchmaker.uncommons.org/>). Librería escrita en java para la construcción de algoritmos evolutivos. Dispone de licencia Apache.
- El proyecto Commons Math, parte de las librerías Apache Commons, contiene el paquete genetics, que proporciona las herramientas necesarias para construir algoritmos genéticos.

En cualquier caso, y dado el objetivo didáctico de este proyecto, se ha decidido prescindir de librerías existentes para la creación de algoritmos genéticos.

Se ha decidido optar por el uso del lenguaje Java por diversos motivos. En primer lugar, se trata de un lenguaje orientado a objetos, por lo que se favorece el uso de técnicas de diseño que favorecen el buen diseño, la reutilización de los componentes y el mantenimiento de código.

Además, el lenguaje proporciona herramientas de un nivel suficientemente bajo como para permitir implementar el motor de optimización de forma sencilla. A la vez, proporciona una extensa librería de componentes que permiten reducir el esfuerzo destinado a incorporar utilidades en la herramienta. Se ha recurrido al uso de librerías externas para implementar aquellos elementos de la herramienta no relacionados con el motor de optimización, por ejemplo, interfaz gráfica de usuario, utilidades de sincronización, utilidades de manejo de ficheros en disco, librerías gráficas o librerías de evaluación de expresiones matemáticas.

Otra de las características que han influido en la elección del lenguaje, son sus capacidades para ejecutarse en cualquier plataforma. Los componentes desarrollados en el lenguaje java, pueden ejecutarse en cualquier plataforma en la que se encuentre instalada una máquina virtual java.

## 2. Algoritmos genéticos

Los algoritmos genéticos se apoyan en los principios de la evolución mediante selección natural, enunciados por Charles Darwin en su libro El Origen de las especies [2]. Estos principios pueden resumirse así:

- Cada individuo tiende a transmitir sus rasgos a su progenie.
- Sin embargo, la naturaleza produce individuos con rasgos diferentes.
- Los individuos más adaptados, tienden a producir mayor progenie.
- Durante largos periodos de tiempo se puede acumular la variación produciendo nuevas especies completamente adaptadas a nichos particulares.

A mediados de la década de 1970, un investigador de la Universidad de Michigan llamado John Holland desarrolló las ideas que más tarde se convertirían en los algoritmos genéticos [7].

Los componentes básicos de un algoritmo genético son los siguientes:

- Una representación para los individuos.
- Una medida del grado de adaptación de un individuo al medio, la función de calidad.
- Un operador de selección, con probabilidad de selección de cada individuo proporcional a su calidad.
- Un operador emparejamiento o reproducción, que dará lugar a nuevos individuos en la siguiente generación.
- Un operador mutación, capaz de alterar las características de los nuevos individuos, incrementando la riqueza genética de la población.

En cada generación se crea un nuevo conjunto de individuos utilizando el material genético de los mejores individuos de la generación anterior.

### ***2.1.Representación de individuos***

Los individuos pueden representarse de diversas formas. Dependiendo de la naturaleza de esta representación, la implementación de los operadores básicos será diferente.

Habitualmente, la teoría clásica de algoritmos genéticos utiliza para la representación la idea de cromosoma. Un cromosoma es el conjunto de los parámetros que determinan la estructura de un individuo. Cada uno de estos parámetros recibe el nombre de gen.

Existen diversos modos de representar los genes y por tanto los cromosomas. Una posibilidad habitual es utilizar valores binarios. Se asigna un número de bits a cada gen.

Sin embargo, también pueden existir representaciones que asignen a cada cromosoma un valor entero, real o cualquier otro tipo de datos específicos al campo de aplicación del algoritmo.

### ***2.2.Algoritmo general***

El algoritmo trabaja sobre una población de individuos, representados por sus cromosomas. Cada uno de estos individuos tiene asociado un valor de su bondad determinado por la función de calidad o de coste.

El funcionamiento genérico de un algoritmo genético podría ser el siguiente:

Generar una población aleatoria de  $N_i$  individuos

Mientras que no se cumpla el criterio de terminación

    Crear una nueva población

    Seleccionar padres



Cruzar los padres con probabilidad  $p_c$

Mutar los individuos resultantes con probabilidad  $p_m$

Finalmente devolver como solución el individuo con mayor calidad de la población

Los criterios de terminación pueden ser variados:

- Los mejores individuos de la población generada representan soluciones suficientemente buenas.
- Se han alcanzado el número de generaciones especificado.
- La población ha convergido. Cuando la mayoría de los cromosomas de la población tienen la mayoría de sus genes iguales se dice que la población ha convergido. Cuando se da esta situación, el hecho de repetir sucesivas iteraciones no va a producir mejores resultados.

### ***2.3. Operador de selección***

Este operador sirve para extraer individuos de una población, de modo que la probabilidad de extracción de un individuo sea proporcional al valor tomado por la función de calidad. De este modo Existen diversos métodos.

#### **2.3.1. Método de la ruleta**

Se trata de un método de selección entre  $N$  elementos no equiprobables. A cada uno de los individuos se le asigna una probabilidad acumulada en función de su calidad. Se genera un número aleatorio dentro del intervalo  $[0,1]$  y se selecciona el individuo correspondiente.

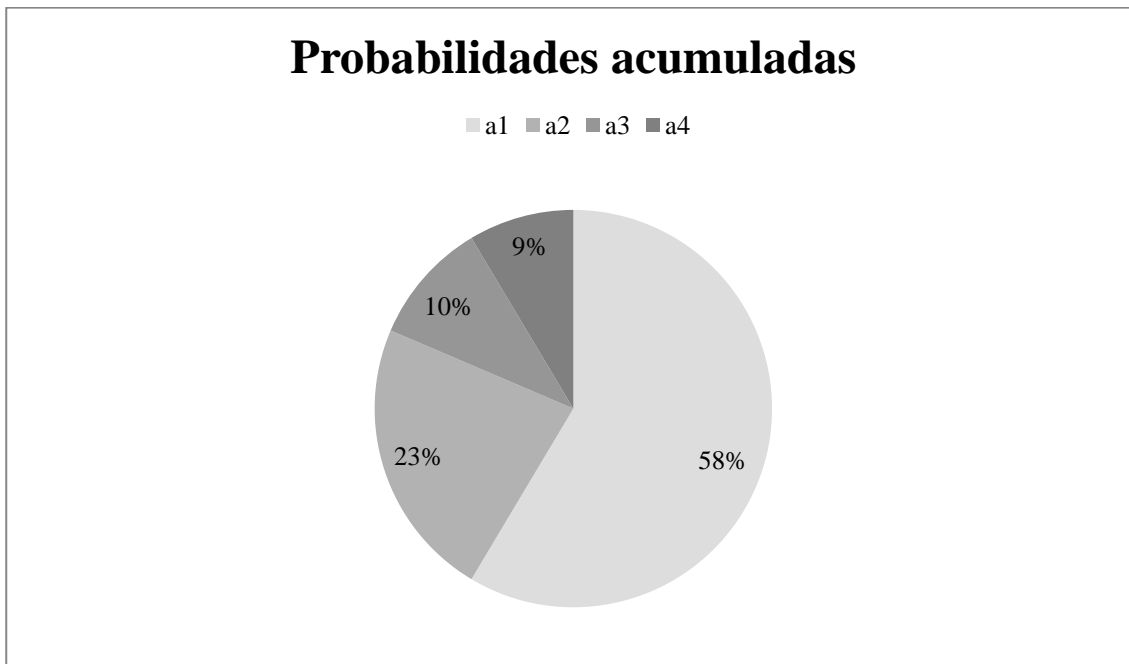
1. En primer lugar se calcula la calidad de cada individuo de la población.

2. Dividiendo cada calidad por la calidad total, obtenemos una serie de probabilidades (suman 1). Estas serán las probabilidades de selección de cada individuo. Se cumple que cuanto mayor es la calidad de un individuo, mayor es su probabilidad de selección.

$$p_i = \frac{c_i}{\sum_{j=1}^n c_j}$$

3. A continuación se calculan las probabilidades acumuladas.

$$a_k = \sum_{i=1}^k p_i$$



4. Finalmente se genera un valor aleatorio entre 0 y 1 y se selecciona el individuo correspondiente.

Este método presenta un inconveniente pues únicamente funciona cuando la función de calidad no devuelve números negativos. Para solventar este problema, se puede normalizar el valor de calidad del individuo para lograr que siempre obtenga valores positivos. El método quedaría del siguiente modo.

1. En primer lugar se calcula la calidad de cada individuo de la población.
2. Se obtiene el menor valor de calidad del conjunto de la población.
3. En caso de que este valor de calidad sea inferior a 0 se obtiene el factor de normalización. El factor de normalización será el valor mínimo de la función de calidad.

$$f = \min(c_i)$$

4. El algoritmo continúa del mismo modo, pero a partir de ahora, se utiliza la calidad normalizada.

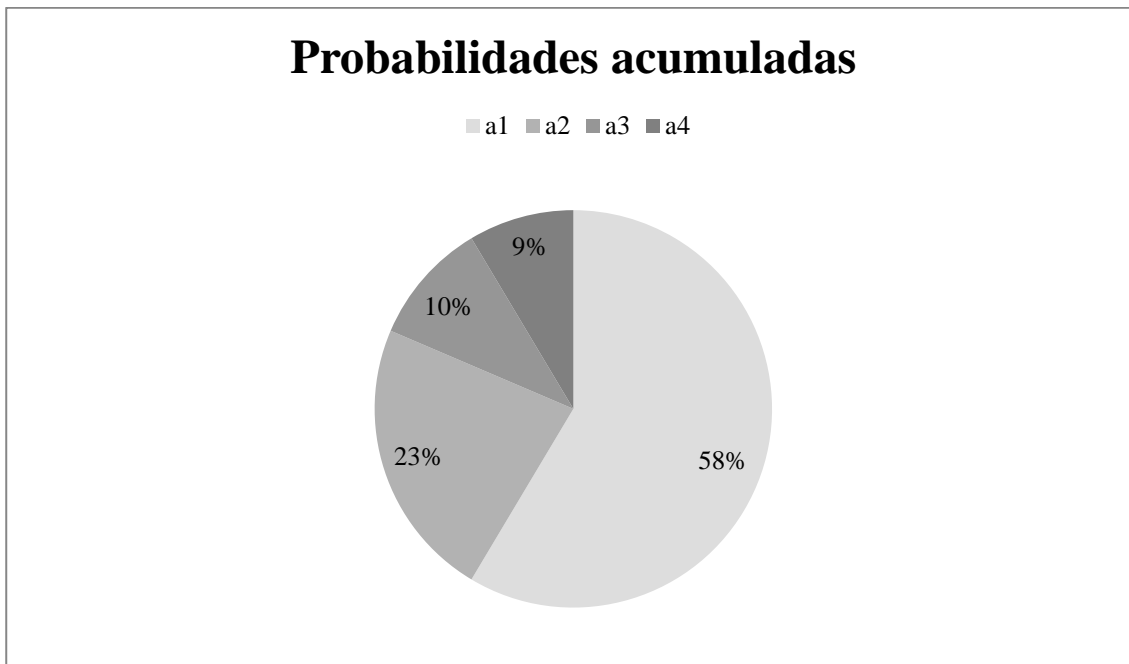
$$c'_i = c_i - f$$

5. Dividiendo cada calidad por la calidad total, obtenemos una serie de probabilidades (suman 1). Estas serán las probabilidades de selección de cada individuo. Se cumple que cuanto mayor es la calidad de un individuo, mayor es su probabilidad de selección.

$$p_i = \frac{c'_i}{\sum_{j=1}^n c'_j}$$

6. A continuación se calculan las probabilidades acumuladas.

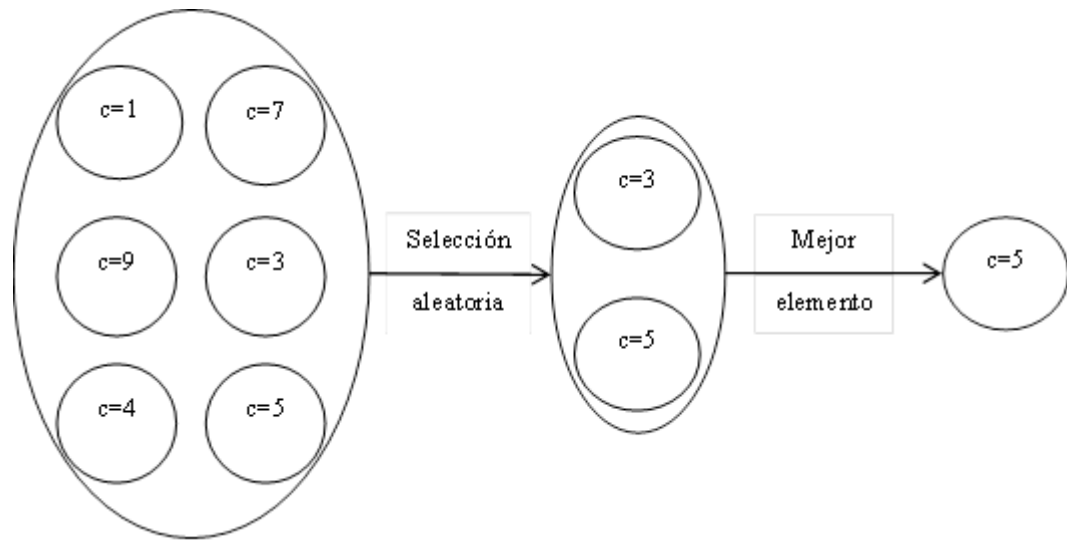
$$a_k = \sum_{i=1}^k p_i$$



7. Finalmente se genera un valor aleatorio entre 0 y 1 y se selecciona el individuo correspondiente.

### **2.3.2. Selección por torneo**

En la selección por torneo, se elige un número determinado de individuos de forma aleatoria de entre los individuos de la población. Los individuos seleccionados se comparan entre sí y se elige el que tiene mayor aptitud.



**Figura 1 Selección por torneo**

En función del número de individuos que participen en el torneo se modifica la presión de selección. Cuando el número de individuos es elevado, la presión de selección es alta, los peores individuos apenas tienen posibilidades de ser seleccionados y la búsqueda de soluciones se centra en el espacio próximo a las mejores soluciones actuales. Por el contrario cuando el número de individuos es reducido, los peores tienen más posibilidades de ser seleccionados y la búsqueda de soluciones puede explorar nuevas regiones del espacio de búsqueda.

Debe tenerse en cuenta que la selección por torneo únicamente comprueba qué elementos son mejores que los otros, pero no tiene en cuenta cuánto mejores son. Esto hace que la presión de selección se mantenga constante. Así, aunque hubiera un individuo extraordinariamente mejor que los demás, este no pasaría a dominar la siguiente generación, lo que comprometería la diversidad de la población. Por contra, se amplifican pequeñas diferencias en la calidad entre dos elementos, descartando elementos que son mínimamente inferiores a sus contendientes en el torneo.

## 2.4. Operador de cruce

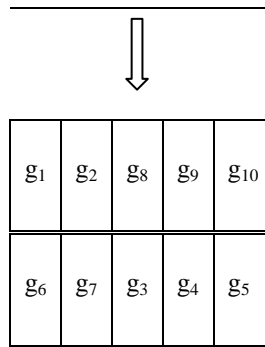
El operador de cruce intercambia información genética entre dos individuos de la población.

Existen diversos algoritmos de cruce que se exponen a continuación.

### 2.4.1. Cruce en un punto

En este algoritmo se forma un vector compuesto por los genes del individuo. A continuación se selecciona un punto de corte en el vector de forma aleatoria generando dos segmentos diferenciados en cada vector. Las colas resultantes de cada vector se intercambian entre sí resultando dos vectores descendientes que heredan información genética de cada uno de los padres.

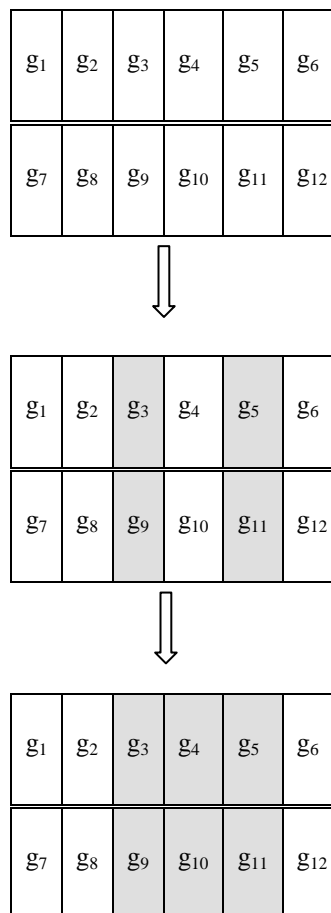


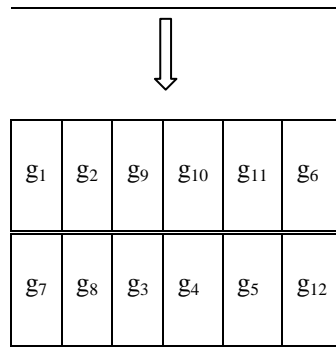


**Figura 2 Cruce en un punto**

### 2.4.2. Cruce en dos puntos

Se trata de una evolución del algoritmo de cruce en un punto. En este caso se seleccionan dos puntos en el vector de genes. Se intercambian entre ambos vectores los intervalos delimitados por los dos puntos.





**Figura 3 Cruce en dos puntos**

### ***2.5. Operador de mutación***

Se trata de un operador que provoca que se modifique alguno de los genes del individuo con un valor aleatorio. Esto suele hacerse con una probabilidad preestablecida generalmente muy baja.

El efecto de este operador es el de mantener la diversidad genética de una población, escapar de los óptimos locales y desplazar a los individuos hacia zonas del espacio de búsqueda que no pueden alcanzarse mediante el resto de operadores.

### ***2.6. Limitaciones de los algoritmos genéticos***

En numerosas ocasiones los algoritmos genéticos tienden a converger hacia valores óptimos locales en lugar de soluciones óptimas globales del problema. Tienden a buscar mejoras a corto plazo desdénando mejoras a largo plazo. Este problema puede solventarse modificando la función de evaluación, aumentando la probabilidad de mutación o mediante el uso de operadores de selección que mantengan la diversidad genética en la población. Otra posible solución al problema es reiniciar el cálculo una vez se realice la evolución durante un número determinado de generaciones. Se denomina era a cada uno de los ciclo de generaciones.

Un posible problema se da cuando el mejor miembro de una población falle en producir descendientes para la siguiente generación. Esto se puede resolver mediante una técnica



denominada elitismo. Esta técnica consiste en copiar el mejor elemento de una generación en la generación siguiente.



### 3. Planteamiento del problema

El objetivo de este proyecto consiste en el desarrollo de una aplicación para resolver problemas de optimización mediante el uso de algoritmos genéticos.

Dada una función de evaluación o de coste, con un recorrido dentro del conjunto de los reales:

$$J: \mathbb{R}^n \rightarrow \mathbb{R}$$

La función de evaluación recibe una serie de parámetros reales acotados entre un valor máximo y un valor mínimo.

$$x_i \in [x_i^{\min}, x_i^{\max}]$$

El problema a resolver consiste en encontrar el conjunto de parámetros óptimo que maximice el valor de la función de evaluación empleando para ello un algoritmo genético.

El algoritmo que se va a emplear es el siguiente:

Durante un número determinado de eras

    Generar una población aleatoria de  $N_i$  individuos

    Durante un número determinado de generaciones

        Crear una nueva población

        Seleccionar padres

        Cruzar los padres con probabilidad  $p_c$

        Mutar los individuos resultantes con probabilidad  $p_m$

        Opcionalmente aplicar un operador de elitismo

Finalmente devolver como solución el individuo con mayor calidad de la población

Se debe proporcionar una interfaz de usuario agradable. Además debe proporcionarse una visualización de los resultados obtenidos clara y deben presentarse detalles de la evolución del programa durante su ejecución.

Adicionalmente, deben proporcionarse mecanismos para almacenar los detalles de la ejecución del programa.

## 4. Diseño

### *4.1.Requisitos*

En el análisis del problema se han detectado los siguientes requisitos funcionales.

#### **4.1.1. Introducción de la función de coste**

Se trata de la función que se va a tratar de optimizar mediante el uso de la herramienta. Esta función tiene como parámetros una serie de números reales y tiene como resultado un valor positivo en el conjunto de los números reales:

$$J: \mathbb{R}^n \rightarrow \mathbb{R}$$

Los posibles elementos que podrán incluirse en la función son

- Números reales.
- Parámetros, expresados por su nombre.
- Los siguientes operadores: +, -, /, \*,  $\sqrt{\phantom{x}}$ , sin, cos, tan, asin, atan, abs, log, log<sub>10</sub>, log<sub>2</sub>
- Los símbolos de paréntesis “(” y “)”.
- El número  $\pi$ , expresado mediante el símbolo “pi”.
- El número e, expresado mediante el símbolo “e”.

#### 4.1.2. Codificación del vector de parámetros o cromosoma

El cromosoma o vector de parámetros es el conjunto de parámetros que se aplican a la función de coste. Está formado por una serie de elementos o genes, que consisten en los parámetros de la función de evaluación.

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

Cada uno de los genes es un número real acotado entre un valor mínimo y un valor máximo.

$$x_i \in [x_i^{\min}, x_i^{\max}]$$

Por este motivo, los valores necesarios para definir un gen son los siguientes:

- Un nombre.
- Un valor real mínimo, para expresar la cota inferior.
- Un valor real máximo, para expresar la cota superior.
- Un valor entero, para expresar la precisión, el número de decimales con los que trabajará.

#### 4.1.3. Introducción del tamaño de la población

El tamaño de la población es el número de elementos (cromosomas) con los que trabajará el algoritmo. Se trata de un valor numérico entero comprendido entre 1 y 1000. Inicialmente tendrá un valor por defecto de 100 elementos.

#### 4.1.4. Establecimiento del número de generaciones

El número de generaciones determinará el número de iteraciones que se realizarán durante la ejecución del algoritmo. Se trata de un valor numérico entero comprendido

entre 1 y 1000. Este valor deberá poderse introducir por pantalla. Inicialmente tendrá un valor por defecto de 100 generaciones.

#### **4.1.5. Establecimiento del número de eras**

El número de eras determinará el número de ejecuciones del algoritmo que se realizarán. Se realizan varias ejecuciones distintas del algoritmo para evitar problemas en la ejecución como la aparición de superelementos que dominen por completo la evolución. Se trata de un valor numérico entero comprendido entre 1 y 100. Este valor debe poder ser introducido por pantalla. Inicialmente tendrá un valor por defecto de 10 eras.

#### **4.1.6. Selección de elitismo**

Se debe poder seleccionar si el algoritmo operará con elitismo o no. El selector estará activado por defecto.

#### **4.1.7. Introducción de la probabilidad de cruce**

La probabilidad de cruce determinará la probabilidad de que los cromosomas sean seleccionados para aplicar el operador de cruce. Se trata de un valor numérico real entre 0 y 1 que debe poderse introducir por pantalla. Inicialmente tendrá un valor por defecto de 0.2.

#### **4.1.8. Introducción de la probabilidad de mutación**

La probabilidad de mutación determina la probabilidad de que a un cromosoma se le aplique el operador de mutación. Se trata de un valor numérico real entre 0 y 1 que debe poderse introducir por pantalla. Inicialmente tendrá un valor por defecto de 0.015.

#### **4.1.9. Selección de un operador de selección**

El usuario debe poder elegir entre distintos operadores de selección. En concreto debe poder elegir entre los siguientes operadores:

- Método de la ruleta
- Selección por torneo

#### **4.1.10. Validación de la configuración**

Antes de comenzar la ejecución del algoritmo debe validarse que la configuración seleccionada es válida. En particular deben validarse los siguientes aspectos:

- Que la función de coste introducida es sintácticamente correcta.
- Que la configuración del cromosoma es correcta y congruente con la función de coste. Esto significa que los genes introducidos coinciden en número y nombre con los parámetros de la función introducida.

#### **4.1.11. Inicialización de la población**

Al inicio de la ejecución del algoritmo genético se debe generar de forma aleatoria una población de individuos que cumplan con las restricciones especificadas para el cromosoma y cada uno de sus genes.

#### **4.1.12. Evaluación de la función de coste**

Dado un cromosoma con valores debe poderse evaluar la función de coste para los genes que contiene.



#### **4.1.13. Operador de selección**

Se ejecutará el operador de selección que se haya configurado en la herramienta.

#### **4.1.14. Operador de cruce**

El operador de cruce que se utilizará es el operador de cruce en un punto con la probabilidad de cruce introducida anteriormente.

#### **4.1.15. Operador de mutación**

El operador de mutación que se aplicará es el de mutación uniforme con la probabilidad configurada inicialmente.

#### **4.1.16. Ejecución del algoritmo**

Debe ejecutarse un algoritmo genético según la configuración introducida. El número de eras determinará el número de ejecuciones distintas del algoritmo que se realizarán. El número de generaciones determinará el número de iteraciones que realizará el algoritmo. El tamaño de la población determinará el número de elementos con los que trabajará y las probabilidades de mutación y de cruce determinarán la probabilidad con que se aplicarán los operadores de mutación y de cruce.

#### **4.1.17. Cancelación de la ejecución**

La ejecución del algoritmo debe poder cancelarse en cualquier momento. Al cancelar la ejecución se mostrarán los resultados finales con los datos calculados hasta el momento de la cancelación.

#### **4.1.18. Resultados parciales**

Durante la ejecución del algoritmo deben mostrarse los datos que se vayan calculando, así como información del progreso de la ejecución y del tiempo transcurrido. En concreto deben mostrarse los siguientes datos:

- Tiempo de ejecución del programa.
- Número de Era actual.
- Número de generación actual.
- Mejor cromosoma obtenido hasta el momento.
- Valor medio de la función de coste durante la generación anterior.
- Desviación estándar de la función de coste en la generación anterior.
- Mejora porcentual obtenido en el mejor valor de la función de coste en la generación anterior.
- Valor medio de los mejores valores de la función de coste obtenidos a lo largo de las distintas eras.

#### **4.1.19. Resultados finales**

Al finalizar la ejecución del algoritmo deben mostrarse los datos de los cálculos realizados. Los datos que se deben mostrar son:

- El mejor cromosoma obtenido en cada era.
- El mejor valor de la función de coste obtenido en cada era.

Además deben mostrarse presentaciones gráficas de los datos obtenidos. En concreto, deben mostrarse de forma gráfica los siguientes datos:

- La evolución del mejor valor de la función de coste obtenido a lo largo de las generaciones en una era.
- La evolución del valor medio de la función de coste para los individuos de la población a lo largo de las generaciones en una era.
- La evolución de la desviación típica de la función de coste para los individuos de la población a lo largo de las generaciones en una era.

#### **4.1.20. Guardado de la ejecución**

La herramienta debe permitir almacenar en un fichero los datos correspondientes a la ejecución del algoritmo. Los datos que deben almacenarse son:

- La función de coste.
- La codificación del cromosoma.
- El tamaño de la población.
- El número de eras a ejecutar.
- El número de generaciones.
- La probabilidad de mutación.
- La probabilidad de cruce.
- La existencia de elitismo.
- El operador de selección a emplear.
- Los resultados de la ejecución.

#### **4.1.21. Carga de la configuración guardada**

La herramienta debe poder cargar una configuración a partir del fichero guardado. Los datos que debe poder cargar son los especificados en el punto 4.1.20

#### **4.2. Casos de uso**

El único actor que se ha identificado es el usuario final de la herramienta. Del análisis de los requisitos se han identificado los siguientes casos de uso:

<b>CU1</b>	<b>Introducción de la función de coste</b>
<b>Descripción:</b>  Permite al usuario introducir la función de coste que se pretende optimizar mediante el uso de la herramienta.	
<b>Actores:</b>  El usuario final.	
<b>Precondiciones:</b>  Ninguna.	
<b>Flujo Normal:</b>  El usuario introduce la función que desea optimizar.	
<b>Flujo Alternativo:</b>	

No aplica.

**Post condiciones:**

No aplica.

**CU 1 Introducción de la función de coste****CU2****Introducción del número de eras****Descripción:**

Permite al usuario introducir un valor numérico entre uno y cien representando el número de eras que se desea que se ejecuten.

**Actores:**

El usuario final.

**Precondiciones:**

Ninguna.

**Flujo Normal:**

El usuario introduce en el campo que contiene las generaciones un número entero entre uno y cien.

**Flujo Alternativo:**

Si el usuario introduce un valor no válido, bien por no ser numérico o bien por estar

fuera del rango aceptable, el valor del campo vuelve a su valor anterior.
<b>Post condiciones:</b>  No aplica.

**CU 2 Introducción del número de eras**

<b>CU3</b>	<b>Introducción del número de generaciones</b>
<b>Descripción:</b>  Permite al usuario introducir un valor numérico entre uno y mil representando el número de generaciones que se desea que se ejecuten.	
<b>Actores:</b>  El usuario final.	
<b>Precondiciones:</b>  Ninguna.	
<b>Flujo Normal:</b>  El usuario introduce un número entero entre uno y mil.	
<b>Flujo Alternativo:</b>  Si el usuario introduce un valor no válido, bien por no ser numérico o bien por estar fuera del rango aceptable, el valor del campo vuelve a su valor anterior.	

**Post condiciones:**

No aplica

**CU 3Introducción del número de generaciones**

<b>CU4</b>	<b>Introducción de la codificación de los cromosomas</b>
<b>Descripción:</b>  Permite al usuario introducir la codificación de los cromosomas que va a emplear el algoritmo.	
<b>Actores:</b>  El usuario final.	
<b>Precondiciones:</b>  Ninguna.	
<b>Flujo Normal:</b>  <ol style="list-style-type: none"> <li>1. Por cada gen que componga el cromosoma el usuario introduce el nombre, valor mínimo, valor máximo y la precisión numérica que se va a emplear.</li> <li>2. El usuario pulsa el botón de añadir.</li> <li>3. La herramienta valida que el gen tenga un nombre, que no exista ningún gen con el mismo nombre y que el valor máximo no sea inferior al valor mínimo.</li> </ol>	

4. La herramienta añade el gen a la lista de genes del cromosoma.

**Flujo Alternativo:**

4. El sistema valida que el gen tenga nombre, que el nombre no esté repetido y que el valor máximo no sea inferior al valor mínimo. Si los datos del gen no son válidos, la herramienta muestra un mensaje indicando el error.

**Post condiciones:**

1. Se creará un listado con los genes configurados para el cromosoma.

**CU 4 Introducción de la codificación de los cromosomas**

<b>CU5</b>	<b>Introducción del tamaño de la población</b>
<p><b>Descripción:</b></p> <p>Permite al usuario introducir el número de cromosomas que contiene la población durante la ejecución del algoritmo genético.</p>	
<p><b>Actores:</b></p> <p>El usuario final.</p>	
<p><b>Precondiciones:</b></p> <p>Ninguna.</p>	
<p><b>Flujo Normal:</b></p>	



El usuario introduce en el campo tamaño de la población un valor numérico entre uno y mil.

**Flujo Alternativo:**

Si el usuario introduce un valor no válido, bien por no ser numérico o bien por estar fuera del rango aceptable, el valor del campo vuelve a su valor anterior.

**Post condiciones:**

No aplica.

**CU 5 Introducción del tamaño de la población**

<b>CU6</b>	<b>Introducción de la probabilidad de cruce</b>
<p><b>Descripción:</b></p> <p>Permite al usuario introducir el valor que se utilizará como probabilidad para ejecutar el operador de cruce en el algoritmo genético.</p>	
<p><b>Actores:</b></p> <p>El usuario final.</p>	
<p><b>Precondiciones:</b></p> <p>Ninguna.</p>	
<p><b>Flujo Normal:</b></p>	

El usuario introduce en el campo probabilidad de cruce un valor numérico entre cero y uno.
<b>Flujo Alternativo:</b>  Si el usuario introduce un valor no válido, bien por no ser numérico o bien por estar fuera del rango aceptable, el valor del campo vuelve a su valor anterior.
<b>Post condiciones:</b>  No aplica.

**CU 6 Introducción de la probabilidad de cruce**

<b>CU7</b>	<b>Introducción de la probabilidad de mutación</b>
<b>Descripción:</b>  Permite al usuario introducir el valor que se utilizará como probabilidad para ejecutar el operador de mutación en el algoritmo genético.	
<b>Actores:</b>  Si el usuario introduce un valor no válido, bien por no ser numérico o bien por estar fuera del rango aceptable, el valor del campo vuelve a su valor anterior.	
<b>Precondiciones:</b>  Ninguna.	

**Flujo Normal:**

El usuario introduce en el campo probabilidad de cruce un valor numérico un valor numérico entre cero y uno.

**Flujo Alternativo:**

Si el usuario introduce un valor no válido, bien por no ser numérico o bien por estar fuera del rango aceptable, el valor del campo vuelve a su valor anterior.

**Post condiciones:**

No aplica.

**CU 7 Introducción de la probabilidad de mutación**

<b>CU8</b>	<b>Guardar datos</b>
<b>Descripción:</b>  Permite al usuario guardar en un fichero los datos de configuración y de ejecución del algoritmo.	
<b>Actores:</b>  El usuario final.	
<b>Precondiciones:</b>  Se han introducido valores de configuración en la herramienta.	

**Flujo Normal:**

1. El usuario pulsa en el botón de guardar configuración.
2. El usuario selecciona un fichero en el que guardar su configuración.
3. La herramienta almacenará en el fichero los datos de configuración existentes:
  - a. Número de eras.
  - b. Número de generaciones.
  - c. Tamaño de la población
  - d. Probabilidad de mutación.
  - e. Probabilidad de cruce.
  - f. Función de coste.
  - g. Codificación de cromosomas.
  - h. Algoritmo de selección.
  - i. Utilización o no de elitismo.
4. La herramienta almacenará en el fichero los datos correspondientes a la ejecución.

**Flujo Alternativo:**

3. En caso de producirse un error al escribir el fichero, la herramienta muestra un mensaje de error.

**Post condiciones:**

Se creará un fichero con los datos de configuración y de ejecución del algoritmo.

**CU 8 Guardar datos**

<b>CU9</b>	<b>Cargar datos</b>
<b>Descripción:</b>  Permite al usuario cargar los datos de configuración y de ejecución almacenados en un fichero.	
<b>Actores:</b>  El usuario final.	
<b>Precondiciones:</b>  Ninguna.	
<b>Flujo Normal:</b>  <ol style="list-style-type: none"><li>1. El usuario pulsa en el botón de cargar configuración.</li><li>2. El usuario selecciona el fichero que contiene la configuración.</li><li>3. En la pantalla principal de la herramienta aparecerán introducidos los parámetros de configuración que contiene el fichero. Estos parámetros son:<ol style="list-style-type: none"><li>a. Número de eras.</li><li>b. Número de generaciones.</li></ol></li></ol>	

- c. Tamaño de la población
  - d. Probabilidad de mutación.
  - e. Probabilidad de cruce.
  - f. Función de coste.
  - g. Codificación de cromosomas.
  - h. Algoritmo de selección.
  - i. Utilización o no de elitismo.
4. En caso de existir datos con resultados de ejecución en el fichero se mostrarán mediante la herramienta.

**Flujo Alternativo:**

3. En caso de que el formato del fichero no sea legible por la herramienta, se muestra un mensaje de error

**Post condiciones:**

Los parámetros de configuración contenidos en el fichero se mostrarán en los campos correspondientes en la herramienta.

La herramienta mostrará los resultados de ejecución contenidos en el fichero si estos existen.

**CU 9 Cargar datos**

<b>CU10</b>	<b>Resolución de problemas de optimización</b>
-------------	--

**Descripción:**

Hace que la herramienta resuelva el problema de optimización configurado.

**Actores:**

El usuario final

**Precondiciones:**

La herramienta debe estar correctamente configurada

**Flujo Normal:**

1. El usuario pulsa en el botón de ejecutar.
2. El sistema comprueba que existe una función de coste correcta.
3. El sistema valida que se han configurado los cromosomas de forma correcta y congruente con la función de coste
4. El sistema comienza la ejecución de un algoritmo genético con las propiedades configuradas.
5. El sistema muestra resultados parciales durante su ejecución.
6. Una vez finalizada la ejecución, el sistema muestra los resultados finales del cálculo.

**Flujo Alternativo:**

1. Si la función de coste no es correcta, el sistema muestra un mensaje de error

2. Si la configuración de los cromosomas no es coherente con la función de coste o no es correcta, el sistema muestra un mensaje de error.
4. Si durante la ejecución del algoritmo se produjese algún error, se detiene la ejecución, se muestra un mensaje de error y se muestran los resultados obtenidos hasta el momento.

**Post condiciones:**

Al finalizar la ejecución, el sistema mostrará los resultados finales del cálculo.

**CU 10 Resolución de problemas de optimización**

<b>CU11</b>	<b>Cancelación de la ejecución</b>
<p><b>Descripción:</b></p> <p>El usuario puede cancelar en todo momento la ejecución.</p>	
<p><b>Actores:</b></p> <p>El usuario final.</p>	
<p><b>Precondiciones:</b></p> <p>El sistema debe estar calculando la solución a un problema de optimización.</p>	
<p><b>Flujo Normal:</b></p> <ol style="list-style-type: none"> <li>1. El usuario pulsa en el botón de cancelar.</li> </ol>	



- |  |
|--|
| <ol style="list-style-type: none"><li>2. La ejecución se detiene antes de finalizar.</li><li>3. La aplicación mostrará los resultados calculados hasta el momento.</li></ol> |
|--|

<b>Flujo Alternativo:</b>
---------------------------

No aplica.
------------

<b>Post condiciones:</b>
--------------------------

La ejecución se detendrá sin haber finalizado.
--

CU 11 Cancelación de la ejecución

### ***4.3.Componentes del sistema***

Los componentes que conforman el sistema, así como sus dependencias con componentes de terceros, se presentan en el siguiente diagrama.

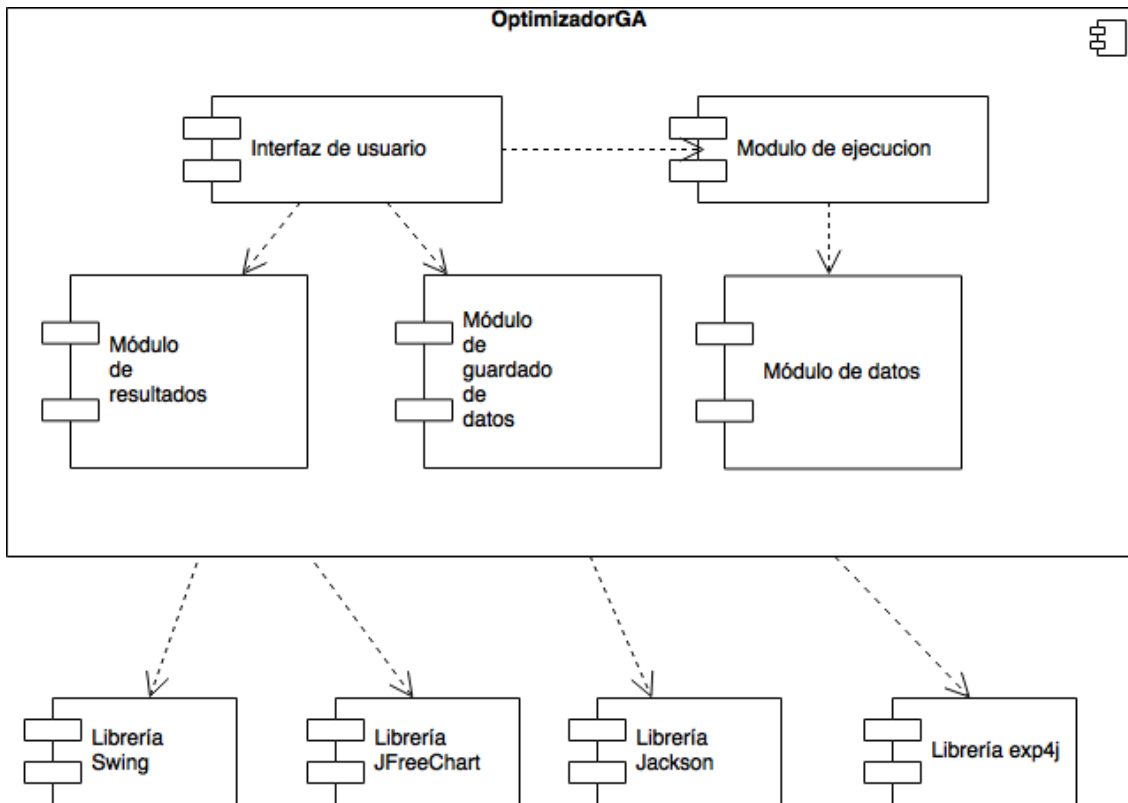


Figura 4 Componentes del sistema

Como puede observarse, el sistema está formado por un único componente. Este componente está compuesto por varios módulos:

- **Interfaz de usuario:** es el módulo encargado de la interacción con el usuario.
- **Módulo de ejecución:** este módulo se encarga de gestionar la ejecución del algoritmo.
- **Módulo de datos:** este módulo se encarga de gestionar el modelo de datos con los que trabaja la aplicación.
- **Módulo de resultados:** este módulo está encargado de mostrar los resultados al usuario.
- **Módulo de guardado de datos:** este módulo se encarga de almacenar los resultados y la configuración de la ejecución.

Además, la herramienta realiza uso de determinadas librerías externas:

- **Swing:** librería estándar de Java empleada para la construcción de interfaces gráficos de usuario.
- **JFreeChart:** librería destinada a la visualización de gráficas. Se utiliza para mostrar los resultados de la ejecución de forma gráfica. Dispone de licencia LGPL, que permite su uso de forma libre. <http://www.jfree.org/jfreechart/>
- **Jackson:** librería destinada al manejo de datos en formato JSON. Se utiliza para el almacenamiento de resultados y configuración en ficheros. Dispone de licencia Apache 2.0, que permite su uso de forma libre. <http://wiki.fasterxml.com/JacksonHome>
- **Exp4j:** librería destinada a la evaluación de expresiones matemáticas. Emplea el algoritmo shunting yard desarrollado por Edsger Dijkstra. Se utiliza para la evaluación de la función de coste. Dispone de licencia Apache 2.0, que permite su uso de forma libre. <http://www.objecthunter.net/exp4j>

#### ***4.4.Diagrama de clases***

A continuación se presenta el diagrama con las clases fundamentales del sistema. Este diseño se verá de forma más detallada en secciones siguientes.

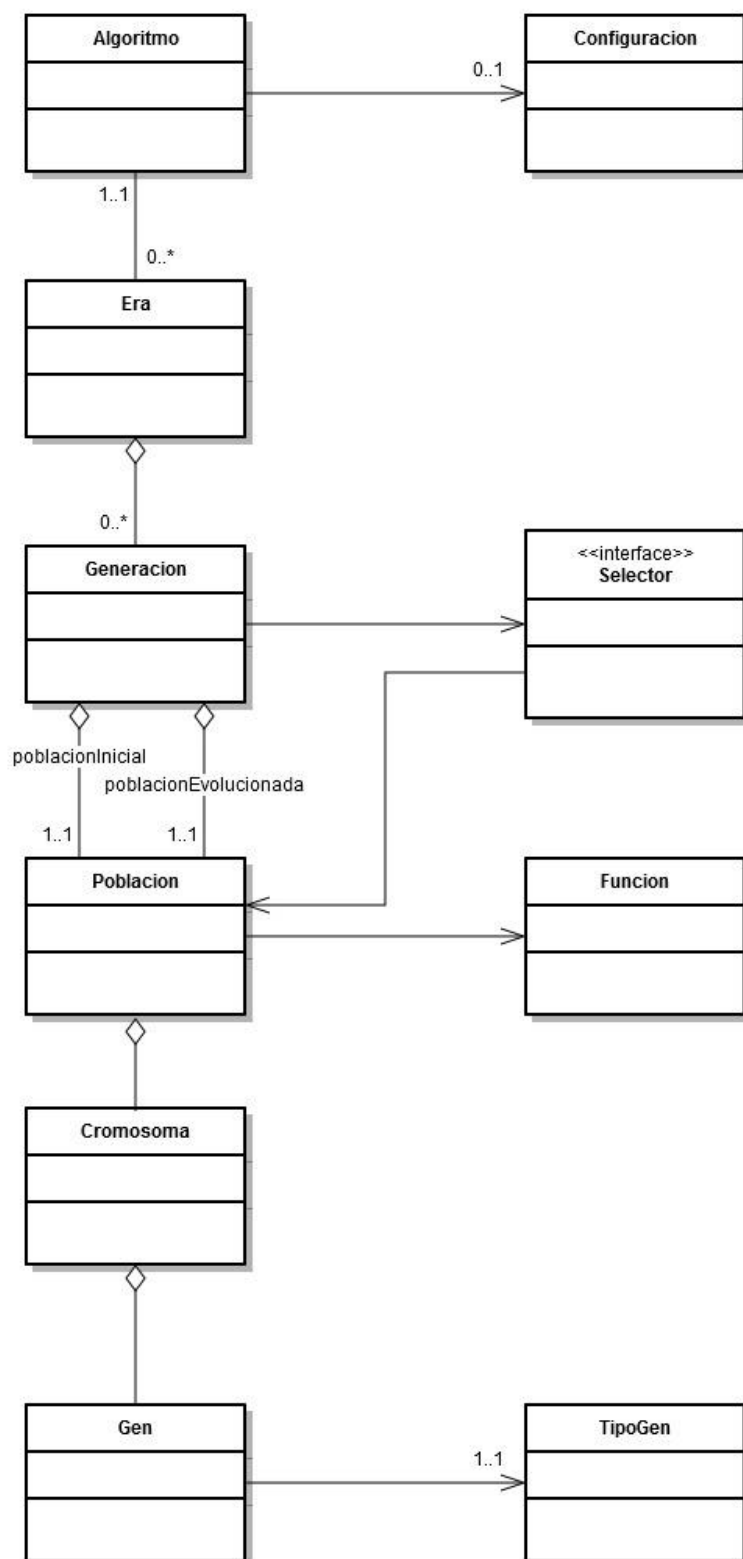


Figura 5 Diagrama de clases

Se han identificado las siguientes clases:

- **Configuracion:** Se trata de una clase diseñada para almacenar los datos de configuración del algoritmo. Contiene número de eras y número de generaciones a ejecutar, tamaño de la población, probabilidad de cruce y de mutación y algoritmo de selección. Al ser necesaria únicamente una instancia de esta clase, se ha diseñado implementando el patrón Singleton.
- **Algoritmo:** Esta clase está diseñada para gestionar el flujo de ejecución del algoritmo genético, se encarga de controlar la ejecución de las sucesivas eras.
- **Era:** esta clase está diseñada para gestionar el flujo de ejecución de una era, se encarga de controlar la ejecución de las generaciones que forman parte de cada era.
- **Generacion:** Esta clase está diseñada para gestionar los pasos evolutivos que se dan en una generación. En su creación recibe una población inicial y aplica un paso en la evolución para crear una población evolucionada.
- **Poblacion:** Esta clase contiene los cromosomas que forman parte de cada población.
- **Cromosoma:** Esta clase representa un individuo de la población. Tiene un valor asociado a la función de evaluación.
- **Gen:** Esta clase representa cada uno de los genes que conforman un individuo. Dispone de dos atributos, un número real representando el valor del propio gen y su codificación, representada por la clase TipoGen.
- **TipoGen:** Esta clase representa la codificación de los genes, dispone de varios atributos: un nombre, un valor máximo, un valor mínimo y un valor de precisión.

- **Funcion:** Representa la función de evaluación o coste asociada al algoritmo. Realiza uso de la clase Expression procedente de la librería Exp4j.
- **Selector:** Se trata de una interfaz empleado para representar el algoritmo de selección empleado. Declara un método, seleccionar, que dada una población inicial debe devolver una población seleccionada para su evolución.

#### 4.4.1. Modelo de datos

En el modelo de datos se incluyen aquellas clases cuyo objetivo fundamental es la de contener los datos de la aplicación. Se trata de clases con muy poco o ningún comportamiento asociado por lo que sus métodos son o bien constructores, o bien métodos de factoría estática o bien métodos de acceso a sus atributos. Estos métodos se han incluido dentro del paquete `com.uned.optimizadorga.elementos`

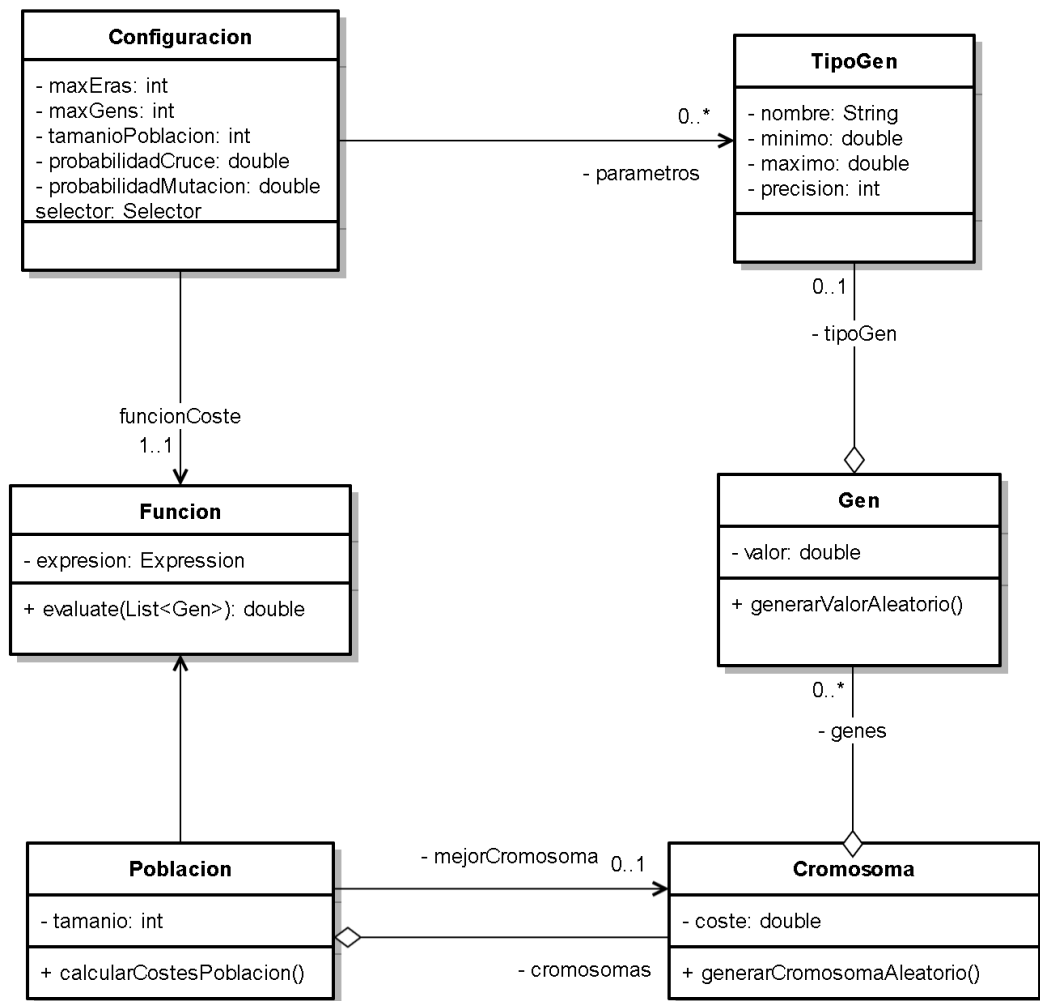


Figura 6 Modelo de datos

El paquete elementos contiene las clases que almacenan los datos de la aplicación. Las clases implicadas en esta área son:

- **TipoGen**
- **Gen**
- **Cromosoma**
- **Población**
- **Función**

- Configuración

#### 4.4.2. Módulo de ejecución del algoritmo

El módulo de ejecución del algoritmo alberga las clases dedicadas a controlar la ejecución del algoritmo genético. Estas se encuentran situadas dentro del paquete `com.uned.optimizadorga.algoritmo` y sus correspondientes subpaquetes.

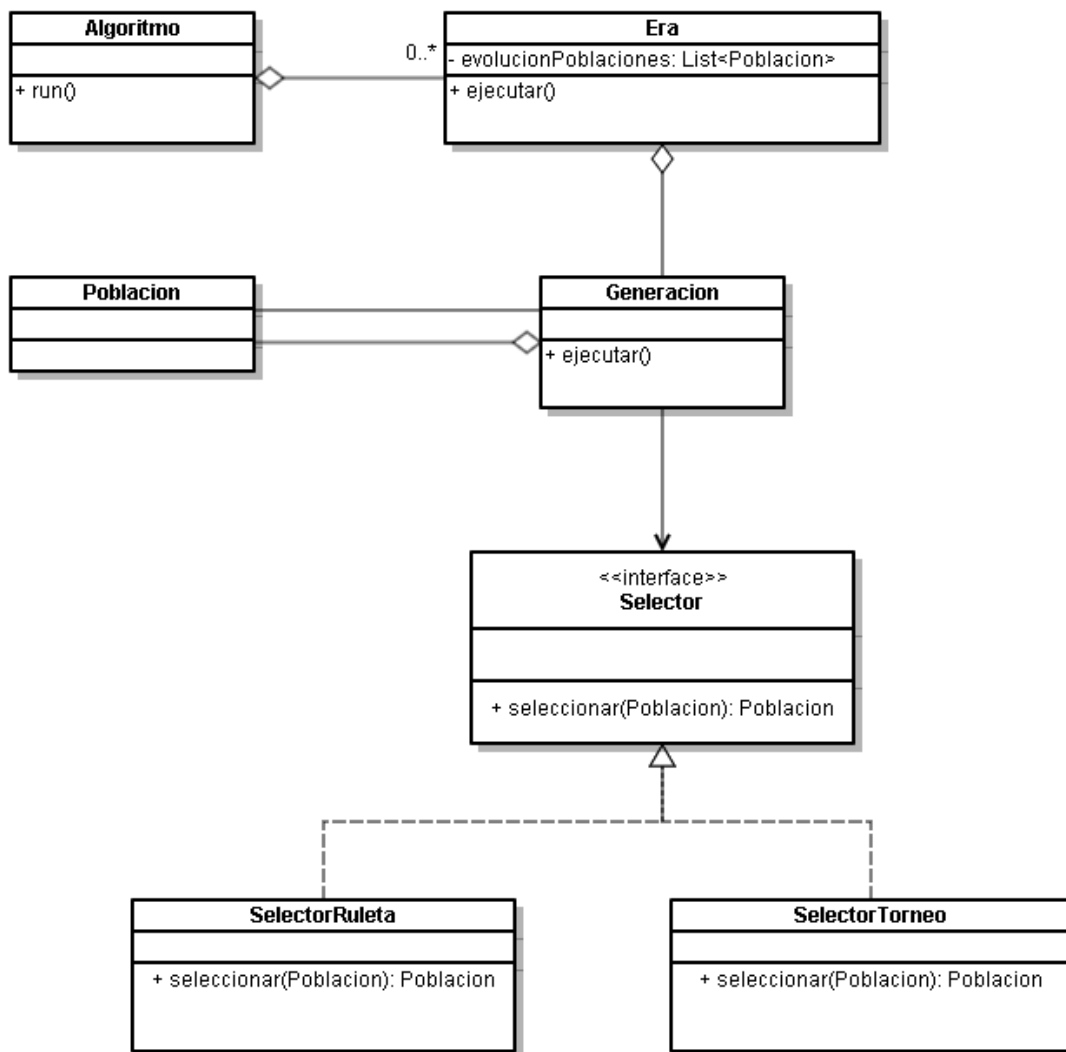


Figura 7 Módulo de ejecución

Las clases identificadas en el módulo que se encarga de la ejecución del algoritmo son:



- **Algoritmo:** Esta clase gestiona la implementación del algoritmo a alto nivel. Implementa la interfaz Runnable de java lo que le permite emplear un hilo de ejecución distinto de la clase que lo invoque. Mediante el método run crea las eras necesarias e invoca su ejecución.
- **Era:** esta clase gestiona la ejecución de una era en concreto. Dispone de un método ejecutar. Este método crea una población inicial con sus genes inicializados con valores aleatorios. Posteriormente la evolución durante las generaciones especificadas.
- **Generación:** Esta clase contiene la implementación de cada generación. El estado que contiene consiste en una población inicial y la correspondiente población resultado de la evolución. Al igual que en la clase era, dispone de un método público ejecutar que realiza la evolución correspondiente. Este método aplica los operadores de selección, cruce, mutación y elitismo que se hayan configurado.
- **Poblacion:** Esta clase es tanto la entrada como la salida de la evolución.
- **Selector:** Dado que se ha optado por implementar diversos operadores de selección, se ha decidido extraer el comportamiento necesario en la interfaz Selector. Esta obliga a implementar el método seleccionar a las clases que lo implementen. Se hace uso de polimorfismo para seleccionar la implementación adecuada en tiempo de ejecución.
- **SelectorRuleta:** Se trata de una implementación de la interfaz Selector que especifica el método de la ruleta.
- **SelectorTorneo:** Se trata de una implementación de la interfaz Selector que especifica una selección por torneo, el número de contendientes que emplea en el torneo es por defecto de dos.

#### **4.4.3. Módulo de presentación de resultados**

En principio la presentación de resultados debería ser extremadamente sencilla, pues bastaría con tomar los resultados de la ejecución y aplicar las transformaciones necesarias para mostrar las métricas de interés. Sin embargo, la necesidad de obtener datos sobre la evolución temporal del programa implica una serie de complicaciones que justifican la necesidad de este módulo. En él intervienen clases del paquete `com.uned.optimizadorga.algoritmo`.

Para mostrar datos sobre la evolución del programa se ha optado por la utilización del patrón Observer tanto sobre las eras como sobre el propio algoritmo. Para ello se han utilizado los interfaces `AlgoritmoSubject`, `AlgoritmoObserver`, `EraSubject` y `EraObserver`.

El patrón Observer define la relación entre un objeto y varios objetos dependientes de modo que cuando el estado del objeto cambie, todos los objetos dependientes sean notificados y actualizados. En este caso, se desea que cuando se produzca un cambio en el estado del cálculo (bien por la finalización del cálculo de una era como por la finalización del cálculo de una generación), se informe a la interfaz de usuario. La estructura básica de este patrón sería la siguiente:

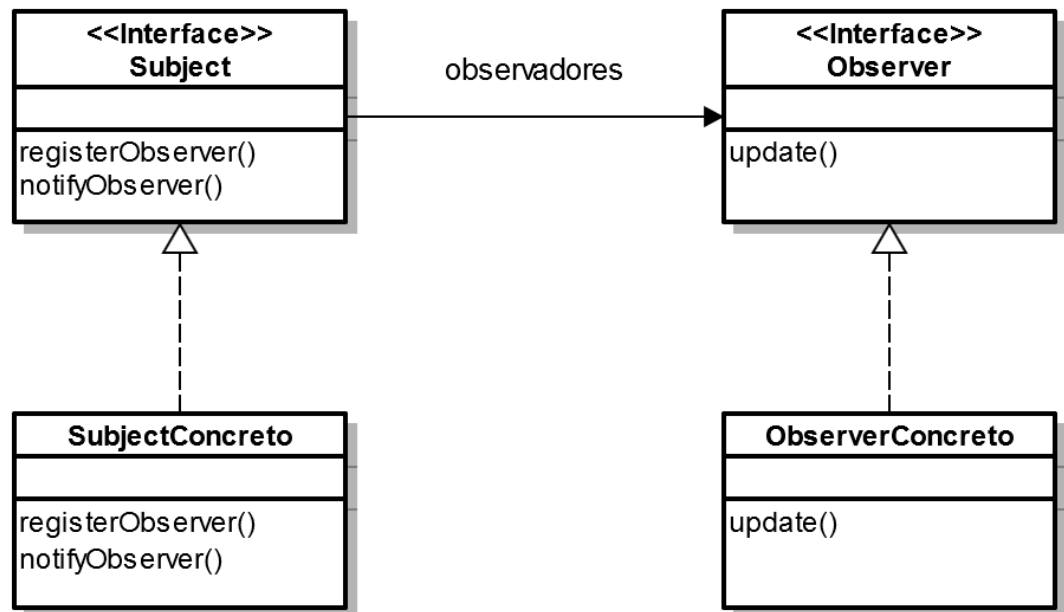


Figura 8 El patrón Observer

Cuando una clase desea mantenerse actualizada de los cambios de estado en una clase, interpreta el papel del Observer, y la clase cuyos cambios son observados se denomina Subject. La clase Subject proporciona el método `registerObserver` para que otras clases puedan incorporarse como observadores de su estado. Además dispone de un método `notifyObserver` que es el que se encargará de notificar cambios en su estado. Este método lo que hace es invocar al método `update` del observador informándole del nuevo estado.

Se explicará su funcionamiento concreto cuando se comenten las clases correspondientes.

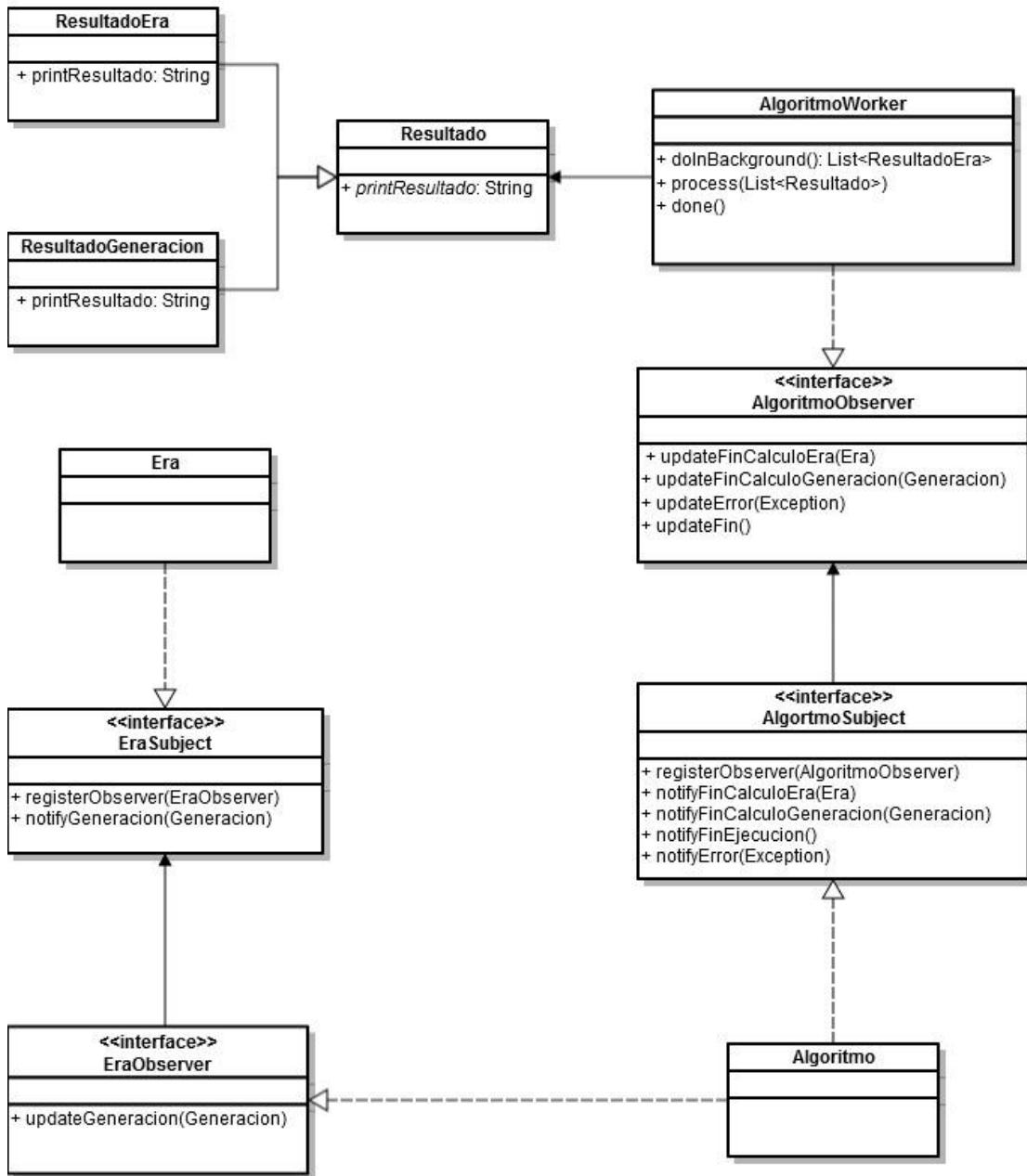


Figura 9 Módulo de presentación de resultados

Las clases que componen este módulo son:

- **Resultado:** Clase abstracta destinada a hacer de superclase para definir los resultados, tanto sean los resultados parciales correspondientes al cálculo de una generación como los correspondientes al cálculo de una era. Dispone de un método denominado `printResultado` que debe ser definido en las clases hijas y obliga a definir el modo en que se mostrará el resultado.
- **ResultadoEra:** Implementación de la clase `Resultado` correspondiente a los resultados procedentes del procesamiento de una era.
- **ResultadoGeneracion:** Implementación de la clase `Resultado` correspondiente a los resultados procedentes de la evolución de una generación.
- **EraSubject:** Interfaz empleada para la implementación del Subject en el patrón Observer. En este caso, el aspecto que se desea observar es la evolución del cálculo dentro de una era, que consiste en la finalización del cálculo de una generación. Lo que hace que la clase `Era` deba implementar esta interfaz.
- **EraObserver:** Interfaz empleada para la implementación de los observadores en el patrón Observer. En este caso, la clase interesada en mantenerse actualizada de los cambios en una era es la clase que ha lanzado su ejecución, la clase `Algoritmo`.
- **AlgoritmoSubject:** Interfaz empleada para la implementación del Subject en el patrón Observer. En este caso, los aspectos a observar serán los correspondientes a la evolución del algoritmo: la finalización del cálculo de una generación, la finalización del cálculo de una era, la finalización de la ejecución o que se haya producido un error. Se proporcionan métodos para notificar cada uno de estos eventos. La clase que implementa esta interfaz es el `Algoritmo`.
- **AlgoritmoObserver:** Interfaz empleada para la implementación de los observadores en el patrón Observer. En este caso, la clase interesada en mantenerse actualizada de los cambios en una era es la clase que ha lanzado su ejecución, la clase `AlgoritmoWorker`.

- **Era:** Esta clase debe implementar la interfaz `EraSubject` para informar de los resultados parciales obtenidos en el cálculo de una generación.
- **Algoritmo:** Esta clase debe implementar la interfaz `EraObserver` para recoger los resultados parciales obtenidos en el cálculo de una generación. Además debe implementar la interfaz `AlgoritmoSubject` para informar de los resultados parciales obtenidos en el cálculo de la era, de errores en la ejecución o del fin del cálculo.
- **AlgoritmoWorker:** Esta clase se utiliza para iniciar el cálculo en un hilo de ejecución diferente del correspondiente a la interfaz de usuario. Esta clase extiende la clase de utilidad `SwingWorker`, proporcionada por la librería estándar `Swing`, que proporciona métodos de utilidad para actualizar a la interfaz de usuario con datos correspondientes a la evolución de la ejecución, y a los datos finales resultado del cálculo. Para poder informar a la interfaz de estos datos, debe establecerse como un observador sobre el algoritmo implementando la interfaz `AlgoritmoObserver`.

#### 4.4.4. Módulo de interfaz gráfica de usuario

La interfaz gráfica está compuesta por una serie de clases que utilizan los componentes definidos en la librería `Swing` de Java. Estas clases se encuentran situadas en el paquete `com.uned.optimizadorga.gui`

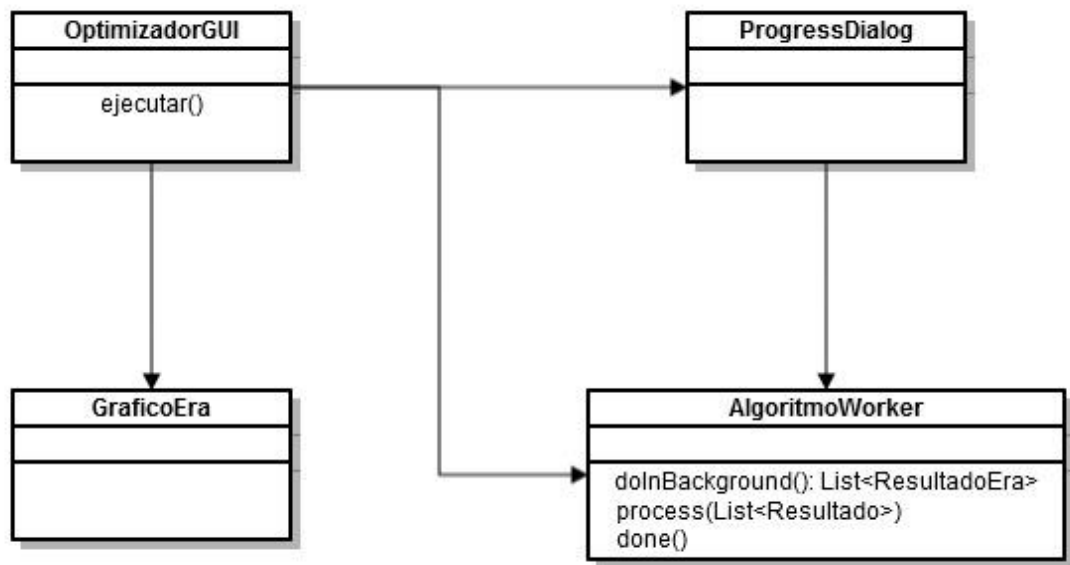


Figura 10 Módulo de interfaz gráfica

- **OptimizadorGUI:** Se trata de la ventana principal de la aplicación. Constituye el punto de entrada a la aplicación y se contiene tanto los elementos necesarios para configurar la ejecución como los resultados finales del cálculo.
- **ProgressDialog:** Se trata de una ventana que informa sobre el progreso del cálculo. Se abre al inicio de la ejecución y muestra los resultados parciales que se van calculando.
- **GraficoEra:** Se trata de una clase empleada para mostrar los datos detallados de los resultados obtenidos en el cálculo de una era.
- **AlgoritmoWorker:** Esta clase es la que actúa de nexo entre el algoritmo en ejecución y la interfaz gráfica.

#### 4.4.5. Módulo de guardado de datos

El módulo de guardado de datos es el más sencillo de todos. El objetivo de este es almacenar los datos de configuración del sistema, y en caso de existir, los resultados de la ejecución. Además proporciona mecanismos para poder recuperar estos datos de un

fichero externo. Se trata de operaciones que afectan a la interfaz de usuario y por este motivo se encuentra situado en clases del paquete `com.uned.optimizadorga.gui`

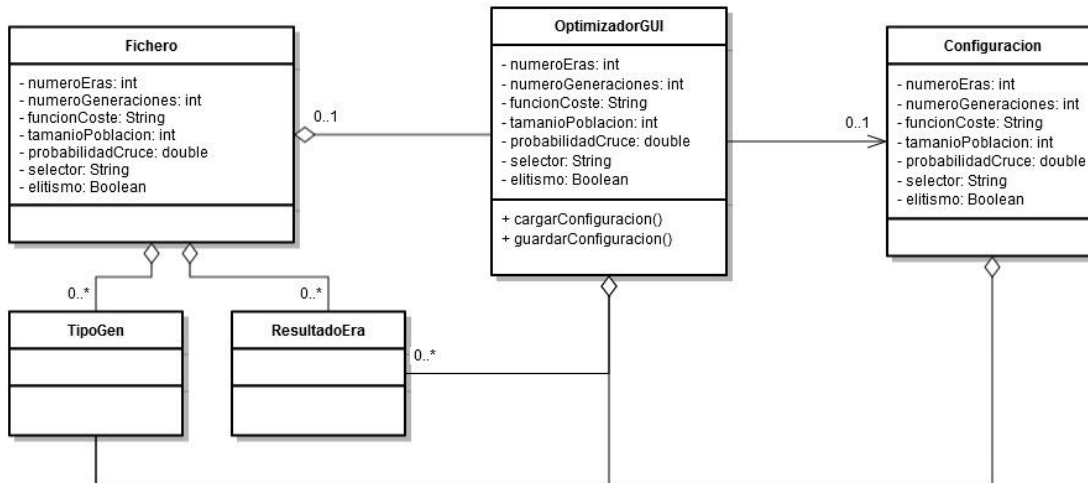


Figura 11 Módulo de guardado de datos

- **OptimizadorGUI:** La clase que gestiona la interfaz de usuario. En esta clase se incluyen los métodos destinados a guardar y cargar datos de un fichero. Además puede contener los datos de configuración si no se ha realizado una ejecución del algoritmo en el momento del guardado.
- **TipoGen:** La clase que contiene los datos de codificación de los cromosomas. Puede pertenecer a la clase OptimizadorGUI o a la clase Configuración en función de si se ha ejecutado el algoritmo.
- **ResultadoEra:** Los resultados de una ejecución del algoritmo.
- **Configuracion:** Esta clase contiene los datos de configuración una vez que el algoritmo se ha ejecutado.
- **Fichero:** En esta clase se almacenan todos los datos, tanto de configuración como resultado de una ejecución. Haciendo uso de la librería Jackson, se almacenará en un fichero en formato JSON.



### ***4.5. Flujo de ejecución***

En esta sección se pretende mostrar los flujos que sigue la ejecución de los distintos procesos que forman la herramienta.

#### **4.5.1. Flujo de ejecución general**

La ejecución general comienza desde la interfaz de usuario, una vez que el usuario ha establecido los datos requeridos por el algoritmo y pulsa en el botón ejecutar.

Desde la interfaz de usuario se crean los objetos necesarios para la ejecución y se pasan a la clase `AlgoritmoWorker`.

El objeto `AlgoritmoWorker` inicia el hilo que ejecutará el algoritmo. Durante su ejecución, el algoritmo envía actualizaciones de forma asíncrona sobre su evolución al objeto `AlgoritmoWorker`, que a su vez se lo reenvía a un objeto `ProgressDialog` que muestra datos sobre la evolución del algoritmo.

Una vez que finalice la ejecución, el algoritmo se lo notifica al `AlgoritmoWorker`, que a su vez envía los resultados a la interfaz de usuario para que los procese y los muestre por pantalla.

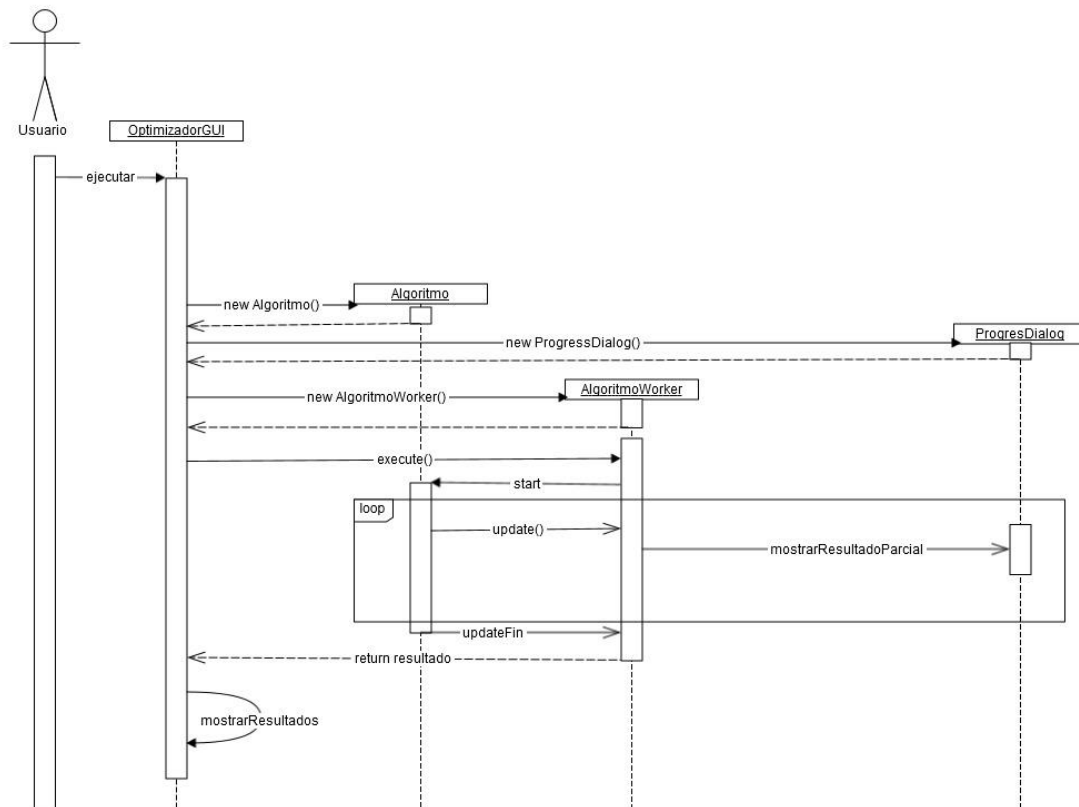


Figura 12 Flujo de la ejecución general

#### 4.5.2. Flujo de ejecución del algoritmo genético

El siguiente diagrama de secuencia muestra la evolución de la ejecución del algoritmo genético. Como puede verse, se ejecutan en bucle todas las eras, asimismo, dentro de cada era se ejecutan en bucle todas las generaciones.

La clase generación recibe una población inicial y durante su ciclo de vida aplica los operadores definidos para obtener una población evolucionada. Una vez finaliza cada generación, es la era a la que pertenece la que se encarga de notificar la finalización para poder computar resultados parciales.

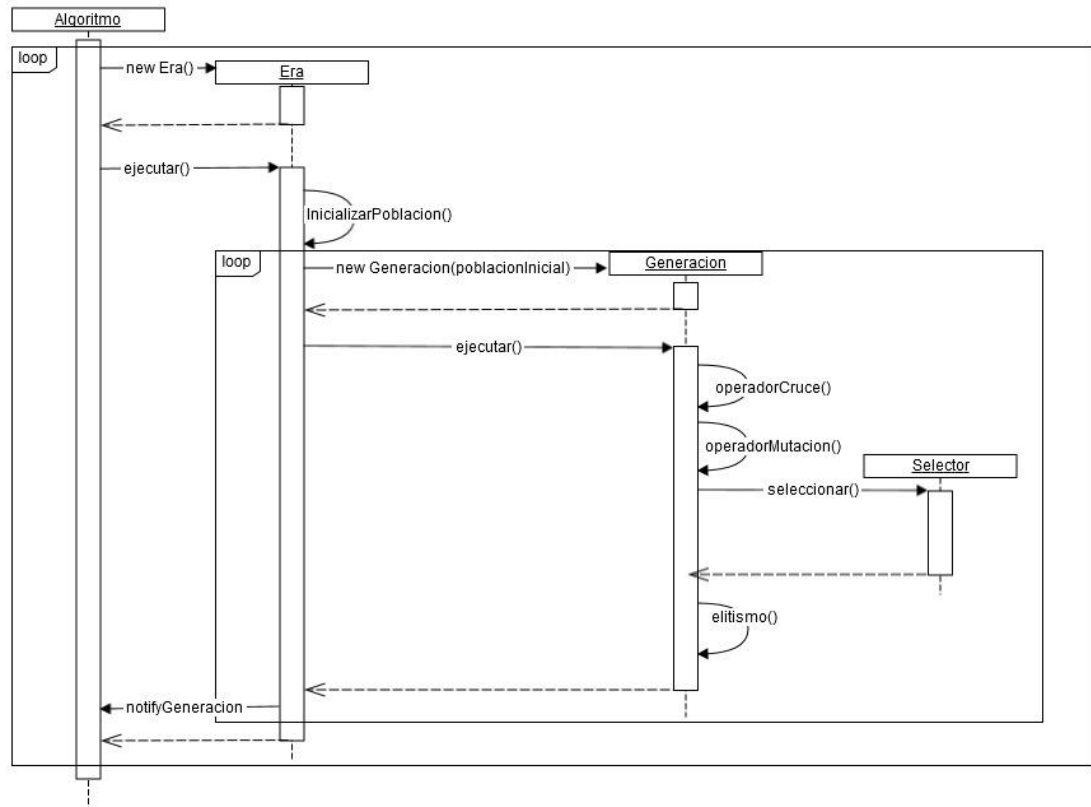


Figura 13 Flujo de ejecución del algoritmo genético

### 4.5.3. Cancelación de la ejecución

El siguiente diagrama de secuencia muestra el evento de la cancelación de una ejecución. El proceso comienza con el usuario pulsando el botón cancelar del diálogo que muestra el progreso. El controlador de eventos de este botón invoca el método cancel heredado de la clase `SwingWorker` que cambia el estado de la clase `AlgoritmoWorker`. La clase `AlgoritmoWorker` se encuentra en un bucle comprobando su estado. En el momento en que detecta este cambio, envía una interrupción a la clase `Algoritmo` para cancelar su ejecución.

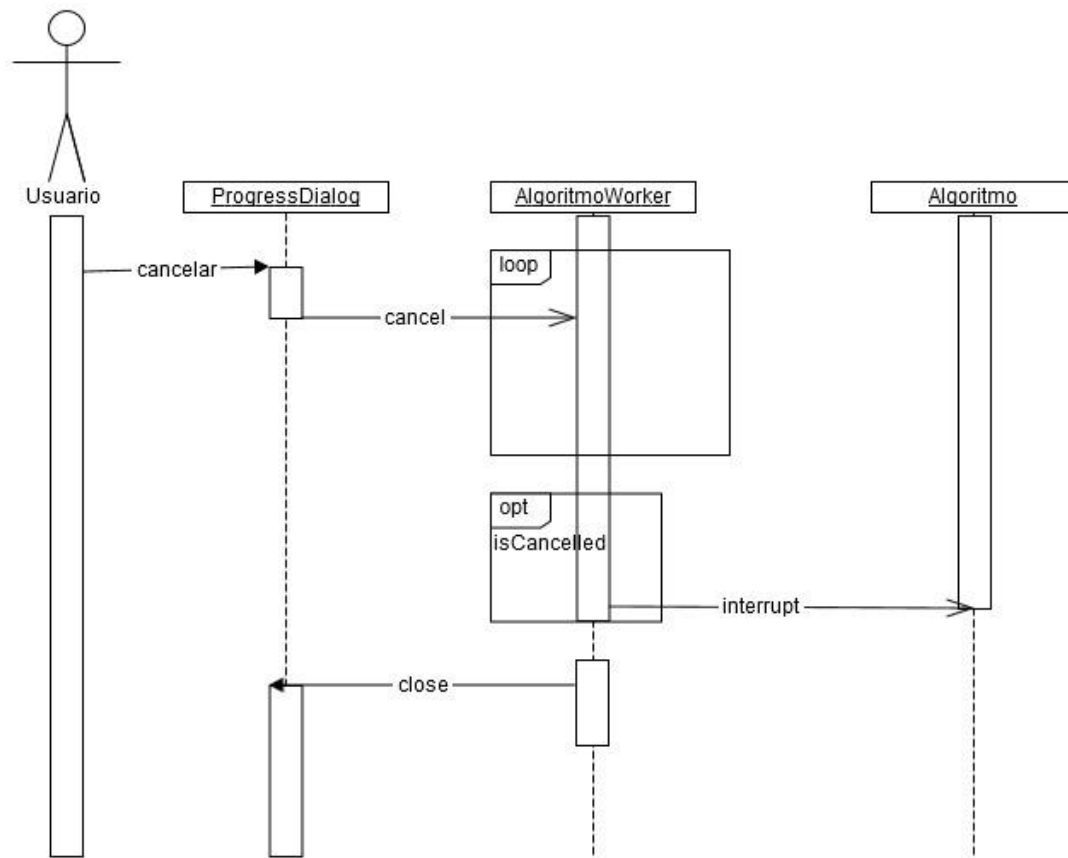


Figura 14 Cancelación de la ejecución

#### 4.5.4. Presentación de resultados parciales

El siguiente diagrama de flujo muestra el flujo de la presentación de los resultados parciales. Se ha implementado el patrón Observer tanto entre la clase era y la clase algoritmo para la notificación de los resultados parciales provenientes de una generación como entre la clase algoritmo y la clase AlgoritmoWorker para la notificación de los resultados parciales provenientes de la generación y de la ejecución de una era.

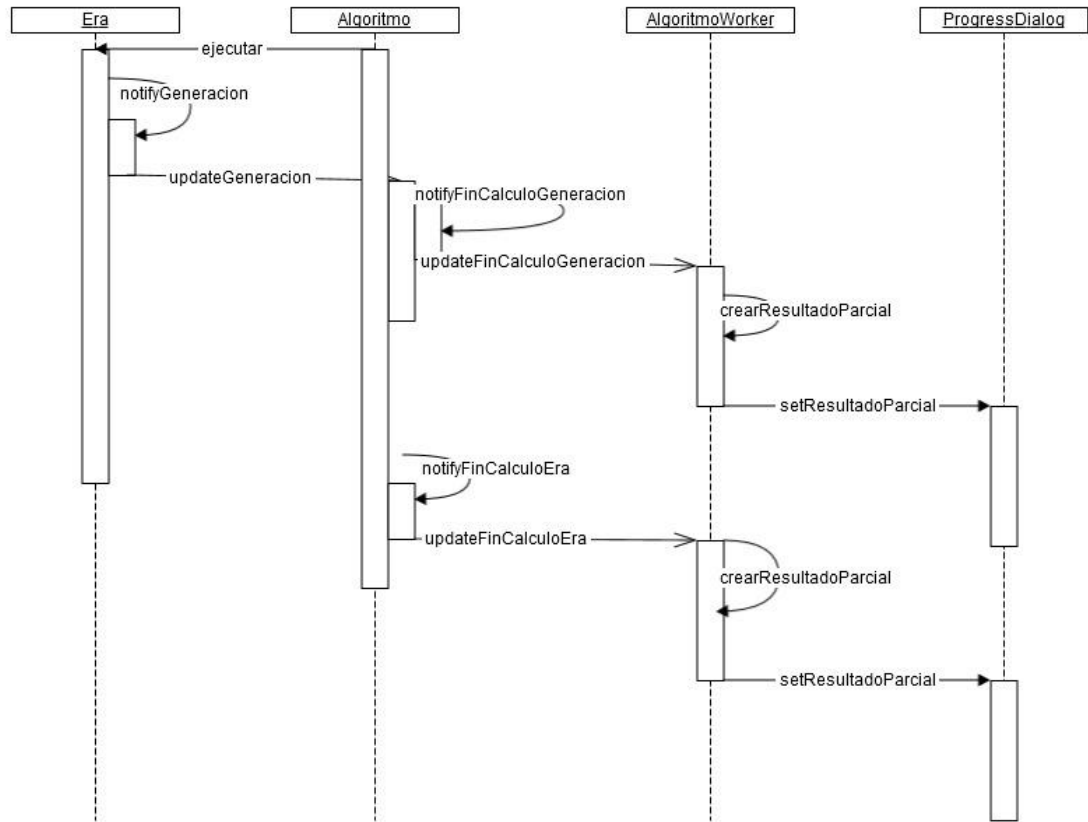


Figura 15 Presentación de resultados parciales

#### 4.6. Interfaz gráfica de usuario

Para implementar la interfaz gráfica de usuario, se ha empleado la librería Swing, incluida en el estándar de Java. Esta librería incluye los componentes gráficos y utilidades de soporte necesarias para gestionar la interacción con el usuario.

El objetivo que se ha buscado en el diseño de la interfaz es la simplicidad, por este motivo se han intentado reducir al mínimo el número de ventanas existentes. La herramienta se ha diseñado compuesta por una ventana principal y dos ventanas secundarias:

- Una ventana principal de la aplicación.

- Una ventana modal para detallar el progreso del cálculo.
- Una ventana modal para detallar los resultados de una era.

#### **4.6.1. Ventana principal de la aplicación**

La ventana principal de la aplicación está destinada a configurar la ejecución, iniciar la ejecución del algoritmo y mostrar los resultados cuando estén disponibles. Esta ventana se ha compuesto por varios paneles, cada uno destinado a una funcionalidad distinta.

1. Panel de configuración del algoritmo.
2. Panel de configuración de la función de coste.
3. Panel de parámetros.
4. Panel de edición de parámetros.
5. Panel de presentación de resultados.

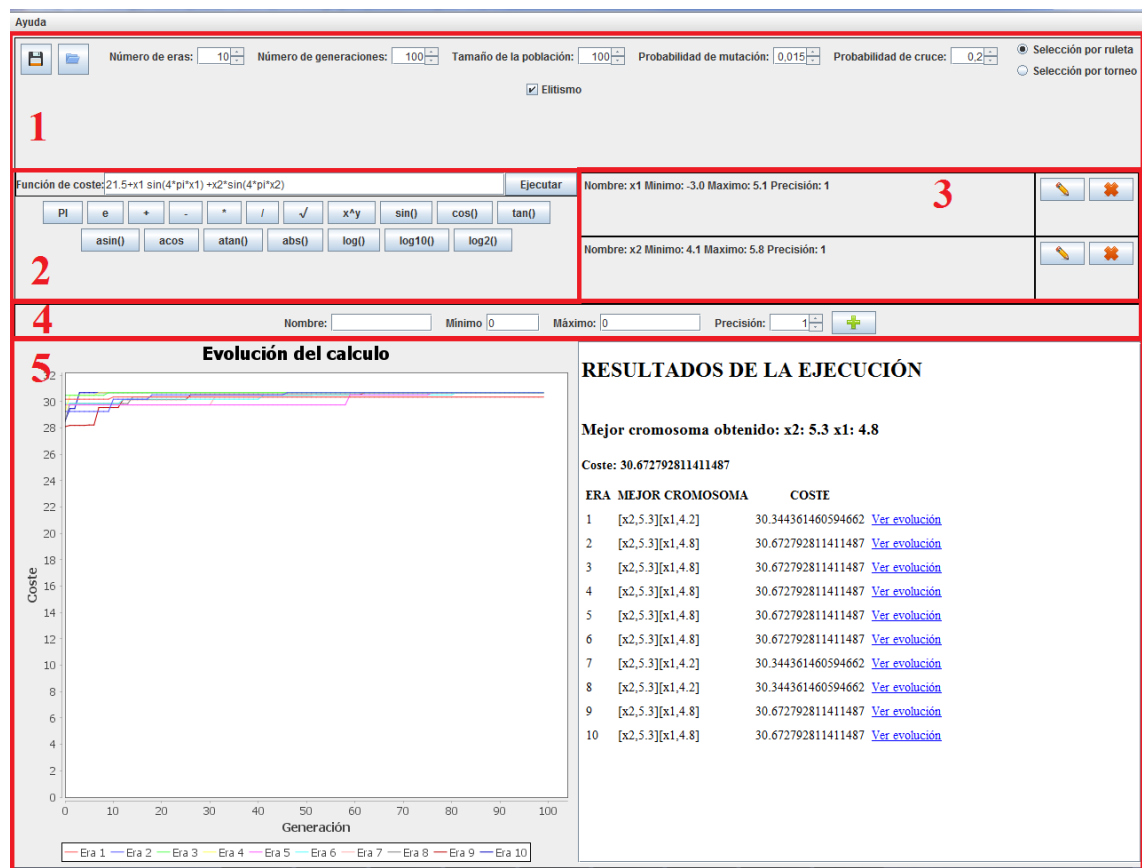


Figura 16 Ventana principal de la aplicación

#### 4.6.2. Ventana de progreso del cálculo

La ventana de progreso se muestra mientras el cálculo se está realizando. Se trata de una ventana modal, que impide el acceso a otras secciones de la herramienta mientras el algoritmo se está ejecutando. Proporciona datos parciales sobre el cálculo y sobre progreso de la ejecución. Además proporciona la posibilidad de cancelar la ejecución del algoritmo. Esta ventana está compuesta por cuatro secciones:

1. Una sección que proporciona información sobre el progreso y tiempo transcurrido durante la ejecución.
2. Una sección que muestra un botón para cancelar la ejecución.
3. Una sección que muestra información sobre la última era calculada.

4. Una sección que muestra información sobre la última generación calculada.



Figura 17 Ventana de progreso



### 4.6.3. Ventana de resultados de una era

En esta ventana se muestra, en detalle y de forma gráfica, la evolución del cálculo realizado para una era, la evolución del valor medio del coste para la población y la evolución de la desviación típica del coste en la población.

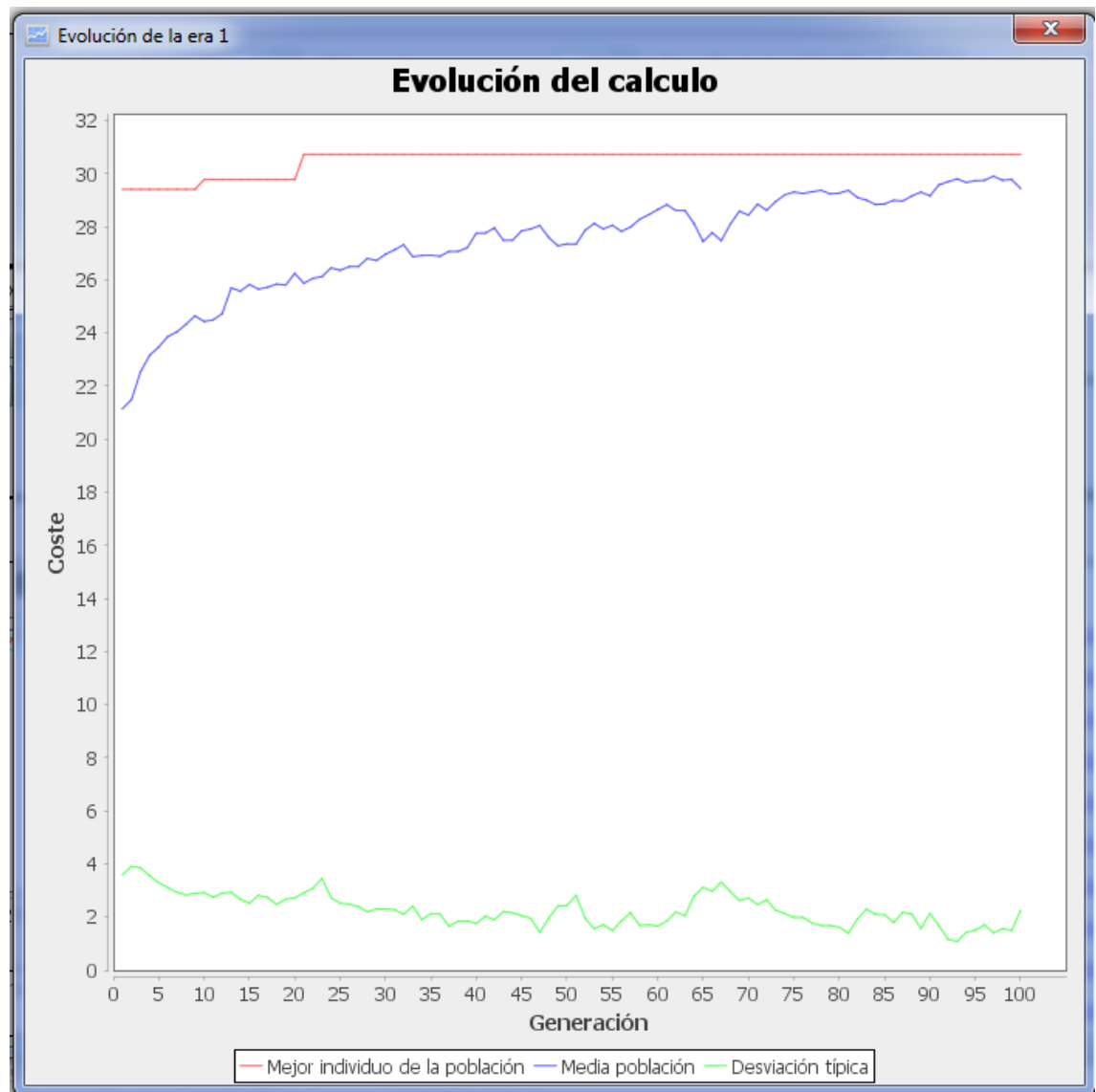


Figura 18 Ventana de resultados de una era



## **5. Funcionamiento de la aplicación**

### ***5.1.Requisitos previos***

Dado que la aplicación está construida en el lenguaje Java, es requisito para su ejecución disponer de una máquina virtual (JRE) adecuada. La versión necesaria debe ser igual o superior a la 1.7.

### ***5.2.Acceso a la aplicación***

Para ejecutar la aplicación debe ejecutarse el fichero en formato jar en el cual se encuentra empaquetada. Este fichero se denomina optimizadorGA.jar.

El modo de ejecución puede ser bien desde el gestor gráfico de ficheros del sistema operativo haciendo doble click en el fichero o bien mediante línea de comandos. El comando adecuado es: `java -jar optimizadorGA.jar`.

Se mostrará la ventana principal de la aplicación.

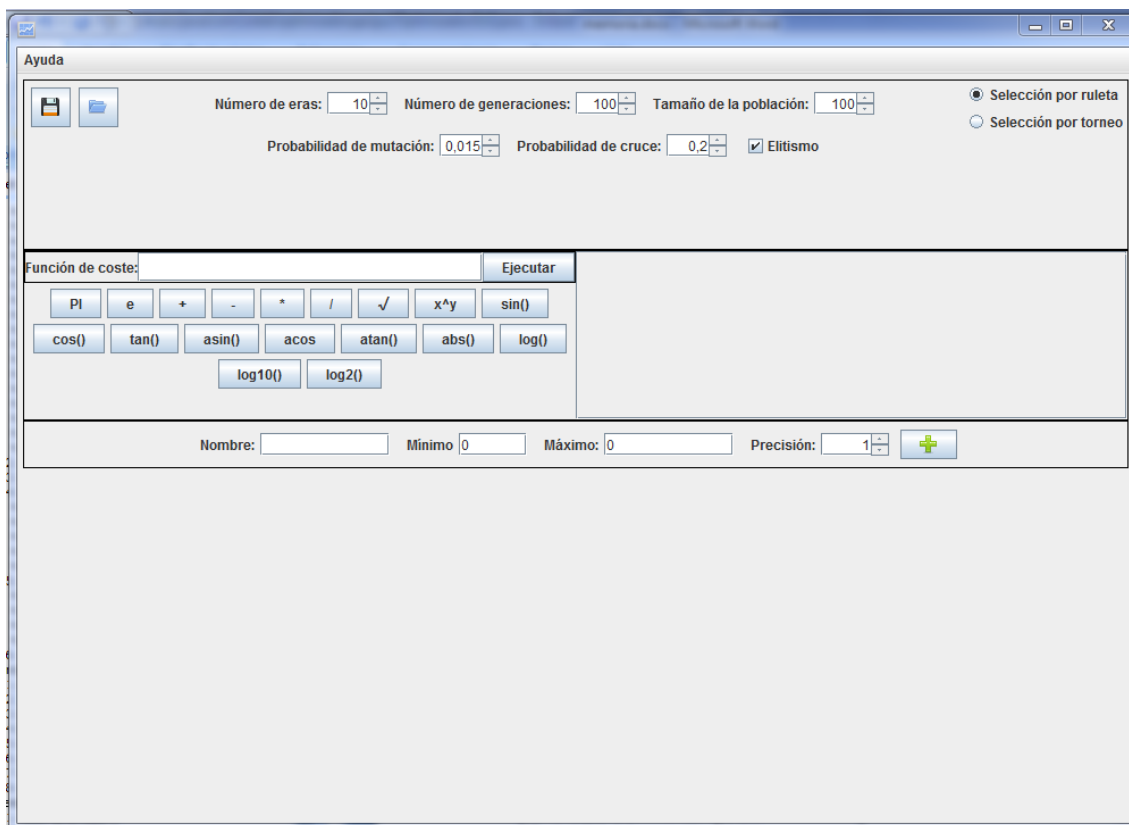


Figura 19 Ventana principal de la aplicación

### 5.3. Introducción de la función de coste

Debe introducirse la función de evaluación en el área reservada para ello. En esta misma área existen una serie de botones destinados a ayudar a introducir las funciones, operadores y constantes predefinidas por la aplicación.

Las funciones y operadores matemáticas predefinidas son:

Función/Operador	Símbolo
Suma	+

Resta	-
Multipliación	*
División	/
Raíz cuadrada	$\sqrt{\phantom{x}}$
Potencia	$\wedge$
Coseno	cos()
Seno	sin()
Tangente	tan()
Arcocoseno	acos()
Arcotangente	atan()
Arcoseno	asin()
Valor absoluto	abs()
Logaritmo neperiano	log()
Logaritmo decimal	log10()

Logaritmo en base 2	log2()
---------------------	--------

Tabla 1 Funciones y operadores predefinidos

Las constantes matemáticas predefinidas en la aplicación son las siguientes:

Constante	Símbolo
Número Pi	PI
Número de Euler	E

Tabla 2 Constantes matemáticas predefinidas

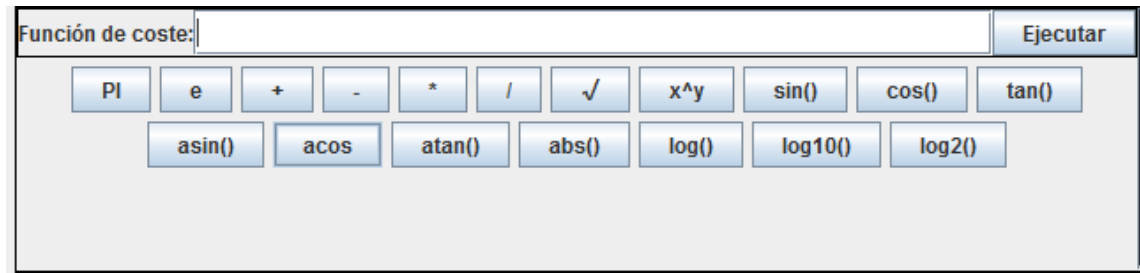



Figura 20 Área de la función de coste

5.4.Codificación del vector de parámetros

Para el correcto funcionamiento de la ejecución, debe especificarse una codificación para el vector de parámetros. La codificación de cada parámetro debe introducirse en el área de introducción de nuevos parámetros.

Nombre:  Mínimo  Máximo:  Precisión:

Figura 21 Área de introducción de nuevos parámetros

Al pulsar el botón , se valida la corrección del nuevo parámetro. En caso de existir alguna incorrección se muestra un mensaje informativo por pantalla. Si la codificación introducida para el parámetro es correcta, se añade el nuevo parámetro a la lista de parámetros declarados.






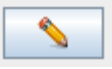
Nombre: x Minimo: -5.0 Maximo: 6.0 Precisión: 1	 
Nombre: y Minimo: 0.0 Maximo: 10.0 Precisión: 1	 

Figura 22 Listado de parámetros declarados

Las validaciones que se realizan sobre un nuevo parámetro son las siguientes:

- El nombre no es vacío.
- El nombre no comienza por un dígito.
- El nombre no coincide con el de otro parámetro ya existente.
- El nombre no contiene el mismo símbolo que algún operador, función o constante predefinida.
- El límite superior no es menor que el límite inferior.

Una vez que se ha declarado un parámetro, este puede eliminarse, para ello debe pulsarse el botón  situado en la misma fila que el parámetro correspondiente.

Asimismo, puede modificarse el parámetro, para ello, debe pulsarse el botón  situado en la misma fila que el parámetro correspondiente. Al pulsarse el botón, se cargan los datos del parámetro a modificar en el área de introducción de parámetros.

### 5.5. Configuración de la ejecución

La aplicación inicialmente contiene una configuración por defecto. Esta configuración es la siguiente:

<b>Número de eras</b>	10
<b>Número de generaciones</b>	100
<b>Tamaño de la población</b>	100
<b>Probabilidad de mutación</b>	0.015
<b>Probabilidad de cruce</b>	0.2
<b>Elitismo</b>	Activado
<b>Tipo de selección</b>	Selección por ruleta

Tabla 3 Configuración por defecto

La configuración puede modificarse en el área de configuración de la aplicación.

The screenshot shows a configuration window with the following settings:


- Número de eras:** 10
- Número de generaciones:** 100
- Tamaño de la población:** 100
- Probabilidad de mutación:** 0.015
- Probabilidad de cruce:** 0.2
- Elitismo:** ☒ Activado
- Tipo de selección:**
  - ☒ Selección por ruleta
  - ☐ Selección por torneo

Figura 23 Área de configuración de la aplicación



### ***5.6.Ejecución***

Una vez configurada la aplicación, puede lanzarse su ejecución. Para ello, se pulsa en el

botón  situado en el área correspondiente a la función de coste.

Al pulsar el botón de ejecución, se valida que la configuración de la función de coste es correcta. Los aspectos que se validan de la función de coste son los siguientes:

- Demasiados puntos decimales en un valor numérico.
- Concuerdan Los símbolos de paréntesis que se abren con los que se cierran.
- Los operadores no se aplican sobre un número o un parámetro.
- El uso de parámetros no declarados.

Una vez realizada la validación, comienza la ejecución del algoritmo. Se muestra una nueva ventana que proporciona información sobre el progreso del algoritmo.

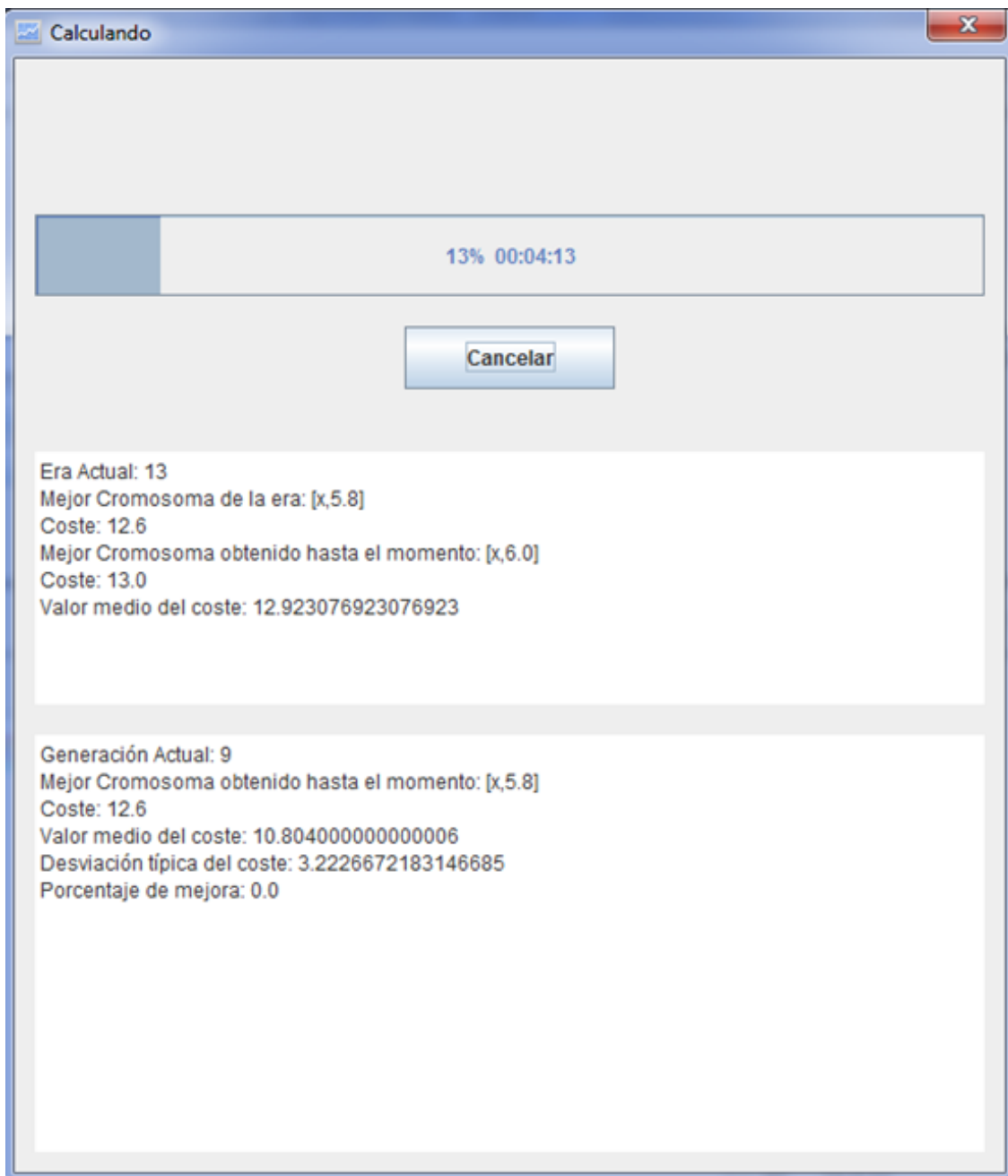


Figura 24 Ventana de progreso

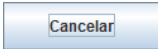
La información que se muestra durante la ejecución es la siguiente:

- Porcentaje completado de la ejecución.
- Tiempo de ejecución transcurrido.

- Número de era que se está calculando.
- Mejor cromosoma de la era actual.
- Coste del mejor cromosoma obtenido en la era actual.
- Mejor cromosoma obtenido en todas las eras.
- Coste del mejor cromosoma obtenido durante toda la ejecución.
- Valor medio del coste en la era actual.
- Número de generación que se está calculando.
- Mejor cromosoma obtenido en la generación.
- Coste del mejor cromosoma obtenido en la generación.
- Valor medio del coste de los cromosomas de la población evolucionada.
- Desviación típica del coste de los cromosomas de la población evolucionada.
- Porcentaje de mejora respecto del mejor cromosoma de la generación anterior.

Esta ventana se cierra automáticamente en el momento en que finalice la ejecución.

### ***5.7.Cancelación de la ejecución***

La ejecución del algoritmo puede cancelarse en todo momento. Para ello debe pulsarse el botón  presente en la ventana de progreso. Al cancelar la ejecución, se cierra la ventana de progreso y se detiene la ejecución.

### ***5.8.Resultados de la ejecución***

Al finalizar la ejecución del algoritmo, bien porque este llegue a su fin o bien porque se cancele su ejecución, se muestran en la ventana principal los resultados obtenidos durante el cálculo. Estos resultados se muestran en la sección destinada para ello.

Esta sección se compone de dos subsecciones. Una parte muestra de forma gráfica la evolución del mejor cromosoma obtenido en cada una de las eras a lo largo de todas las generaciones. La otra sección muestra el mejor cromosoma obtenido durante toda la ejecución así como el mejor cromosoma obtenido en cada una de las eras y un enlace para mostrar detalles sobre la evolución de cada era en particular.

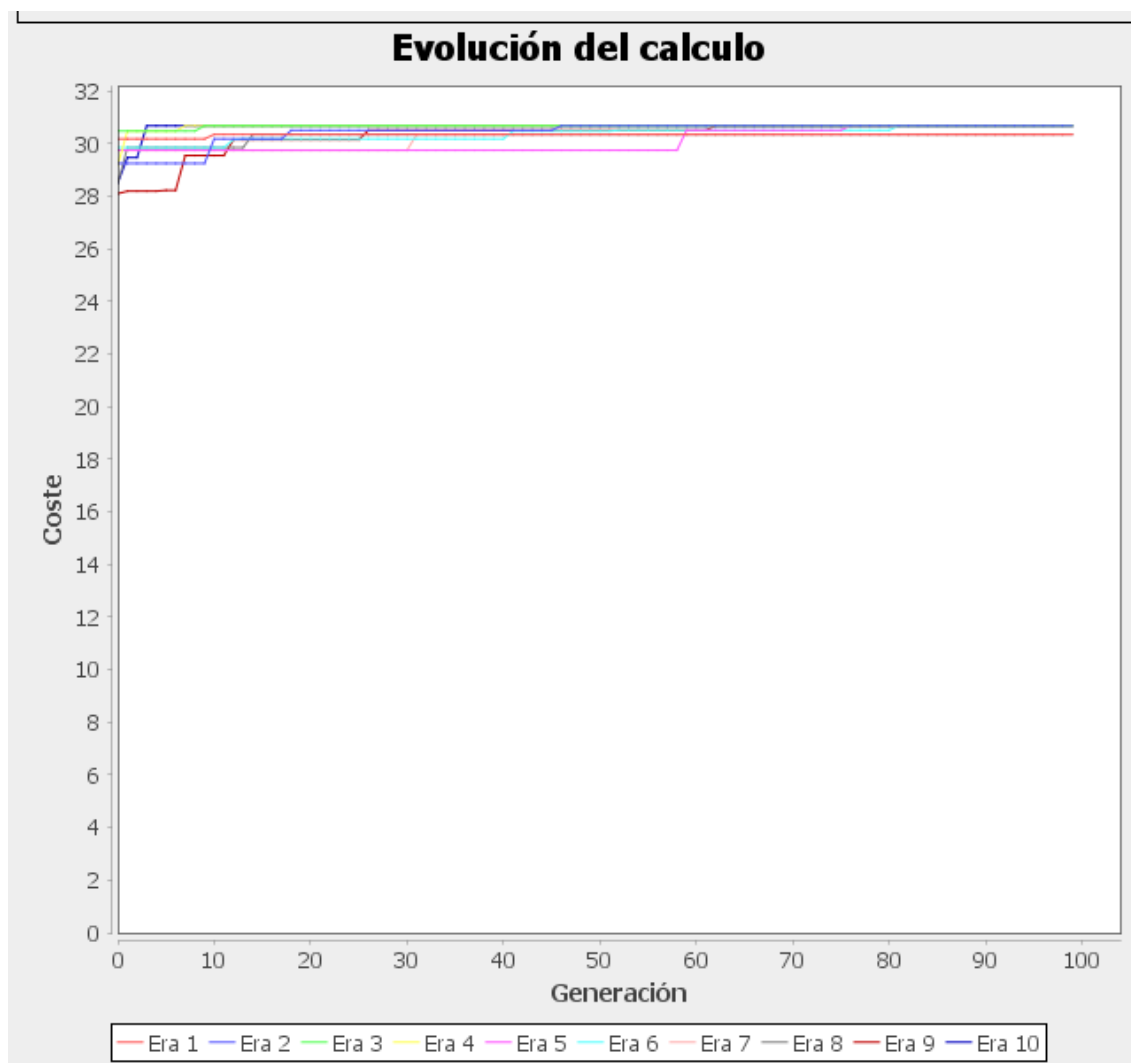


Figura 25 Sección gráfica de los resultados

## RESULTADOS DE LA EJECUCIÓN

**Mejor cromosoma obtenido: x2: 5.3 x1: 4.8**

**Coste: 30.672792811411487**

ERA	MEJOR CROMOSOMA	COSTE
1	[x2,5.3][x1,4.2]	30.344361460594662 <a href="#">Ver evolución</a>
2	[x2,5.3][x1,4.8]	30.672792811411487 <a href="#">Ver evolución</a>
3	[x2,5.3][x1,4.8]	30.672792811411487 <a href="#">Ver evolución</a>
4	[x2,5.3][x1,4.8]	30.672792811411487 <a href="#">Ver evolución</a>
5	[x2,5.3][x1,4.8]	30.672792811411487 <a href="#">Ver evolución</a>
6	[x2,5.3][x1,4.8]	30.672792811411487 <a href="#">Ver evolución</a>
7	[x2,5.3][x1,4.2]	30.344361460594662 <a href="#">Ver evolución</a>
8	[x2,5.3][x1,4.2]	30.344361460594662 <a href="#">Ver evolución</a>
9	[x2,5.3][x1,4.8]	30.672792811411487 <a href="#">Ver evolución</a>
10	[x2,5.3][x1,4.8]	30.672792811411487 <a href="#">Ver evolución</a>

**Figura 26 Mejores cromosomas de la ejecución**

### ***5.9.Detalles de la evolución de una era***

Al pulsar en el enlace “Ver evolución”, situado junto al mejor cromosoma de cada era, se abre una ventana que muestra detalles sobre la evolución del cálculo de la era. En concreto, muestra de forma gráfica, a lo largo de las generaciones, el mejor cromosoma obtenido, la media de la población y la desviación típica.

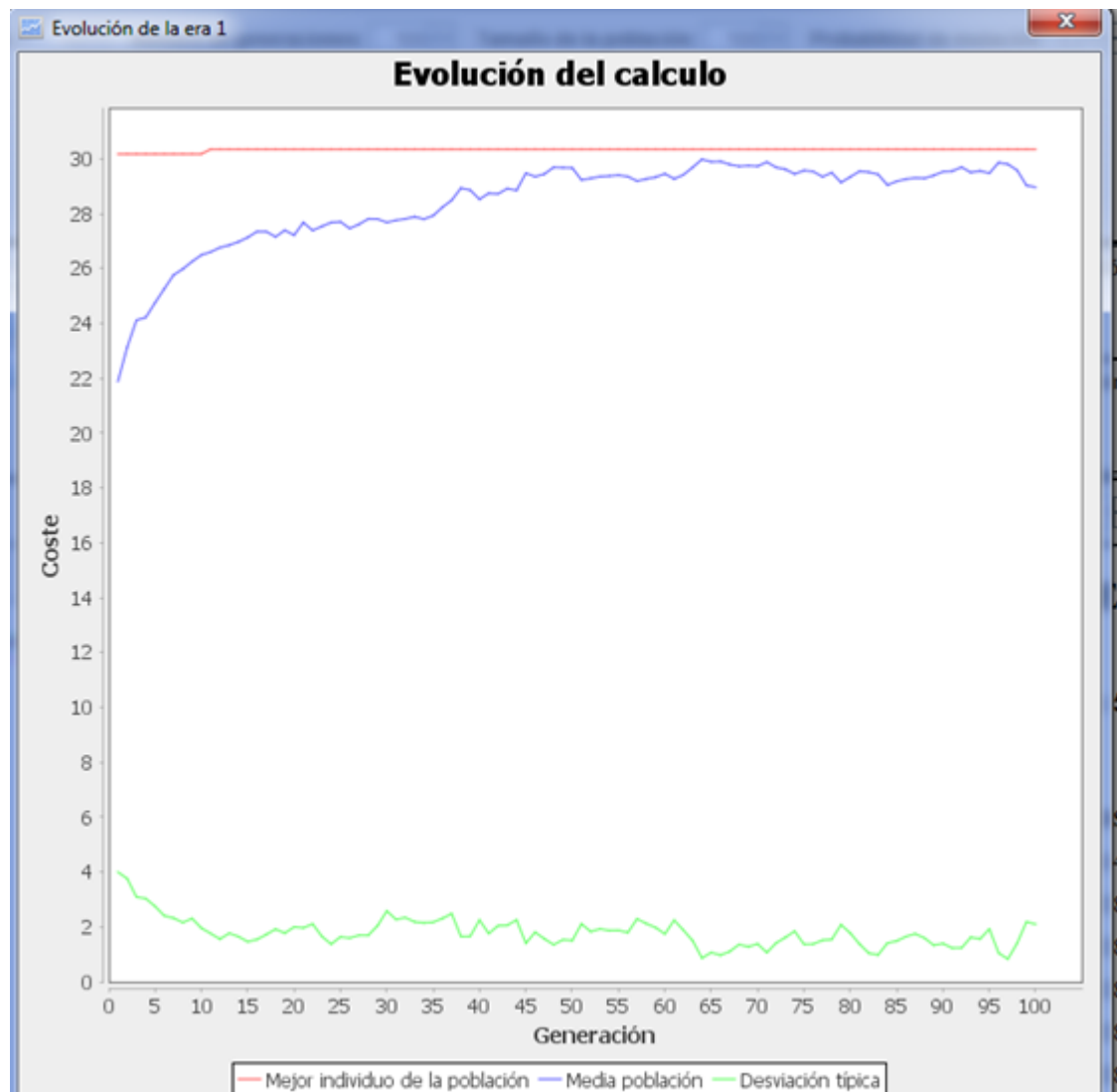



Figura 27 Evolución del cálculo de una era

### 5.10. Guardado de datos



Pueden almacenarse los datos de la aplicación pulsándose el icono . Al pulsar en el botón se abre un diálogo para seleccionar el archivo en el que se desea guardar.

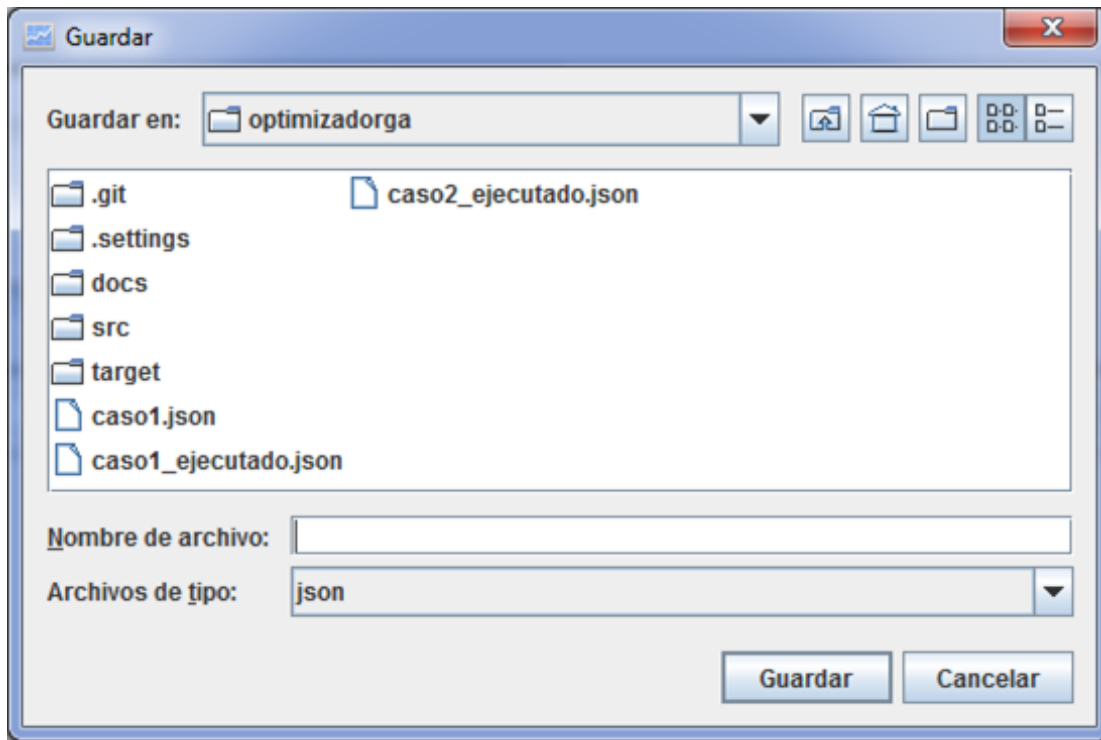


Figura 28 Diálogo de guardado de datos

Al introducir un nombre para el archivo y pulsar en el botón Guardar, se almacenan, en un fichero en formato json, los datos de la aplicación. Si se ha realizado una ejecución, también se almacenarán los resultados correspondientes a los cálculos realizados.

### 5.11. *Carga de datos guardados*

Para recuperar los datos almacenados en un fichero de datos, debe pulsarse el icono



. Al pulsarse el botón, se abre un diálogo para seleccionar el archivo.



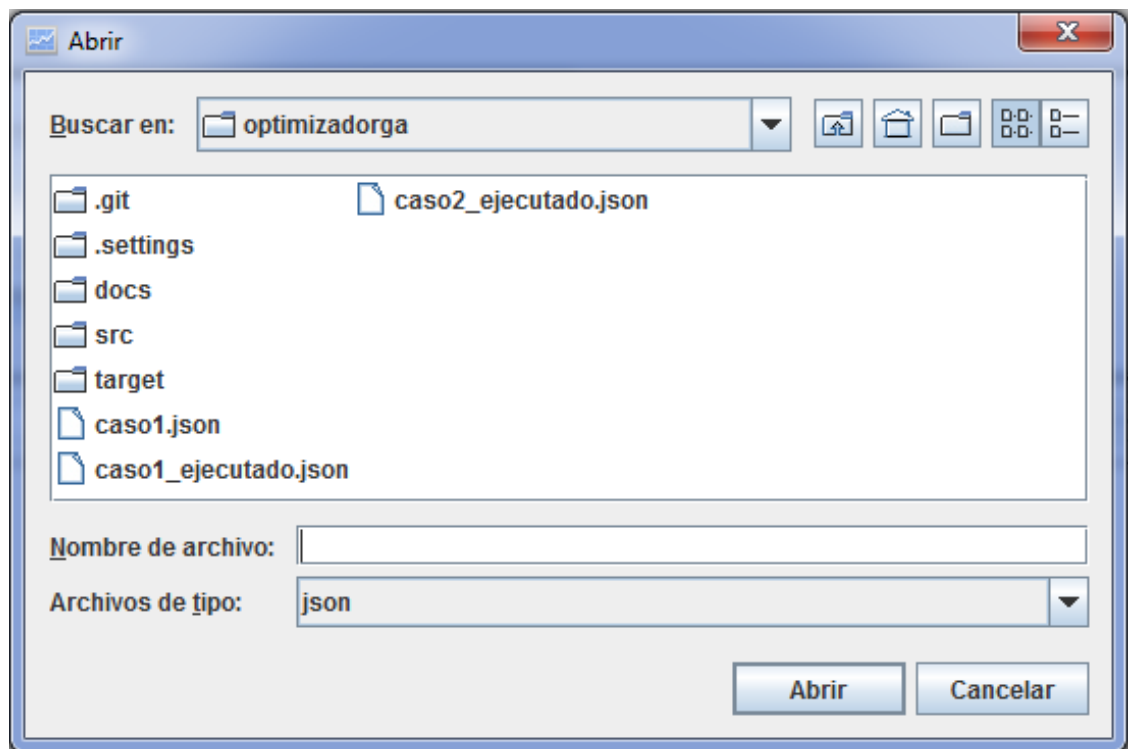


Figura 29 Diálogo de apertura de datos

Al seleccionar el archivo y pulsar en el botón Abrir, se cargan los datos correspondientes en la aplicación.

### 5.12. Ayuda de la aplicación

El manual de usuario es accesible en todo momento desde la propia aplicación. Para acceder, debe pulsarse en la barra de herramientas situada en la parte superior y acceder al apartado ayuda/ayuda. Al pulsarse esta opción se abre el manual de usuario.

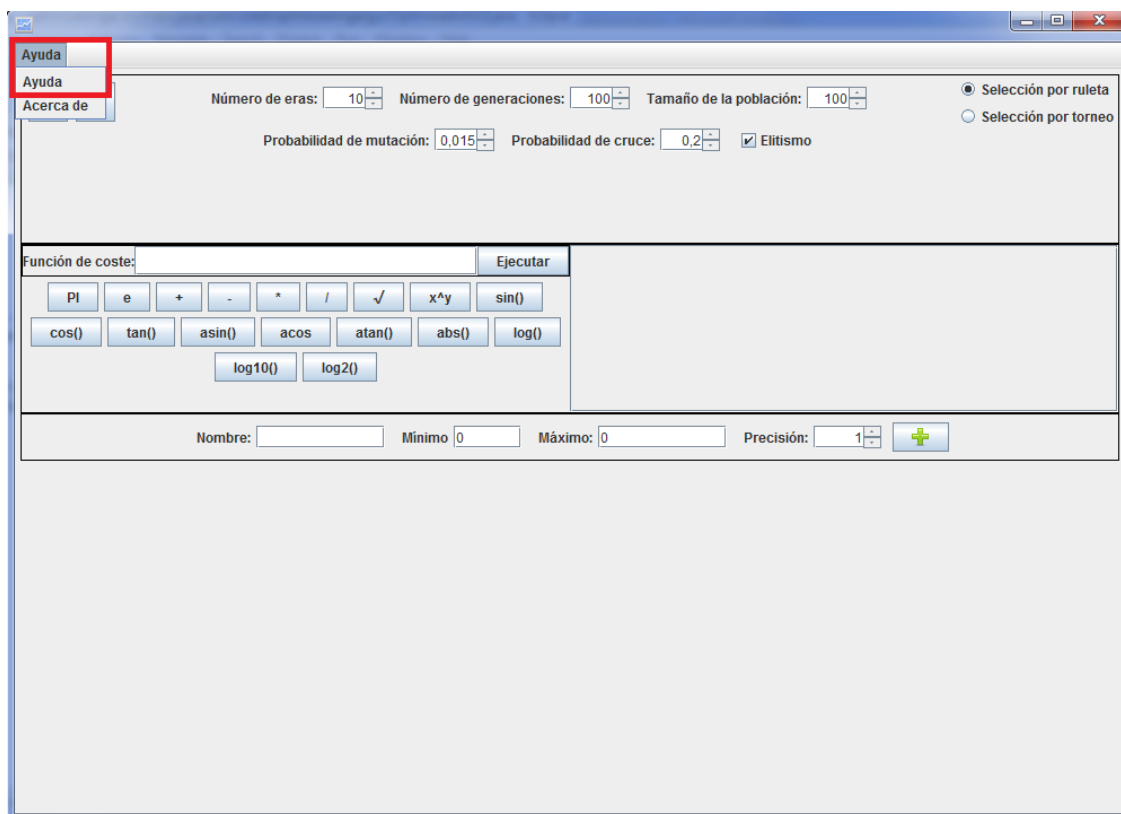


Figura 30 Acceso a la ayuda de la aplicación

## 6. Ejemplos de ejecución

Para probar la ejecución del algoritmo se han utilizado varias diversas funciones proporcionadas en las directrices del trabajo. A continuación se muestran los resultados de estas pruebas.

Las pruebas han sido realizadas utilizando la configuración por defecto. Esto es:

<b>Número de eras</b>	10
<b>Número de generaciones</b>	100
<b>Tamaño de la población</b>	100
<b>Probabilidad de mutación</b>	0.015
<b>Probabilidad de cruce</b>	0.2
<b>Elitismo</b>	Activado
<b>Tipo de selección</b>	Selección por ruleta

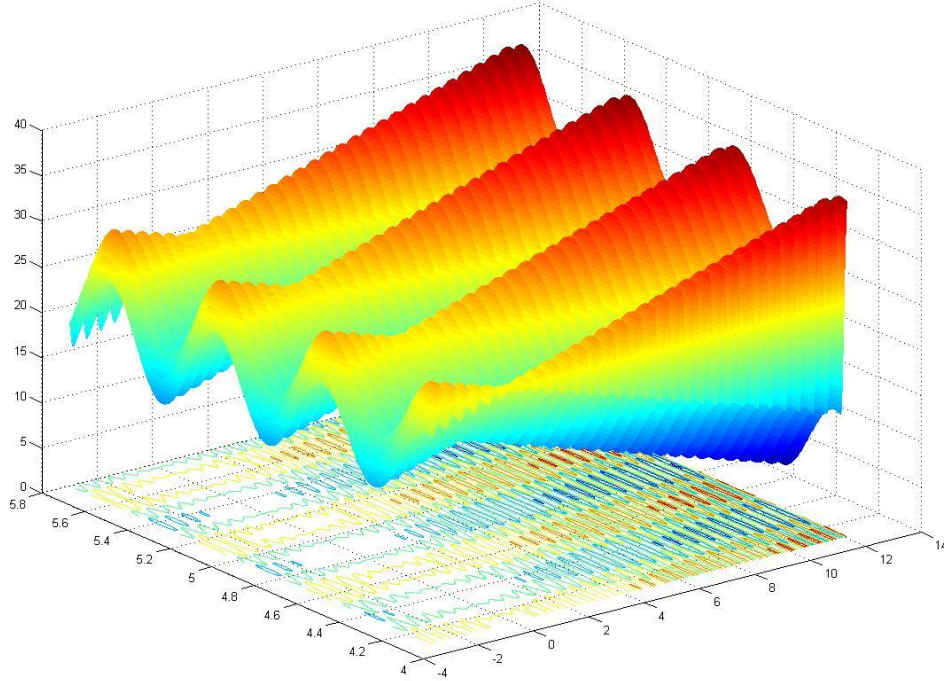
Tabla 4 Configuración por defecto

### 6.1.Caso de prueba 1

**Función de evaluación:**

$$f(x_1, x_2) = 21.5 + x_1 \cdot \text{sen}(4 \cdot \pi \cdot x_1) + x_2 \cdot \text{sen}(4 \cdot \pi \cdot x_2)$$

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



**Figura 31 Caso de prueba 1**

Como se puede observar, la función tiene varios máximos con valores próximos a 38,77

El mejor cromosoma obtenido por la herramienta con la configuración por defecto es el formado por  $x_1=12,1$  y  $x_2=5,6$  con un coste de 38,33.

## 6.2. Caso de prueba 2

**Función de evaluación:**

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

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

Como se puede observar, la función toma su valor máximo cuando todos los parámetros  $x_i$  tienen el valor más próximo a 0. Se trata del cromosoma formado por [0, 2.1, 0, 0, 0, 0]. El coste de este cromosoma es de 95.59.

El mejor cromosoma obtenido por la herramienta con la configuración por defecto es el formado por los valores {-0.21, 2.1, 0.5, 0.0, -0.2, -0.06} con un coste de 95,25.

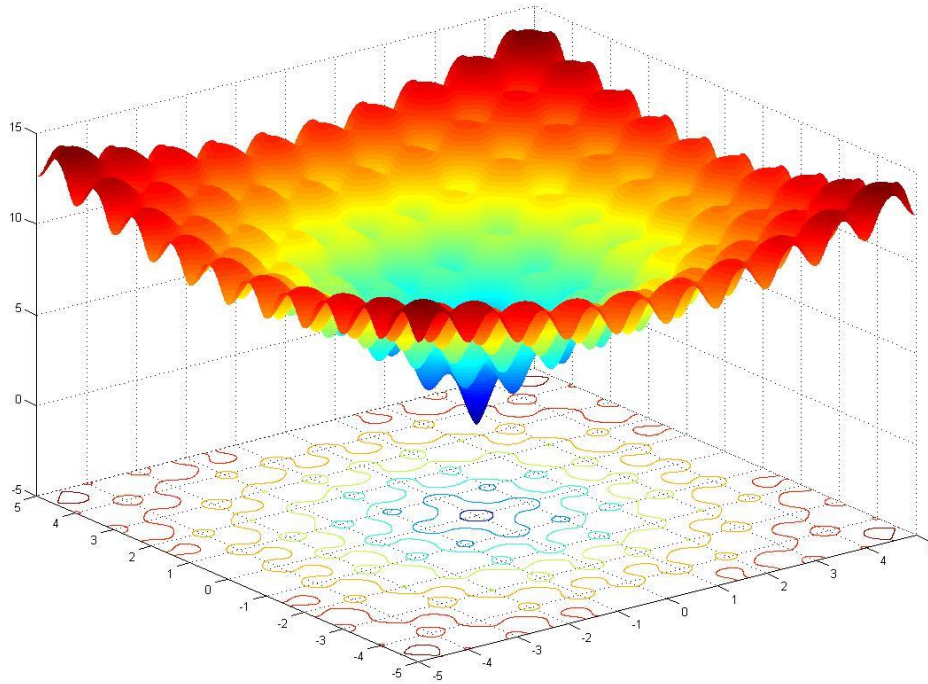
### ***6.3.Caso de prueba 3: Función de Ackley***

**Función de evaluación:**

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

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

$$S = \{-5 \leq x_1 \leq 5; -5 \leq x_2 \leq 5\}$$



**Figura 32 Caso de prueba 3. Función de Ackley**

La función tiene cuatro máximos en los valores  $\{(-4.6, 4.6), (4.6, -4.6), (-4.6, -4.6), (4.6, 4.6)\}$  con costes de 14.30.

El mejor cromosoma obtenido por la herramienta con la configuración por defecto es el formado por los valores  $x_1 = 4.6$  y  $x_2 = -4.6$  con un coste de 14.30.

La función de Ackley suele usarse para probar algoritmos de optimización en su variante de minimización. Dispone de varios mínimos locales por lo que conlleva dificultades para los algoritmos evolutivos. Dispone de un mínimo global en (0,0) con un coste de 0.

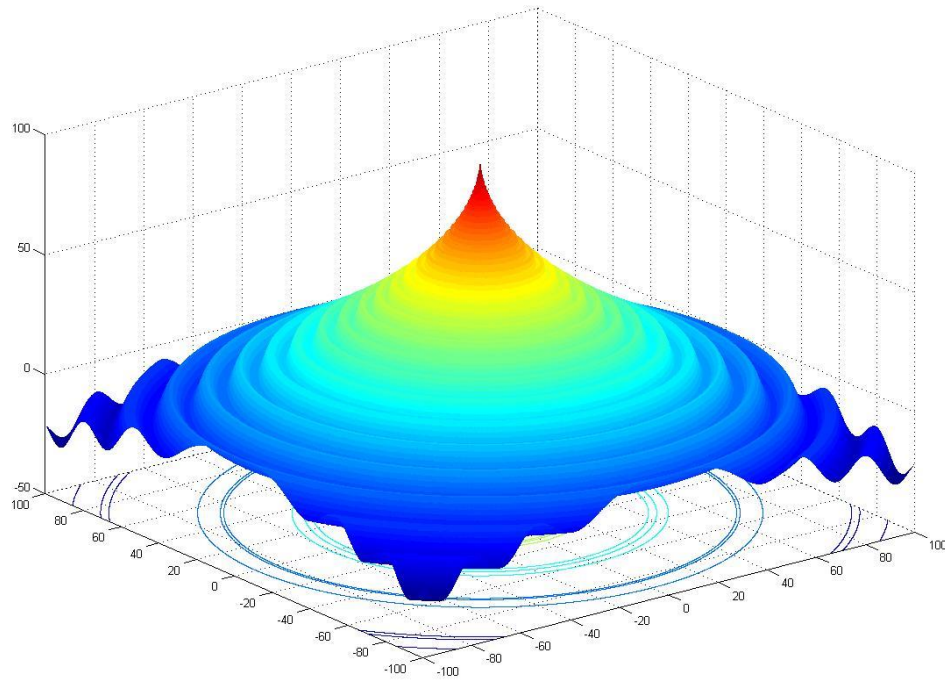
Al probar la herramienta para minimizar la función de Ackley (optimizar el valor inverso de la función) se obtiene el mejor cromosoma en (0.0, 0.0) con un coste de 0

#### 6.4.Caso de prueba 4: Función de Schaffer

**Función de evaluación:**

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

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



**Figura 33 Caso de prueba 4. Función de Schaffer**

Como puede observarse, esta función dispone de numerosos máximos locales y un máximo global claro alrededor del punto (0,0) con un valor de 100.

El mejor cromosoma obtenido por la herramienta con la configuración por defecto es el formado por los valores  $x_1 = -0.1$  y  $x_2 = -0.4$  con un coste asociado de 93.1

Se observa que el mejor valor de las distintas eras es relativamente bajo, con cromosomas cuyos costes oscilan entre 84 y 93.

ERA	MEJOR CROMOSOMA	COSTE	
1	[x1,-0.2][x2,1.3]	88.17640770634893	<a href="#">Ver evolución</a>
2	[x1,2.4][x2,0.8]	83.7174784340698	<a href="#">Ver evolución</a>
3	[x1,1.0][x2,0.1]	89.92916719732565	<a href="#">Ver evolución</a>
4	[x1,0.1][x2,0.6]	91.48261393130707	<a href="#">Ver evolución</a>
5	[x1,-0.1][x2,-0.4]	93.10124238116529	<a href="#">Ver evolución</a>
6	[x1,0.7][x2,1.9]	84.72666789190603	<a href="#">Ver evolución</a>
7	[x1,-0.6][x2,-1.4]	86.8038525436497	<a href="#">Ver evolución</a>
8	[x1,0.9][x2,1.7]	85.8789760871191	<a href="#">Ver evolución</a>
9	[x1,2.2][x2,1.0]	84.45204911606933	<a href="#">Ver evolución</a>
10	[x1,-0.2][x2,0.8]	90.28902339772215	<a href="#">Ver evolución</a>

**Figura 34 Resultados con la configuración por defecto**

Al modificar la configuración y aumentar el número de generaciones a 1000, el mejor cromosoma obtenido es el formado por los valores  $x1 = 0$  y  $x2 = 0$  con un coste asociado de 100, encontrando el máximo global a partir de la generación número 500.



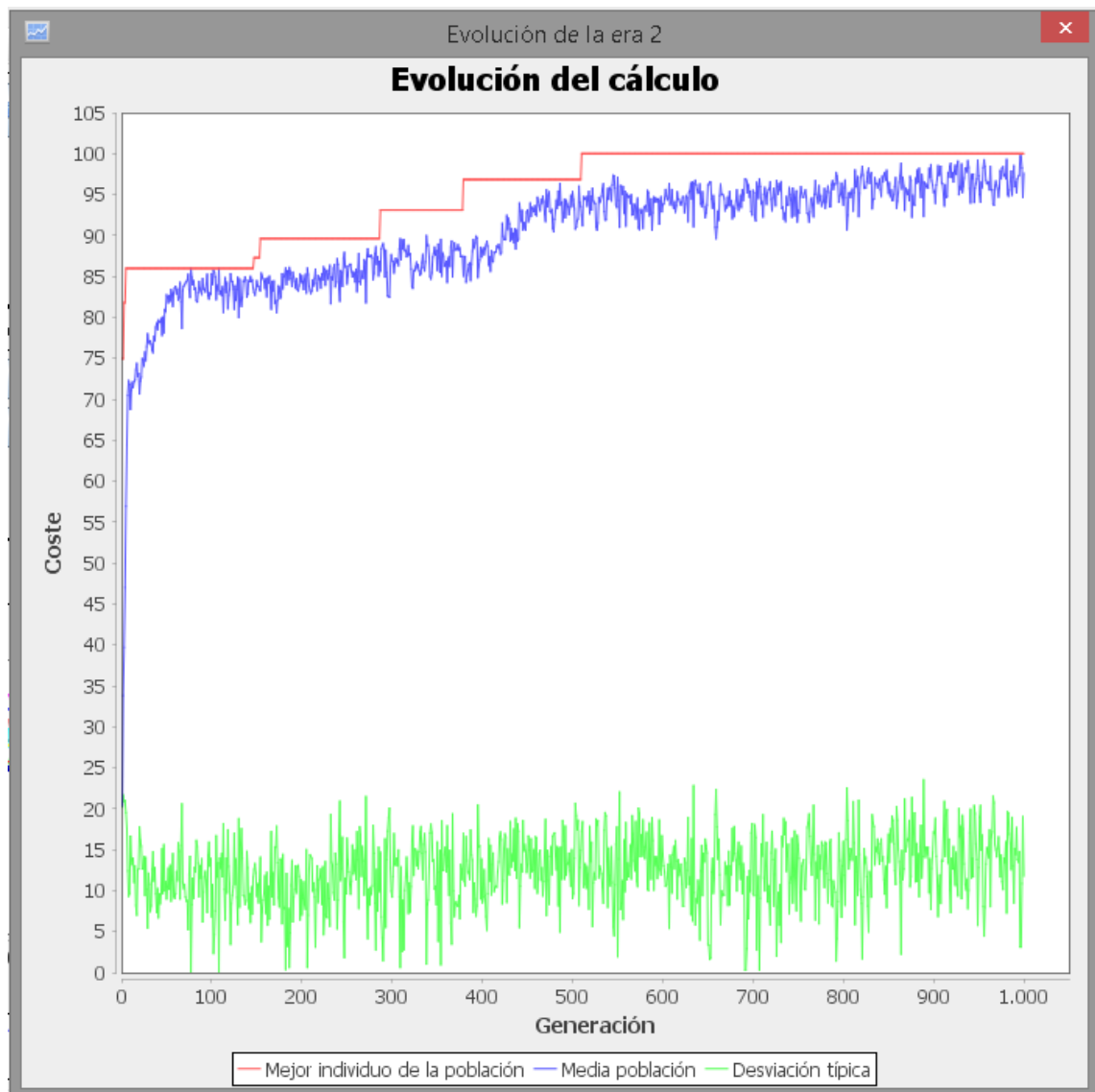


Figura 35 Evolución de la mejor era

Asimismo se observa que los resultados de todas las eras mejoran en general, obteniéndose cromosomas con costes entre 92 y 100.

ERA	MEJOR CROMOSOMA	COSTE
1	[x1,-0.2][x2,-0.1]	95.10156297372293 <a href="#">Ver evolución</a>
2	[x1,-0.0][x2,-0.0]	100.0 <a href="#">Ver evolución</a>
3	[x1,-0.2][x2,0.2]	94.24226738809473 <a href="#">Ver evolución</a>
4	[x1,-0.5][x2,-0.1]	92.80330476463598 <a href="#">Ver evolución</a>
5	[x1,-0.0][x2,0.1]	96.83224920799687 <a href="#">Ver evolución</a>
6	[x1,-0.0][x2,-0.0]	100.0 <a href="#">Ver evolución</a>
7	[x1,0.2][x2,-0.0]	95.08610529939716 <a href="#">Ver evolución</a>
8	[x1,-0.2][x2,0.1]	95.10156297372293 <a href="#">Ver evolución</a>
9	[x1,-0.1][x2,0.1]	96.0664103170124 <a href="#">Ver evolución</a>
10	[x1,-0.1][x2,0.1]	96.0664103170124 <a href="#">Ver evolución</a>

Figura 36 Resultados con 1000 generaciones

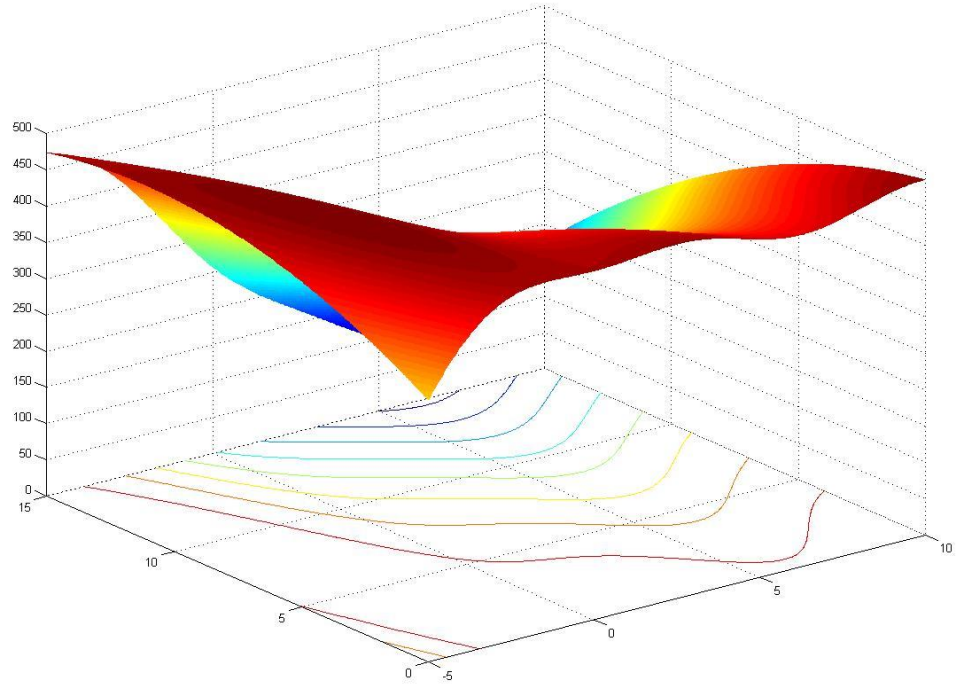
### 6.1. Caso de prueba 5

#### Función de evaluación:

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

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

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



**Figura 37 Caso de prueba 5**

Existen numerosos valores que hacen máximo de la función con un valor cercano a 500.

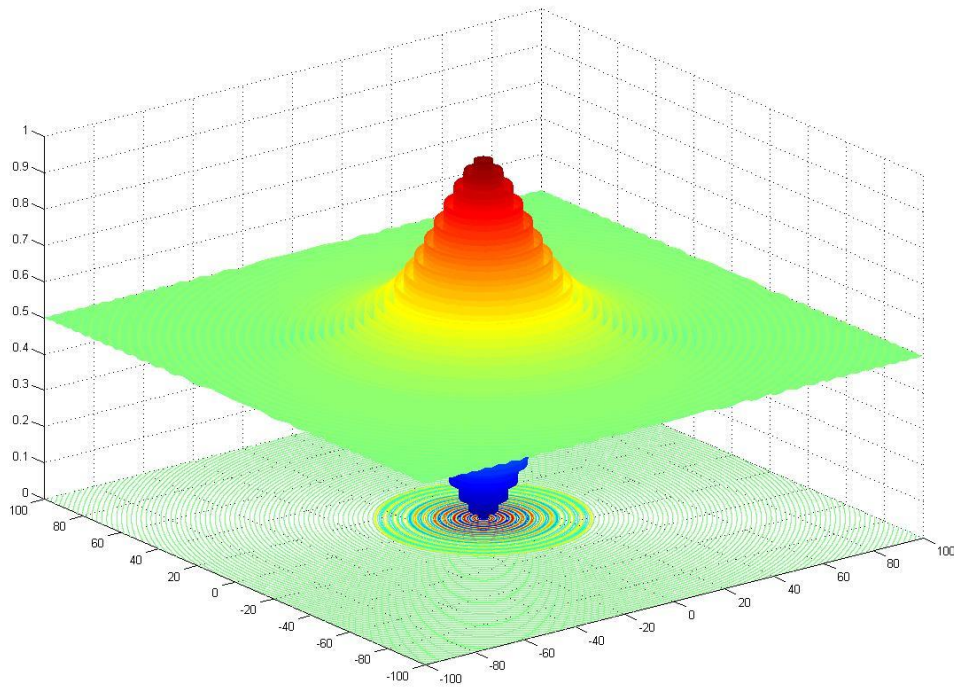
El mejor cromosoma obtenido por la herramienta con la configuración por defecto es el formado por los valores  $x_1 = 9.4$  y  $x_2 = 2.5$  con un coste asociado de 499.88649.

## 6.2. Caso de prueba 6

**Función de evaluación:**

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

$$S = \{x, y \in [-100, 100]\}$$



**Figura 38 Caso de prueba 6**

Puede observarse que la función dispone de un máximo en (0,0) con un coste de 1.

El mejor cromosoma obtenido por la herramienta con la configuración por defecto es el formado por los valores  $x = 0.1$  e  $y = -3.1$  con un coste asociado de 0.9889.

## 7. Planificación y presupuesto

Desde el comienzo el trabajo se ha planteado en varias fases:

1. Fase de planificación y documentación. En esta fase se han consultado referencias bibliográficas, se han valorado diversas posibilidades técnicas y se ha planteado el resto del trabajo.
2. Fase de desarrollo de un motor de ejecución para el algoritmo genético. En esta fase se ha abordado el análisis, desarrollo y pruebas de los módulos encargados de la ejecución del algoritmo. Al final de esta fase se ha obtenido el proceso que ejecuta el algoritmo genético.
3. Fase de desarrollo de una herramienta gráfica básica. En esta fase se ha diseñado e implementado una interfaz gráfica capaz de dar soporte a las funcionalidades básicas de la herramienta. A su vez se ha integrado esta interfaz con el motor que se ha creado anteriormente. Al final de esta fase se ha obtenido un ejecutable capaz de introducir los parámetros, la función de evaluación, ejecutar el algoritmo genético y mostrar de forma básica los resultados de la ejecución.
4. Fase de mejora de la interfaz gráfica. En esta fase se han abordado las mejoras funcionales en el ejecutable anteriormente. En concreto, se han abordado los aspectos correspondientes al procesamiento de los resultados, el guardado en ficheros y la carga de ficheros. Al final de esta fase se ha obtenido el producto final.
5. Fase de documentación. En esta fase se han recopilado los datos correspondientes al desarrollo para redactar de forma definitiva esta memoria. Al finalizar esta fase se ha obtenido la memoria del proyecto.

A continuación se puede ver un diagrama de Gantt con la evolución de las diversas fases.

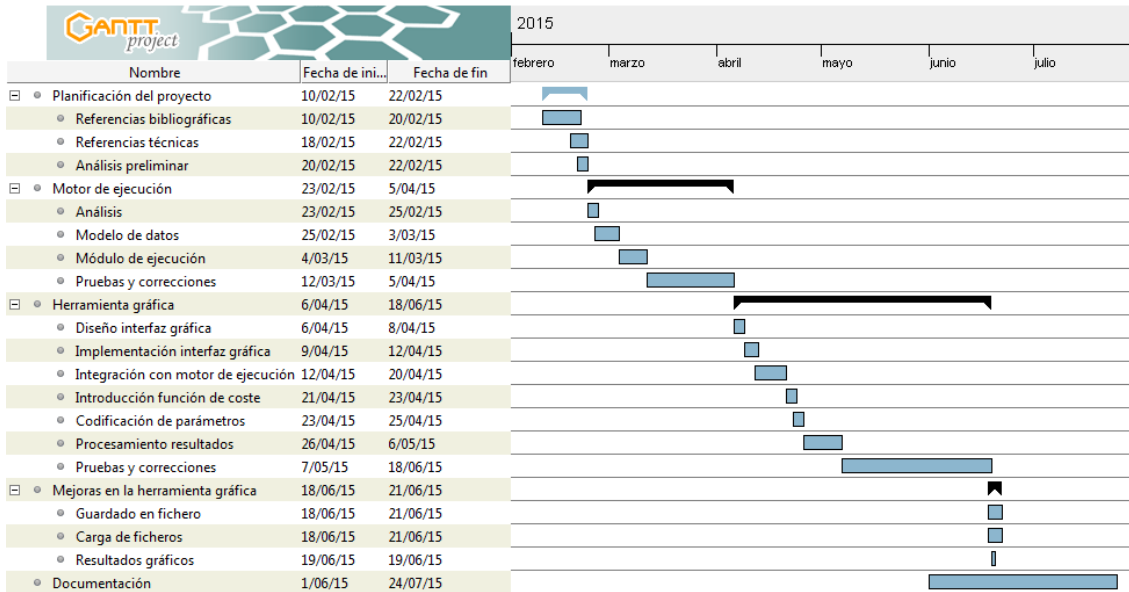


Figura 39 Fases y tareas del proyecto

## 7.1. Fase de planificación y documentación del proyecto

En esta fase se ha buscado información, consultado referencias bibliográficas, evaluado diversas posibilidades técnicas y se ha planteado el modo en que se aborda el resto del proyecto.

### 7.1.1. Búsqueda de referencias bibliográficas

En esta tarea se han buscado y consultado referencias bibliográficas con el objetivo de conocer los algoritmos genéticos.

Fecha de comienzo: 10/02/2015

Fecha de fin: 20/02/2015

Tiempo dedicado: 20 horas.

### **7.1.2. Búsqueda de referencias técnicas**

En esta tarea se han buscado y evaluado diversas opciones para decidir el entorno técnico que se ha utilizado. Se han valorado el lenguaje de programación a emplear y la posibilidad de desarrollar librerías propias o externas según requisitos del proyecto. Asimismo se incluye en esta tarea el trabajo dedicado al aprendizaje necesario para lograr el correcto funcionamiento de estas librerías externas.

Fecha de comienzo: 18/02/2015

Fecha de fin: 22/02/2015

Tiempo dedicado: 8 horas.

### **7.1.3. Análisis preliminar**

En esta tarea se han valorado diversos métodos de trabajo y se ha optado por dividir el trabajo en las diversas fases que se están presentando.

Fecha de comienzo: 20/02/2015

Fecha de fin: 22/02/2015

Tiempo dedicado: 6 horas.

## ***7.2.Fase de desarrollo de motor de ejecución***

En esta fase se ha desarrollado una librería capaz de dar soporte a los algoritmos necesarios para cumplir con los requisitos.

### **7.2.1. Análisis**

En esta tarea se han analizado los requisitos y se ha obtenido la estructura de las capas que dan soporte a la aplicación.

Fecha de inicio: 23/02/2015

Fecha de fin: 25/02/2015

Tiempo dedicado: 6 horas.

### **7.2.2. Desarrollo del modelo de datos**

En esta fase se ha codificado una solución a la capa del modelo de datos de la aplicación.

Fecha de inicio: 25/02/2015

Fecha de fin: 03/03/2015

Tiempo dedicado: 15 horas.

### **7.2.3. Desarrollo del módulo de ejecución**

En esta tarea se ha codificado la capa que da solución al módulo de ejecución de la aplicación.

Fecha de inicio: 4/03/2015

Fecha de fin: 11/03/2015

Tiempo dedicado: 26 horas.

### **7.2.4. Pruebas y correcciones**

En esta tarea se han sometido a pruebas de integración y de rendimiento el modelo de datos y el módulo de ejecución. Se han aplicado los cambios y correcciones pertinentes.

Fecha de inicio: 12/03/2015

Fecha de fin: 5/04/2015



Tiempo dedicado: 20 horas

### ***7.3.Desarrollo de la herramienta gráfica***

En esta fase se ha desarrollado una herramienta gráfica que de soporte a los requisitos de usuario de la herramienta.

#### **7.3.1. Diseño de la interfaz gráfica**

En esta tarea se ha diseñado la apariencia de la interfaz gráfica.

Fecha de inicio: 6/04/2015

Fecha de fin: 8/04/2015

Tiempo dedicado: 8 horas

#### **7.3.2. Implementación de la interfaz gráfica**

En esta tarea se ha construido la interfaz gráfica diseñada en la fase anterior.

Fecha de inicio: 9/04/2015

Fecha de fin: 12/04/2015

Tiempo dedicado: 20 horas

#### **7.3.3. Integración con el motor de ejecución**

En esta tarea se ha dotado de funcionalidad a la interfaz gráfica enlazándola con el motor de ejecución desarrollado anteriormente.

Fecha de inicio: 12/04/2015

Fecha de fin: 20/04/2015

Tiempo dedicado: 30 horas

#### **7.3.4. Introducción de la función de coste**

En esta tarea se ha desarrollado las funcionalidades relativas a la introducción de la función de coste por parte del usuario. Validaciones, control de errores y su enlace con el motor de ejecución.

Fecha de inicio: 21/04/2015

Fecha de fin: 23/04/2015

Tiempo dedicado: 6 horas

#### **7.3.5. Codificación de cromosomas**

En esta tarea se ha dotado de funcionalidad a la sección dedicada a la codificación de cromosomas por parte del usuario. Introducción de genes, edición, validaciones, control de errores y enlace con el motor de ejecución.

Fecha de inicio: 23/04/2015

Fecha de fin: 25/04/2015

Tiempo dedicado: 16 horas

#### **7.3.6. Procesamiento de resultados**

En esta tarea se ha implementado de forma básica el procesamiento de los resultados de la ejecución para obtener las métricas solicitadas en los requisitos.

Fecha de inicio: 26/04/2015

Fecha de fin: 6/05/2015

Tiempo dedicado: 20 horas.

#### **7.3.7. Pruebas y correcciones**

En esta tarea se han realizado pruebas a las funcionalidades construidas y se han aplicado los cambios y correcciones necesarios.

Fecha de inicio: 07/05/2015

Fecha de fin: 18/06/2015

Tiempo dedicado: 20 horas.

### ***7.4. Mejoras en la herramienta gráfica***

En esta fase se han realizado mejoras sobre la herramienta gráfica básica desarrollada para dar solución a aquellos requisitos no relacionados directamente con la ejecución del algoritmo.

#### **7.4.1. Guardado de datos en ficheros**

En esta tarea se ha implementado la funcionalidad de guardado de los datos en ficheros.

Fecha de inicio: 18/06/2015

Fecha de fin: 21/06/2015

Tiempo dedicado: 6 horas.

#### **7.4.2. Carga de ficheros**

En esta tarea se ha implementado la funcionalidad de carga de datos almacenados en ficheros.

Fecha de inicio: 18/06/2015

Fecha de fin: 21/06/2015

Tiempo dedicado: 4 horas.

### **7.4.3. Presentación de resultados gráficos**

En esta tarea se ha mejorado la visualización de resultados y se han introducido los componentes gráficos con datos sobre los resultados.

En esta tarea se ha implementado la funcionalidad de guardado de los datos en ficheros.

Fecha de inicio: 19/06/2015

Fecha de fin: 19/06/2015

Tiempo dedicado: 6 horas.

### ***7.5.Documentación***

En esta fase se han recopilado los datos recopilados durante las fases de análisis y desarrollo para la redacción de esta memoria.

En esta tarea se ha implementado la funcionalidad de guardado de los datos en ficheros.

Fecha de inicio: 01/06/2015

Fecha de fin: 24/07/2015

Tiempo dedicado: 80 horas.

### ***7.6.Presupuesto***

Como puede observarse, aunque el proyecto se haya desarrollado a lo largo de seis meses periodo, la dedicación no ha sido la misma en todos los momentos debido a la disponibilidad del autor. En cualquier caso, se han dedicado 317 horas. Estimando el coste de una hora de trabajo de un ingeniero a 35 euros, el coste final del proyecto sería de 11095 euros.



## 8. Conclusiones

En el desarrollo de este proyecto se ha ilustrado el funcionamiento de los algoritmos genéticos. Se ha presentado el proceso de desarrollo y se ha construido una aplicación que muestra el funcionamiento de estos algoritmos.

Cabe destacar que se han cumplido con éxito los objetivos del proyecto, pues se ha implementado una herramienta que cumple con los requisitos del proyecto en los tiempos previstos, con unos costes razonables.

Como ha podido verse, la bondad de estos algoritmos depende fuertemente de su adaptación al problema particular que se desea resolver. En este sentido, la utilidad de la herramienta desarrollada es académica, siendo especialmente útil para ilustrar el progreso de un algoritmo genético en diversos tipos de problemas.

Sin embargo, pese a la carencia de aplicaciones concretas de esta herramienta y dada la estructuración de su diseño, se han creado componentes que pueden ser de utilidad para aplicar algoritmos genéticos para resolver diversos problemas.

En este sentido cabe enunciar posibles ampliaciones a este trabajo. Existe la posibilidad de aplicar la herramienta a problemas concretos. Para ello, podría estudiarse las características particulares de un problema para obtener una función de evaluación y una codificación de cromosomas que sean de utilidad y podría emplearse la propia herramienta para resolverlo.

Gracias a la modularización que se ha seguido durante el diseño, podrían emplearse los componentes para crear aplicaciones válidas para distintos requisitos. Resultarían de especial utilidad los módulos que forman el modelo de datos y el módulo de ejecución. Estos módulos pueden utilizarse para crear aplicaciones adaptadas a distintos problemas. Podrían implementarse distintas interfaces de usuario para, crear aplicaciones que permitan utilizarse de manera distinta a la creada o aplicaciones que se ejecuten en entornos diferentes, entornos web, móvil o tareas que se ejecuten en

segundo plano. Además, podría modificarse el módulo de resultados para construir métricas más adaptadas a los deseos del usuario.



## 9. Referencias y bibliografía

1. Champanard, A. (2007). Genetic Algorithms Libraries. <http://geneticalgorithms.ai-depot.com/Libraries.html> . Artificial Intelligence Depot. Fecha de consulta: Agosto 2015.
2. Darwin, C. (1859). El origen de las especies.
3. Davis, L. (1991). *Handbook of genetic algorithms*. New York: Van Nostrand Reinhold. ISBN: 978-0442001735.
4. Gen, M., & Cheng, R. (1997). *Genetic algorithms and engineering design*. New York: Wiley. ISBN: 978-0471127413
5. Gestal Pose, M. (2006). *Introducción a los algoritmos genéticos*. <http://sabia.tic.udc.es/mgestal/cv/aaggtutorial/aagg.html> Universidade da Coruña. Fecha de consulta: Agosto 2015.
6. Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley. ISBN: 978-0201157673.
7. Holland, John H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press isbn: 978-0472084609.
8. Michalewicz, Z. (1996). *Genetic algorithms + data structures =: Evolution programs*. Berlin: Springer-Verlag. ISBN: 978-3540606765.
9. Poli, R., Langdon, W. B., McPhee, N. F., & Koza, J. R. (2008). *A field guide to genetic programming*. S.I.: Lulu Press, lulu.com. ISBN: 978-1409200734
10. Serradilla, F., Barros, B. (1996). *Representación e inferencia en inteligencia artificial, un enfoque práctico*. Universidad Politécnica de Madrid.

11. Surjanovic, S., Bingham, D. (2015). *Optimization Test Problems* <http://www.sfu.ca/~ssurjano/optimization.html>. Simon Fraser University. Fecha de consulta: Agosto 2015.