



**UNIVERSIDAD DE SEVILLA**  
**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA**

**Grado en Ingeniería Informática – Tecnologías Informáticas**  
**Curso 2020/2021**

**OPTIMIZACIÓN DE FUNCIONES MEDIANTE  
ALGORITMO MULTIOBJETIVO BASADO EN  
AGREGACIÓN**

Realizado por:

Jaime Emilio Sala Mascort - jaisalmas

Jesús Fernández García - jesfergar5

Correo electrónico:

[jesala31@gmail.com](mailto:jesala31@gmail.com)

[jesusfernandez199815@gmail.com](mailto:jesusfernandez199815@gmail.com)

**Sevilla, febrero de 2021**

# **Índice**

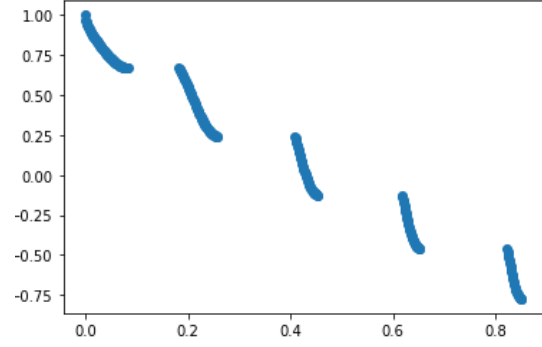
1. Temática y problema por resolver .....	3
2. Estructura del código y elementos utilizados.....	4
3. Librerías utilizadas .....	6
4. Utilización del programa y ejemplos de uso .....	7

# 1. Temática y problema por resolver

La temática elegida para el trabajo ha sido la resolución de problemas de optimización multiobjetivo con y sin restricciones. En concreto, intentaremos obtener soluciones óptimas para las funciones ZDT3 y CF6 en 4 y 16 dimensiones. Estas son sus fórmulas y sus frentes Pareto óptimos:

$$\begin{aligned}
 f_1(\vec{x}) &= x_1 \\
 f_2(\vec{x}) &= g(\vec{x})[1 - \sqrt{x_1/g(\vec{x})} - x_1/g(\vec{x}) \sin(10\pi x_1)] \\
 g(\vec{x}) &= 1 + \frac{9}{n-1} \left( \sum_{i=2}^n x_i \right) \\
 0 \leq x_i &\leq 1, \quad i = 1, \dots, n
 \end{aligned}$$

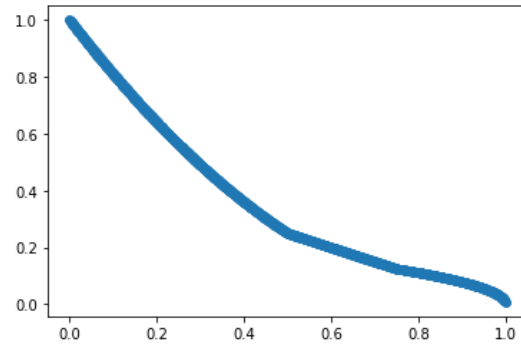
**Función ZDT3**



**Frente Pareto Óptimo ZDT3**

$$\begin{aligned}
 f_1 &= x_1 + |G(t)| + \sum_{j \in J_1} (y_j)^2 \\
 f_2 &= (1 - x_1)^2 + |G(t)| + \sum_{j \in J_2} (y_j)^2 \\
 y_j &= \begin{cases} x_j - 0.8x_1 \sin\left(6\pi x_1 + \frac{j\pi}{n}\right), & j = 2, 4 \\ x_j - 0.8x_1 \cos\left(6\pi x_1 + \frac{j\pi}{n}\right) - |G(t)|, & j \in J_1 \\ x_j - 0.8x_1 \sin\left(6\pi x_1 + \frac{j\pi}{n}\right) - |G(t)|, & j \in J_2 - \{2, 4\} \end{cases} \\
 x &\in [0, 1] \times [-2, 2]^{n-1} \\
 g_1(x, t) &= \begin{cases} x_2 - |G(t)| - 0.8x_1 \sin\left(6\pi x_1 + \frac{2\pi}{n}\right) - \text{sign}(k_1)\sqrt{|k_1|} \\ k_1 = 0.5(1 - x_1) - (1 - x_1)^2 \end{cases} \\
 g_2(x, t) &= \begin{cases} x_4 - |G(t)| - 0.8x_1 \sin\left(6\pi x_1 + \frac{4\pi}{n}\right) - \text{sign}(k_2)\sqrt{|k_2|} \\ k_2 = 0.25\sqrt{1 - x_1} - 0.5(1 - x_1) \end{cases} \\
 POF : \begin{cases} f_1 = y_1 + |G(t)|; & f_2 = y_2 + |G(t)| \\ y_2 = \begin{cases} (1 - y_1)^2 & \text{if } 0 \leq y_1 \leq 0.5 \\ 0.5(1 - y_1) & \text{else if } y_1 \leq 0.75 \\ 0.25\sqrt{1 - y_1} & \text{else if } y_1 \leq 1 \end{cases} \\ 0 \leq y_1 \leq 1 \end{cases}
 \end{aligned}$$

**Función CF6**



**Frente Pareto Óptimo CF6**

El algoritmo propuesto para buscar soluciones óptimas de estas funciones es un algoritmo evolutivo basado en agregación. Esto consiste en descomponer el problema en varios subproblemas de un solo objetivo. Para cada subproblema a resolver, la función objetivo será una agregación de los objetivos del problema principal. En este trabajo, se utilizará la formulación de Tchebychef:

$$g^{te}(x|\lambda, z^*) = \max\{\lambda|f_i(x) - z_i^*|, 1 \leq i \leq m\}$$

donde  $\lambda = (\lambda_1, \dots, \lambda_m)^T$  es un vector peso y  $z^* = (z_1^*, \dots, z_m^*)^T$  es un punto de referencia donde cada una de sus componentes está dada por:

$$z_i^* = \min\{f_i(x) | x \in \Omega\}$$

Para dotar al algoritmo de manipulación de restricciones, se ha optado por el mecanismo basado en selección, el cual consiste en:

- Si un individuo de la generación  $k - 1$  y otro de la generación  $k$  no incumplen las restricciones, la actualización se lleva a cabo de la misma forma que si se tratara de un problema sin restricciones.
- Si un individuo de la generación  $k - 1$  y otro de la generación  $k$  incumplen las restricciones, la actualización se realiza sumando los errores cometidos por cada individuo en cada restricción y se selecciona el que tenga un error más pequeño (número más positivo).
- Si un individuo de la generación  $k - 1$  no cumple las restricciones, pero el individuo de la generación  $k$  sí, entonces la actualización solo se produce en el propio individuo y en los vecinos que no incumplan las restricciones. La actualización total o parcial del vecindario de un individuo, causaría la pérdida de diversidad.

## 2. Estructura del código y elementos utilizados

El proyecto está estructurado de la siguiente forma:

$$\left\{ \begin{array}{l} \text{Funciones} \left\{ \begin{array}{l} Cf6.hs \\ FileTreatment.hs \\ Zdt3.hs \end{array} \right. \\ \\ \begin{array}{l} AgregacionCR.hs \\ AgregacionSR.hs \\ Principal.hs \\ SolucionCF6.hs \\ SolucionZDT3.hs \end{array} \end{array} \right.$$

- **Funciones:** Es un módulo compuesto por submódulos, en estos están programadas las funciones CF6, ZDT3 y las funciones necesarias para leer desde los ficheros .dat el frente Pareto óptimo de las funciones.
- **AgregacionCR/SR:** Son los módulos donde está el mayor peso de la programación. En ellos está implementado el algoritmo evolutivo basado en agregación con y sin manejo de restricciones.
- **Principal:** Es el módulo principal del programa. En él se realiza el control interactivo por consola con el usuario.
- **SolucionCF6/ZDT3:** Estos módulos crean cada uno un tipo de dato nuevo, con lo que facilita el manejo de la solución aportada por el algoritmo en el módulo Principal.

A continuación, indicaremos dónde y cómo se han utilizado cada uno de los elementos mínimos exigidos:

- **2 usos de prelude y Data.List**

- **Módulo Cf6.hs**: Función **abs** y **sqrt** en línea 23, calculo de valor absoluto para raíz cuadrada.
- **Módulo AgregacionSR**: Función **sort** en línea 60, ordenación de una lista que contiene índices. Función **zip4** en línea 261, creación de lista de tuplas de tamaño 4.

- **2 usos de funciones recursivas**

- **Módulo AgregacionSR**: Función **algoritmo\_agregacion\_ZDT3**, se usa la recursión para ejecutar el algoritmo las  $n$  generaciones.
- **Módulo AgregaciónSR**: Función **evaluaciones**, se usa la recursión para evaluar cada individuo de una lista.

- **2 usos patrones**

- **Modulo AgregacionSR**: Función **parte**, se usa para definir el caso base de la recursión.
- **Modulo AgregacionSR**: Función **seleccion\_aleatoria**, se usa para definir el caso base de la recursión.

- **2 usos guardas**

- **Modulo AgregacionSR**: Función **limitador\_aux**, se usa para comprobar que las mutaciones de los individuos no se salen del rango.
- **Modulo AgregacionCR**: Función **evalua\_cf6**, se usa para comprobar el número de restricciones que se incumple.

- **2 usos de case of**

- **Modulo Principal**: Función **main**, se usa para el control de la aplicación según lo elegido por el usuario.
- **Modulo Principal**: Función **siguiente\_accion\_zdt3**, se usa para el control de la aplicación según lo elegido por el usuario.

- **2 usos de listas por comprensión**

- **Modulo AgregacionSR**: Función **calc\_subproblemas**, se usa para calcular una resta con todas las evaluaciones con el punto z.
- **Modulo AgregaciónSR**: Función **puntos\_de\_cruce**, se usa para generar una lista de True, False, para hacer cruces con los individuos.

- **2 usos de orden superior**

- **Modulo AgregacionSR**: Función **calc\_vecindario**, se usa para calcular el vecindario de un individuo (los índices).

- **Modulo AgregaciónCR:** Función **cumple**, se usa para saber si un individuo cumple las restricciones.
- **Declaraciones de tipos para todas las funciones definidas**
  - Como se puede observar en el código, todas las funciones cuentan con su tipado.
- **Creación de módulos**
  - Como se ha visto en la estructura del proyecto, se han creado varios módulos y submódulos.
- **Creación de 2 tipos de datos**
  - Se han creado 2 tipos de datos nuevos, SolucionCF6 y SolucionZDT3 que encapsulan las soluciones proporcionadas por el algoritmo. Estos tipos tienen funciones propias que nos ayudan a manejar las soluciones en el módulo Principal.
- **Uso de dos tipos de datos abstractos o librerías**
  - Hemos usado Random, Array y gnuplot, para la generación de números aleatorios, utilización de vectores y generar gráficas.

### 3. Librerías utilizadas

Las librerías utilizadas para este proyecto han sido las siguientes:

- System.Directory
  - System.Random
  - System.IO
  - Data.Array
  - Data.Time
  - Data.List
  - Data.Char
  - Graphics.Gnuplot, para hacer uso de ella hemos tenido que instalarla mediante cabal.
- `cabal install gnuplot`

## 4. Utilización del programa y ejemplos de uso

Para utilizar el programa lo primero que hay que hacer es compilarlo con la siguiente instrucción:

```
ghc -o Principal.hs principal.exe
```

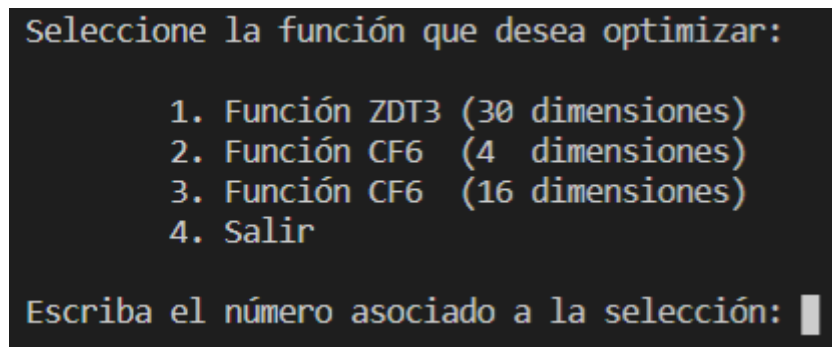
Si nos diese el siguiente error:

```
Could not load module 'System.Random'
```

simplemente tendremos que ejecutar:

```
ghc -o Principal.hs principal.exe -package random
```

Una vez lanzado el programa desde la consola con `./principal.exe` o haciendo doble clic en él, nos solicitará seleccionar una función a optimizar o salir del programa:



```
Seleccione la función que desea optimizar:

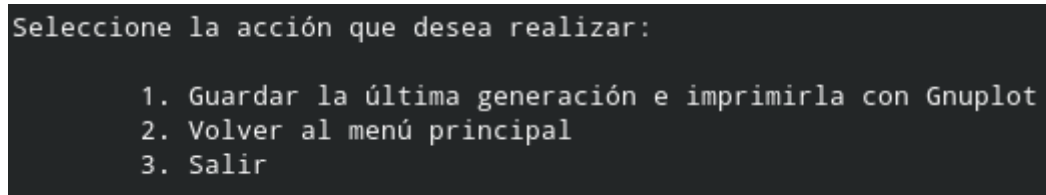
1. Función ZDT3 (30 dimensiones)
2. Función CF6 (4 dimensiones)
3. Función CF6 (16 dimensiones)
4. Salir

Escriba el número asociado a la selección: █
```

*Ejecución del programa. Primer menú*

Una vez seleccionada la función, introduciendo el número asociado a cada opción, nos irá solicitando los parámetros necesarios para la ejecución del algoritmo, como pueden ser: tamaño población, número de generaciones, etc.

Cuando el algoritmo haya terminado de ejecutarse, nos saldrá el siguiente menú:



```
Seleccione la acción que desea realizar:

1. Guardar la última generación e imprimirla con Gnuplot
2. Volver al menú principal
3. Salir
```

*Ejecución del programa. Segundo menú*

En él, podemos optar por:

- **Guardar la última generación e imprimirla con Gnuplot:** Esto nos guardará las evaluaciones de los individuos de la última generación en un `.dat` y nos mostrará gráficamente estas evaluaciones en Gnuplot.
- **Volver al menú principal:** Lo que nos dirigirá de nuevo al primer menú sin realizar ninguna operación sobre la solución calculada.

- **Salir:** Producirá la finalización del programa.