Universidad de Matanzas
Facultad de Ciencias Técnicas

APLICACIÓN CON ARQUITECTURA ORIENTADA A SERVICIOS PARA OPTIMIZACIÓN MONO-OBJETIVO BASADA EN HEURÍSTICAS SIN USO DE GRADIENTE

Tesis Presentada como Requisito Parcial
para la Obtención del Título de
Máster en Ingeniería Asistida por Computadora

Autor: Ing. Michel Fernández González

Tutores: Dr.C. Ramón Quiza Sardiñas
Dr.C. Rodolfo Elías Haber Guerra

Matanzas, 2019

DECLARACIÓN DE AUTORIDAD Y NOTA LEGAL

Yo, Michel Fernández González, declaro que soy el único autor de la siguiente tesis, titulada *Aplicación con arquitectura orientada a servicios para optimización mono-objetivo basada en heurísticas sin uso de gradiente* y, en virtud de tal, cedo el derecho de copia de la misma a la Universidad de Matanzas, bajo la licencia *Creative Commons* de tipo *Reconocimiento No Comercial Sin Obra Derivada*, con lo cual se permite su copia y distribución por cualquier medio siempre que mantenga el reconocimiento de sus autores, no haga uso comercial de la obra y no realice ninguna modificación de ella.

Matanzas, 16 de febrero de 2019.

Michel Fernández González

RESUMEN

La optimización juega un papel crucial en la industria manufacturera actual. Este trabajo se enfoca al diseño y la implementación de una aplicación para computación en la nube, basada en una arquitectura orientada a servicios, para resolver problemas de optimización mono-objetivo, usando heurísticas sin uso de gradiente. Este enfoque permite aplicar la usualmente costosa computacionalmente optimización basada en heurísticas en el entorno altamente eficiente de la computación en la nube. Una biblioteca de clases fue primeramente desarrollada para implementar los principales elementos de un problema de optimización, así como tres populares heurísticas mono-objetivo: algoritmo genético, entropía cruzada y enjambre de partícula. El diseño de esta biblioteca también garantiza su futura extensibilidad, con el objetivo de incluir nuevas heurísticas. Sobre la base de esta biblioteca de clases, se diseñó y desarrolló un sistema cliente-servidor. Se establecieron, además, los correspondientes protocolos de comunicación. El funcionamiento de la aplicación cliente-servidor fue validada utilizando un conjunto de problemas estándares de prueba, los cuales fueron tomados de la literatura. Las tres heurísticas propuestas mostraron un adecuado funcionamiento, aunque el método de entropía cruzada mostró mejores resultados que los otros dos algoritmos en todos los problemas considerados.

Palabras claves: Optimización; Heurísticas sin uso de gradiente; Computación en la nube.

ABSTRACT

Optimization plays an important role in current manufacturing industry. This work is focused on designing and implementing a cloud-computing application, based on a service-oriented architecture approach, for solving single-objective optimization problems by using gradient-free heuristics. This approach allows to apply the usually computationally expensive heuristic-based optimization in high-performance cloud-computing environment. A class library was firstly developed for implementing the main features of an optimization problem and three very popular single-objective heuristics: genetic algorithm, cross-entropy and particle swarm optimization. The design of this library also guarantees its future extensibility, in order to include new heuristics. On the basis of this class library, a client-server system was designed and developed. The corresponding communication protocols were also established. The performance of the client-server application was validated by using a set of standard test problems, which were taken from the literature. The three proposed heuristic show an acceptable performance but the cross-entropy method outperformed the other two algorithms in all the considered problems.

Keyworks: Optimization; Gradient-free heuristics; Cloud-computing

TABLA DE CONTENIDO

INTRODUCCIÓN

La optimización de los procesos tecnológicos juega un papel fundamental en la manufactura moderna, al permitir obtener productos más competitivos, dentro de un mercado exigente y dinámico (Quiza, *et al.*, 2014). No obstante, en la práctica industrial, las variables involucradas se relacionan por modelos caracterizados por la no linealidad, la incertidumbre y la presencia de ruido (Minisci, *et al.*, 2019), lo cual hace que no se cumplan los presupuestos de continuidad, suavidad y unimodalidad exigidos por los métodos de optimización analíticos o iterativos (Bronshtein, *et al.*, 2015).

Con vistas a superar esta dificultad, se han desarrollado un grupo de técnicas, conocidas genéricamente como heurísticas sin gradientes (*gradient-free heuristics*), las cuales son algoritmos inspirados en la naturaleza y que, mayormente, se basan en el uso de un conjunto de soluciones que, iterativamente, convergen acerca al óptimo (o un valor cercano a él) (Bozorg-Haddad, 2018). A pesar de que no hay una heurística universalmente efectiva (Wolpert y Macready, 1997), dichas técnicas son notablemente robustas, en el sentido de que son capaces de solucionar un amplio espectro de problemas sin necesidad de realizar modificaciones en las mismas (Fidanova, 2018).

Dentro de las heurísticas sin gradiente más utilizadas están los algoritmos genéticos (Affenzeller, *et al.*, 2009), el enjambre de partículas (Bonyadi y Michalewicz, 2017), el recocido simulado (Bouhmala, 2018) y la entropía cruzada (Rubinstein y Kroese 2004). Dichos métodos han sido ampliamente aplicados a un grupo de problemas industriales, dentro de los cuales cabe mencionar al maquinado convencional (La Fé, *et al.*, 2018; Umer,

et al., 2014), el micromaquinado (Attanasio, *et al.*, 2017; La Fé, *et al.*, 2019) y la soldadura (La Fé, *et al.*, 2017; Yan, *et al.*, 2016).

La principal desventaja de las heurísticas sin gradiente, es el alto costo computacional de las mismas, dadas por el uso de poblaciones de soluciones que son evaluadas en paralelo (Yang, 2018). Esto hace que sea altamente conveniente ejecutarlos en sistemas de cómputo de alto rendimiento (Gong, *et al.*, 2015). En este sentido, los sistemas de arquitectura orientada a servicios (Roten-Gal-Oz, 2012) y la computación en la nube (Hwang, 2017) son paradigmas de gran utilidad.

A partir de todo lo anterior, se identifica como **problema** a resolver en la presente investigación, *la no existencia, hasta donde se pudo comprobar, de una aplicación orientada a servicios, para la optimización basada en heurísticas sin gradiente*.

Como posible solución de este problema, se planteó como **hipótesis** que, mediante el uso de herramientas de software libre, *es posible desarrollar una aplicación, basada en una arquitectura orientada a servicios, para resolver problemas de optimización mediante el uso de heurísticas sin gradiente*.

Como **objetivo general**, se trazó, entonces, *diseñar, implementar y validar una aplicación cliente/servidor, basada en la suite de protocolos TCP/IP, para la solución de problemas de optimización mono-objetivo, mediante heurísticas sin gradiente*.

Para cumplir dicho objetivo general, se definieron como **objetivos específicos**:

1. Diseñar e implementar la biblioteca de clases para optimización mono-objetivo, mediante el uso de heurísticas sin gradiente.

2. Establecer los protocolos de comunicación entre la aplicación de servidor para optimización mono-objetivo mediante el uso de heurísticas sin gradiente, y sus clientes.

3. Diseñar e implementar del software de servidor, para optimización mono-objetivo, mediante el uso de heurísticas sin gradiente.

4. Diseñar e implementar una aplicación cliente para optimización mono-objetivo, mediante el uso de heurísticas sin gradiente.

5. Validar el sistema cliente/servidor desarrollada mediante casos de estudio tomados de la literatura especializada.

La investigación desarrollada es un resultado del proyecto “Heurísticas inteligentes para optimización multiobjetivo con aplicación a los procesos industriales”, asociado al programa de Ciencias Básicas.

CAPÍTULO 1 ESTADO DEL ARTE

En este capítulo se presenta el estado del arte en la temática tratada, a través de una revisión crítica de la literatura. Se hace hincapié en los conceptos básicos de optimización, en la descripción de las heurísticas mono-objetivo, y en los fundamentos de la computación basada en la nube.

1.1 Conceptos y técnicas básicas de optimización

1.1.1 Formulación intuitiva

La optimización, entendida en un sentido elemental, es la selección de la mejor opción ante un problema dado. Vista así, juega un papel crucial en todas las ramas del hacer humano, aunque, muchas veces, esta selección se lleva a cabo de un modo puramente instintivo (Fidanova, 2018). Por el contrario, en la ciencia y la tecnología, las decisiones se basan en modelos matemáticos que relacionan las diversas variables o factores que intervienen en un determinado proceso o sistema (Henderson, *et al.*, 2003).

En primer lugar, se llama objetivo, a aquella magnitud que se desea optimizar. Esta optimización, en dependencia del problema considerado, puede ser una maximización o una minimización. No obstante, para el tratamiento matemático, ambas son equivalentes, por lo cual, se puede considerar que, sin pérdida de generalidad, la optimización es siempre una minimización (Coello, *et al.*, 2007). En este sentido, es conveniente aclarar que un objetivo a maximizar se puede convertir en uno a minimizar, simplemente multiplicándolo por menos uno.

Se llaman variables de decisión, a los factores que influyen sobre el objetivo de optimización y que pueden ser modificados, arbitrariamente, en el problema considerado (Zhang, 2018). Esta modificación, sin embargo, está restringida, normalmente, por los intervalos de definición de las variables de decisión y por determinadas relaciones entre ellas (Minisci, *et al.*, 2019).

1.1.2 Conceptos formales

Con el objetivo de llevar a cabo la optimización, mediante técnicas matemáticas, se requieren definiciones formales de los elementos enunciados en la sección anterior. La *función objetivo*, se define, entonces, por la relación funcional:

$$f : \Omega \subset \mathbb{R}^n \to \mathbb{R},\ \Omega \neq \varnothing; \tag{1.1}$$

donde $\mathbf{x} \in \Omega$ es el vector de *variables de decisión*, cuyo dominio de definición, Ω, está limitado por los intervalos:

$$l_i \leq x_i \leq u_i,\ i = 1, ..., n; \tag{1.2}$$

así como por las restricciones de desigualdad:

$$g_i(\mathbf{x}) \leq 0,\ i = 1, ..., p; \tag{1.3}$$

y las restricciones de igualdad:

$$h_i(\mathbf{x}) \leq 0,\ i = 1, ..., q. \tag{1.4}$$

El valor $f^* \triangleq f(\mathbf{x}^*) > -\infty$ es llamado un *mínimo global*, si y sólo si, $\forall \mathbf{x} \in \Omega: \ f(\mathbf{x}^*) \leq f(\mathbf{x})$ (Van Veldhuizen, 1999). Un problema de optimización, donde se considera un único objetivo, se denomina *mono-objetivo* (Mirjalili, 2016).

1.2 Técnicas de optimización

1.2.1 Técnicas analíticas

Las técnicas de optimización analíticas son las técnicas más antiguas de todas las utilizadas, pero también las más exactas. Éstas consisten en determinar el punto estacionario, es decir el punto donde la primera derivada de la función es cero (Quiza, *et al.*, 2012):

$$\frac{df}{dx} = 0 \tag{1.7}$$

En casos en que la variable de decisión es un vector en lugar de un escalar, el punto estacionario será encontrado en donde todos los componentes del gradiente de la función se hacen cero:

$$\nabla f = \frac{\partial f}{\partial x_1} e_1 + ... + \frac{\partial f}{\partial x_n} e_n = 0 \tag{1.8}$$

Como la condición antes mencionada es necesaria pero no suficiente, es necesario que la segunda derivada (o la matriz Hessiana) para el vector de decisión, sea verificada con el objetivo de saber si el punto estacionario corresponde un máximo, un mínimo, o un punto de montura (Yang, *et al.*, 2014).

Las técnicas analíticas tienen un fuerte basamento matemático, pero sólo trabajan bien para problemas de baja complejidad. Otra de sus limitaciones está en las restricciones de optimización, aunque algunos métodos como el multiplicador de Lagrange han sido desarrollado con este propósito (Bronshtein, *et al.*, 2015).

1.2.2 Técnicas numéricas

Las técnicas numéricas están, también, basadas en el gradiente de la función objetivo, pero a diferencia de las técnicas analíticas no necesitan del cálculo de las raíces de las derivadas, las cuales a menudo son ecuaciones trascendentales. Las técnicas numéricas empiezan en algún punto $\mathbf{x}_0$ y, de manera iterativa, calculan un nuevo punto $\mathbf{x}_i$ de un punto $\mathbf{x}_{i-1}$ previo, por seguimiento del gradiente hasta que alguna condición de parada es alcanzada (Quiza, *et al.*, 2014).

Dentro de las técnicas numéricas existen dos grandes grupos: aquellas que usan la matriz Hessiana (como el método de Newton) o aquellos que usan gradiente (como son, el gradiente conjugado o el gradiente descendiente). En algunas ocasiones las derivadas no son directamente evaluadas sino aproximadas usando el método de diferencias finitas.

Dos inconvenientes tienen los métodos iterativos. En primer lugar, la elección del punto de comienzo (o semilla) influye significativamente en la convergencia del método. El otro de los contratiempos es que el óptimo encontrado podría ser un óptimo local en vez de uno global. También, para una aplicación exitosa de la mayoría de los métodos iterativos la función objetivo debe de ser continua y diferenciable (Ravindran, *et al.*, 2006).

1.2.3 Técnicas heurísticas

1.2.3.1 Características generales

Las heurísticas no son métodos exactos de optimización, sino técnicas que ofrecen soluciones cercanas al óptimo (quasi-óptimas). Sin embargo, con una adecuada configuración, estos métodos pueden dar precisiones arbitrariamente bajas, lo que las hace especialmente atractivas para la solución de problemas técnicos. Las heurísticas, a diferencia de los métodos analíticos o numéricos, no requieren que la función objetivo cumpla con las condiciones de continuidad, suavidad y unimodalidad, por lo que pueden ser aplicadas a una amplia gama de problemas industriales (Talbi y Nakib, 2019).

Dentro de las heurísticas mono-objetivos más empleadas se encuentran los algoritmos genéticos, la optimización por enjambre de partículas y el método de entropía cruzada (Gupta y Ong, 2019), las cuales se describen a continuación.

1.2.3.2 Algoritmo genético

Los algoritmos genéticos (*genetic algorithma*, GA) son el grupo más popular dentro de las heurísticas sin gradiente para resolver problemas de optimización mono-objetivo. Su principal característica distintiva es la codificación de cada solución individual en una cadena llamada cromosoma (Punia y Kaur, 2013). Esta codificación hace que el algoritmo sea independiente del problema, por lo que pueden considerarse de naturaleza robusta**.** Los GA utilizan un conjunto de soluciones (conocidas como población) que se evalúan y procesan en paralelo (Saborido*, et al.*, 2016).

En los GA (ver Pseudocódigo 1.1) una población inicial de individuos, también llamada soluciones candidatas o cromosomas es generada aleatoriamente. Durante cada paso de la iteración, llamada también generación, los individuos de la población actual son evaluados y se les asigna un determinado valor de ajuste o aptitud, según la función objetivo y el cumplimiento de las restricciones. Con el objetivo de formar una nueva población, los individuos son previamente seleccionados, comúnmente con una probabilidad que está en correspondencia con su valor de aptitud; cada uno de estos valores seleccionados producirá posteriormente una descendencia que serán la próxima generación de padres (Buontempo, 2019).

Pseudocódigo 1.1. Algoritmo genético

```
INICIO ga
   crear una población inicial de forma aleatoria
   MIENTRAS alcance la condición de parada
      evaluar la población
      crear la nueva población (selección + cruzamiento + mutación)
   FIN MIENTRAS
FIN ga
```

Para generar nuevas soluciones candidatas los AG usan dos operadores conocidos como cruzamiento y mutación. El cruzamiento es el operador genético primario. Este es el encargado de tomar dos individuos que serán llamados padres y que van a producir uno o dos nuevos individuos, llamados descendencia, producto a la combinación de los padres. En la forma más simple del procedimiento el operador trabaja intercambiando sub-cadenas antes y después de un punto de entrecruzamiento seleccionado aleatoriamente (Minisci, *et al.*, 2019).

El operador mutación es, básicamente, una modificación de manera arbitraria para evitar convergencias prematuras mediante el muestreo de nuevos puntos en el espacio de

búsqueda. En el caso de las cadenas de bits, la mutación ocurre colocando bits de manera aleatoria con una determinada probabilidad (Fidanova, 2018).

Este proceso se repite hasta lograr cualquiera de las condiciones de parada, usualmente, al alcanzar el número máximo de iteraciones o una determinada convergencia en las soluciones. (Affenzeller, *et al.*, 2009). Todos los operadores incorporan algún comportamiento aleatorio, lo que garantiza la capacidad de encontrar la solución óptima global incluso cuando no está incluida en la población inicial (Jacobson y Kanber, 2015).

La selección se realiza, en los algoritmos genéticos, siguiendo dos enfoques fundamentales (Talbi y Nakib, 2019):

- *Método de selección por ruleta o selección proporcional*: En este enfoque, el número de descendientes esperados para un individuo está dado es proporcional a su ajuste. Por consiguiente, cada individuo de la población es representado por un espacio de tamaño proporcional a su aptitud. La selección es aleatoria, pero dentro de los espacios de tamaño previamente calculados.
- *Método de selección por torneo*: Se basa en seleccionar un número, igual o mayor de dos, de individuos y compararlos entre sí. El más apto de ellos, es considerado como progenitor. Este método, con respecto al de selección por ruleta, tiene la ventaja de no requerir ordenamiento (que es un proceso computacionalmente complejo), pero tiene el inconveniente de que la presión de selección (y, por consiguiente, la calidad de los resultados, es muy dependiente del tamaño de la selección).

En su forma más simple para codificación binaria, la técnica entrecruzamiento convencional incluye (Siarry, 2016):

- *Entrecruzamiento de punto único:* Un corte es hecho de manera aleatoria, generando dos cadenas cada una con su cola y su cabeza. Las dos colas son entonces intercambiadas de posición para producir dos nuevos individuos llamados cromosomas.
- *Entrecruzamiento multipunto:* En el entrecruzamiento multipunto se crean n puntos de entrecruzamiento, las subcadenas conformadas son intercambiadas alrededor de los n puntos. De acuerdo con varios autores el entrecruzamiento multipunto es más apropiado para combinar las buenas características presentes en una cadena, dado que muestrea de manera uniforme a lo largo de un cromosoma. Al mismo tiempo múltiples puntos de entrecruzamiento hacen un mayor número de sub-cadenas a medida que el número de puntos de entrecruzamiento aumenta. Disminuir el número de puntos de entrecruzamiento durante la ejecución de GA puede ser un buen compromiso.
- *Entrecruzamiento uniforme:* Dado dos padres, cada gen en la descendencia es creado por la copia del correspondiente gen de uno de los padres. La selección del padre correspondiente es llevada a cabo por una máscara de entrecruzamiento aleatoria. A cada índice, el gen de la descendencia es tomado del primer padre si hay un 1 en la máscara del índice, por otra parte, si hay un cero en la máscara del índice entonces el gen es tomado del segundo padre.

Los operadores de mutación permiten de manera indirecta hacer pequeños saltos dentro de diferentes áreas del espacio de búsqueda (Kramer, 2017). El operador básico de mutación

para problemas con codificación de tipo binaria es el *bitwise*. La mutación ocurre, gen a gen, de manera aleatoria y con una probabilidad baja (usualmente por debajo de 0,01%). En muchos casos la mutación es interpretada como la generación de un nuevo *bit* y en otros casos como intercambio del *bit*. En la formulación de numeración entera (*integer numbering formulation*) la mutación se lleva a cabo remplazando un alelo con un valor aleatoriamente escogido en el rango apropiado, con cierta probabilidad (Li, *et al.*, 2013).

El elitismo es otro concepto importante, en los algoritmos genéticos. Consiste en tomar el mejor individuo o los mejores n individuos respectivamente, de la generación anterior y conservarlos para la próxima generación lo cual puede llegar a ser crítico para la convergencia anticipada. La estrategia más usada de mantener el mejor individuo de la generación anterior es conocido como el modelo de la jaula dorada. Si la mutación es aplicada al valor elite con el objetivo de evitar una convergencia prematura, el mecanismo de remplazo se conoce como de bajo elitismo (Chouhan, *et al.*, 2018).

1.2.3.3 Recocido simulado

El recocido simulado (*simulated annealing*, SA) es otra heurística popular de optimización sin gradientes. Se basa en el proceso de enfriamiento de metales (Haber, *et al.*, 2009). En el proceso de recocido metalúrgico, el enfriamiento lento tiene como objetivo obtener un estado de energía mínima global en un metal, lo que proporciona un estado estructural estable y evita los estados meta-estables con mayor energía. De manera similar, el método de SA se enfoca en alcanzar el óptimo global de una función matemática, evitando los óptimos locales (Siarry, 2016). El algoritmo de SA trabaja con un único punto de solución, que se selecciona aleatoriamente. También se inicializa un parámetro del algoritmo,

llamado temperatura, que determina la probabilidad de moverse de un estado a otro (Pseudocódigo 1.2).

Pseudocódigo 1.2. Algoritmo de recocido simulado

```
INICIO sa
   elegir una solución inicial aleatoria y una temperatura inicial
   MIENTRAS alcance la condición de fin de ciclo
      crear una solución a través de una distribución gaussiana
      SI (nueva solución es mejor que la anterior)
         aceptar nueva solución
      SINO SI (la prob. asociada es mayor que un valor elegido al azar)
         aceptar nueva solución
      SINO
         mantener solución anterior
      FIN SI
      actualizar con la mejor solución y la temperatura
   FIN MIENTRAS
END sa
```

El proceso de optimización se lleva a cabo en un ciclo, que finaliza cuando se logran algunas condiciones: generalmente, cuando el parámetro de temperatura alcanza algún valor prescrito y después de un número máximo de iteración. En cada iteración, se intenta una nueva solución que se mueve, desde la existente, una distancia aleatoria con una distribución gaussiana. Si la nueva solución es mejor (es decir, tiene un mejor valor de función de aptitud), se acepta. Por el contrario, si es peor, la probabilidad, $p = \exp(\Delta f / T)$ se calcula y se compara con un número aleatorio, *r*. Si $p > r$, la nueva solución es aceptada (incluso siendo peor que la anterior). De lo contrario, se conserva la vieja solución. En cada iteración, el valor óptimo global obtenido en todo el proceso se actualiza (si corresponde), el valor de la temperatura también se actualiza (Yang, 2018).

1.2.3.4 Optimización de enjambre de partícula

La optimización por enjambre de partículas (particle swarm optimization, PSO) es un método estocástico, basado en la población, de optimización global mayormente usado en

problemas de tipo continuo o sea con continuidad en el espacio de búsqueda. Fue desarrollado en el año 1995 y se basa en el comportamiento social de algunas poblaciones de animales (aves o peces). Este método ha probado su eficiencia en problemas de ingeniería y ha causado un gran impacto en las comunidades de la optimización y de la ingeniería. Esta depende de un grupo de parámetros para su puesta en marcha (Bonyadi y Michalewicz, 2017).

Inicialmente la PSO fue propuesta para resolver problemas sin restricciones. El método es inicializado con un grupo de partículas, cada una correspondiente a una solución candidata en particular. La PSO encuentra el óptimo global moviendo cada una de las partículas con una velocidad, $v_i(t)$, que es actualizada en cada iteración, t, teniendo en cuenta tres parámetros que consideran la resistencia al cambio (inercialidad, w), la influencia de la mejor posición alcanzada por la partícula (aceleración cognitiva, c_1) y la influencia de la mejor posición de todo el enjambre (aceleración social, c_2):

$$v_i(t) = wv_i(t-1) + c_1r_1[x_i^* - x_i(t-1)] + c_2r_2[x^{**} - x_i(t-1)], \quad i = 1 \ldots Z \ ; \qquad (1.9)$$

ronde r_1 y r_2 son vectores aleatorios de una distribución aleatoria uniforme, $\mathcal{U}_{[0,1]}$, introducidos para mantener la diversidad del enjambre. La mejor posición de cada partícula, x_i^*, y de toda la población, x^{**}, son actualizadas durante cada iteración.

La posición $x_i(t)$ de cada partícula es modificada, entonces, como se muestra a continuación:

$$x_i(t) = x_i(t-1) + v_i(t), \quad i = 1 \ldots Z \ . \qquad (1.10)$$

El algoritmo (Pseudocódigo 1.3) se encarga de que cada posición ocupada por la partícula esté dentro del espacio de búsqueda, modificando los valores correspondientes cuando caen fuera de dicho espacio.

Pseudocódigo 1.3. Optimización de enjambre de partículas

```
INICIO pso
   generar una posición y velocidad aleatoria
   MIENTRAS alcance las condiciones de parada
      evaluar población
      elegir la mejor posición p. cada part. y la mejor posición global
      actualizar velocidad
      actualizar posición
   FIN MIENTRAS
FIN pso
```

1.2.3.5 Algoritmo de entropía cruzada

El método de entropía cruzada (*cross-entropy*, CE) es una heurística basada en la población que resuelve los problemas de optimización al transformarlos en problemas estocásticos asociados con una probabilidad muy pequeña utilizando alguna técnica de minimización de la varianza (Haber, *et al.*, 2010). El método de CE se basa en la construcción de una secuencia aleatoria de soluciones que convergen, probabilísticamente, hacia una solución óptima o quasi-óptima (Beruvides, *et al.*, 2017).

El método de CE ha sido satisfactoriamente utilizado en una variedad de problemas de optimización de tipo combinatorio y de estimación de eventos de baja probabilidad de ocurrencia, este último con distribuciones tanto de tipo ligeras como densas (Arab, *et al.*, 2018). Las áreas de aplicaciones del algoritmo de CE incluyen buffers de asignación o distribución, modelos para colas de sistemas de telecomunicaciones, computación neuronal, control y navegación, secuenciación de ADN, procesamiento de señales, ruteo

de vehículos, reforzamiento del aprendizaje, manejo de proyectos y sistemas confiables. Es importante tener en cuenta que el método CE trabaja de manera exitosa tanto con problemas determinísticos como es el caso del problema del vendedor viajero, así como con problemas ruidosos como es el caso de problemas de distribución de buffer (Gao, *et al.*, 2018).

El método de CE (ver Pseudocódigo 1.4) comienza con la inicialización de una distribución gaussiana para cada una de las variables de decisión, las cuales se llevan a través del cálculo de las respectivas medias y varianzas, a partir de los dominios de definición de cada variable, con una componente aleatoria (De Boer, *et al.*, 2005).

Pseudocódigo 1.4. Algoritmo de entropía cruzada

```
INICIO ce
   inicializar media y varianza
   MIENTRAS condición de parada sea alcanzada
      generar una población de trabajo aleatoria gaussiana
      evaluar la población de trabajo
      extraer la población elite de la población de trabajo
      recalcular media y varianza a partir de la población élite
   FIN MIENTRAS
FIN ce
```

Se ejecuta, entonces, un ciclo en cada una de cuyas iteraciones se genera una población de trabajo mediante una distribución gaussiana, con la media y varianza dadas, y acotada al intervalo definido para cada variable de decisión. Esta población es evaluada y, de la misma, se extrae la población de élite, formada por una fracción de sus individuos con mejor función objetivo.

Finalmente, se recalculan las medias y varianza de cada variable de decisión, utilizando para ello los respectivos valores de la población de élite (Olivares-Mendez, *et al.*, 2014).

El ciclo es repetido hasta que se alcanza la condición de parada que, usualmente, es el número máximo de iteraciones o la convergencia de las soluciones, dada por valores de varianza pequeños (Pérez, *et al.*, 2019).

1.3 Arquitectura orientada a servicios y computación en la nube

1.3.1 Conceptos básicos

La arquitectura orientada a servicios (*service-oriented architecture*, SOA) es un estilo utilizado para la construcción de sistemas informáticos, basada en las interacciones de componentes autónomos, levemente acopladas, llamados servicios. Cada servicio muestra procesos y comportamientos a través de contratos, los cuales se componen de mensajes a direcciones conocidas (o descubribles) llamadas puntos finales (*endpoints*). El comportamiento de un servicio es regido por políticas que son externas al propio servicio. Los contratos y mensajes son usados por componentes externos llamados consumidores del servicio (*service consumers*) (Roten-Gal-Oz, 2012)

La SOA se basa en un grupo de principios dentro de los cuales pueden destacarse los siguientes (Kanneganti y Chodavarapu, 2008):

- Las aplicaciones deben dar acceso a sus funcionalidades para que sean utilizadas por otras aplicaciones existentes o futuras. Debe ser posible combinar los servicios ofrecidos por diferentes aplicaciones para crear servicios de alto nivel o aplicaciones compuestas.
- Las diferencias tecnológicas no deben ser importar y la interoperatividad debe ser un aspecto clave.

- Deben adoptarse estándares abiertos para habilitar la integración entre empresas. La orquestación de procesos de negocios a través de múltiples suministradores, socios y clientes debe ser posible.
- Debe prestarse atención a la gobernabilidad y manejabilidad de los sistemas, con el objetivo de evitar que los tres principios anteriores no lleven al caos.

Para ser efectivos, los servicios deben poseer atributos, dentro de los que se destacan:

- Estar definido en el nivel adecuado de granularidad, desde el punto de vista del usuario.
- Describirse a sí mismo, de modo que los usuarios entiendan, con facilidad, cómo invocarlo.
- Ser independientes de la tecnología, de forma que los usuarios no se vean restringidos por el hardware o los sistemas utilizados.
- Ser fáciles de encontrar por los usuarios a partir de búsquedas en registros de servicios.
- Poder ser integrados con otros servicios para conformar servicios de nivel superior.
- Ser contextualmente independientes, es decir, responder de la misma forma, sin importar lo que haya hecho el usuario antes de invocarlo.
- No mantener información entre una transacción y otra, lo cual hace fácil para proveedores y usuarios crear y utilizar servicios, respectivamente.

Una parte importante de los servicios se implementa sobre la suite protocolos TCP/IP (*Transmission Control Protocol/Internet Protocol*), especialmente, en forma de servicios

web, es decir, aquellos que se basan en el uso del lenguaje de descripción de servicios web (*web services description language*, WSDL) (Wang y Kissel, 2015).

Computación en la nube (*cloud computing*) o servicios en la nube, es un nuevo paradigma computacional que se pudiera explicar como un entramado tecnológico donde la propiedad y responsabilidad de la distribución de los servicios de informática y las telecomunicaciones (*informatics and tele-comunications*, IT) es proporcionado por una organización centralizada que puede ser una empresa o una organización pública. Estas organizaciones manejan la arquitectura de la nube y brindan los servicios de la nube a los consumidores, a la vez que se responsabilizan por el manejo apropiado de la infraestructura de la nube y ofrecen los servicios basados en la misma a los clientes. Los clientes que utilizan los servicios de la nube aceptan los términos de la misma debido a que esperan obtener bajos costos y alta flexibilidad en vez de ser ellos los propietarios y proveedores de la infraestructura. La tecnología de la nube permite reducir costos y acceder a tecnologías y soluciones de IT que son demasiado complejas de implementar y ejecutar por el usuario (Oppitz y Tomsu, 2018).

La computación en la nube ha surgido a partir de la convergencia de cuatro tecnologías (Fig. 1.1): (i) la virtualización del hardware y los procesadores multi-núcleos; (ii) las tecnologías de utilidades y la computación en rejilla; (iii) los avances recientes en la arquitectura orientada a servicios, la Web 2.0 y otras plataformas de servicios y (iv) la computación autónoma y las operaciones de centros de datos automatizados (Hwang, 2017).

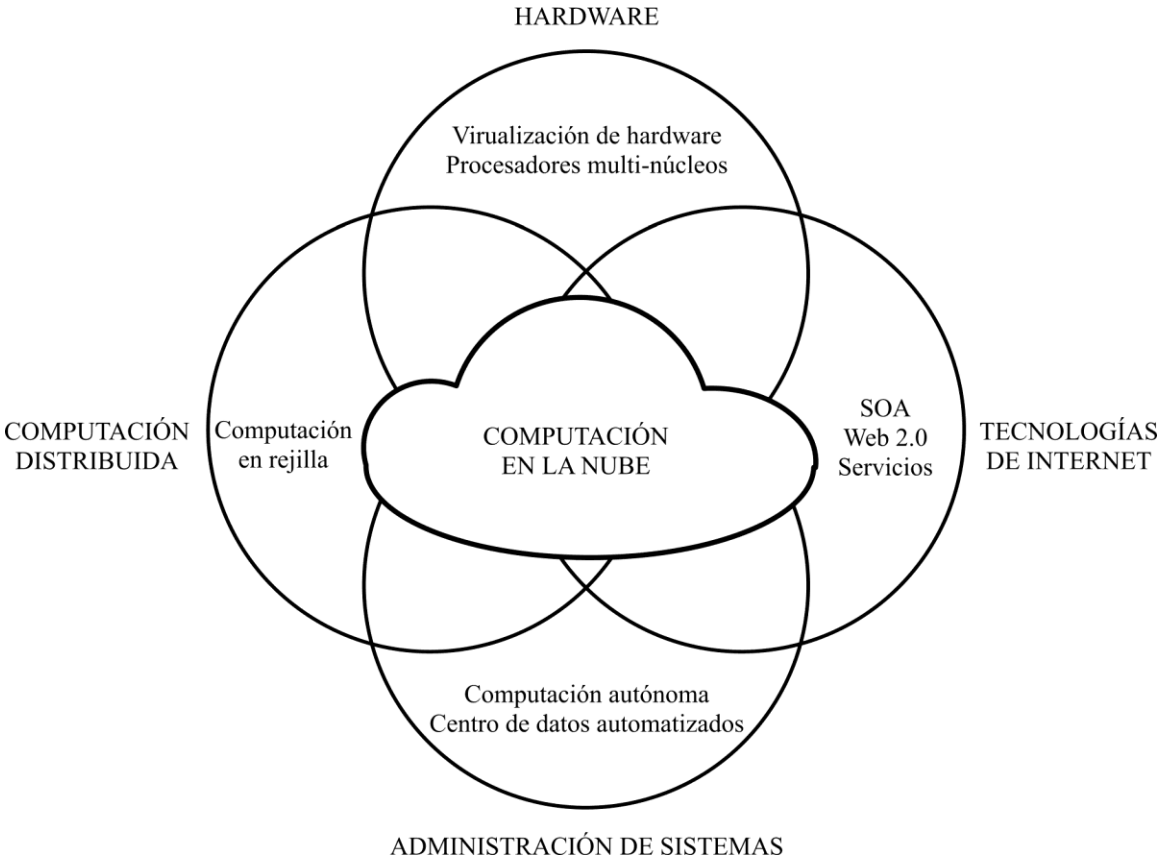

Figura 1.1 Convergencia de tecnologías en la computación en la nube (Hwang, 2017)

La computación en la nube pude implementase en base a cuatro modelos de servicios (Wang y Kissel, 2015):

- Software como servicio (*software-as-a-service*, SaaS): Proporciona, a los usuarios, acceso sólo a determinadas aplicaciones. Es el modelo de mayor antigüedad y el que menos libertad deja al cliente.
- Plataforma como servicio (*platform-as-a-service*, PaaS): Permite a los clientes desarrollar sus propias aplicaciones en la nube. El control y administración de la infraestructura recae, totalmente, en la nube.

- Infraestructura como servicio (*infrastructure-as-a-service*, IaaS): Permite a los clientes instalar y utilizar sus propios sistemas operativos en la nube. Ésta sólo proporciona el mantenimiento del hardware.
- Almacenamiento como servicio (*storage-as-a-service*, STaaS): El proveedor mantiene una capacidad de almacenamiento accesible para el usuario. A diferencia de los tres modelos anteriores, no está oficialmente definido Instituto Nacional de Estándares y Tecnología (*National Institute of Standards and Technology*, NIST) de los EE.UU., pero ha ganado gran popularidad recientemente.

1.3.3 Informática de utilidad

La informática de utilidad se basa en un modelo de negocios mediante el cual los usuarios o clientes reciben una gama de recursos informáticos por parte del distribuidor del servicio que puede ser del tipo internet de las cosas (*internet of things*, IoT) o sencillamente por parte de la nube. La informática de la utilidad ha posibilitado que los usuarios puedan exigir nuevos recursos a la nube como procesadores más eficientes en la red, memoria ampliable y estructuras de almacenamiento, sistemas operativos distribuidos, *middleware* para virtualizar las máquinas, nuevos modelos y métodos de programación, administración efectiva de recursos y el desarrollo de aplicaciones (Oppitz y Tomsu, 2018).

En los últimos años ha habido un cambio en la tendencia al pasar de un uso de la computación de alto rendimiento (*high-performance computing*, HPC) a un uso masivo de la computación de alto flujo (*high-throughput computing*, HTC). Las plataformas punto a punto (*point-to-point*, P2P) de computación en la nube y los servicios *web* se enfocan más en la HTC que en aplicaciones HPC. El diseño tecnológico HTC se enfoca más en el

manejo de un alto flujo, en el que millones de usuarios solicitan de manera simultaneas servicios disímiles en la red. Es de suprema importancia en los servicios de tipo HTC tener en cuenta el rendimiento energético, el costo, la seguridad y la confiabilidad en la nube.

Una arquitectura en la nube está formada básicamente por hardware y tecnología de red de tipo básica, procesadores de tipo x86, dispositivos de almacenamiento de terabytes de capacidad e internet, son básicamente los componentes que conforman la nube además de la disposición de grandes centros de datos. En cuanto a los centros de procesamientos de datos es importante tener en cuenta que estos están diseñados para centrarse sobre todo en la relación rendimiento-precio. La forma en que se gestiona la eficiencia energética y el rendimiento son mucho más importantes que la velocidad (Hwang, *et al.*, 2012).

1.3.4 Recursos virtualizados en sistemas en la nube

La nube es capaz de brindar servicios a pedido de tipo *hardware*, *software* o plataformas. En cuanto a las plataformas, se puede citar a *MapReduce* que proporciona una nueva forma o modelo de programación que permite manejar el paralelismo de datos con capacidad de tolerancia a fallas natural. Se prevé que, en un futuro no muy lejano, tratar, manipular e interpretar grandes cantidades de datos va a significar enviar la parte computacional del proceso, dígase los distintos tipos de programas a la ubicación de los datos, en lugar de mandar los datos a los grandes centros de procesamientos.

Es la computación en la nube una nueva forma de informática que ha sido definida por muchos usuarios y diseñadores, entre ellos podemos encontrar a IBM, que ha definido la computación en la nube como:

> *Una nube es un conjunto de recursos informáticos virtualizados. Una nube puede alojar una variedad de diferentes cargas de trabajo, incluso trabajo de back-end de estilos por lotes y aplicaciones interactivas de acceso al usuario (Hwang, 2017).*

De acuerdo a esta definición una nube nos permite de cierta manera que las cargas de trabajo se implementen y además aumente de manera considerable gracias a la virtualización de los servicios y el aprovisionamiento rápido de todo tipo de tecnologías físicas y virtuales. Esta tecnología informática asume modelos de programación redundantes, de recuperación automática, altamente escalable o ampliable que hacen posible la recuperación de los sistemas ante fallas de software o hardware de manera invariable y rápida. De manera consecuente los sistemas de tecnologías de tipo nube deberían permitir poder monitorear los recursos informáticos de su sistema en tiempo real para poder hacer un acomodamiento de los mismos según los requerimientos para cuando sea necesaria la reasignación de los recursos (Oppitz y Tomsu, 2018).

1.4 Conclusiones parciales del capítulo

Una vez finalizado el presente capítulo, se ha podido arribar a las siguientes conclusiones parciales:

1. Las heurísticas permiten la solución de problemas optimización, mono-objetivo, sin los requerimientos de continuidad, derivabilidad y unimodalidad, usualmente exigidos por los métodos analíticos y numéricos.

2. Dentro de las heurísticas más empleadas para la optimización uni-objetivo se encuentran los algoritmos genéticos, el recocido simulado, el enjambre de partículas y la entropía cruzada

3. La computación en la nube, es un paradigma que permite aprovechar las capacidades de cómputos de los grupos de ordenadores, desde una arquitectura orientada a servicios, y resulta, por lo tanto, especialmente conveniente para herramientas de alto costo computacional como las heurísticas de optimización.

CAPÍTULO 2 BIBLIOTECA DE CLASES PARA OPTIMIZACIÓN SIN GRADIENTE

En este capítulo se presenta la descripción de la biblioteca de clases desarrollada para la optimización mono-objetivo mediante heurísticas sin gredientes, explicando su estructura y describiendo sus principales componentes.

2.1 Estructura general de la librería

La estructura general de la librería (Fig. 2.1) se construye en base a la librería `LibGFO` (*library for gradient free optimization*), las cual contiene, estructuradas, un grupo de funciones sobre las cuales la aplicación define e implementa las heurísticas. La clase `GFOException` implementa el tratamiento de excepciones, dentro de la librería. Las clases `GFOVariable`, `GFOConstraint` y `GFOObjective`, definen los aspectos claves de los problemas de optimización: las variables de decisión, las restricciones del problema y los objetivos a optimizar, respectivamente. Las clases `GFOVariables`, `GFOConstrains y GFOObjectives`, por su parte, implementa grupos de objetivos de las clases mencionadas previamente. La clase `GFOParameters` contiene, genéricamente, los parámetros de las heurísticas de optimización. La clase abstracta `GFOFunction` define los elementos comunes a las clases que serán utilizadas como función objetivo. De ella, hereda la clase `GFOScriptedFunction`, que es un case específico donde la función es evaluada utilizando el lenguaje *ECMAScript*. En conjunto estas clases sirven como componentes para la clase abstracta `GFOHeuristic`, la cual define los rasgos comunes a todas las heurísticas de optimización. De ella se derivan las clases abstractas `GFOHeuristicSOO` y `GFOHeuristicMOO`. La primera sirve de base a todas las heurísticas mono-objetivo, mientras que la segunda, a las multi-objetivo que, en un futuro, pudieran incorporarse a la

biblioteca. Sobre `GFOHeuristicSOO` se implementan las clases `GFOHeuristicGA`, `GFOHeuristicCE` y `GFOHeuristicPSO`, que implementan los métodos de algoritmo genético, entropía cruzada y enjambre de partícula, respectivamente.

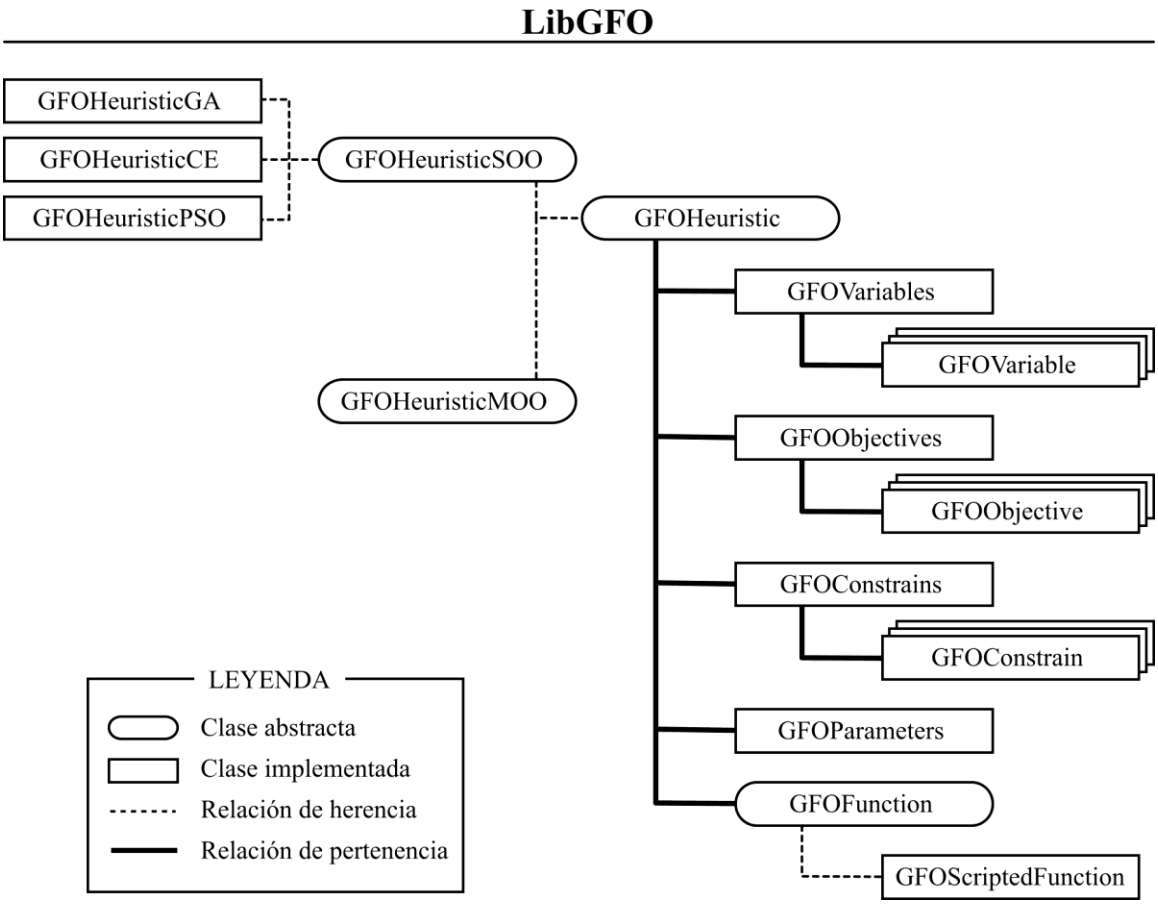

Figura 2.1. Estructura de clases de la librería

2.2 Clases bases comunes

2.2.1 Archivo de cabecera `gfolib.h`

En el archivo de cabecera `gfolib.h` es el fichero principal dentro de la librería de las 85heurísticas, en el están referenciadas todas las clases de la librería para poder incluirlas de una sola vez al ser utilizadas por otro archivo.

2.2.2 Clase `GFOException`

La clase `GFOException` (Fig. 2.2) permite implementar los objetos que generan excepciones ante fallos en el sistema. Se definen tres atributos para esto, el primero una variable numérica de tipo `unsigned int` llamado `my_id` el cual será el identificador numérico del tipo de error, luego dos atributos de tipo `QString`, cadenas de caracteres, llamados `my_description` y `my_message`, el primero de estos almacenará una descripción de la excepción como tal, lo cual permitirá hacer una trazabilidad del error dentro del sistema, el segundo es contendrá un mensaje con una descripción más ampliada para el mejor entendimiento del usuario que trabaja directamente con la aplicación. Los métodos dan la funcionalidad a la clase comenzando por el constructor `FGOException` que permite inicializar el objeto, el método `id()`, devuelve el valor numérico del identificador, `description()` permite generar la descripción del objeto antes explicada y `message()`, crea el mensaje para el usuario.

GFOException
-my_id:UINT -my_description:QString -my_message:QString
+id():UINT +description():QString +message():QString +GFOException(id:UINT,desc:QString,msg:QString)

Figura 2.2 Esquema UML de la clase `GFOException`

2.2.3 Clase `GFOVariable`

La clase `GFOVaiable` (Fig. 2.3), permite crear objetos que representan las variables de decisión que usara la aplicación para resolver los problemas en cuestión. La misma cuenta con tres atributos que serán los que contendrán: el nombre de la variable de decisión,

my_name de tipo QString; el valor mínimo que puede asumir la misma, my_min de tipo double y el valor máximo de la variable, my_max de tipo double también. La clase presenta además su correspondiente constructor para inicializar el objeto de tipo GFOVariable *y* operaciones para acceder a los valores los atributos como son name(), min() y max(), métodos de tipo *getter* y los métodos *setter* dentro de la clase como son setName y setInterval que permiten modificar los valores de dichos atributos para esa clase.

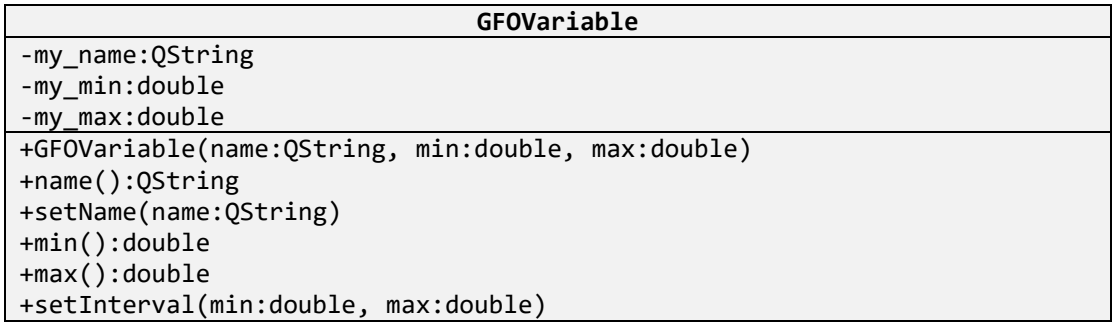

Figura 2.3 Esquema UML de la clase GFOVariable

2.2.4 Clase GFOVariables

La clase GFOVariables (Fig. 2.4) representa el vector conformado por todas las variables de decisión que se pueden representar dentro del problema de optimización. Como único atributo presenta un listado de las instancias de GFOVariable. Los distintos métodos con los que cuenta están en función de realizar operaciones sobre los elementos de la lista como agregar elementos de tipo *variable* a la lista, método append(), modificar el tamaño de la misma eliminado algún elemento con remove() o limpiar la lista con clear() o referenciar algún elemento en específico a través de la sobrecarga del operador [] entre otros. El método toJson() permite crear la estructura para contener en correspondencia

cada variable de decisión con su valor mínimo y máximo asociado. `fromJson()`permite referenciar el elemento.

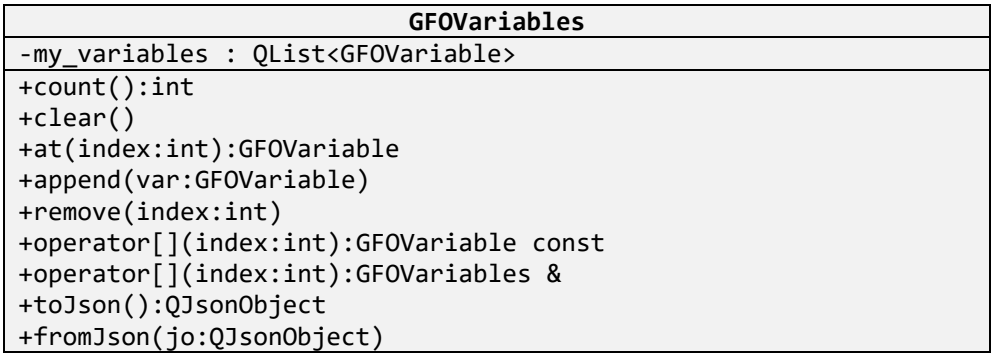

Figura 2.4 Esquema UML de la clase `GFOVariables`

2.2.5 Clase `GFOObjective`

La clase `GFOObjective` (Fig. 2.5) va a permitir describir e implementar a las funciones objetivos de los problemas de optimización dentro de la aplicación. Ésta cuenta con los atributos para almacenar el nombre del objetivo, una variable de tipo `QString` llamada `my_name` y el tipo de objetivo (minimización o maximización) a través de una variable llamada `my_type` de tipo `GFOObjectiveType` así como con las operaciones de acceso a los miembros `name()` y `type()`, métodos *getter* y la realización de las modificaciones correspondientes ocurre con los métodos *setter* `setType()` y `setName()`, su constructor permite inicializar los objetos de la clase.

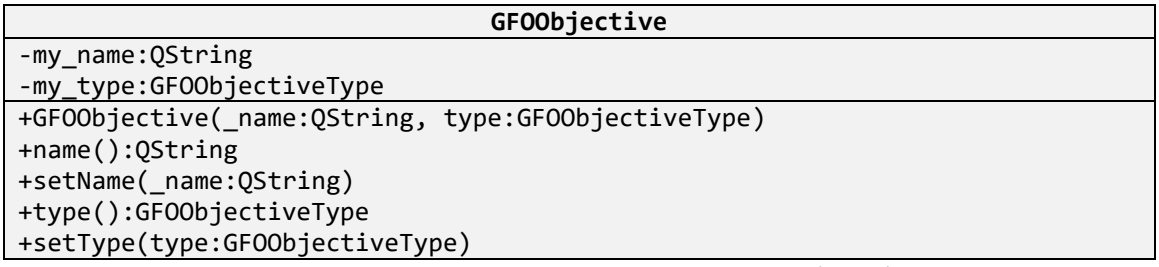

Figura 2.5 Esquema UML de la clase `GFOObjective`

2.2.6 Clase `GFOObjectives`

De forma similar a la clase `GFOVariables`, `GFOObjectives` (Fig. 2.6) es la herramienta que le va a permitir a la aplicación manipular el conjunto de las funciones objetivo con las que cuenta un problema de tipo multiobjetivo. Cuenta, como atributo, con un listado de instancias de la clase `GFOObjective` llamado `my_objectives` y con un conjunto de operaciones típicas para el trabajo con listas como consultar o apuntar a un elemento (o función objetivo) en específico, esto se logró sobrecargando al operador [], modificar dicho listado agregándole o quitándole algún elemento, se volvieron a usar para esto los métodos `append()` y `remove()` respectivamente. Como se explicó previamente el método `toJson()` una vez más permite crear el lazo entre nombre y tipo asociado con cada objeto de tipo `GFOObjective`. `fromJson()`permite tener una referencia de ese objeto.

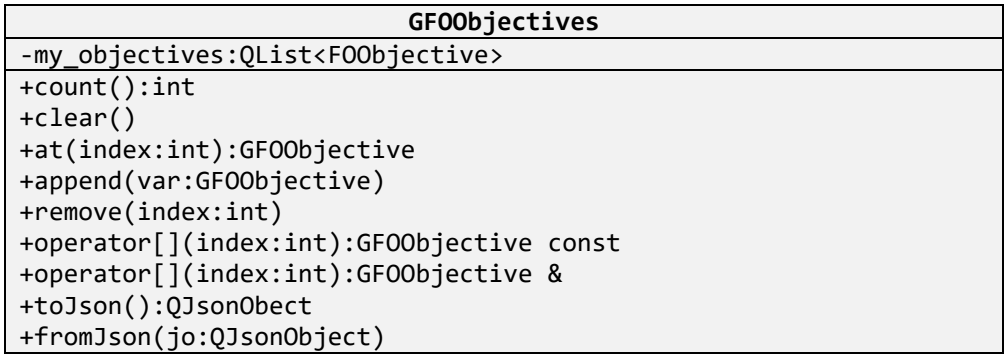

Figura 2.6 Esquema UML de la clase `GFOObjectives`

2.2.7 Clase `GFOParameters`

La clase `GFOParameters` (Fig. 2.7) permite representar el conjunto de pares nombre/valores que contienen los parámetros del algoritmo de optimización. Esto es posible gracias a la utilización de un objeto tipo `QHash` llamado `my_params`, que sirve de

atributo a la clase y permite generar el objeto conformado por los pares de valores de tipo cadena `QString`, para almacenar los nombres y otro listado de tipo numérico `double`, para los valores correspondientes. Cuenta, además, con un grupo de operaciones que permiten agregar, eliminar y consultar los valores de los parámetros, como se ha visto con anterioridad en clases anteriores que usan trabajo con listas, métodos como `append()`, `remove()`, `clear()`, ya fueron explicados así como `toJson()` y `fromJson()`.

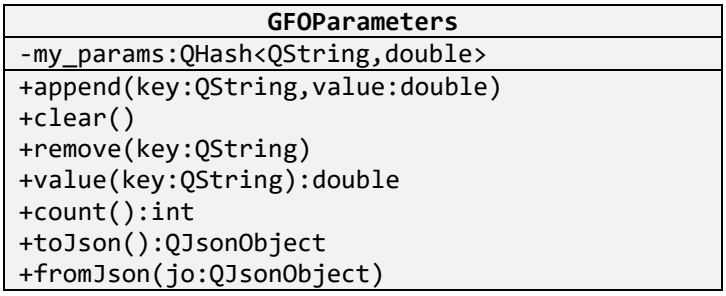

Figura 2.7 Esquema UML de la clase `GFOParameters`

2.2.8 Clase `GFOFunction`

La clase abstracta `GFOFunction` (Fig. 2.8) tiene como funcionalidad permitir evaluar cada una de las heurísticas que se implementa en la aplicación, usando como atributo un apuntador a un objeto de tipo `GFOHeuristic` llamado `my_heuristic` la clase servirá de base para la evaluación de cualquier función objetivo. La misma al ser abstracta le da la libertad al usuario de implementar a placer la forma en la que esta se evaluara, así como la fuente desde la que se llevara a cabo. Cuenta con una operación abstracta llamada `evaluate()`, que toma como argumento una matriz, con los valores de las variables de decisión y devolviendo otra matriz, con los valores de los objetivos. Repiten métodos para su implementación ya mencionados como son `toJson()` y `fromJson()`.

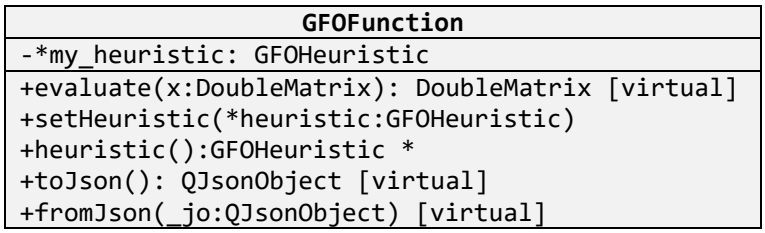

Figura 2.8 Esquema UML de la clase abstracta `GFOFunction`

2.2.9 Clase `GFOScriptedFunction`

La clase `GFOScriptedFunction` (Fig. 2.9) implementa una función a la que se le establece un código en forma de texto que es evaluado mediante el lenguaje ECMAScript (Ecma International, 2018). La clase implementa los métodos para acceder y modificar el código, e implementa, también, la evaluación de dicho código.

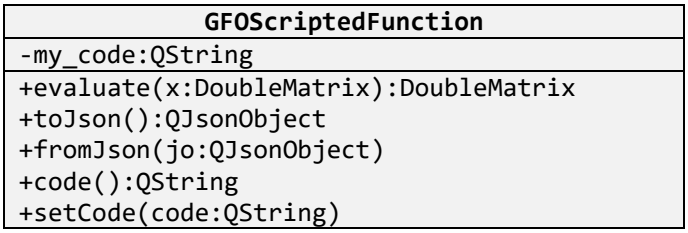

Figura 2.9 Esquema UML de la clase abstracta `GFOScriptedFunction`

2.2.10 Clase `GFOConstraint`

La clase `GFOConstraint` (Fig. 2.10) permite implementar objetos que van a contener todas las características y la funcionalidad de las restricciones asociadas a un problema de optimización, tanto mono-objetivo como multi-objetivo. En esta clase se define el nombre (variable `my_name` de tipo `QString`) y el tipo (variable `my_type` de tipo `GFOConstraintType`) para cada una de las restricciones a tener en cuenta. Como métodos que le otorgan la funcionalidad a la clase está el constructor de la misma, así como los métodos `setName` y `setType` con la misma funcionalidad explicada en casos anteriores que se usaron métodos del mismo nombre así mismo sucede con los métodos `name()` y `type()`.

GFOConstraint
-my_name:QString -my_type:GFOConstraintType
+GFOConstraint(name:QString, type:GFOConstraintType) +name():QString +type():GFOCOnstraint +setName(name:QString) +setType(type:GFOConstraintType)

Figura 2.10 Esquema UML de la clase GFOConstraint

2.2.11 Clase `GFOConstraints`

La clase `GFOConstraints` (Fig. 2.11) implementa objetos que como en el caso de `GFOVariables` y `GFOObjectives` van a contener el conjunto de todas las restricciones asociadas a un problema típico de optimización Su atributo es una lista de tipo `QList` de objetos de tipo `GFOConstraint`, como era de esperarse. Como en los casos de las clases antes mencionadas sus métodos van a permitir el acceso individual a cada restricción, adicionar una nueva o eliminar alguna otra, con los operadores antes mencionados como son: `append()`, `remove()`, `at()`, `clear()`, entre otros y enlazar las características que conforman las restricciones a través de los objetos a través de `toJson()` y `fromJson()`.

GFOConstraints
-my_constraints:QList<GFOConstriant>
+count():int +clear() +at(index:int):GFOConstraint +append(var:GFOConstraint) +remove(index:int) +operator[](index:int):GFOConstraint const +operator[](index:int):GFOConstraint & +toJson():QJsonObject +fromJson(&jo:QJsonObject)

Figura 2.11 Esquema UML de la clase GFOConstraints

2.3 Clases bases para optimización mono-objetivo

2.3.1 Clase `GFOHeuristic`

La clase `GFOHeuristic`, (Fig. 2.12) es la clase abstracta sobre la cual se van a desarrollar los diferentes tipos de heurísticas que van a ser posteriormente implementadas de manera individual tanto las mono-objetivo como las multi-objetivo. Esta clase contiene como sus atributos las variables definidas desde sus respectivas clases que conforman el conjunto de valores y parámetros utilizados dentro de un problema de optimización como son variables de la clase `GFOVariables`, definidos anteriormente para las variables de decisión; también `objectives` para definir los objetivos, `constraints` para las restricciones y `paramenters` para definir otros parámetros dentro del problema, cada uno objeto de sus respectivas clases ya antes mencionadas. Posee otros atributos como es el puntero *my*_`function`, relacionado con la forma con la que se va a evaluar la heurística y otro parámetro como `my_exec_time` de tipo `double` que define el tiempo de ejecución de la heurística. Los metidos consisten en una gama de funciones que dotan de versatilidad y funcionalidad a la heurística que se va a definir. Métodos como `readFromFile()` y `writeFromFile()` permiten cargar datos desde un fichero externo o guardar los mismos, también saveToFile() y `loadFromFile()` permite cargar el archivo pasándole como argumento el nombre del mismo. También se definen métodos virtuales para que sean implementados con posterioridad en cada heurística de manera individual, son los métodos `execute()` y `readParameters()`, relacionado con la forma en que será ejecutada la heurística así como la manera en que serán introducidos los parámetros. Otros métodos que ya han sido explicados con anterioridad como son los métodos `toJson()` y `fromJson()`.

En esta clase aparecen implementadas dos *signals*, `endEpoch()` y `endOptimization()`.

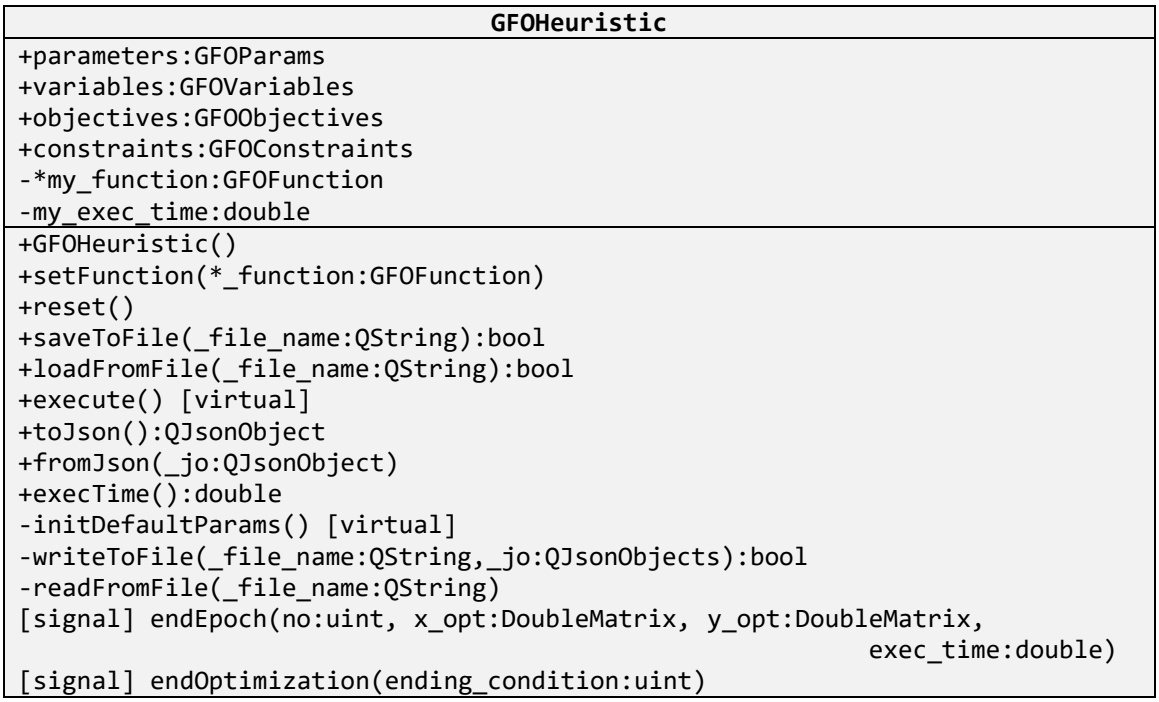

Figura 2.12 Esquema UML de la clase `GFOHeuristic`

2.3.2 Clase `GFOHeuristicSOO`

Para la optimización mono-objetivo, se implementó una clase base común, `GFOHeuristicSOO` (Fig. 2.13). La misma define como atributos tres variables de tipo `DoubleMatrix`, `X`, `Y` y `G` correspondiente a las matrices de variables de decisión, los valores de la función objetivo y las restricciones. Los métodos `optimalX()` y `optimalXStdDev()` permiten determinar, primeramente, el valor optimo dentro de las variables de decisión así como conocer la desviación estándar en el proceso de convergencia de la variable de decisión, lo mismo ocurre en los casos correspondientes de las funciones objetivo con `optimalY()` y `optimalXStdDev()`.

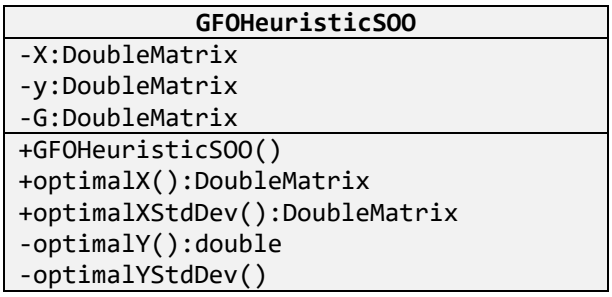

Figura 2.13Esquema UML de la clase GFOHeuristicBaseSOO

Debe hacerse notar que, en varias de las clases antes descritas, se utiliza la clase `DoubleMatrix`, implementada en la biblioteca `EcMatrix`, que constituye una interfaz para el uso de las funcionalidades de operaciones con matrices proporcionadas por la biblioteca `Eigen`.

2.4 Clases para heurísticas mono-objetivo

2.4.1 Clase `GFOHeuristicGA`

La clase `GFOHeuristicGA` (Fig. 2.14) implementa un algoritmo genético general. En el mismo, se definen los atributos que van a permitir almacenar la cadena de cromosomas en la cual se codifican las soluciones individuales en la variable `chromosome` de tipo `QString`, así como los límites de las variables de decisión a través de las variables *l* y *u* de tipo `DoubleMatrix`. Los métodos privados `initPopulation()`, permite generar la población inicial de variables de decisión de manera aleatoria para la inicialización, dos métodos más se encargan de codificar las soluciones ya evaluadas de manera individual y decodificar posteriormente las mismas, estos son `encodePopulation()` y `decodePopulation()` respectivamente. También se implementan los operadores genéticos para la selección, el cruzamiento y la mutación dentro del método

createNewPopulation(), el método initDefaultParameters(), permite inicializar valores y parámetros propios del problema .

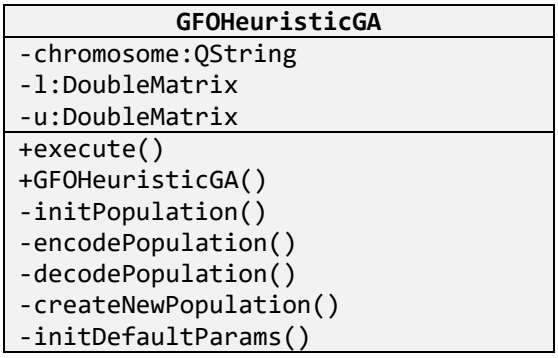

Figura 2.14 Esquema UML de la clase GFOHeuristicGA

2.4.2 Clase GFOHeuristicCE

La clase GFOHEuristicCE, que implementa la heurística CE (Fig. 2.15), cuenta con atributos que le van a permitir a la clase primeramente almacenar las soluciones aleatorias, E variable de tipo DoubleMatrix generadas con su correspondiente media y desviación estándar (variables mu y sigma, ambas de tipo DoubleMatrix). Los métodos que van a llevar a cabo el proceso de convergencia hacia la solución óptima, initMuAndSigma*()* va a permitir inicializar los valores de media y varianza, luego de ser seleccionada la nueva población a partir de la población *elite* anterior, los nuevos valores de mu y sigma son recalculados y actualizados con los métodos computeNewMuAndSigma() y updateMuAndSigma(), el método initDefaultParams(), se encarga de la puesta a punto de otros parámetros relacionados con el problema de optimización.

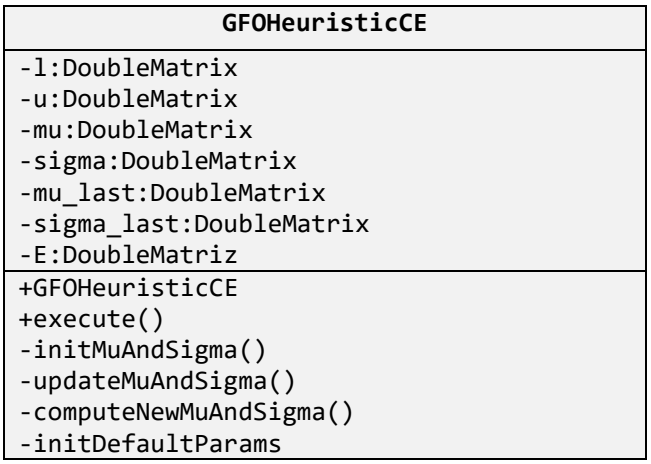

Figura 2.15. Esquema UML de la clase `GFOHeuristicCE`

2.4.3 Clase `GFOHeuristicPSO`

La clase `GFOHEuristicPSO` (Fig. 2.15) implementa la heurística PSO. Cuenta con atributos que le van a permitir a la clase realizar el procesamiento correspondiente. En primer lugar, las variables de tipo `DoubleMatrix`, `u` y `l`, permiten almacenar los valores mínimo y máximo de las variables de decisión. La variable `V` almacena la velocidad de cada solución, mientras que `X_p`, `Y_p`, `X_g`, `Y_g`, todas de tipo `DoubleMatrix`, almacenan la mejor solución, y su correspondiente imagen, para cada individuo y para la población en general, respectivamente.

Dentro de los métodos que proporciona la clase, además del constructor `GFOHeuristicPSO()`, está el método público `execute()`, que implementa la ejecución el proceso de optimización. También proporciona los métodos privados `initPopulation()`, que inicializa la población, aleatoriamente, `updateBestValues()`, que actualiza los mejores valores tanto para cada individuo como para la población, y `updatePopulation()`, que actualiza las posiciones de todas las

soluciones la población. Finalmente, está la implementación del método protegido `initDefaultParams()`, que inicializa los valores predeterminados de los parámetros del algoritmo.

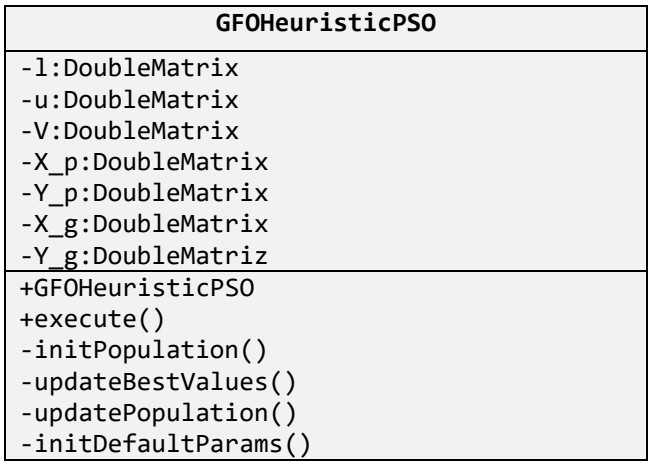

Figura 2.16. Esquema UML de la clase `GFOHeuristicPSO`

2.4 Conclusiones parciales del capítulo

Una vez finalizado el presente capítulo, se ha podido arribar a las siguientes conclusiones parciales:

1. Se ha diseñado e implementado una biblioteca de clases en C++, basada en Qt, y denominada *GFOLib*, que permite llevar a cabo la optimización mono-objetivo mediante heurísticas sin gradiente.

2. La biblioteca *GFOLib* implementa las clases para manipular los principales componentes de un problema de optimización. Implementa, también los métodos de algoritmos genéticos, entropía cruzada y enjambre de partículas.

3. La biblioteca *GFOLib* permite su extensión futura para incorporar tanto otras heurísticas mono-objetivos como heurísticas multi-objetivos.

CAPÍTULO 3 IMPLEMENTACIÓN DE LA APLICACIÓN BASADA EN TECNOLOGÍA DE NUBE

En el capítulo se describe el diseño e implementación de la aplicación cliente/servidor para el proceso de optimización, así como su validación a partir de casos de estudio tomados de la literatura y de problemas tecnológicos reales.

3.1 Aplicación servidor para optimización

3.1.1 Características generales

Para realizar las funciones de servidor, se desarrolló una aplicación (denominada COS Server). La misma está basada en una arquitectura orientada a servicios, extensible a computación en la nube.

COS Server es una aplicación para ejecutarse en sistemas operativos MS Windows o Linux, desarrollada en C++ utilizando las librerías de Qt.

3.1.2 Diseño de la aplicación

COS Server es una aplicación de consola. La misma implementa un servidor de TCP/IP que se mantiene a la escucha, por un puerto determinado, de las peticiones de los clientes. Para la misma se diseñaron las clases *TCPServer* y *ClientThread*.

La clase *TCPServer* recibe las peticiones del cliente y levanta un hilo, con una instancia de la clase *ClientThread*, para cada una de ellas. La comunicación entre dichas clases, se realiza a través del par de *signals/slots writeLog/onWriteLog* (Fig. 3.1), con el cual la instancia de *ClientThread* le indica a la de *TCPServer*, que acciones va realizando para que

ésta las asiente en el archivo de registro. También, en dependencia de la configuración, el programa muestras dichas acciones, en la pantalla de la consola.

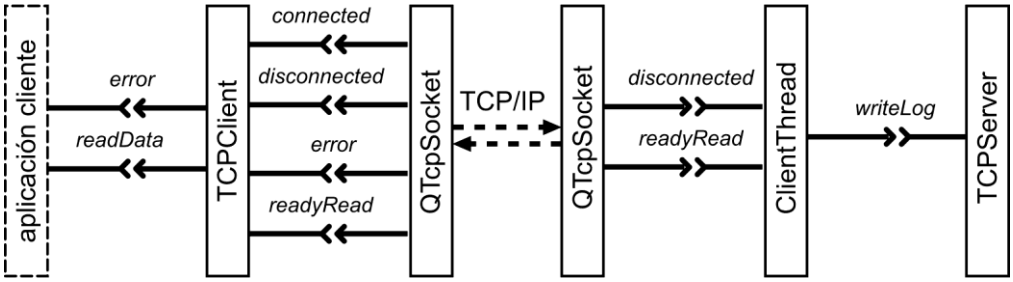

Figura 3.1. Arquitectura cliente/servidor de COS

A su vez, cada instancia de *ClientThread*, crea una de *QTcpSocket*, con la que se comunica mediante los pares de *signals/slots readyRead/onReadyRead*, para la lectura de envíos de datos desde el cliente, y *disconnected*, para la notificación de la desconexión. La instancia de *QTcpSocket* es la encargada de comunicarse con la instancia equivalente, en el cliente, a través de los protocolos TCP/IP.

3.1.3 Descripción de las clases

3.1.3.1 Clase TCPServer

La clase *TCPServer*, es una clase derivada de *QTcpServer*, proporcionada por la librería de Qt (Fig. 3.2). En la misma se implementan las variables privadas *port* [entera] (puerto por el que se comunicará la aplicación), *echo* [booleana] (indica si se escribe o no, en la consola, las acciones que el servidor va realizando) y *log_file_name* [cadena de caracteres] (nombre del archivo de registro de sucesos). También implementa, de forma protegida, el método *incomingConnection*, el cual, ante una petición del cliente, crea y lanza la instancia de *ClientThread* que procesará dicha petición.

Como métodos privados, además del creador, se implementa el método *startServer*, con el cual abre y lee el archivo de configuración e inicia el proceso del servidor. Finalmente, se implementa el slot público *onWriteLog*, el cual escribe en el archivo de registro la información de cada acción realizada por el servidor. Si la variable echo tiene valor verdadero, este *slot*, además, muestra dicha información en la consola de la aplicación.

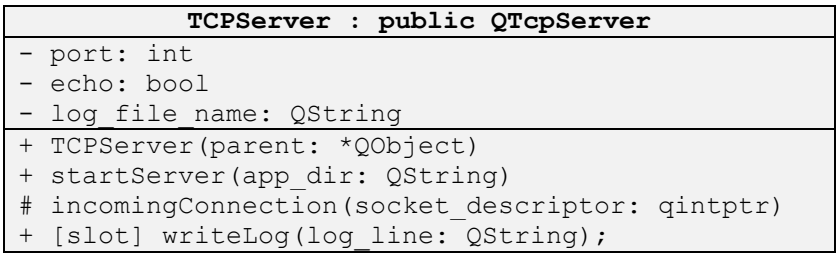

Figura 3.2. UML de la clase TCPServer

3.1.3.2 Clase ClientThread

La clase *ClientThread* se deriva de la clase *QThread* e implementa un hilo donde se procesa y responde la petición del cliente (Fig. 3.3). La misma contiene como variables privadas, un puntero a una instancia se *QTcpSocket* y un puntero a un entero que contiene el descriptor de la conexión. Contiene, además, como públicos, el creador *ClientThread*, donde se asigna el valor del descriptor de la conexión y donde se conecta el par *signal*/*slot*, que la comunica con la instancia *QTcpSocket* que la contiene; el método *run*, donde se crea y lanza la instancia de *QTcpSocket* que se comunicará con el cliente y se conectan los pares *signals/slots* correspondientes; el método *write*, donde se escriben los datos al socket y se emite la *signal writeLog* y el método *processData*, donde se lee y procesan los datos recibidos del cliente, lo cual se hace creando una instancia de clase derivada de *GFOHeuristic*, de la librería GFO, cargando los datos y ejecutando el proceso de optimización.

La clase *ClientThread* implementa, además, los slots *readyRead*, donde se procesa la lectura del cliente y *disconnected*, donde se notifica la desconexión del cliente. También contiene las señales, *error* y *writeLog*, las que se emiten ante un error de la conexión y para pedir una escritura en el archivo de registros.

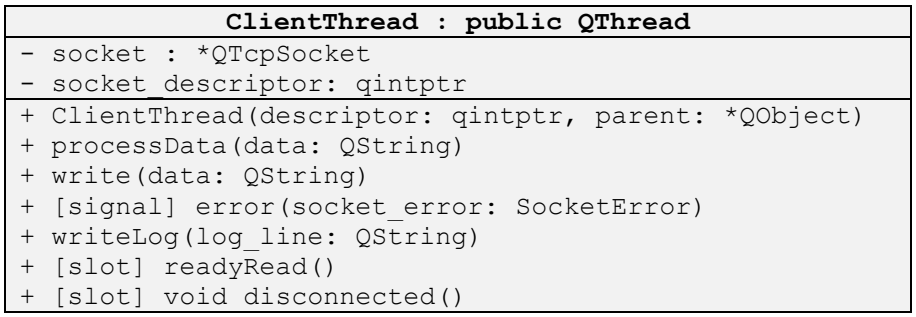

Figura 3.3. UML de la clase ClientThread

3.1.4 Comunicación cliente/servidor

La comunicación cliente/servidor se realiza codificando la información, tanto de la petición como de la respuesta, mediante el protocolo JSON, con estructuras específicas para la petición y la respuesta.

La petición (Fig. 3.4), incorpora la heurística a utilizar para la solución (de las tres implementadas), y sus parámetros, así como la definición del problema de optimización (variables de decisión, objetivos y restricciones) y el código con los modelos correspondientes.

```
object
    version: string
    heuristic: string
    variables: object
        items: array of object
            name: string
            min: double
            max: double
    objectives: object
        items: array of object
            name: double
            type: int
    constraints: object
        items: array of object
            name: double
            type: int
    parameters: object
        items: array of object
            key: string
            value: double
    function: object
        type: string
        code: array of object
            line: string
```

Figura 3.4. Estructura JSON de la petición

La respuesta (Fig. 3.5), por su parte, incluye la solución con sus valores de variables y objetivos, y el tiempo que tardó en ejecutarse el problema.

```
object
    points: object
        point: array of object
            x: array of double
            y: array of double
    exec_time: double
```

Figura 3.5. Estructura JSON de la respuesta

3.2 Aplicación cliente

3.2.1 Características generales

Como cliente del servidor descrito en la sección anterior, se puede desarrollar cualquier aplicación que cumpla con las condiciones de comunicación establecidas. Como ejemplo,

se ha desarrollado un programa, denominado COS WinGUI, para servir de cliente, con interfaz gráfica de usuario, sobre MS Windows.

3.2.2 Interfaz gráfica de usuario

La aplicación COS WinGUI cuenta con una ventana principal donde hay un panel con tabulaciones y dos cuadros de texto (Fig. 3.6). El primero, denominado *Problema*, contiene la definición del problema de optimización, el cual se establece según el formato JSON descrito en la sección anterior. Dicho panel de texto permite editar el código a través de escritura o cargarlo desde un archivo prestablecido.

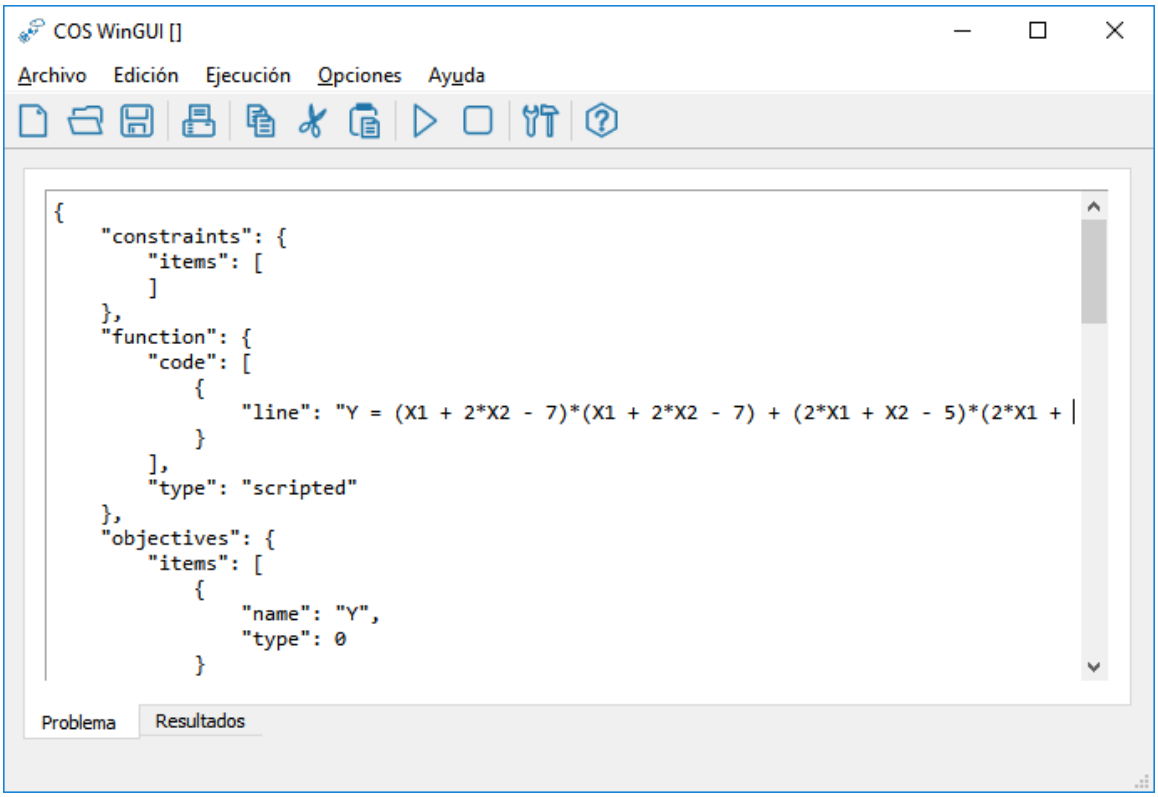

Figura 3.6. Interfaz gráfica de la aplicación COS WinGUI

Por su parte, el panel de texto *Resultados* es de sólo lectura y permite visualizar los resultados de la optimización.

La ventana principal del programa cuenta con un grupo de opciones que se resumen en la Tabla 3.1.

Tabla 3.1. Opciones del programa

Menú	*Opción*	*Descripción*
Archivo	Nuevo	Crea un nuevo problema de optimización
	Abrir…	Abre un archivo de texto en formato JSON que contiene el problema de optimización
	Guardar	Guarda el archivo de texto en formato JSON que contiene el problema de optimización
	Guardar como…	Guarda el archivo del problema de optimización con un nombre diferente
	Imprimir…	Imprime el contenido del problema de optimización
	Cerrar	Termina la sesión de trabajo con el programa
Edición	Copiar	Copia el contenido seleccionado en el texto del problema, al portapapeles
	Cortar	Corta el contenido seleccionado en el texto del problema, al portapapeles
	Pegar	Pega, en el texto del problema, el contenido del portapapeles
	Seleccionar todo	Selecciona todo el contenido del texto del problema
Ejecución	Ejecutar	Ejecuta el problema de optimización enviando la petición al servidor
	Abortar	Aborta la ejecución del problema de optimización enviando la petición al servidor

Tabla 3.1. Opciones del programa (continuación)

Menú	*Opción*	*Descripción*
Opciones	Configuración…	Muestra el cuadro de diálogo de configuración de las opciones del programa
Ayuda	Contenido…	Muestra el contenido de la ayuda del programa
	Acerca de…	Muestra el cuadro de diálogo de créditos del programa

3.3 Estudios de caso

3.3.1 Problemas tipo de la literatura

Con el objetivo de comprobar el funcionamiento de la aplicación, se ejecutaron un grupo de problemas tipos de prueba, reportados en la literatura especializada (Surjianovic y Bingham, 2017). Se trataron cinco tipos de problemas, entre los que se encuentran problemas de muchos mínimos locales, problemas de forma de tasa, de forma de plato, problemas de forma de valles, problemas de crestas escarpadas y otros tipos de problemas.

3.3.2 Problemas con muchos mínimos locales.

3.3.2.1 Función de Bukin No. 6.

La función de Bukin No. 6 es un problema de tipo bidimensional el cual cuenta con una gran cantidad de mínimos locales. Cada uno de estos mínimos locales están dispuestos en forma de una cresta. La función objetivo de este problema es:

$$f(\mathbf{x}) = 100\sqrt{\left|x_2 - 0.01x_1^2\right|} + 0.01\left|x_1 + 10\right| \tag{3.1}$$

El dominio de las variables de decisión para la función es $x_1 \in [-15,-5], x_2 \in [-3,3]$ y el mínimo global se encuentra en $f(\mathbf{x}^*) = 0,$ para $\mathbf{x}^* = (-10,1)$.

A continuación, se presentan los resultados arrojados por la simulación de las heurísticas de optimización CE, GA y PSO para el caso de la función Bukin N.6. Teniendo en cuenta que los valores óptimos para la función son conocidos, $f(\mathbf{x}^*) = 0,$ para $\mathbf{x}^* = (-10,1)$, para cada una de las heurísticas se presenta una tabla en la cual se muestran los resultados de cada una de las corridas. La tabla cuenta con los parámetros independientes correspondiente a cada una de las heurísticas. Para CE (Tabla 3.2), Tamaño de población $Z = 250$, cantidad de generaciones $N = 10^5$, límite de convergencia $\varepsilon = 10^{-6}$, factor de suavizado $\alpha = 0.8$, factor de elitismo $\eta = 0.1,$ para GA (Tabla 3.3) los valores de tamaño de la población $Z = 2500$, cantidad de generaciones $N = 10^5$ límite de convergencia $\varepsilon = 10^{-6}$, probabilidad de mutación $p_m = 10^{-5}$, longitud de codificación $L_c = 16$ y para PSO (Tabla 3.4) son población $Z = 300$, cantidad de generaciones $N = 10^3$,aceleración cognitiva $x_1 = 0.5$ y aceleración social $x_2 = 0.5$ y la inercialidad $w = 0.1$. El algoritmo devuelve el valor óptimo para la variable de decisión $\mathbf{x}^*$ y de la función objetivo y^* además del tiempo *t* de convergencia para cada corrida medido en segundos. Para cada una de las heurísticas viene asociada en la tabla correspondiente los errores de aproximación asociados a cada una de las variables de decisión $\mathbf{x}_1^*$, $\mathbf{x}_2^*$ y y^* para la función objetivo, en el tiempo t. Los valores medios en cada uno de los casos son tenidos en cuenta para las variables de decisión (Tabla 3.5) $vma(\mathbf{x}^*)$, la función objetivo $vma(y^*)$ y el tiempo

de las corridas $vm(t)$ y también son graficados (Figura 3.7) para una mejor observación. Se arribó a la conclusión que el mejor resultado se alcanzaba a través de la CE.

Tabla 3.2 Resultados de la optimización de la función Bukin No. 6 con CE

No.	x_1^*	x_2^*	y^*	$t(s)$
1	-1,03E+01	1,07E+00	6,93E-02	102,12
2	-9,47E+00	8,97E-01	6,66E-02	99,14
3	-9,17E+00	8,40E-01	7,19E-02	98,64
4	-1,02E+01	1,05E+00	6,78E-02	100,04
5	-9,47E+00	8,97E-01	6,28E-02	101,00
---	$e(x_1^*)$	$e(x_2^*)$	$e(y^*)$	$t(s)$
1	3,47E-01	7,07E-02	6,93E-02	102,12
2	5,28E-01	1,03E-01	6,66E-02	99,14
3	8,35E-01	1,60E-01	7,19E-02	98,64
4	2,23E-01	4,52E-02	6,78E-02	100,04
5	5,30E-01	1,03E-01	6,28E-02	101,00
MAE	2,95E-01	---	6,77E-02	100,19
MAX	8,35E-01	---	7,19E-02	102,12

Tabla 3.3 Resultados de la optimización de la función Bukin No. 6 con GA

No.	x_1^*	x_2^*	y^*	$t(s)$
1	-9,28421	0,861969	0,200212	185,15
2	-7,71896	0,595825	2,17E-01	136,58
3	-8,37158	0,700836	0,176153	229,99
4	-7,99057	0,638489	0,276055	169,34
5	-7,66541	0,587585	0,132865	175,66
---	$e(x_1^*)$	$e(x_2^*)$	$e(y^*)$	$t(s)$
1	7,16E-01	1,38E-01	2,00E-01	185,15
2	2,28E+00	4,04E-01	2,17E-01	136,58
3	1,63E+00	2,99E-01	1,76E-01	229,99
4	2,01E+00	3,62E-01	2,76E-01	169,34
5	2,33E+00	4,12E-01	1,33E-01	175,66
MAE	1,06E+00	---	2,00E-01	179,34
MAX	2,33E+00	---	2,76E-01	229,99

Tabla 3.4 Resultados de la optimización de la función Bukin No. 6 con PSO

No.	x_1^*	x_2^*	y^*	$t(s)$
1	-11,5584	1,33631	0,560586	132,217
2	-9,55E+00	9,12E-01	6,88E-01	129,017
3	-9,53E+00	9,33E-01	1,55E+01	127,92
4	-1,32E+01	1,75E+00	6,32E-01	128,796
5	-1,24E+01	1,54E+00	1,59E-01	127,889
---	$e(x_1^*)$	$e(x_2^*)$	$e(y^*)$	$t(s)$
1	1,56E+00	3,36E-01	5,61E-01	132,22
2	4,52E-01	8,79E-02	6,88E-01	129,02
3	4,71E-01	6,67E-02	1,55E+01	127,92
4	3,23E+00	7,53E-01	6,32E-01	128,80
5	2,41E+00	5,40E-01	1,59E-01	127,89
MAE	9,90E-01	---	3,51E+00	129,17
MAX	3,23E+00	---	1,55E+01	132,22

Tabla 3.5 Valores medios obtenidos para la función de Bakin No. 6

	$vma(\mathbf{x}^*)$	$vma(y^*)$	$vm(t)$
CE	2,95E-01	6,77E-02	1,00E+02
GA	1,06E+00	2,00E-01	1,79E+02
PSO	9,90E-01	3,51E+00	1,29E+02

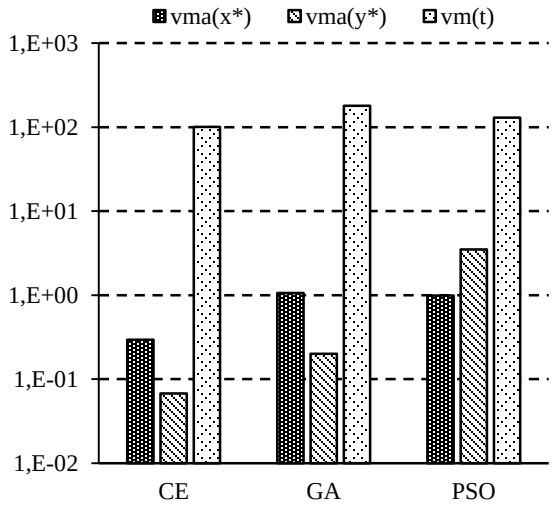

Figura 3.7. Valores medios obtenidos durante la simulación para las heurísticas CE, GA y PSO para la función Bukin N.6

3.3.2.2 Función de Schaffer No. 2

La función Schaffer No. 2 es una función bidimensional y su función objetivo la define de la siguiente manera:

$$f(\mathbf{x}) = 0.5 + \frac{\sin^2\left(x_1^2 - x_2^2\right) - 0.5}{\left[1 + 0.001\left(x_1^2 + x_2^2\right)\right]^2} \tag{3.2}$$

El dominio de la variable de decisión queda definido en el cuadrante $x_i \in [-100,100]$ para todos los valores de $i = 1,2$. Su mínimo global en $f(\mathbf{x}^*) = 0$ para $\mathbf{x}^* = (0,0)$.

A continuación, se presentan los resultados arrojados por la simulación de las heurísticas de optimización CE, GA y PSO para el caso de la función Schaffer No. 2. Teniendo en cuenta que los valores óptimos para la función son conocidos, $f(\mathbf{x}^*) = 0,$ para $\mathbf{x}^* = (0,0)$ .Como en el caso analizado anteriormente los parámetros independientes correspondiente a cada una de las heurísticas son mostrados. Para CE (Tabla 3.6), tamaño de población $Z = 100$, cantidad de generaciones $N = 10^5$, límite de convergencia $\varepsilon = 10^{-6}$, factor de suavizado $\alpha = 0.8$, factor de elitismo $\eta = 0.1,$ para GA (Tabla 3.7) los valores de tamaño de la población $Z = 1000,$ cantidad de generaciones $N = 10^5$,límite de convergencia $\varepsilon = 10^{-6}$, probabilidad de mutación $p_m = 10^{-5}$, longitud de codificación $L_c = 16$ y para PSO (Tabla 3.8) son, población $Z = 500$, cantidad de generaciones $N = 10^5$,aceleración cognitiva $x_1 = 0.5$ y aceleración social $x_2 = 0.5$ y la inercialidad $w = 0.1$. Como fue analizado el algoritmo devuelve el valor óptimo para la variable de decisión $\mathbf{x}^*$ y de la función objetivo y^* además del tiempo *t* en segundos de convergencia para cada corrida.

Tabla 3.6 Resultados de la función de Schaffer No. 2 con CE

No.	x_1^*	x_2^*	y^*	$t(s)$
1	1,88E-07	-1,37E-06	1,48E-14	11,315
2	4,81E-07	-3,50E-07	1,03E-14	11,298
3	9,30E-08	-7,28E-07	2,60E-14	11,617
4	-4,29E-07	4,92E-07	1,32E-14	9,829
5	1,22E-06	4,75E-07	1,65E-14	12,002
---	$e(x_1^*)$	$e(x_2^*)$	$e(y^*)$	$t(s)$
1	1,88E-07	1,37E-06	1,48E-14	11,32
2	4,81E-07	3,50E-07	1,03E-14	11,30
3	9,30E-08	7,28E-07	2,60E-14	11,62
4	4,29E-07	4,92E-07	1,32E-14	9,83
5	1,22E-06	4,75E-07	1,65E-14	12,00
MAE	5,83E-07	---	1,62E-14	11,21
MAX	1,37E-06	---	2,60E-14	12,00

Tabla 3.7 Resultados de la función de Schaffer No. 2 con GA

No.	x_1^*	x_2^*	y^*	$t(s)$
1	0	0	5,64E-11	126,123
2	-0,0030518	0	9,58E-09	66,648
3	-0,0030518	0	0,0001192	85,51
4	0	-0,0030518	1,95E-08	84,041
5	-0,0030518	-0,0030518	0,0004717	99,261
---	$e(x_1^*)$	$e(x_2^*)$	$e(y^*)$	$t(s)$
1	0,00E+00	0,00E+00	5,64E-11	126,12
2	3,05E-03	0,00E+00	9,58E-09	66,65
3	3,05E-03	0,00E+00	1,19E-04	85,51
4	0,00E+00	3,05E-03	1,95E-08	84,04
5	3,05E-03	3,05E-03	4,72E-04	99,26
MAE	1,53E-03	---	1,18E-04	92,32
MAX	3,05E-03	---	4,72E-04	126,12

Cada una de las heurísticas tiene asociada a su tabla correspondiente los errores de aproximación de cada una de las variables de decisión u optimas $\mathbf{x}_1^*$, $\mathbf{x}_2^*$ y y^* para la función objetivo, en el tiempo ***t*.** Los valores medios en cada uno de los casos son tenidos en cuenta para las variables de decisión $vma(\mathbf{x}^*)$, la función objetivo $vma(y^*)$ y el tiempo de las corridas $vm(t)$ (Tabla 3.9) y también son graficados (Figura 3.8) para una mejor

observación como se realizó anteriormente. Se arribó a la conclusión que el mejor resultado se alcanzaba a través de la CE.

Tabla 3.8 Resultado de la función de Schaffer No. 2 con PSO

No.	x_1^*	x_2^*	y^*	$t(s)$
1	-2,78E-06	-4,07E-05	1,66E-12	241,87
2	2,48E-07	-1,96E-07	1,66E-15	412,939
3	-3,86E-08	-3,86E-07	9,68E-16	959,372
4	-1,06E-07	3,99E-07	1,43E-15	157,376
5	-1,48E-08	-2,77E-07	1,74E-15	530,463
---	$e(x_1^*)$	$e(x_2^*)$	$e(y^*)$	$t(s)$
1	2,78E-06	4,07E-05	1,66E-12	241,87
2	2,48E-07	1,96E-07	1,66E-15	412,94
3	3,86E-08	3,86E-07	9,68E-16	959,37
4	1,06E-07	3,99E-07	1,43E-15	157,38
5	1,48E-08	2,77E-07	1,74E-15	530,46
MAE	4,51E-06	---	3,34E-13	460,40
MAX	4,07E-05	---	1,66E-12	959,37

Tabla 3.9 Valores medios obtenidos para la función de Schaffer No. 2

	$vma(\mathbf{x}^*)$	$vma(y^*)$	$vm(t)$
CE	5,83E-07	1,62E-14	1,12E+01
GA	1,53E-03	1,18E-04	9,23E+01
PSO	4,51E-06	3,34E-13	4,60E+02

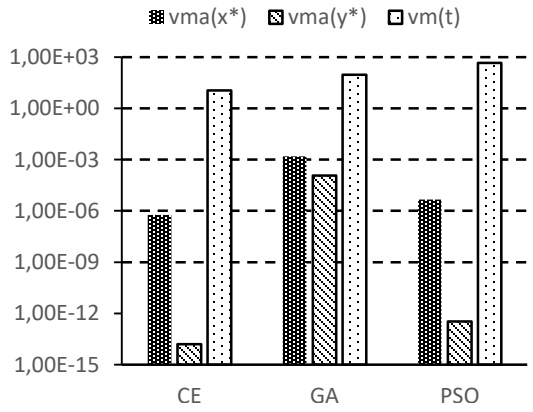

Figura 3.8 Valores medios obtenidos durante la simulación para las heurísticas CE, GA y PSO para la función Schaffer N.2

3.3.3Problemas con forma de taza.

3.3.3.1 Función de Bohachevsky

Las funciones bidimensionales Bohachevsky tienen todas las mismas formas de taza. Su función objetivo es:

$$f(x_1, x_2) = x_1^2 + 2x_2^2 - 0.3\cos(3\pi x_1) - 0.4\cos(4\pi x_2) + 0.7 \qquad (3.3)$$

El dominio de la variable de decisión queda definido en el cuadrante $x_i \in [-100,100]$ para todos los valores de $i = 1,2$. Sus mínimos globales quedan definido de la forma $f_j(\mathbf{x}^*) = 0$ para $\mathbf{x}^* = (0,0)$ para todas las $j = 1,2,3$.

Para la función Bohachevsky también fue aplicada la misma simulación con los mismos tipos de heurísticas conociendo que sus mínimos globales quedan definido de la forma $f_j(\mathbf{x}^*) = 0$ para $\mathbf{x}^* = (0,0)$. Para CE (Tabla 3.10), tamaño de población $Z = 50$, cantidad de generaciones $N = 10^5$, límite de convergencia $\varepsilon = 10^{-6}$, factor de suavizado $\alpha = 0.8$, factor de elitismo $\eta = 0.1$, para GA (Tabla 3.11) los valores de tamaño de la población $Z = 250$, cantidad de generaciones $N = 10^5$,límite de convergencia $\varepsilon = 10^{-6}$, probabilidad de mutación $p_m = 10^{-5}$, longitud de codificación $L_c = 16$ y para PSO (Tabla 3.12) son, población $Z = 500$, cantidad de generaciones $N = 10^5$,aceleración cognitiva $x_1 = 0.5$ y aceleración social $x_2 = 0.5$ y la inercialidad $w = 0.1$. Cada una de las heurísticas tiene asociada a su tabla correspondiente los errores de aproximación de cada una de las variables de decisión u optimas $\mathbf{x}_1^*$, $\mathbf{x}_2^*$ y *y****** para la función objetivo, en el tiempo *t***.** Los valores

medios en cada uno de los casos son tenidos en cuenta para las variables de decisión $vma(\mathbf{x}^*)$, la función objetivo $vma(y^*)$ y el tiempo de las corridas $vm(t)$ (Tabla 3.13) y también son graficados (Figura 3.9) para una mejor observación como se realizó anteriormente. Se arribó a la conclusión que el mejor resultado se alcanzaba a través de la CE.

Tabla 3.10 Resultados de la función Bohachevsky con CE

No.	x_1^*	x_2^*	y^*	$t(s)$
1	7,19E-07	6,09E-07	1,46E-10	1,019
2	1,95E-07	1,12E-07	1,10E-10	0,969
3	-7,34E-07	6,20E-08	1,28E-10	1,016
4	-8,34E-07	1,10E-07	3,36E-10	1
5	-7,21E-09	1,84E-07	2,53E-10	0,984
---	$e(x_1^*)$	$e(x_1^*)$	$e(y^*)$	$t(s)$
1	7,19E-07	6,09E-07	1,46E-10	1,02
2	1,95E-07	1,12E-07	1,10E-10	0,97
3	7,34E-07	6,20E-08	1,28E-10	1,02
4	8,34E-07	1,10E-07	3,36E-10	1,00
5	7,21E-09	1,84E-07	2,53E-10	0,98
MAE	3,56E-07	---	1,95E-10	1,00
MAX	8,34E-07	---	3,36E-10	1,02

Tabla 3.11 Resultados de la función Bohachevsky con GA

No.	x_1^*	x_2^*	y^*	$t(s)$
1	0	-0,0518799	0,71362	5,297
2	0	0	2,14E-05	5,375
3	-0,0030518	-0,0030518	0,0121545	5,282
4	-0,0030518	-0,0030518	0,0004777	4,391
5	-0,0274658	0	0,0108571	4,875
---	$e(x_1^*)$	$e(x_1^*)$	$e(y^*)$	$t(s)$
1	0,00E+00	5,19E-02	7,14E-01	5,30
2	0,00E+00	0,00E+00	2,14E-05	5,38
3	3,05E-03	3,05E-03	1,22E-02	5,28
4	3,05E-03	3,05E-03	4,78E-04	4,39
5	2,75E-02	0,00E+00	1,09E-02	4,88
MAE	9,16E-03	---	1,47E-01	5,04
MAX	5,19E-02	---	7,14E-01	5,38

Tabla 3.12 Resultados de la función Bohachevsky con PSO

No.	x_1^*	x_2^*	y^*	$t(s)$
1	-0,0107716	0,0020939	0,0082347	31,629
2	-1,37E-05	2,53E-06	2,92E-09	23,362
3	5,24E-06	7,74E-07	4,47E-10	14,768
4	4,46E-08	-3,77E-08	1,65E-11	14,458
5	-1,11E-05	-9,52E-06	1,09E-06	28,581
---	$e(x_1^*)$	$e(x_2^*)$	$e(y^*)$	$t(s)$
1	1,08E-02	2,09E-03	8,23E-03	31,63
2	1,37E-05	2,53E-06	2,92E-09	23,36
3	5,24E-06	7,74E-07	4,47E-10	14,77
4	4,46E-08	3,77E-08	1,65E-11	14,46
5	1,11E-05	9,52E-06	1,09E-06	28,58
MAE	1,29E-03	---	1,65E-03	22,56
MAX	1,08E-02	---	8,23E-03	31,63

Tabla 3.13 Valores medios obtenidos a partir de cada heurística

	$vma(\mathbf{x}^*)$	$vma(y^*)$	$vm(t)$
CE	3,56E-07	1,95E-10	9,98E-01
GA	9,16E-03	1,47E-01	5,04E+00
PSO	1,29E-03	1,65E-03	2,26E+01

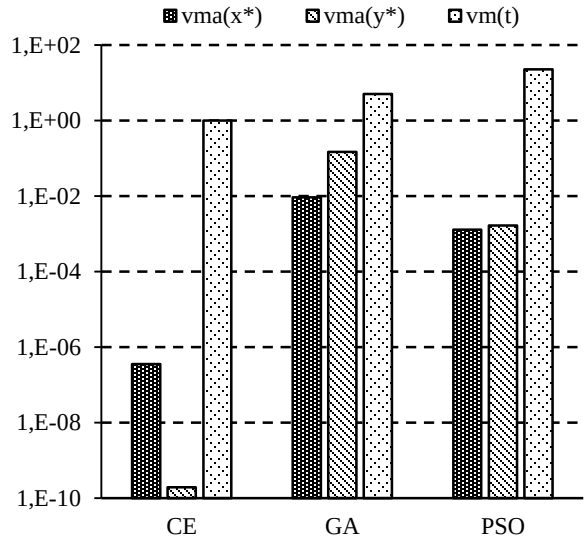

Figura 3.9. Valores medios obtenidos para la función Bohachevsky

3.3.4 Problemas con forma de plato

3.3.4.1 Función de Booth

La función bidimensional de Booth queda definida por su función objetivo de la forma:

$$f(x) = \left(x_1 + 2x_2 - 7\right)^2 + \left(2x_1 + x_2 - 5\right)^2 \tag{3.4}$$

La variable de decisión queda definida en el intervalo $x_i \in [-10,10]$ para todas las $i = 1,2$ .Su optimo global queda definido en $f\left(\mathbf{x}^*\right) = 0$ para $\mathbf{x}^* = (1,3)$.

Los resultados arrojados por la simulación de las heurísticas de optimización CE, GA y PSO son presentados para el caso de la función de Booth. Teniendo en cuenta que los valores óptimos para la función son conocidos, $f\left(\mathbf{x}^*\right) = 0$, para $\mathbf{x}^* = (1,3)$. Como en el caso analizado anteriormente los parámetros independientes correspondiente a cada una de las heurísticas son mostrados. Para CE (Tabla 3.14), tamaño de población $Z = 150$, cantidad de generaciones $N = 10^5$, límite de convergencia $\varepsilon = 10^{-6}$, factor de suavizado $\alpha = 0.8$, factor de elitismo $\eta = 0.1$, para GA (Tabla 3.15) los valores de tamaño de la población $Z = 1000$, cantidad de generaciones $N = 10^5$,límite de convergencia $\varepsilon = 10^{-6}$, probabilidad de mutación $p_m = 10^{-5}$, longitud de codificación $L_c = 16$ y para PSO (Tabla 3.16) son, población $Z = 500$, cantidad de generaciones $N = 10^5$,aceleración cognitiva $x_1 = 0.5$ y aceleración social $x_2 = 0.5$ y la inercialidad $w = 0.1$. El algoritmo devuelve el valor óptimo para la variable de decisión $\mathbf{x}^*$ y de la función objetivo y^* además del tiempo t , en segundos, de convergencia para cada corrida. Para cada una de las heurísticas viene

asociada a su tabla los errores de aproximación de cada una de las variables optimas $\mathbf{x}_1^*$, $\mathbf{x}_2^*$ y y^* para la función objetivo, en el tiempo t. Los valores medios en cada uno de los casos son tenidos en cuenta, como en los casos anteriores para las variables de decisión $vma(\mathbf{x}^*)$, la función objetivo $vma(y^*)$ y el tiempo de las corridas $vm(t)$ (Tabla 3.17) y también son graficados (Figura 3.10) para una mejor observación como se realizó anteriormente. Se arribó a la conclusión que el mejor resultado se alcanzaba a través de la CE.

Tabla 3.14 Resultados de la función de Booth con CE

No.	x_1^*	x_2^*	y^*	$t(s)$
1	1,00E+00	3,00E+00	4,18E-11	1,73
2	1,00E+00	3,00E+00	2,01E-09	2,05
3	1,00E+00	3,00E+00	5,80E-11	1,86
4	1,00E+00	3,00E+00	4,70E-11	2,03
5	1,00E+00	3,00E+00	8,79E-10	1,97
---	$e(x_1^*)$	$e(x_2^*)$	$e(y^*)$	$t(s)$
1	1,00E-06	0,00E+00	4,18E-11	1,73
2	3,00E-05	3,00E-05	2,01E-09	2,05
3	0,00E+00	0,00E+00	5,80E-11	1,86
4	0,00E+00	0,00E+00	4,70E-11	2,03
5	2,00E-05	2,00E-05	8,79E-10	1,97
MAE	1,01E-05	---	6,08E-10	1,93E+00
MAX	3,00E-05	---	2,01E-09	2,05E+00

Tabla 3.15 Resultados de la función de Booth con GA

No.	x_1^*	x_2^*	y^*	$t(s)$
1	9,95E-01	3,00E+00	2,00E-03	8,14
2	9,76E-01	3,02E+00	1,02E-03	10,47
3	9,37E-01	3,05E+00	7,13E-03	12,00
4	1,25E+00	2,81E+00	1,14E-01	8,24
5	9,75E-01	3,02E+00	1,25E-03	13,47
---	$e(x_1^*)$	$e(x_2^*)$	$e(y^*)$	$t(s)$
1	5,13E-03	4,15E-03	2,00E-03	8,14
2	2,37E-02	1,85E-02	1,02E-03	10,47
3	6,28E-02	4,90E-02	7,13E-03	12,00
4	2,51E-01	1,88E-01	1,14E-01	8,24
5	2,53E-02	2,46E-02	1,25E-03	13,47
MAE	6,51E-02	---	2,51E-02	1,05E+01
MAX	2,51E-01	---	1,14E-01	1,35E+01

Tabla 3.16 Resultados de la función de Booth con GA

No.	x_1^*	x_2^*	y^*	$t(s)$
1	1,00E+00	3,00E+00	3,72E-02	3,36
2	1,00E+00	3,00E+00	9,26E-03	8,06
3	1,00E+00	3,00E+00	7,32E-04	3,00
4	1,00E+00	3,00E+00	3,44E-05	14,03
5	1,00E+00	3,00E+00	1,39E-06	3,38
---	$e(x_1^*)$	$e(x_2^*)$	$e(y^*)$	$t(s)$
1	3,70E-04	4,50E-04	3,72E-02	3,36
2	1,88E-04	2,30E-04	9,26E-03	8,06
3	5,00E-05	6,00E-05	7,32E-04	3,00
4	1,40E-05	1,00E-05	3,44E-05	14,03
5	2,00E-06	0,00E+00	1,39E-06	3,38
MAE	1,37E-04	---	9,45E-03	6,37E+00
MAX	4,50E-04	---	3,72E-02	1,40E+01

Tabla 3.1 7Valores medios obtenidos a partir de cada heurística

	$vma(\mathbf{x}^*)$	$vma(y^*)$	$vm(t)$
CE	1,01E-05	6,08E-10	1,93E+00
GA	6,51E-02	2,51E-02	1,05E+01
PSO	1,37E-04	9,45E-03	6,37E+00

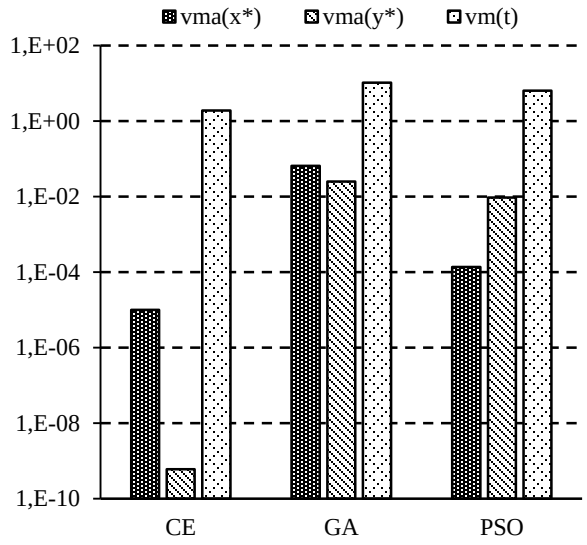

Figura 3.10. Valores medios obtenidos para la función de Booth

3.3.4.2 Función de Matyas

La función de Matyas tiene dos dimensiones y además no tiene mínimos locales, sólo un único mínimo global, su función objetivo es:

$$f(x) = 0.26\left(x_1^2 + x_2^2\right) - 0.48 x_1 x_2 \tag{3.5}$$

Las variables de decisión son evaluadas en el espacio $x_i \in [-10,10]$ para toda $i = 1,2$. Su optimo global se encuentra en $f(\mathbf{x}^*) = 0$ para $\mathbf{x}^* = (0,0)$.

A continuación, son presentados los resultados arrojados por la simulación de las heurísticas de optimización CE, GA y PSO para el caso de la función Matyas. Teniendo en cuenta que los valores óptimos para la función son conocidos, $f(\mathbf{x}^*) = 0,$ para $\mathbf{x}^* = (0,0)$. Una vez más los parámetros independientes correspondiente a cada una de las heurísticas son mostrados. Para CE (Tabla 3.18), tamaño de población $Z = 150$, cantidad de

generaciones $N=10^5$, límite de convergencia $\varepsilon=10^{-6}$, factor de suavizado $\alpha=0.8$, factor de elitismo $\eta=0.1$, para GA (Tabla 3.19) los valores de tamaño de la población $Z=1000$, cantidad de generaciones $N=10^5$,límite de convergencia $\varepsilon=10^{-6}$, probabilidad de mutación $p_m=10^{-5}$, longitud de codificación $L_c=16$ y para PSO (Tabla 3.20) son, población $Z=500$, cantidad de generaciones $N=10^5$,aceleración cognitiva $x_1=0.5$ y aceleración social $x_2=0.5$ y la inercialidad $w=0.1$.

Tabla 3.18 Resultados de la función de Matyas con CE

No.	$\mathbf{x}_1^*$	$\mathbf{x}_2^*$	y^*	$t(s)$
1	-4,16E-07	-5,85E-07	1,23E-12	11,439
2	-8,55E-07	-3,61E-07	4,47E-12	11,61
3	4,16E-07	7,01E-07	1,21E-12	11,996
4	2,31E-07	2,85E-07	1,31E-12	12,095
5	-4,68E-07	-5,18E-07	5,09E-12	11,72
	$e(\mathbf{x}_1^*)$	$e(\mathbf{x}_2^*)$	$e(x_1^*)$	$t(s)$
	4,16E-07	5,85E-07	1,23E-12	11,439
	8,55E-07	3,61E-07	4,47E-12	11,61
	4,16E-07	7,01E-07	1,21E-12	11,996
	2,31E-07	2,85E-07	1,31E-12	12,095
	4,68E-07	5,18E-07	5,09E-12	11,72
MAE	4,83E-07		2,66E-12	1,18E+01
MAX	8,55E-07		5,09E-12	1,21E+01

Para cada una de las heurísticas viene asociada a su tabla los errores de aproximación de cada una de las variables optimas x_1^*, x_2^* y y^* para la función objetivo, en el tiempo t. Los valores medios en cada uno de los casos son tenidos en cuenta, como en los casos anteriores para las variables de decisión $vma(\mathbf{x}^*)$, la función objetivo $vma(y^*)$ y el tiempo de las corridas $vm(t)$ (Tabla 3.21) y también son graficados (Figura 3.11) para una mejor

observación como se realizó anteriormente. Se arribó a la conclusión que el mejor resultado se alcanzaba a través de la CE.

Tabla 3.19 Resultados de la función de Matyas con GA

No.	x_1^*	x_2^*	y^*	$t(s)$
1	-0,156555	-0,144653	0,0009427	11,798
2	0,078125	0,0726318	0,0002348	15,158
3	-0,0784302	-0,0723267	0,0002366	11,533
4	-0,0784302	-0,0784302	0,0002461	13,47
5	-0,0100708	-0,0100708	0,0008178	13,282
---	$e(x_1^*)$	$e(x_2^*)$	$e(y^*)$	$t(s)$
1	1,57E-01	1,45E-01	9,43E-04	11,798
2	7,81E-02	7,26E-02	2,35E-04	15,158
3	7,84E-02	7,23E-02	2,37E-04	11,533
4	7,84E-02	7,84E-02	2,46E-04	13,47
5	1,01E-02	1,01E-02	8,18E-04	13,282
MAE	7,80E-02	---	4,96E-04	1,30E+01
MAX	1,57E-01	---	9,43E-04	1,52E+01

Tabla 3.20 Resultados de la función de Matyas con PSO

No.	x_1^*	x_2^*	y^*	$t(s)$
1	3,51E-03	3,87E-03	5,86E-07	38,957
2	-7,61E-08	-3,82E-08	1,24E-13	28,128
3	-8,42E-07	-1,15E-06	3,28E-13	26,815
4	1,90E-03	2,21E-03	1,94E-07	42,176
5	5,42E-07	5,05E-07	3,24E-13	28,222
---	$e(x_1^*)$	$e(x_2^*)$	$e(y^*)$	$t(s)$
1	3,51E-03	3,87E-03	5,86E-07	38,957
2	7,61E-08	3,82E-08	1,24E-13	28,128
3	8,42E-07	1,15E-06	3,28E-13	26,815
4	1,90E-03	2,21E-03	1,94E-07	42,176
5	5,42E-07	5,05E-07	3,24E-13	28,222
MAE	1,15E-03	---	1,56E-07	3,29E+01
MAX	3,87E-03	---	5,86E-07	4,22E+01

Tabla 3.21 Valores medios para la función de Matyas

	$vma(\mathbf{x}^*)$	$vma(y^*)$	$vm(t)$
CE	4,83E-07	2,66E-12	1,18E+01
GA	7,80E-02	4,96E-04	1,30E+01
PSO	1,15E-03	1,56E-07	3,29E+01

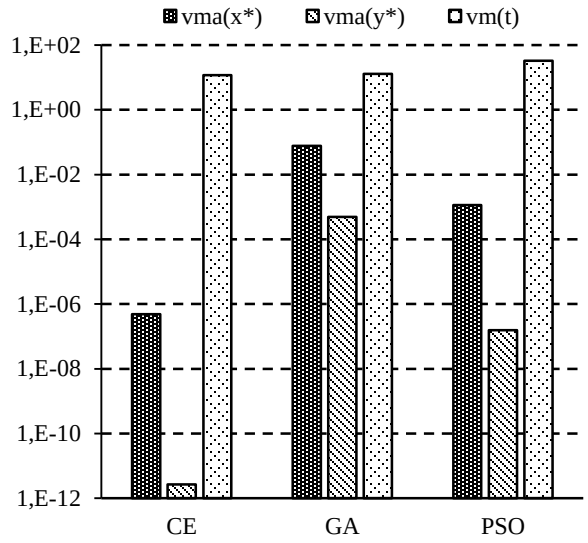

Figura 3.11. Valores medios para la función de Matyas

3.3.4.3 Función Power Sum

La función Power Sum de tipo multimodal tiene como valores recomendados para la evaluación del vector **b**, para *d*=4, es $\mathbf{b} = (8,18,44,114)$ y su función objetivo es la siguiente:

$$f(\mathbf{x}) = \sum_{i=1}^{d} \left[\left(\sum_{j=1}^{d} x_j^i \right) - b_i \right]^2 \tag{3.6}$$

La variable de decisión queda definida en el rango $x_i \in [0, d]$ para todas las $i = 1, \ldots, d.$

A continuación, se presentan los resultados arrojados por la simulación de las heurísticas de optimización CE, GA y PSO para el caso de la función Power Sum. Teniendo en cuenta que los valores óptimos para la función son conocidos, $f\left(\mathbf{x}^*\right)=0,$ para $x_1^*=0, x_2^*=0, x_3^*=0, x_4^*=0$. Como en los casos analizado anteriormente los parámetros independientes correspondiente a cada una de las heurísticas son mostrados. Para CE (Tabla 3.22), tamaño de población $Z=50$, cantidad de generaciones $N=10^5$, límite de convergencia $\varepsilon=10^{-6}$, factor de suavizado $\alpha=0.8$, factor de elitismo $\eta=0.1$, para GA (Tabla 3.23) los valores de tamaño de la población $Z=250$, cantidad de generaciones $N=10^5$,límite de convergencia $\varepsilon=10^{-6}$, probabilidad de mutación $p_m=10^{-5}$, longitud de codificación $L_c=16$ y para PSO (Tabla 3.24) son, población $Z=100$, cantidad de generaciones $N=10^5$,aceleración cognitiva $x_1=0.5$ y aceleración social $x_2=0.5$ y la inercialidad. Para cada una de las heurísticas viene asociada a su tabla los errores de aproximación asociados a cada una de las variables optimas $\mathbf{x}_1^*$, $\mathbf{x}_2^*$ y y^* para la función objetivo, en el tiempo t**.** Los valores medios en cada uno de los casos son tenidos en cuenta, como en los casos anteriores para las variables de decisión $vma(\mathbf{x}^*)$, la función objetivo $vma(y^*)$ y el tiempo de las corridas $vm(t)$ (Tabla 3.25) y también son graficados (Figura 3.12) para una mejor observación como se realizó anteriormente. Se arribó a la conclusión que el mejor resultado se alcanzaba a través de la CE

Tabla 3.22 Resultados de la función Power Sum con CE

No.	x_1^*	x_2^*	x_3^*	x_4^*	y^*	$t(s)$
1	1,34E-08	-4,06E-08	8,56E-09	-7,15E-08	1,78E-16	2,24
2	-5,19E-08	-1,28E-06	-1,11E-06	-4,83E-03	2,63E-12	2,72
3	-1,26E-08	-2,06E-09	-4,02E-08	-2,59E-07	1,59E-16	2,17
4	-1,70E-09	-6,82E-05	-1,32E-03	-3,11E-04	3,34E-12	3,27
5	3,12E-08	5,95E-08	-1,43E-08	2,09E-06	9,72E-16	1,80
---	$e(x_1^*)$	$e(x_2^*)$	$e(x_3^*)$	$e(x_4^*)$	$e(y^*)$	$t(s)$
1	1,34E-08	4,06E-08	8,56E-09	7,15E-08	1,78E-16	2,24
2	5,19E-08	1,28E-06	1,11E-06	4,83E-03	2,63E-12	2,72
3	1,26E-08	2,06E-09	4,02E-08	2,59E-07	1,59E-16	2,17
4	1,70E-09	6,82E-05	1,32E-03	3,11E-04	3,34E-12	3,27
5	3,12E-08	5,95E-08	1,43E-08	2,09E-06	9,72E-16	1,80
MAE	3,27E-04	---	---	---	1,19E-12	2,44
MAX	4,83E-03	---	---	---	3,34E-12	3,27

Tabla 3.23 Resultados de la función Power Sum con GA

No.	x_1^*	x_2^*	x_3^*	x_4^*	y^*	$t(s)$
1	-3,052E-05	-0,0019836	0,0156555	0,0048828	6,883E-08	9,50
2	0	-0,0039368	-3,052E-05	-3,052E-05	6,10E-08	10,31
3	-3,052E-05	0,0039978	-0,0011292	-6,104E-05	6,495E-08	9,95
4	-0,0002747	6,104E-05	0,0007324	-0,0156555	1,087E-06	11,17
5	-3,052E-05	0,0007935	-3,052E-05	-3,052E-05	1,431E-09	10,60
---	$e(x_1^*)$	$e(x_2^*)$	$e(x_3^*)$	$e(x_4^*)$	$e(y^*)$	$t(s)$
1	3,05E-05	1,98E-03	1,57E-02	4,88E-03	6,88E-08	9,50
2	0,00E+00	3,94E-03	3,05E-05	3,05E-05	6,10E-08	10,31
3	3,05E-05	4,00E-03	1,13E-03	6,10E-05	6,49E-08	9,95
4	2,75E-04	6,10E-05	7,32E-04	1,57E-02	1,09E-06	11,17
5	3,05E-05	7,93E-04	3,05E-05	3,05E-05	1,43E-09	10,60
MAE	2,47E-03	---	---	---	2,57E-07	10,31
MAX	1,57E-02	---	---	---	1,09E-06	11,17

Tabla 3.24 Resultados de la función Power Sum con PSO

No.	x_1^*	x_2^*	x_3^*	x_4^*	y^*	$t(s)$
1	2,357E-06	-0,0052242	-0,0264995	0,0824291	4,441E-06	2,375
2	6,65E-04	-1,49E-02	-9,15E-02	-1,09E-01	8,94E-05	21,659
3	-8,35E-05	1,21E-02	2,88E-02	8,62E-02	7,21E-06	22,799
4	2,26E-06	5,79E-03	5,60E-03	6,30E-02	1,18E-06	14,673
5	3,59E-04	3,97E-03	2,45E-02	1,50E-01	7,75E-05	22,674
---	$e(x_1^*)$	$e(x_2^*)$	$e(x_3^*)$	$e(x_4^*)$	$e(y^*)$	$t(s)$
1	2,36E-06	5,22E-03	2,65E-02	8,24E-02	4,44E-06	2,36E-06
2	6,65E-04	1,49E-02	9,15E-02	1,09E-01	8,94E-05	6,65E-04
3	8,35E-05	1,21E-02	2,88E-02	8,62E-02	7,21E-06	8,35E-05
4	2,26E-06	5,79E-03	5,60E-03	6,30E-02	1,18E-06	2,26E-06
5	3,59E-04	3,97E-03	2,45E-02	1,50E-01	7,75E-05	3,59E-04
MAE	3,56E-02	---	---	---	3,59E-05	3,56E-02
MAX	1,50E-01	---	---	---	8,94E-05	1,50E-01

Tabla 3.25 Valores medios para la función Power Sum

	$vma(\mathbf{x}^*)$	$vma(y^*)$	$vm(t)$
CE	3,27E-04	1,19E-12	2,44E+00
GA	2,47E-03	2,57E-07	1,03E+01
PSO	3,56E-02	3,59E-05	1,68E+01

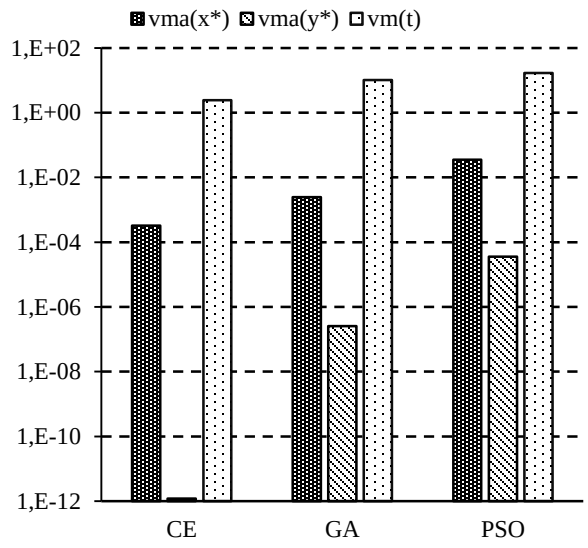

Figura 3.12. Valores medios para la función Power Sum

3.3.5 Problemas con forma de Valle

3.3.5.1 Función Three-Hump Camel

La función Three-Hump Camel (Camello de Tres Jorobas) de tipo bidimensional, es una función que tiene tres mínimos locales y su función objetivo es:

$$f(\mathbf{x}) = 2x_1^2 - 1.05x_1^4 + \frac{x_1^6}{6} + x_1x_2 + x_2^2 \tag{3.7}$$

El dominio de la variable de decisión es $x_i \in [-5,-5]$ para toda $i = 1,2$. Su optimo global se encuentra en $f(\mathbf{x}^*) = 0$ para $\mathbf{x}^* = (0,0)$.

A continuación, se presentan los resultados arrojados por la simulación de las heurísticas de optimización CE, GA y PSO para el caso de la función Three Hump Camel. Teniendo en cuenta que los valores óptimos para la función son conocidos, $f(\mathbf{x}^*) = 0,$ para $\mathbf{x}^* = (0,0)$. Los parámetros independientes correspondiente a cada una de las heurísticas son mostrados. Para CE (Tabla 3.26), tamaño de población $Z = 100$, cantidad de generaciones $N = 10^5$, límite de convergencia $\varepsilon = 10^{-6}$, factor de suavizado $\alpha = 0.8,$ factor de elitismo $\eta = 0.1,$ para GA (Tabla 3.27) los valores de tamaño de la población $Z = 500$, cantidad de generaciones $N = 10^5$,límite de convergencia $\varepsilon = 10^{-6}$, probabilidad de mutación $p_m = 10^{-5}$, longitud de codificación $L_c = 16$ y para PSO (Tabla 3.28) son, población $Z = 500$, cantidad de generaciones $N = 10^5$,aceleración cognitiva $x_1 = 0.5$ y aceleración social $x_2 = 0.5$ y la inercialidad. Para cada una de las heurísticas viene asociada a su tabla los errores de aproximación de cada una de las variables de

decisión optimas $\mathbf{x}_1^*$, $\mathbf{x}_2^*$ y y^* para la función objetivo, en el tiempo t. Los valores medios en cada uno de los casos son tenidos en cuenta, como en los casos anteriores para las variables de decisión $vma(\mathbf{x}^*)$, la función objetivo $vma(y^*)$ y el tiempo de las corridas $vm(t)$ (Tabla 3.29) y también son graficados (Figura 3.13) para una mejor observación como se realizó anteriormente. Se arribó a la conclusión que el mejor resultado se alcanzaba a través de la CE.

Tabla 3.26 Resultados de la función Three Hump Camel con CE

No.	x_1^*	x_2^*	y^*	$t(s)$
1	6,18E-09	1,27E-07	1,14E-11	6,49
2	3,15E-07	4,24E-07	1,64E-11	6,04
3	1,47E-07	3,90E-07	2,61E-11	6,42
4	1,09E-07	-3,33E-07	1,10E-11	6,26
5	1,24E-06	-1,52E-06	5,12E-11	6,11
---	$e(x_1^*)$	$e(x_2^*)$	$e(y^*)$	$t(s)$
1	6,18E-09	1,27E-07	1,14E-11	6,49
2	3,15E-07	4,24E-07	1,64E-11	6,04
3	1,47E-07	3,90E-07	2,61E-11	6,42
4	1,09E-07	3,33E-07	1,10E-11	6,26
5	1,24E-06	1,52E-06	5,12E-11	6,11
MAE	4,61E-07	---	2,32E-11	6,26E+00
MAX	1,52E-06	---	5,12E-11	6,49E+00

Tabla 3.27 Resultados de la función Three Hump Camel con GA

No.	x_1^*	x_2^*	y^*	$t(s)$
1	0,00E+00	0,00E+00	3,05E-06	10,74
2	0,00E+00	-1,53E-04	4,89E-05	11,56
3	-1,53E-04	1,53E-04	4,66E-08	44,66
4	-1,53E-04	1,53E-04	4,66E-08	41,88
5	0,00E+00	0,00E+00	1,22E-05	16,22
---	$e(x_1^*)$	$e(x_2^*)$	$e(y^*)$	$t(s)$
1	0,00E+00	0,00E+00	3,05E-06	10,74
2	0,00E+00	1,53E-04	4,89E-05	11,56
3	1,53E-04	1,53E-04	4,66E-08	44,66
4	1,53E-04	1,53E-04	4,66E-08	41,88
5	0,00E+00	0,00E+00	1,22E-05	16,22
MAE	7,63E-05	---	1,29E-05	2,50E+01
MAX	1,53E-04	---	4,89E-05	4,47E+01

Tabla 3.28 Resultados de la función Three Hump Camel con PSO

No.	x_1^*	x_2^*	y^*	$t(s)$
1	2,85E-05	-1,04E-04	9,51E-09	19,49
2	-3,59E-08	7,14E-08	4,45E-13	11,31
3	5,08E-07	-3,04E-07	1,08E-12	5,92
4	-2,80E-04	-4,47E-03	2,14E-05	20,33
5	7,05E-08	4,65E-08	1,04E-12	6,95
---	$e(x_1^*)$	$e(x_2^*)$	$e(y^*)$	$t(s)$
1	2,85E-05	1,04E-04	9,51E-09	19,49
2	3,59E-08	7,14E-08	4,45E-13	11,31
3	5,08E-07	3,04E-07	1,08E-12	5,92
4	2,80E-04	4,47E-03	2,14E-05	20,33
5	7,05E-08	4,65E-08	1,04E-12	6,95
MAE	4,88E-04	---	4,29E-06	1,28E+01
MAX	4,47E-03	---	2,14E-05	2,03E+01

Tabla 3.29 Valores medios obtenidos para la función Three Hump Camel

	$vma(\mathbf{x}^*)$	$vma(y^*)$	$vm(t)$
CE	4,61E-07	2,32E-11	6,26E+00
GA	7,63E-05	1,29E-05	2,50E+01
PSO	4,88E-04	4,29E-06	1,28E+01

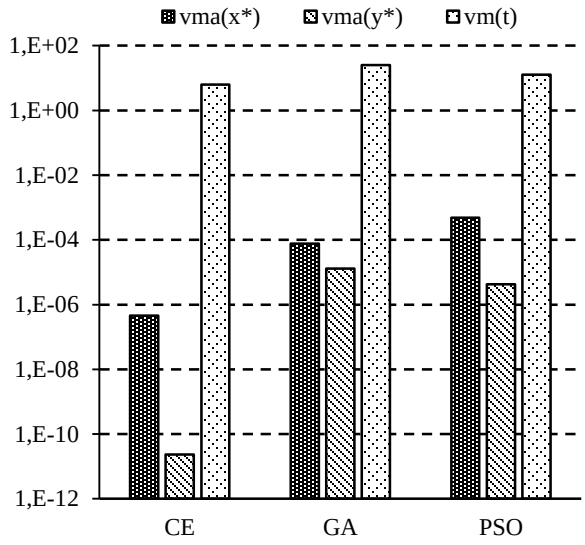

Figura 3.13. Valores medios para la función Three Hump Camel

3.5 Conclusiones parciales del capítulo

Una vez finalizado el presente capítulo, se ha podido arribar a las siguientes conclusiones parciales:

1. Se diseñó un sistema cliente-servidor, con una arquitectura orientada a servicios, para la optimización mono-objetivo utilizando heurísticas sin gradiente, definiendo los correspondientes protocolos de comunicación.

2. Se implementó una aplicación servidor, sobre la suite de protocolos TCP/IP y mediante el uso de la biblioteca GFOLib, que permite resolver problemas de optimización mono-objetivo.

3. Se implementó una aplicación cliente para comprobar el correcto funcionamiento del servidor.

4. Se validó el funcionamiento del sistema cliente-servidor con un grupo de problemas tipo, mostrando un correcto funcionamiento las tres heurísticas implementadas, aunque con superando la entropía cruzada a los otros dos métodos tanto en eficiencia computacional como en calidad de los resultados.

CONCLUSIONES

Como resultado final del trabajo desarrollado, se ha podido arribar a las siguientes conclusiones:

1. Se diseñó, basándose en la revisión crítica de la bibliografía, un sistema cliente-servidor, con arquitectura orientada a servicios, para optimización mono-objetivo basada en heurísticas sin gradiente.

2. Se implementó una biblioteca de clases y, basada en ella una aplicación para servidor y otra para cliente, para llevar a cabo la solución de problemas de optimización mono-objetivo, mediante los métodos de algoritmo genético, entropía cruzada y enjambre de partículas.

3. Se validó el funcionamiento del sistema cliente-servidor implementado con un grupo de problemas de prueba tomados de la literatura, mostrando las tres heurísticas un buen funcionamiento, aunque, la entropía cruzada fue superior a las otras dos.

RECOMENDACIONES

Basadas en las conclusiones obtenidas y para la extensión futura del presente trabajo, se realizan las siguientes recomendaciones:

1. Validar el sistema desarrollado con problemas de optimización correspondientes a procesos industriales reales.

2. Implementar clientes para otros sistemas operativos, tales como Linux y Android.

3. Incrementar las funcionalidades de la aplicación incorporando otras heurísticas mono-objetivos así como capacidad para optimización multi-objetivo.

REFERENCIAS BIBLIOGRÁFICAS

Affenzeller, M.; Wagner, S.; Winkler, S.; Beham, A., 2009. *Genetic algorithms and genetic programming: Modern concepts and practical applications.* Boca Raton, FL (USA): Chapman and Hall/CRC ISBN 9781138114272.

Arab, H.G.; Rashki, M.; Rostamian, M.; Ghavidel, A.; Shahraki, H.; Keshtega, B., 2018. "Refined first-order reliability method using cross-entropy optimization method". *Engineering with Computers* DOI: 10.1007/s00366-018-0680-9.

Attanasio, A.; Ceretti, E.; Giardini, C., 2017. "SWARM optimization of force model parameters in micromilling". *Procedia CIRP*, 58, pp. 434-439, DOI: 10.1016/j.procir.2017.03.248.

Beruvides, G.; Castaño, F.; Haber, R.E.; Quiza, R.; Villalonga, A., 2017. "Coping with complexity when predicting surface roughness in milling processes: Hybrid incremental model with optimal parametrization". *Complexity*, DOI: 10.1155/2017/7317254.

Bonyadi, M.R.; Michalewicz, Z., 2017. "Particle swarm optimization for single objective continuous space problems: A review". *Evolutionary Computation*, 25 (1), pp. 1-54, DOI: 10.1162/EVCO_r_00180.

Bouhmala, N., 2018. "Combining simulated annealing with local search heuristic for MAX-SAT". *Journal of Heuristics*, [online], DOI: 10.1007/s10732-018-9386-9.

Bozorg-Haddad, O. (ed.), 2018. *Advanced optimization by nature-inspired algorithms*. Singapore: Springer.

Bronshtein, I.N.; Semendyayev, K.A.; Musiol, G.; Mühlig, H., 2015. *Handbook of mathematics.* 6th Ed. Berlin (Germany): Springer-Verlag, ISBN 978-3-662-46220-1.

Buontempo, F., 2019. *Genetic algorithms and machine learning for programmers: Create AI models and evolve solutions.* Raleigh, NC (USA): The Pragmatic Programmers, ISBN 978-1-68050-620-4.

Coello, C.A.; Lamont, G.B.; Veldhuizen, D.A., 2007. *Evolutionary algorithms for solving multi-objective problems.* 2nd Ed. New York: Springer, ISBN 978-0-387-33254-3.

Chouhan, S.S.; Kaul, A.; Singh, U.P., 2018. "Soft computing approaches for image segmentation: a survey". *Multimedia Tools and Applications*, 77 (21), pp. 28483-28537, DOI: 10.1007/s11042-018-6005-6.

De Boer, P.T.; Kroese, D.P.; Mannor, S.; Rubinstein, R.Y., 2005. "A Tutorial on the cross-entropy method". *Annals of Operations Research*, 134, pp. 19-67, DOI: 10.1007/s10479-005-5724-z.

Ecma International, 2018. *ECMA-262: ECMAScript (r) 2018 language specification* Geneva (Switzerland): Ecma International.

Fidanova, S. (ed.), 2018. *Recent advances in computational optimization (Results of the Workshop on Computational Optimization WCO 2016)*. Cham (Switzerland): Sprimger, ISBN 978-3-319-59860-4.

Gao, X.; Jia, L.; Kar, S., 2018. "A new definition of cross-entropy for uncertain variables". *Soft Computing*, DOI: 10.1007/s00500-017-2534-6.

Gong, Y.J.; Chen, W.N.; Zhan, Z.H.; Zhang, J.; Li, Y.; Zhang, Q.; Li, J.J., 2015. "Distributed evolutionary algorithms and their models: A survey of the state-of-the-art". *Applied Soft Computing*, 34, pp. 286-300, DOI: 10.1016/j.asoc.2015.04.061.

Gupta, A.; Ong, Y.S., 2019. *Memetic computation: The mainspring of knowledge transfer in a data-driven optimization era.* Cham (Switzerland): Springer Nature, ISBN 978-3-030-02728-5.

Haber, R.E.; Del Toro, R.M.; Gajate, A., 2010. "Optimal fuzzy control system using the cross-entropy method. A case study of a drilling process". *Information Sciences*, 180, pp. 2777-2792, DOI: 10.1016/j.ins.2010.03.030.

Haber, R.E.; Haber-Haber, R.; Jiménez, A.; Galán, R., 2009. "An optimal fuzzy control system in a network environment based on simulated annealing. An application to a drilling process". *Applied Soft Computing*, 9 (3), pp. 889-895, DOI: https://doi.org/10.1016/j.asoc.2008.11.005.

Henderson, D.; Jacobson, S.H.; Johnson, A.W., 2003. "Handbook of Metaheuristics". En: Glover, F. ; Kochenberger, G.A. (eds.), *Handbook of Metaheuristics*. Boston, MA (USA): Springer, ISBN 978-1-4020-7263-5.

Hwang, K., 2017. *Cloud computing for machine learning and cognitive applications.* Cambridge, MA (USA): MIT Press, ISBN 978-0-26203641-2

Hwang, K.; Fox, G.C.; Dongarra, J.J., 2012. *Distributed and cloud computing: From parallel processing to the Internet of Things.* Singapore: Elsevier, ISBN 978-0-12-385880-1.

Jacobson, L.; Kanber, B., 2015. *Genetic algorithms in Java basics.* New York (USA): Apress Media, ISBN 978-1-4842-0329-3.

Kanneganti, R.; Chodavarapu, P., 2008. *SOA security.* Greenwich, CT (USA): Manning, ISBN 1-932394-68-0.

Kramer, O., 2017. *Genetic algorithm essentials.* Cham (Switzerland): Springer International, ISBN 978-3-319-52155-8.

La Fé, I.; Beruvides, G.; Quiza, R.; Haber, R.E.; Rivas, M., 2019. “Automatic selection of optimal parameters based on simple soft computing methods: A case study on micro-milling processes”. *IEEE Transactions on Industrial Informatics*, 5 (2), pp. 800, DOI: 10.1109/TII.2018.2816971.

La Fé, I.; Quiza, R.; Rivas, M.; Ramtahalsing, V., 2018. “Hybrid modelling and optimization of the oblique cutting of AISI 1045 steel”. *International Journal of Manufacturing, Materials, and Mechanical Engineering*, 8 (4), DOI: 10.4018/IJMMME.2018100101.

La Fé, I.; Rivas, M.; Quiza, R.; Zambrano-Robledo, P., 2017. “Optimización multiobjetivo de la soldadura automática bajo fundente del acero JIS 3116 ”. *Ingeniería Mecánica,* 20 (2), pp. 77.

Li, K.; Kwong, S.; Wang, R.; Tang, K.-S.; Man, K.-F., 2013. “Learning paradigm based on jumping genes: A general framework for enhancing exploration in evolutionary multiobjective optimization”. *Information Sciences*, 226, pp. 1-22, DOI: 10.1016/j.ins.2012.11.002.

Minisci, E.; Vasile, M.; Periaux, J.; Gauger, N.R.; Giannakoglou, K.C.; Quagliarella, D. (eds.), 2019. *Advances in evolutionary and deterministic methods for design, optimization and control in engineering and sciences*. Cham (Switzerland): Springer, ISBN 978-3-319-89986-2.

Mirjalili, S., 2016. “Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems”. *Neural Computing and Applications*, 27 (4), pp. 1053-1073, DOI: 10.1007/s00521-015-1920-1.

Olivares-Mendez, M.A.; Fu, C.; Kannan, S.; Voos, H.; Campoy, P., 2014 of Conference. “Using the cross-entropy method for control optimization: A case study of see-and-avoid on unmanned aerial vehicles”. *22nd Mediterranean Conference on Control and Automation (MED2014)*. Palermo (Italy): IEEE, pp. 1183-1189.

Oppitz , M.; Tomsu, P., 2018. *Inventing the cloud century: How cloudiness keeps changing our life, economy and technology*. Cham (Switzerland): Springer Nature, ISBN 978-3-319-61161-7.

Pérez, T.; Cruz, Y.J.; La Fé, I.; Quiza, R., 2019. “Biblioteca de clases para optimización multiobjetivo mediante el método de entropía cruzada”. *Revista Cubana de Ciencias Informáticas*, 13 (1), pp. 90-104.

Punia, P.; Kaur, M., 2013. “Various genetic approaches for solving single and multi-objective optimization problems: A review”. *International Journal of Advanced Research in Computer Science and Software Engineering*, 3 (7), pp. 1014-1020.

Quiza, R.; Beruvides, G.; Davim, J.P., 2014. “Modeling and optimization of mechanical systems and processes”. En: Davim, J.P. (ed.), *Modern mechanical engineering*. Springer Berlin Heidelberg, ISBN 978-3-642-45175-1, pp. 169-198.

Quiza, R.; López-Armas, O.; Davim, J.P., 2012. *Hybrid modeling and optimization of manufacturing: Combining artificial intelligence and finite element method.* Heidelberg (Germany): Springer, ISBN 9783642260849.

Ravindran, A.; Ragsdell, K.M.; Reklaitis, G.V., 2006. *Engineering optimization: Methods and applications.* 2nd. Ed. Ed. Hoboken, NJ (USA): John Wiley & Sons, ISBN 978-0-471-55814-9.

Roten-Gal-Oz, A., 2012. *SOA Patterns.* Shelter Island, NY (USA): Manning Publications, ISBN 978-1-93398826-9.

Rubinstein, R.Y.; Kroese , D.P., 2004. *The cross-entropy method: A unified approach to combinatoriaJ optimization, Monte-Carlo simulation, and machine learning.* New York (USA): Springer Science+Business Media, ISBN 978-1-4419-1940-3.

Saborido, R.; Ruiz, A.B.; Bermúdez, J.D.; Vercher, E.; Luque, M., 2016. "Evolutionary multi-objective optimization algorithms for fuzzy portfolio selection". *Applied Soft Computing*, 39, pp. 48-63, DOI: 10.1016/j.asoc.2015.11.005.

Siarry, P. (ed.), 2016. *Metaheuristics*. Cham (Switzerland): Springer, ISBN 978-3-319-45401-6.

Surjianovic, S.; Bingham, E., 2017. *Virtual library of simulations experiments: Test functions and datasets* [online]. Simon Fraser University. Disponible en: *https://www.sfu.ca/~ssurjano/optimization.html*. [Citado: 2018.01.12].

Talbi, E.-G.; Nakib, A. (eds.), 2019. *Bioinspired heuristics for optimization*. Cham (Switzerland): Springer Nature, ISBN 978-3-319-95103-4.

Umer, U.; Qudeiri, J.A.; Hussein, H.A.M.; Khan, A.A.; Al-Ahmari, A.R., 2014. "Multi-objective optimization of oblique turning operations using finite element model and genetic algorithm". *International Journal of Advanced Manufacturing Technology*, 71 (1-4), pp. 593-603, DOI: 10.1007/s00170-013-5503-y.

Van Veldhuizen, D.A., 1999. *Multiobjective evolutionary algorithms: Classications, analyses, and new innovations*. Ph.D. Wright-Patterson AFB: Air Force Institute of Technology.

Wang, J.; Kissel, Z.A., 2015. *Introduction to network security: Theory and practice.* Singapore: John Wiley and Sons ISBN 978-1-118-93948-2.

Wolpert, D.H.; Macready, W.G., 1997. "No free lunch theorems for optimization". *IEEE Transactions on Evolutionary Computation*, 1 (1), pp. 67-82.

Yan, W.; Zhang, H.; Jiang, Z.-G.; Hon, K.K.B., 2016. "Multi-objective optimization of arc welding parameters: the trade-offs between energy and thermal efficiency". *Journal of Cleaner Production*, [in press], DOI: 10.1016/j.jclepro.2016.03.171.

Yang, X.-S. (ed.), 2018. *Nature-inspired algorithms and applied optimization*. Cham (Switzerland): Springer International, ISBN 978-3-319-67668-5.

Yang, X.S.; Koziel, S.; Leifsson, L., 2014. "Computational optimization, modelling and simulation: Past, present and future". *Procedia Computer Science*, 29, pp. 754-758, DOI: 10.1016/j.procs.2014.05.067.

Zhang, X., 2018. *LTE optimization engineering handbook.* Singapore: John Wiley and Sons, ISBN 9781119159001.