Resumen

Palabras clave

Overview

Keywords

Índice

1. Introducción 9
2. Algoritmos genéticos 10
   2.1. Representación de individuos 10
   2.2. Algoritmo general 11
   2.3. Operador de selección 11
      2.3.1. Método de la ruleta 12
      2.3.2. Selección por torneo 12
   2.4. Operador de cruce 12
      2.4.1. Cruce en un punto 12
      2.4.2. Cruce en dos puntos 12
   2.5. Operador de mutación 13
   2.6. Limitaciones de los algoritmos genéticos 13
3. Planteamiento del problema 14
4. Diseño 15
   4.1. Requisitos 15
      4.1.1. Introducción de la función de coste 15
      4.1.2. Codificación del vector de parámetros o cromosoma 15
      4.1.3. Introducción del tamaño de la población 16
      4.1.4. Establecimiento del número de generaciones 16
      4.1.5. Establecimiento del número de eras 16
      4.1.6. Selección de elitismo 16
      4.1.7. Introducción de la probabilidad de cruce 17
      4.1.8. Introducción de la probabilidad de mutación 17

4.1.9. Selección de un operador de selección 17
4.1.10. Validación de la configuración 17
4.1.11. Inicialización de la población 17
4.1.12. Evaluación de la función de coste 17
4.1.13. Operador de selección 18
4.1.14. Operador de cruce 18
4.1.15. Operador de mutación 18
4.1.16. Ejecución del algoritmo 18
4.1.17. Cancelación de la ejecución 18
4.1.18. Resultados parciales 18
4.1.19. Resultados finales 19
4.1.20. Guardado de la ejecución 19
4.1.21. Carga de la configuración guardada 20
4.2. Casos de uso 20
4.3. Componentes del sistema 30
4.4. Diagrama de clases 31
4.4.1. Modelo de datos 33
4.4.2. Módulo de ejecución del algoritmo 35
4.4.3. Módulo de presentación de resultados 36
4.4.4. Módulo de interfaz gráfica de usuario 40
4.4.5. Módulo de guardado de datos 41
4.5. Flujo de ejecución 42
4.5.1. Flujo de ejecución general 42
4.5.2. Flujo de ejecución del algoritmo genético 44
4.6. Interfaz gráfica 44

5. Funcionamiento de la aplicación 45
5.1. Requisitos previos 45
5.2. Acceso a la aplicación 45
5.3. Configuración de la ejecución 45
5.4. Ejecución 45
5.5. Resultados de la ejecución 45
5.6. Guardado de datos 45
5.7. Carga de datos guardados 45
6. Ejemplos de ejecución 46
6.1. Caso de prueba 1 46
6.2. Caso de prueba 2 46
6.3. Caso de prueba 3: Función de Ackley 46
6.4. Caso de prueba 4: Función de Schaffer 46
6.5. Caso de prueba 5 47
6.6. Caso de prueba 6 47
7. Planificación y presupuesto 48
8. Conclusiones 49
9. Bibliografía 50

Figura 1 Componentes del sistema 30
Figura 2 Diagrama de clases 32
Figura 3 Modelo de datos 34
Figura 4 Módulo de ejecución 35
Figura 5 El patrón Observer 37
Figura 6 Módulo de presentación de resultados 38

Figura 7 Módulo de interfaz gráfica 40
Figura 8 Módulo de guardado de datos 41
Figura 9 Flujo de la ejecución general 43
Figura 10 Flujo de ejecución del algoritmo genético 44

1. Introducción

Es un poco hablar de lo que estás haciendo y de que otras herramientas existen actualmente para hacer lo mismo. También deberías hablar aquí de las decisiones básicas a la hora de hacer la aplicación (ej: ¿Por qué has usado Java?) y los objetivos de la misma.

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. 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.

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 básico será diferente.

Habitualmente, la teoría clásica de algoritmos genéticos utiliza par a 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 rango 0,1 y se selecciona el individuo correspondiente.

2.3.2. Selección por torneo

En la selección por torneo, se elige un número 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.

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.

2.4.Operador de cruce

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

Existen diversos algoritmos de cruce.

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.

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.

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 desdeñando 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.

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:

$$J: R^n \rightarrow R$$

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 Ni individuos

      Durante un número determinado de generaciones

            Crear una nueva población

            Seleccionar padres

            Cruzar los padres con probabilidad pc

            Mutar los individuos resultantes con probabilidad pm

            Opcionalmente aplicar un operador de elitismo

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

Se debe proporcionar un 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

Con los requisitos, análisis de casos de uso, etc. También debes de especificar los componentes que tiene, poner un diagrama de clases e indicar que hace cada una de ellas. Indicar cómo se evalúan las expresiones, cómo se organiza la interfaz gráfica... No incluyas código (esto irá en el CD) a no ser que fuese imprescindible. Se trata de documentar las clases indicando que hacen y cómo está organizada la aplicación no de describir exhaustivamente cada línea del programa...

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:

$$J: R^n \rightarrow R$$

Los posibles elementos que podrán incluirse en la función son

- Números reales.
- Parámetros, expresados por su nombre.
- Los siguientes operadores: +, -, /, *, √, sin, cos, tan, asin, atan, abs, log, $\log_{10}$, $\log_2$
- Los símbolos de paréntesis "(" y ")".
- El número π, 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.

$$\theta = [x_{1, \ldots,} x_n]$$

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

- El mejor valor de la función de coste obtenido a lo largo de cada 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:

CU1	**Introducción de la función de coste**
Descripción: Permite al usuario introducir la función de coste que se pretende optimizar mediante el uso de la herramienta	
Actores: El usuario final	
Precondiciones: Ninguna	
Flujo Normal: El usuario introduce la función que desea optimizar.	
Flujo Alternativo: No aplica.	
Post condiciones: No aplica.	

CU 1Introducción 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: No aplica.	
Post condiciones: No aplica.	

CU 2Introducción del número de eras

CU3	**Introducción del número de generaciones**
Descripción: Permite al usuario introducir un valor numérico entre uno y mil representando el número de generaciones que se desea que se ejecuten.	

Actores: El usuario final.
Precondiciones: Ninguna.
Flujo Normal: El usuario introduce un número entero entre uno y mil.
Flujo Alternativo: No aplica
Post condiciones: No aplica

CU 3Introducción del número de generaciones

CU4	**Introducción de la codificación de los cromosomas**
Descripción: Permite al usuario introducir la codificación de los cromosomas que va a emplear el algoritmo.	
Actores: El usuario final.	
Precondiciones: Ninguna.	

Flujo Normal:

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.
2. El usuario pulsa el botón de añadir.
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.
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 4Introducción de la codificación de los cromosomas

CU5	**Introducción del tamaño de la población**

Descripción:

Permite al usuario introducir el número de cromosomas que contiene la población durante la ejecución del algoritmo genético.

Actores:

El usuario final.

Precondiciones:

Ninguna.

Flujo Normal: El usuario introduce en el campo tamaño de la población un valor numérico entre uno y mil.
Flujo Alternativo: No aplica.
Post condiciones: No aplica.

CU 5Introducción del tamaño de la población

CU6	**Introducción de la probabilidad de cruce**
Descripción: Permite al usuario introducir el valor que se utilizará como probabilidad para ejecutar el operador de cruce en el algoritmo genético.	
Actores: El usuario final.	
Precondiciones: Ninguna.	
Flujo Normal: El usuario introduce en el campo probabilidad de cruce un valor numérico entre cero y uno.	
Flujo Alternativo: No aplica.	

Post condiciones: No aplica.

CU 6Introducción de la probabilidad de cruce

CU7	**Introducción de la probabilidad de mutación**
Descripción: Permite al usuario introducir el valor que se utilizará como probabilidad para ejecutar el operador de mutación en el algoritmo genético.	
Actores: El usuario final.	
Precondiciones: Ninguna.	
Flujo Normal: El usuario introduce en el campo probabilidad de cruce un valor numérico un valor numérico entre cero y uno.	
Flujo Alternativo: No aplica.	
Post condiciones: No aplica.	

CU 7Introducción de la probabilidad de mutación

CU8	**Guardar datos**

Descripción:

Permite al usuario guardar en un fichero los datos de configuración y de ejecución del algoritmo.

Actores:

El usuario final.

Precondiciones:

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
5. La herramienta almacenará en el fichero los datos correspondientes a la ejecución.

Flujo Alternativo:

No aplica

Post condiciones:

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

CU 8Guardar datos

CU9	**Cargar datos**

Descripción:

Permite al usuario cargar los datos de configuración y de ejecución almacenados en un fichero.

Actores:

El usuario final.

Precondiciones:

Ninguna.

Flujo Normal:

1. El usuario pulsa en el botón de cargar configuración.
2. El usuario selecciona el fichero que contiene la configuración.
3. En la pantalla principal de la herramienta aparecerán introducidos los parámetros de configuración que contiene el fichero. Estos parámetros podrán ser:
    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.

4. En caso de existir datos con resultados de ejecución en el fichero se mostrarán mediante la herramienta.

Flujo Alternativo:

No aplica.

Post condiciones:

La herramienta marcará los parámetros de configuración que contiene el fichero.

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

CU 9Cargar datos

CU10	**Resolución de problemas de optimización**

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.

Post condiciones:

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

CU 10Resolución de problemas de optimización

CU11	**Cancelación de la ejecución**

Descripción:

El usuario puede cancelar en todo momento la ejecución.

Actores:

El usuario final

Precondiciones:

El sistema debe estar calculando la solución a un problema de optimización.

Flujo Normal:

1. El usuario pulsa en el botón de cancelar.
2. La ejecución se detiene antes de finalizar.

3. La aplicación mostrará los resultados calculados hasta el momento.
Flujo Alternativo: No aplica.
Post condiciones: La ejecución se detendrá sin haber finalizado.

CU 11Cancelación 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.

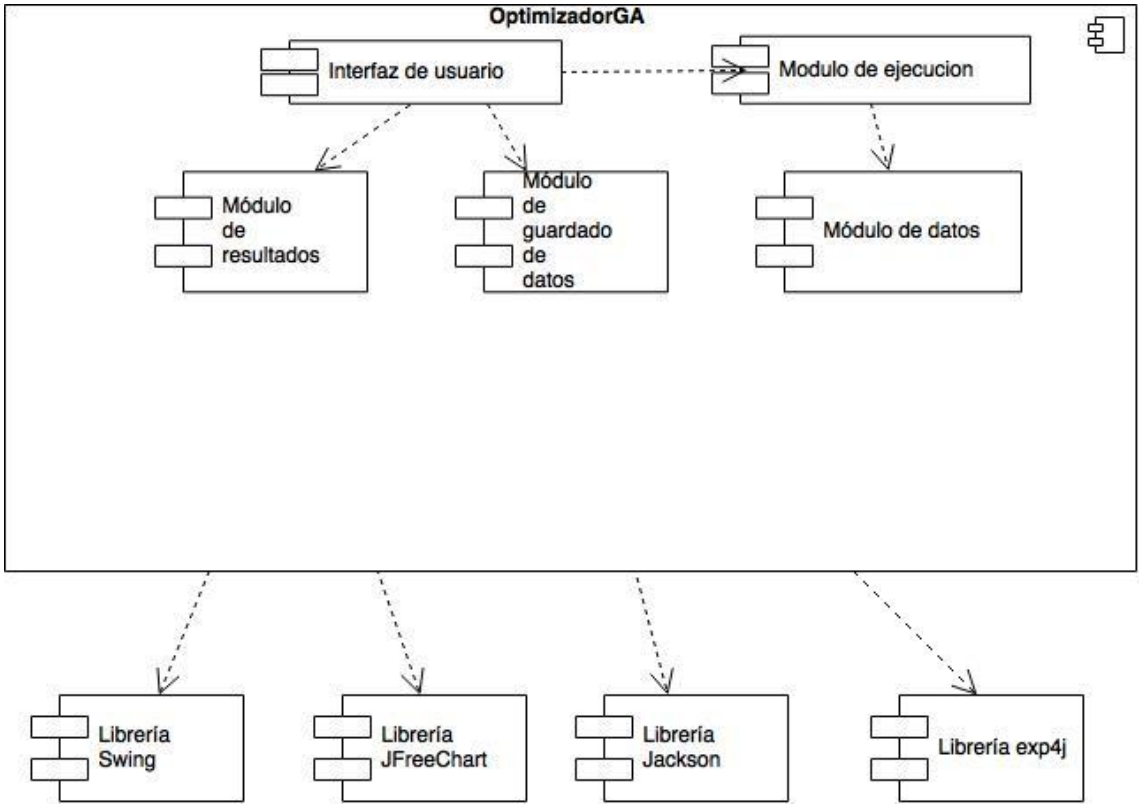

Figura 1 Componentes del sistema

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.

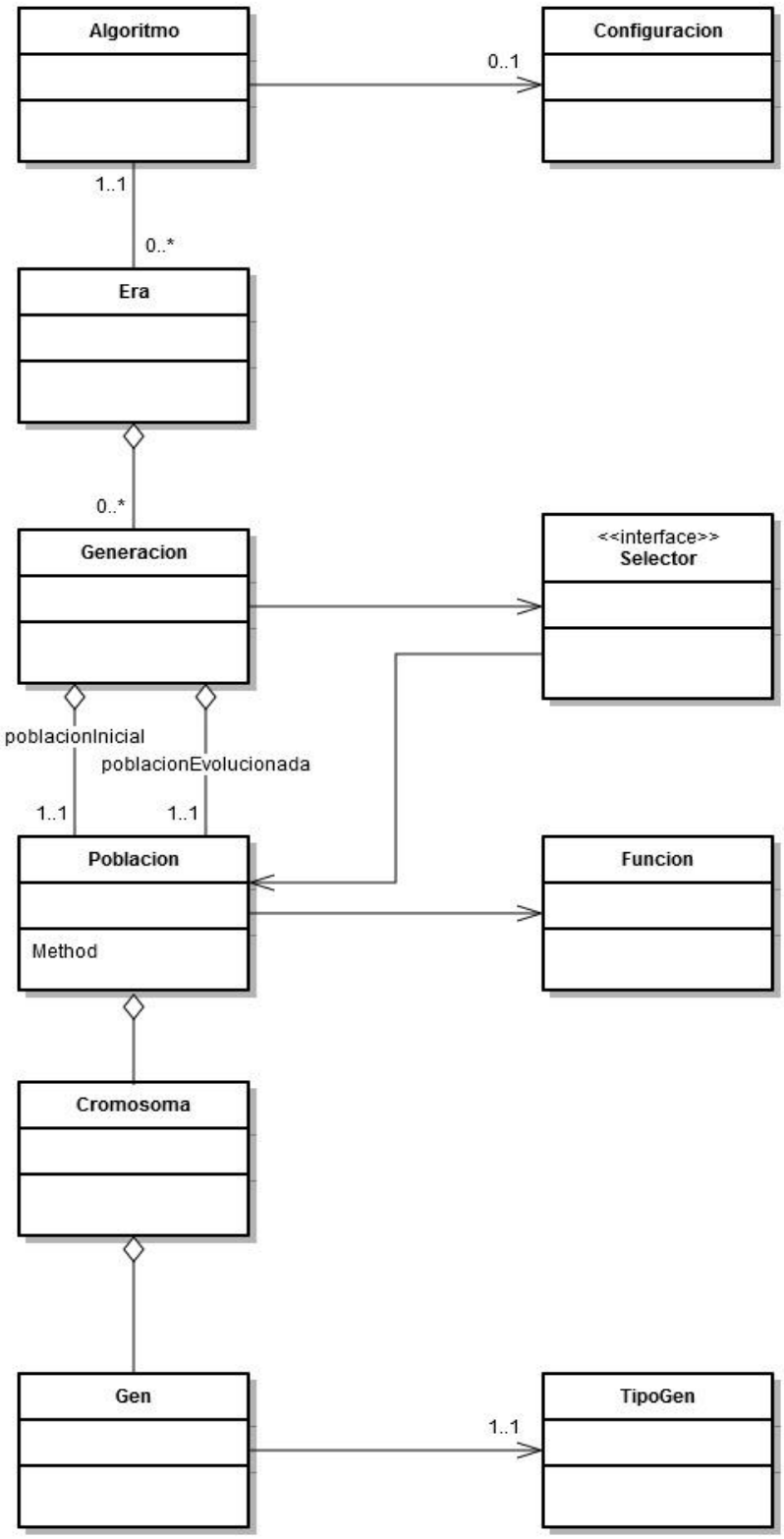

Figura 2 Diagrama de clases

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 un 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

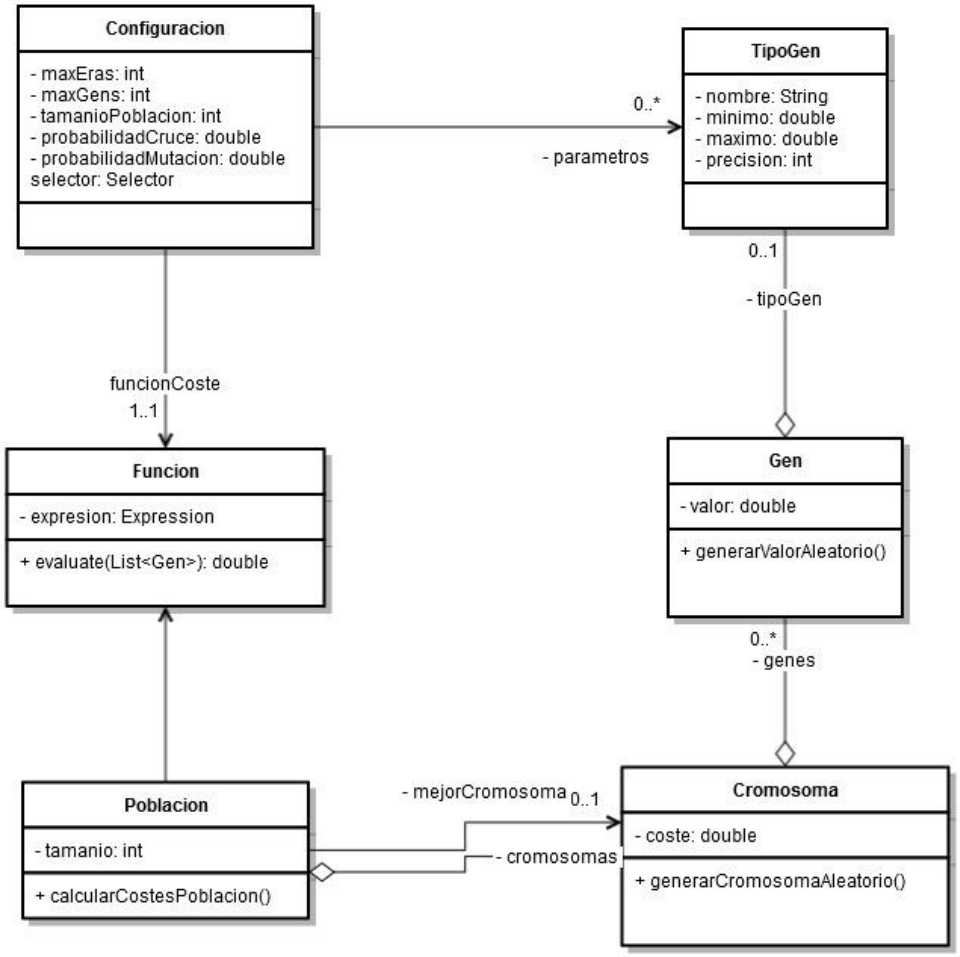

Figura 3 Modelo de datos

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.

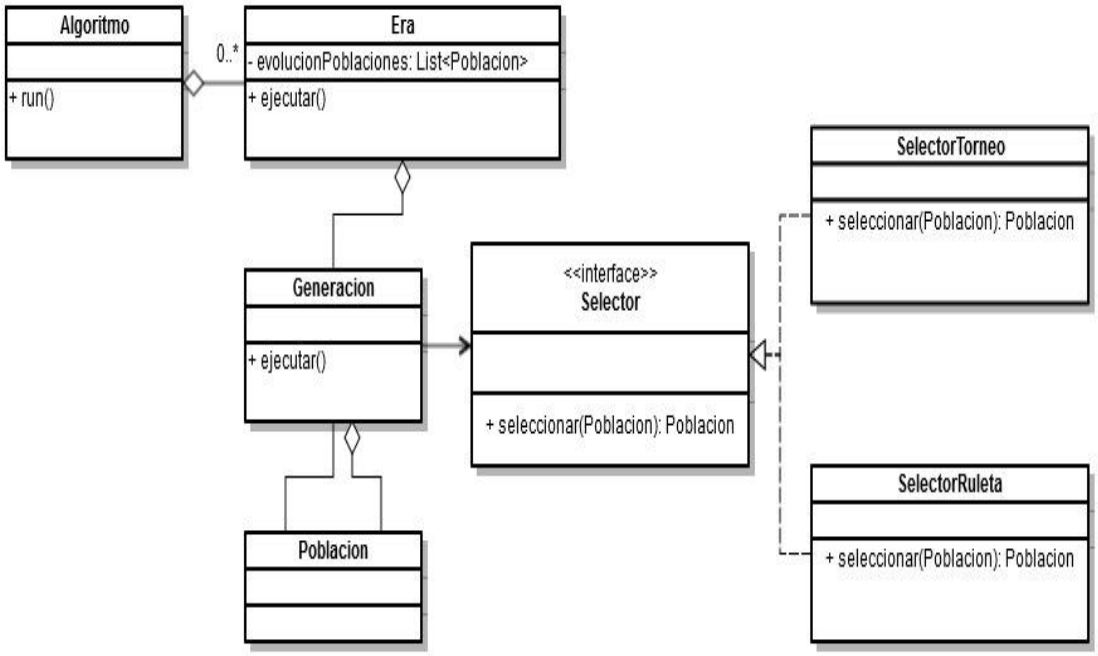

Figura 4 Módulo de ejecución

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 el 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 el interfaz Selector. Este interfaz 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 del interfaz Selector que especifica el método de la ruleta.

- **SelectorTorneo:** Se trata de una implementación del 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 al interfaz de usuario. La estructura básica de este patrón sería la siguiente:

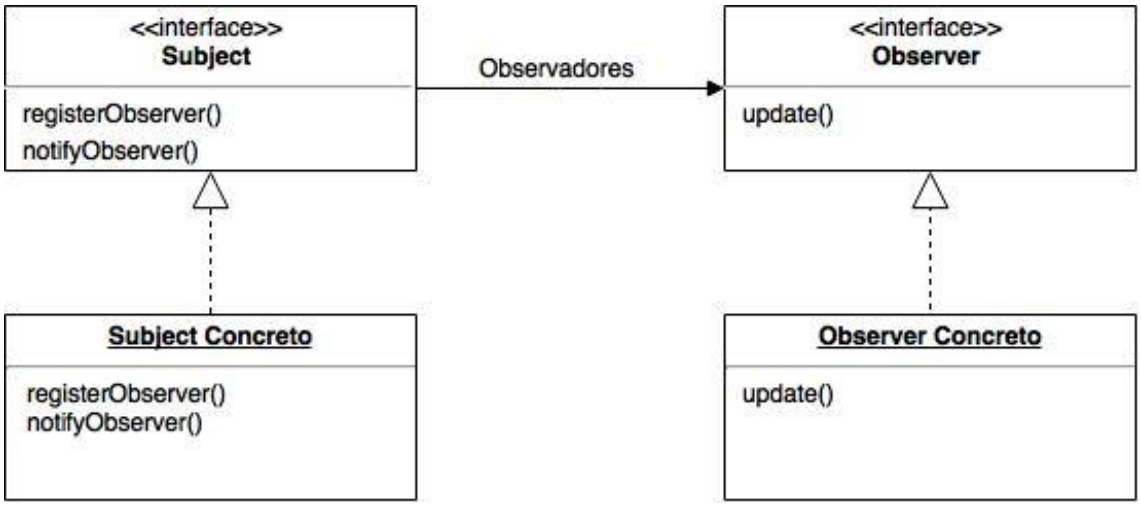

Figura 5 El patrón Observer

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.

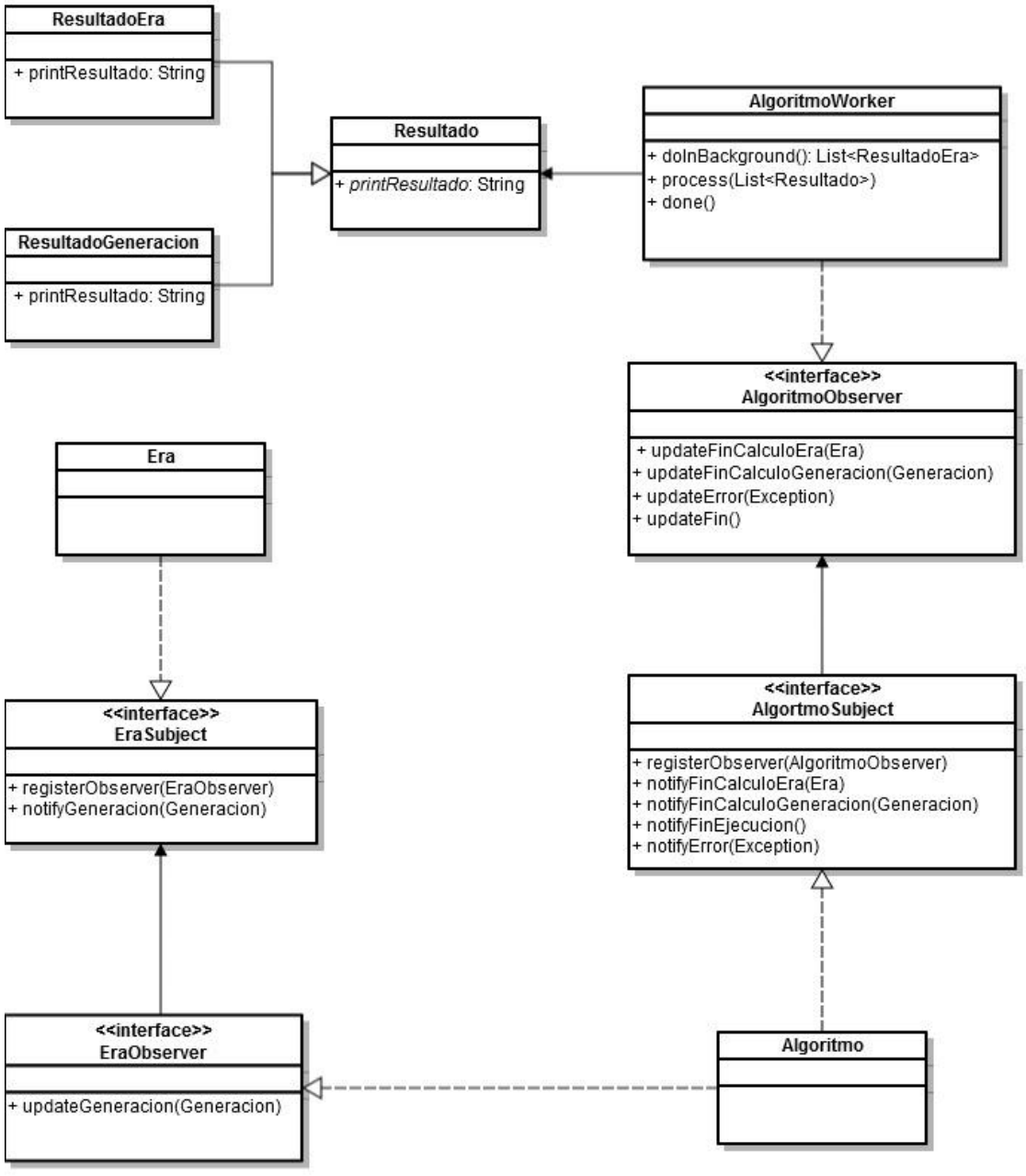

Figura 6 Módulo de presentación de resultados

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 empleado 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 este interfaz.

- **EraObserver:** Interfaz empleado 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 empleado 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 este interfaz es el Algoritmo.

- **AlgoritmoObserver:** Interfaz empleado 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 el interfaz EraSubject para informar de los resultados parciales obtenidos en el cálculo de una generación.

- **Algoritmo:** Esta clase debe implementar el interfaz EraObserver para recoger los resultados parciales obtenidos en el cálculo de una generación. Además debe implementar el 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 al interfaz de usuario. Esta clase extiende la clase de utilidad SwingWorker, proporcionada por la librería estándar Swing, que proporciona métodos de utilidad para actualizar al interfaz de usuario con datos correspondientes a la evolución de la ejecución, y a los datos finales resultado del cálculo. Para poder informar al interfaz de estos datos,

debe establecerse como un observador sobre el algoritmo implementando el interfaz AlgoritmoObserver.

4.4.4. Módulo de interfaz gráfica de usuario

El interfaz gráfico está compuesto 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

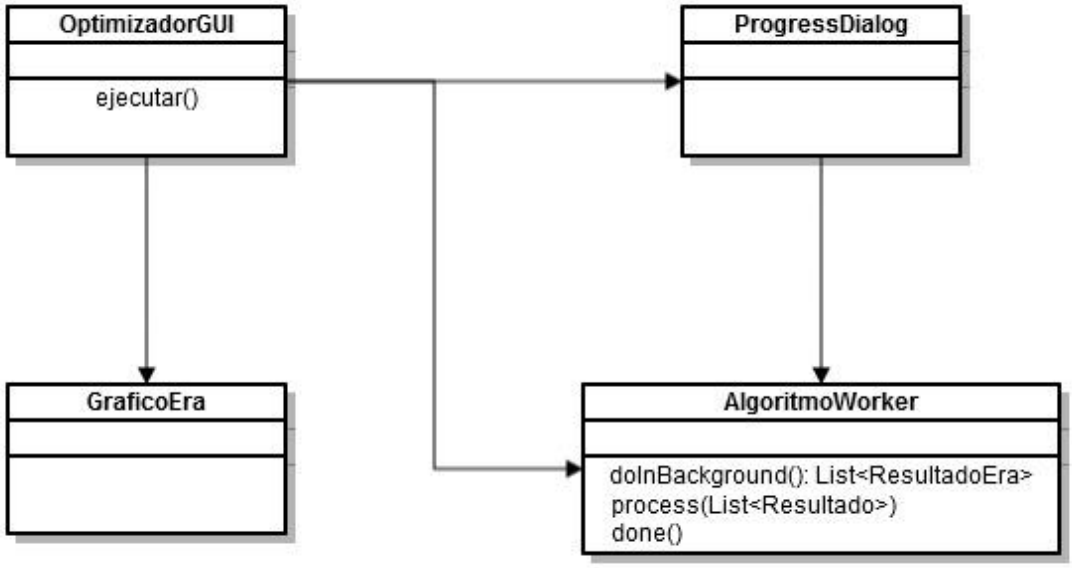

Figura 7 Módulo de interfaz gráfica

- **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. Ademñas 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

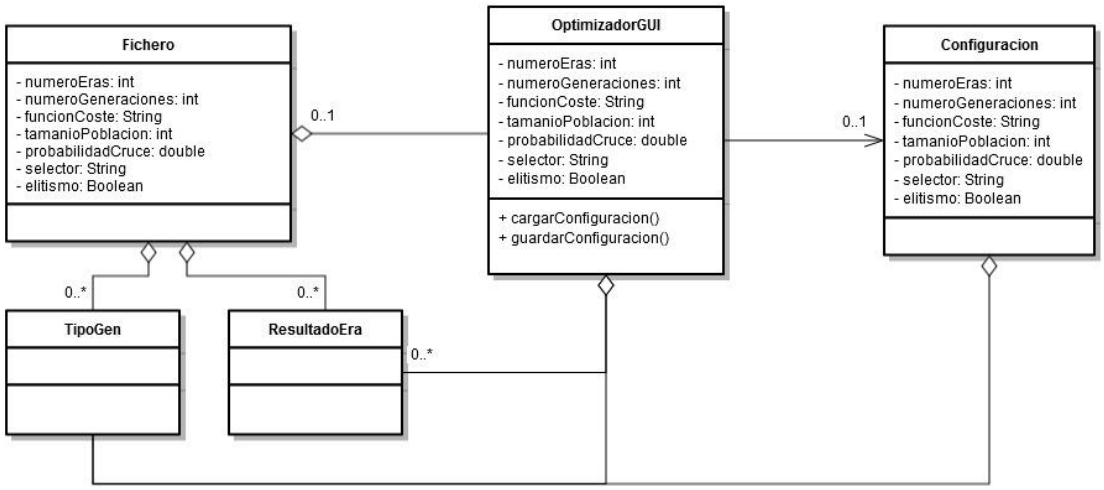

Figura 8 Módulo de guardado de datos

- **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.

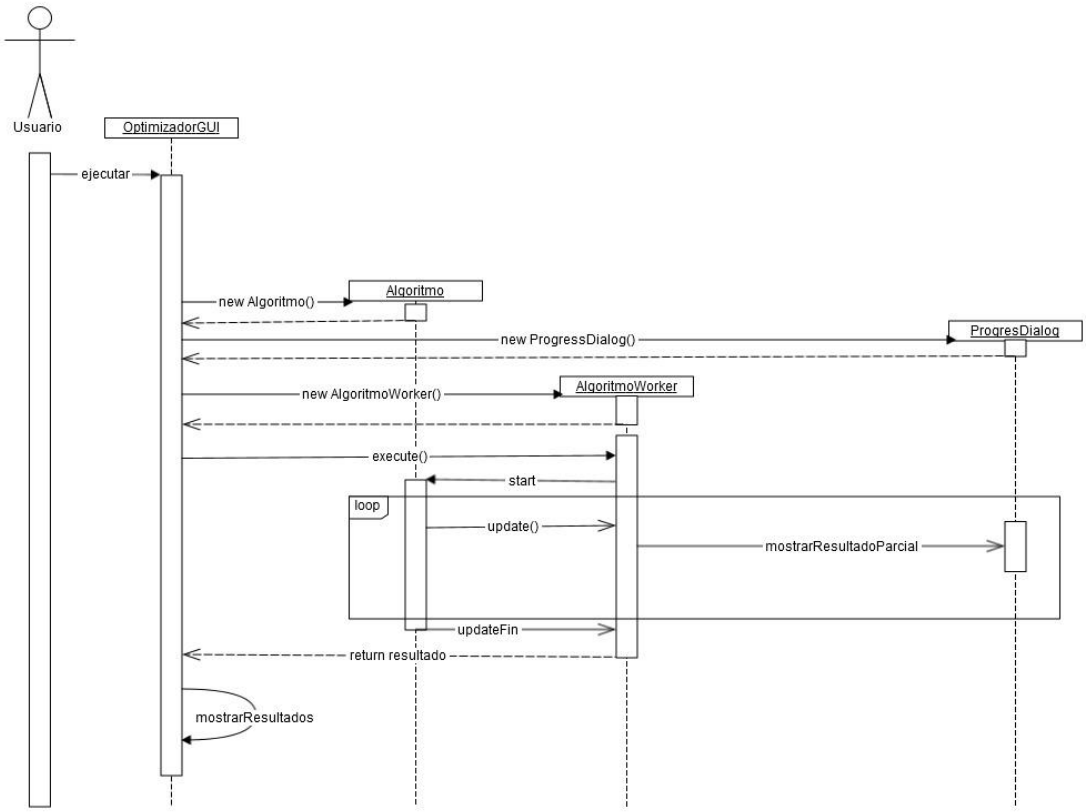

Figura 9 Flujo de la ejecución general

4.5.2. Flujo de ejecución del algoritmo genético

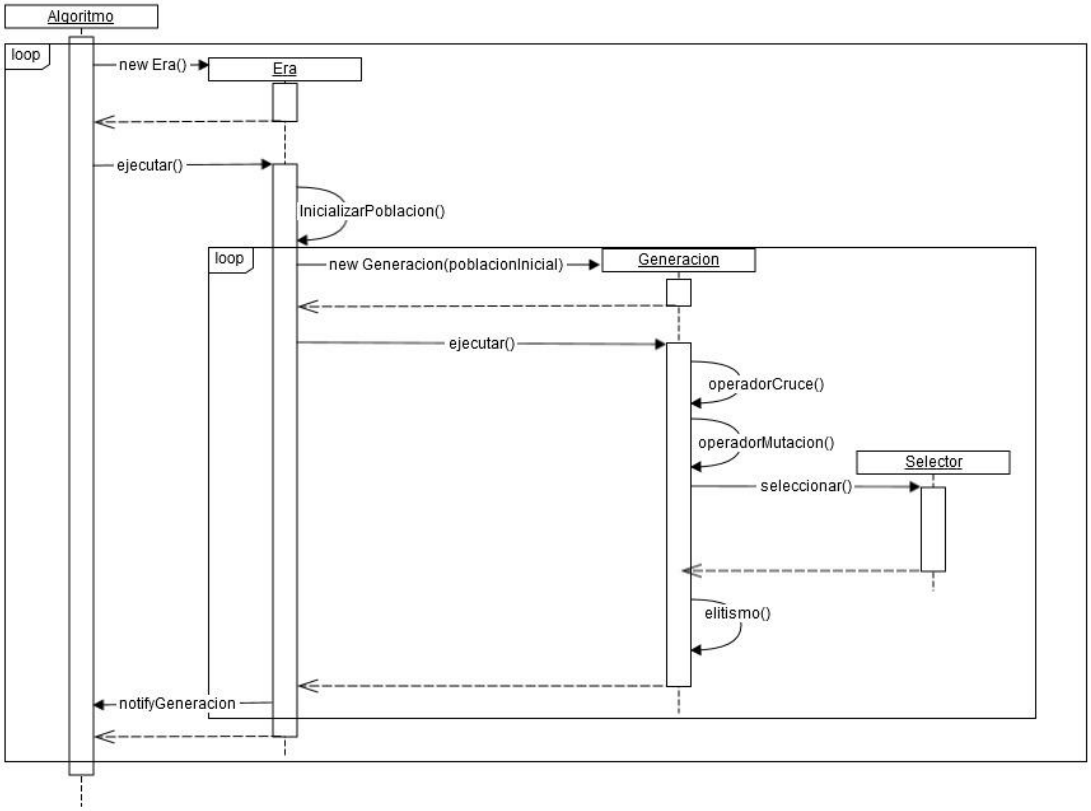

Figura 10 Flujo de ejecución del algoritmo genético

4.6. Interfaz gráfica

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.

5.3. Configuración de la ejecución

La aplicación inicialmente contiene una configuración por defecto

5.4. Ejecución

5.5. Resultados de la ejecución

5.6. Guardado de datos

5.7. Carga de datos guardados

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.

6.1. Caso de prueba 1

Función de evaluación:

$f(x1, x2) = 21.5 + x1 * sen(4 * \pi * x1) + x2 * sen(4 * \pi * x2)$

S= {x1 ∈ [-3.0, 5.1], x2 ∈ [4.1, 5.8]}

6.2. Caso de prueba 2

Función de evaluación:

$f(x1, x2, x3, x4, x5, x6) = 100 - (x1^2 + x2^2 + x3^2 + x4^2 + x5^2 + x6^2)$

S= { x1 ∈[-3.0, 5.1], x2 ∈[2.1, 7.8], x3 ∈[-10.1, 20.3],
x4 ∈[-3.3, 4.2], x5 ∈[-15.3, 70.1], x6 ∈[-0.25, 0.35]

6.3. Caso de prueba 3: Función de Ackley

Función de evaluación:

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

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

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

6.4. Caso de prueba 4: Función de Schaffer

Función de evaluación:

6.5.Caso de prueba 5

Función de evaluación:

6.6.Caso de prueba 6

Función de evaluación:

7. Planificación y presupuesto

Describe cuanto has tardado, cómo has dividido el trabajo y cuanto costaría tu aplicación (un diagrama de gant con las partes del proyecto ayuda mucho en esta fase)

8. Conclusiones

9. Bibliografía