/UCIENCIA/.18
III Conferencia Científica Internacional
Universidad de las Ciencias Informáticas

Temática: Software libre

Biblioteca de clases para optimización multiobjetivo mediante el método de entropía cruzada

Class library for multi-objective optimization by using the cross-entropy method

Teresa Pérez-Sosa[*], Yarens Cruz, Iván La Fé, Ramón Quiza

Centro de Estudio de Fabricación Avanzada y Sostenible (CEFAS), Universidad de Matanzas. Autopista a Varadero km 3½, Matanzas 44740, Cuba. Teléf.: +(53)45256812, Web: *http://cefas.umcc.cu*
[*] Autor para correspondencia: teresa.perez@umcc.cu

Resumen
La optimización multiobjetivo juega un papel fundamental en la ingeniería moderna, ya que permite diseñar sistemas y procesos donde se mejoran varios criterios simultáneamente. En este campo, se destaca el uso del llamado enfoque *a posteriori*, basado en la obtención de las soluciones no dominadas (conocidas también como el conjunto de Pareto) para, posteriormente, seleccionar de ellas la alternativa más conveniente para las condiciones reales. Éste ha mostrado ser la opción más efectiva, ya que evita el suministro previo de cualquier indicación de preferencia con respecto a los objetivos. Este trabajo presenta la implementación de una biblioteca de clases para la optimización multiobjetivo a través del método de entropía cruzada. Ésta se basa en una biblioteca previa, de MATLAB, desarrollada y publicada por los autores. La biblioteca fue implementada en C++, evitando el uso de otras bibliotecas aparte de la estándar, para garantiza su portabilidad y su capacidad de integración. Será distribuida bajo la Licencia Pública General Reducida de GNU (versión 3). Para comprobar no sólo el funcionamiento de la biblioteca, sino también su inserción en otros programas, se desarrolló una aplicación. Todas las pruebas realizadas mostraron un buen desempeño sin errores identificados. Como desarrolló futuro de este trabajo, se prevé la implementación de otras heurísticas de optimización multiobjetivo, tales como los algoritmos genéticos y la optimización de hormiguero. También deberá ser integrada esta biblioteca en aplicaciones para resolver problemas industriales reales, como son la optimización de la sostenibilidad de procesos de fabricación y el diseño de microindustrias alimentarias.

Palabras clave: Biblioteca de clases, optimización multiobjetivo, entropía cruzada, frente de Pareto, software libre.

Abstract
Multi-objective optimization plays a key role in modern engineering practice, as it allows designing systems and processes where several criteria are simultaneously improved. In this field, it should be highlighted the use of the so-called a posteriori *approach, which is based on obtaining the non-dominated solutions (also known as the Pareto set) for, after that, choosing from them the most convenient alternative for the actual conditions. It has emerged as the most effective choice, as it avoids the previous supply of any preference indication with respect to the objectives. This work presents the implementation of a class library for multi-objective optimization by using the cross-entropy method. It is based in a previous MATLAB toolbox developed and published by the authors. The library was implemented in C++, avoiding the use of any library but the standard ones, for guarantying the portability and integration capability. It will be distributed under the GNU Lesser General Public License (version 3). For testing not only the behavior of class library but also its insertion in other programs, an application was developed. All the tests carried out to the software have shown a good performance without identified mistakes. As a future development of this work, it has been foreseeing the implementation of other multi-objective optimization heuristic, such as genetic algorithms and ant colony optimization. It should be also integrated this library in applications for solving actual industrial problems, such as optimization of sustainability of manufacturing processes and design of micro-industries for food processing.*

Keywords: *Class library, multi-objective optimization, cross-entropy, Pareto front, free software.*

Introducción

En la industria moderna, la inmensa mayoría de los asuntos de diseño de sistemas y procesos, pasa por la solución de un problema de optimización. Como, usualmente, se requiere considerar, simultáneamente, varios criterios contradictorios, dicho problema de optimización es multiobjetivo. Este tipo de problemas puede resolverse integrando los diversos objetivos en uno solo, ya sea por agregación lineal o no lineal, por ordenamiento, o por otra técnica. Este enfoque, llamado *a priori*, que realmente transforma el problema multiobjetivo en uno uniobjetivo, tiene el inconveniente de que debe suministrarse cierta información sobre las preferencias con respecto a los objetivos, antes de proceder al proceso de optimización, siendo la materialización de estas preferencias usualmente subjetiva y poco realista (Yang at al., 2014).

Por el contrario, el llamado enfoque *a posteriori*, evita este problema al no requerir ninguna información previa acerca de la preferencia de los objetivos. En ella, el proceso de optimización conduce a un grupo de soluciones óptimas, que lo son en el sentido amplio de que ninguna otra, en el espacio de soluciones factibles considerado, mejora uno de los objetivos sin, a su vez, empeorar algunos de los otros. A este conjunto de soluciones se les llama no dominadas o conjunto de Pareto, y a su representación en el espacio de objetivos, frente de Pareto (Xiong et al.,

2015). Una vez obtenido el frente de Pareto, se selecciona la solución más conveniente, teniendo en cuenta las condiciones reales del problema.

La literatura especializada refleja una gran diversidad de técnicas de optimización multiobjetivo a posteriori, siendo la inmensa mayoría de ellas heurísticas no basadas en el uso del gradiente de la función objetivo (Coello et al., 2007). Esto les confiere la importante ventaja, frente a los métodos numéricos basados en gradiente, de no requerir que dicha función objetivo cumpla un conjunto de requisitos matemáticos como la continuidad, la derivabilidad o la unimodalidad (Gong et al., 2015).

Dentro de las heurísticas para optimización multiobjetivo, el método de entropía cruzada se destaca por su capacidad para resolver problemas donde la probabilidad de encontrar la solución es particularmente baja (De Boer et al. 2005), lo que lo hace muy apropiado para situaciones donde el conjunto de Pareto está localizado en una parte muy pequeña del especio de búsqueda, sin gradientes marcados que lleven a él. Varias propuestas, en este sentido, has sido publicadas (Ünveren y Acan, 2007; Bekker y Aldrich, 2011; Hauman, 2012; Sebaa et al., 2013; Giagkiozis et al., 2014). Coautores de este trabajo, han participado en el desarrollo y la implementación de una heurística multi-objetivo basada en entropía cruzada, que ha demostrado simplicidad (en el sentido de requerir el ajuste de pocos parámetros) y efectividad en la solución de problemas complejos (Beruvides et al., 2016; Haber et al., 2017). Esta heurística, que ha sido aplicada con éxito a la solución de problemas prácticos (La Fé et al., 2018), está, sin embargo, implementada en MATLAB, lo que dificulta tanto su distribución (por el carácter propietario de MATLAB) como su incorporación a otros programas.

En consecuencia, se ha trazado como objetivo del presente trabajo, la implementación y comprobación de una biblioteca de clases, en lenguaje C++, que permita llevar a cabo la solución de problemas de optimización multiobjetivo, mediante una heurística basada en entropía cruzada. El trabajo se ha dividido en xxx secciones. Luego de la presente introducción, se describe sucintamente el algoritmo utilizado. La tercera sección describe la implementación de la biblioteca de clases y de una aplicación desarrollada para validad su funcionamiento. Finalmente, la comprobación de los resultados de la clase se presenta en la cuarta sección y, para finalizar, se resumen las conclusiones y proyecciones de trabajo futuras.

Materiales y métodos

Descripción del algoritmo

El algoritmo para optimización multiobjetivo con entropía cruzada simple (*simple multi-objective cross-entropy*, SMOCE) (Haber et al., 2017), se dirige, a resolver el problema:

$$\min(\mathbf{y}) = \mathbf{F}(\mathbf{x}): \quad \mathbf{x} \in \mathbb{R}^n \ \mathbf{y} \quad \mathbb{R}^m; \tag{1}$$

donde:

$$l_i \le x_i \le u_i, \quad i = \{1, \dots \ n \ ; \tag{2}$$

y que, además, puede encontrarse restringida por:

$$g_i(\mathbf{x}) \le 0, \quad i = \{1, \dots \ p \ . \tag{3}$$

El núcleo del algoritmo de SMOCE es la población de trabajo, **Q**, la cual, en la *t*-ésima iteración:

$$\mathbf{Q}^{(t)} = \{(\mathbf{x}_1, \mathbf{y}_1)^{(t)}, \dots \ \mathbf{x}_Z \ \mathbf{y}_Z)^{(t)}\}; \tag{4}$$

está compuesta por *Z* soluciones $\mathbf{x}_k = \{x_{k,1}, \dots \ x_{k\,n}$ y sus respectivas funciones objetivo evaluadas, de forma que para el *k*-ésimo vector de variables, $\mathbf{x}_k$, el correspondiente vector de objetivos, $\mathbf{y}_k$, toma la forma:

$$\mathbf{y}_k = \{y_{k,1} = f_1(\mathbf{x}_k), \dots \ y_{k\,m} \quad f_m \ \mathbf{x}_k \quad k \quad \dots \ Z \ . \tag{5}$$

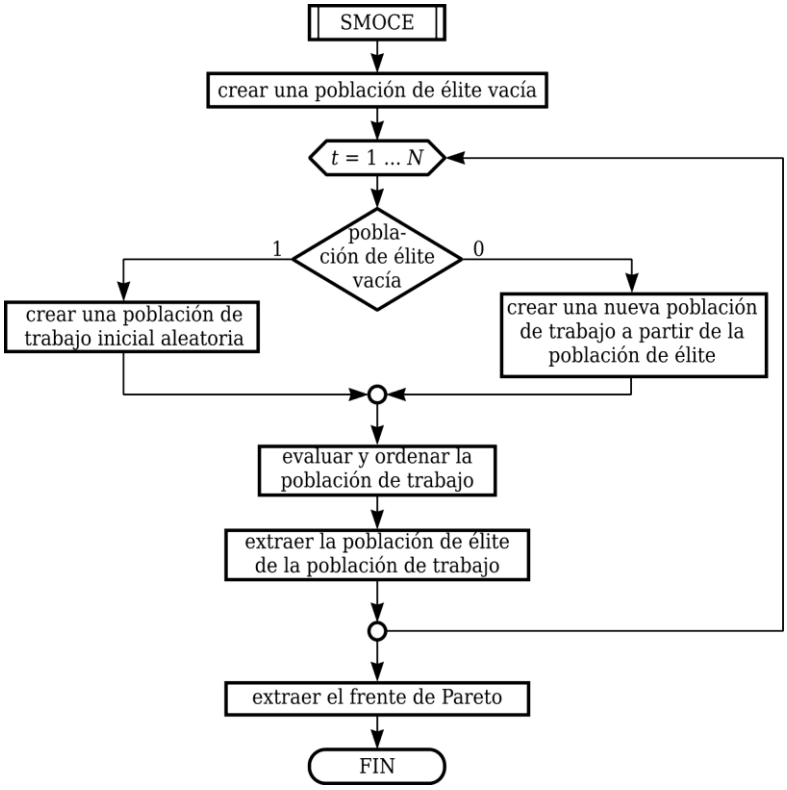

Figura 1 Diagrama de bloques del algoritmo SMOCE

El proceso evolutivo (ver Fig. 1) tiene lugar en un ciclo con una única condición de término: el arribo al número máximo de generaciones, N. En la primera iteración, se crea una población de trabajo con valores de sus variables de decisión tomados aleatoriamente de una distribución uniforme entre sus respectivos valores mínimos y máximos. En cada una de las iteraciones siguientes, se crea una nueva población de trabajo, $\mathbf{Q}^{(t)}$, a partir de la de la iteración anterior, $\mathbf{Q}^{(t-1)}$. Esto se lleva a cabo evaluando cada solución de la población y ordenándolos según su dominancia (es decir, la cantidad de soluciones que la dominan), para extraer de ella los $E = \alpha Z$ mejores individuos, siendo $0 \leq \alpha \leq 1$, un parámetro del algoritmo denominado fracción de élite.

A partir de la población de élite, se crea un histograma donde la dimensión correspondiente a cada una de las funciones objetivos, se divide en D intervalos, por lo que las soluciones de élite se agrupan en D^m clases. Para cada una de dichas clases, se crea una fracción de la nueva población, de tamaño proporcional a la cantidad de soluciones contenidas en la clase, y seleccionados aleatoriamente, a partir de una distribución normal con media y varianza igual

a cada una de las variables de las soluciones de dicha clase, truncada a los valores mínimos y máximos de dicha variable.

Una vez concluido el proceso evolutivo, se extrae el frente de Pareto filtrando la población de élite resultante para eliminar todas las soluciones dominadas.

El nivel de complejidad, *C*, del algoritmo propuesto es similar al de otras heurísticas basadas en ordenamiento de Pareto, el cual está dado por la expresión (Coello et al. 2007 p. 317):

$$C = \tau NZm + NZ^2m - NZm\,; \tag{6}$$

donde τ es el tiempo de evaluación de la función objetivo, N es el número de iteraciones, Z es el tamaño de la población y m es la cantidad de funciones objetivos. Un estudio detallado de la eficiencia del algoritmo y, especialmente, de la influencia de sus parámetros en dicha eficiencia, ha sido previamente publicado por Haber y coautores (2017). Ese propio trabajo incluye, además, una comparación de los resultados obtenidos con la librería propuesta (MOCE+) con otras cinco heurísticas de optimización multiobjetivo: NSGA-II, MOEA/D, MOPSO, SPEA-II y PESA-II.

Descripción de la librería de clases

La librería de clases, denominada libGFO (*gradient free optimization*) incluye aquellas clases necesarias para llevar a cabo la optimización multiobjetivo con el método de entropía cruzada, aunque fue diseñada teniendo en cuenta su extensión futura a otras heurísticas.

En primer lugar, se diseñó la clase `GFOVariable` (Fig. 2), que representa a una variable de decisión, del problema de optimización. La misma cuenta con tres atributos: el nombre (cadena de caracteres) y los valores mínimo y máximo (numérico) que puede tomar la variable. La clase presenta, además, operaciones para acceder a y modificar los valores de dichos atributos.

GFOVariable
-my_name : string -my_min : double -my_max : double
+GFOVariable(_name : string = “”, _min : ouble = 0.0, _max : double = 1.0 +name() : string +setName(_name : string) +min() : double +setInterval(_min : double = 0.0, _max : double = 1.0)

Figura 2 Esquema UML de la clase `GFOVariable`

La clase `GFOVariables` (Fig. 3) representa al conjunto de todas las variables de decisión del problema. Su único atributo es un listado de las instancias de `GFOVariable`. La clase cuenta con operaciones para agregar, modificar y eliminar variables de la lista.

GFOVariables
-my_vars : GFOVariables []
+count() : int +operator[](_index : int) : GFOVariable +set(_index : int, _name : string) +set(_index : int, _min : double, _max : double) +set(_index : int, _name : string, _min : double, _max : double) +append(_var : GFOVariable) +append(_name : string, _min : double = 0.0, _max : double = 1.0) +remove(_index _int) +clear()

Figura 3 Esquema UML de la clase `GFOVariables`

La clase `GFOObjective` (Fig. 4) representa a los objetivos de optimización. Ésta cuenta con los atributos para almacenar el nombre y el tipo de objetivo (minimización o maximización) así como con las operaciones de acceso y modificación correspondientes.

GFOObjective
-my_name : string -my_type : GFOObjectiveType
+GFOObjective(_name : string = “”, _type : GFOObjectiveType = otMinimization) +name() : string +setName(_name : string) +type() : GFOObjectiveType +setType(_type : GFOObjectiveType)

Figura 4 Esquema UML de la clase `GFOObjective`

De forma similar a la clase `GFOVariables`, `GFOObjectives` (Fig. 5) permite manipular el conjunto de las funciones objetivo del problema. Cuenta, como atributo, con un listado de instancias de la clase `GFOObjective` y con un conjunto de operaciones que permiten consultar y modificar dicho listado.

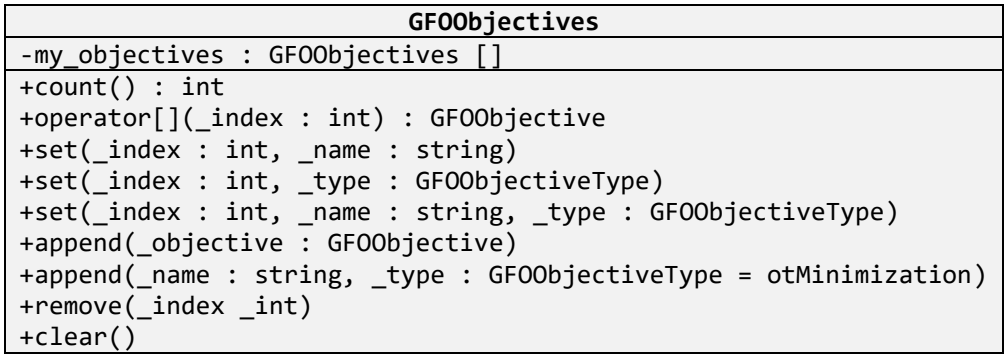

Figura 5 Esquema UML de la clase `GFOObjectives`

La clase `GFOParameters` (Fig. 6) permite representar el conjunto de pares nombre/valores que contienen los parámetros del algoritmo de optimización. Dicha clase contiene, como atributos, un listado de valores de cadena, para almacenar los nombres y otro listado, numérico, para los valores correspondientes. Cuenta, además, con un grupo de operaciones que permiten agregar, eliminar y consultar los valores de los parámetros, dado su nombre.

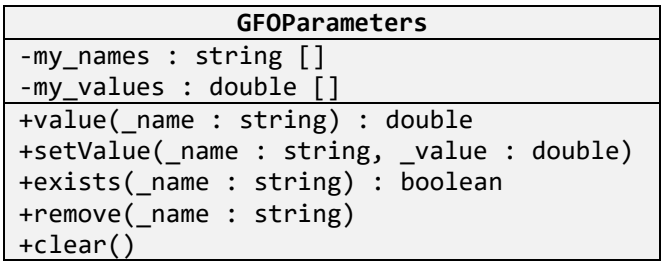

Figura 6 Esquema UML de la clase `GFOParameters`

La clase abstracta `GFOFunction` (Fig. 7) permite definir la clase que servirá de base para cualquier función objetivo. La misma cuenta con una única operación abstracta, consistente en la evaluación de las funciones objetivos, tomando como argumento una matriz, con los valores de las variables de decisión y devolviendo otra matriz, con los valores de los objetivos.

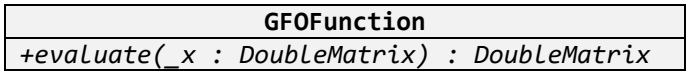

Figura 7 Esquema UML de la clase abstracta `GFOFunction`

Para la representación del algoritmo de optimización, se definió una clase base denominada `GFOHeuristicBase`, (Fig. 8) con el objetivo de, a través de clases derivadas, implementar las diversas heurísticas. La misma implementa las funcionalidades comunes a cualquier algoritmo de optimización. En primer lugar, dispone de los atributos públicos para manipular los parámetros, las variables de decisión y los objetivos del problema, así como de atributos privados para la función objetivo y el almacenamiento de la población. Cuenta, además, con operaciones para asignar y devolver la función objetivo y para devolver el frente de Pareto. También presenta una operación abstracta, para implementar la ejecución del algoritmo, en las clases derivadas.

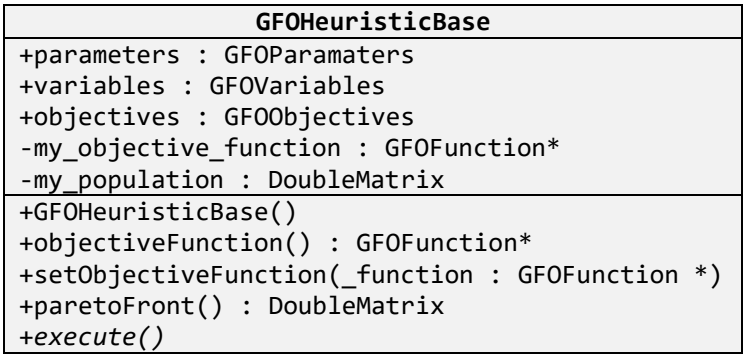

Figura 8 Esquema UML de la clase `GFOHeuristicBase`

Finalmente, se implementó la clase `GFOHeuristicMOCE` (Fig. 9), derivada de `GFOHeuristicBase`, para representar el algoritmo de optimización multiobjetivo, basado en entropía cruzada.

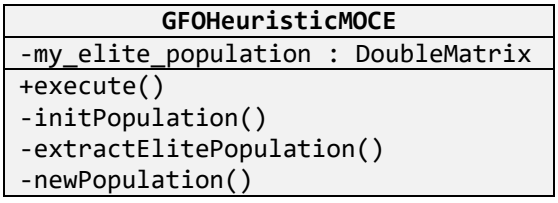

Figura 9 Esquema UML de la clase `GFOHeuristicMOCE`

La clase `GFOHeuristicMOCE` cuenta con un atributo para almacenar la población de élite. También dispone de operaciones para inicializar la población de trabajo, para extraer, de ésta, a la población de élite, y para crear la nueva población de trabajo a partir de la de élite.

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`.

Aplicación para el uso de la librería

Con el propósito de comprobar tanto la funcionalidad de la librería desarrollada como su capacidad de integración en otros programas, se desarrolló una aplicación para optimización multiobjetivo, con interfaz gráfica de usuario. La aplicación (Fig. 10), denominada Hicaco Archer, fue implementada utilizando la plataforma Qt, con el objetivo de garantizar su portabilidad tanto a MS Windows como a Linux.

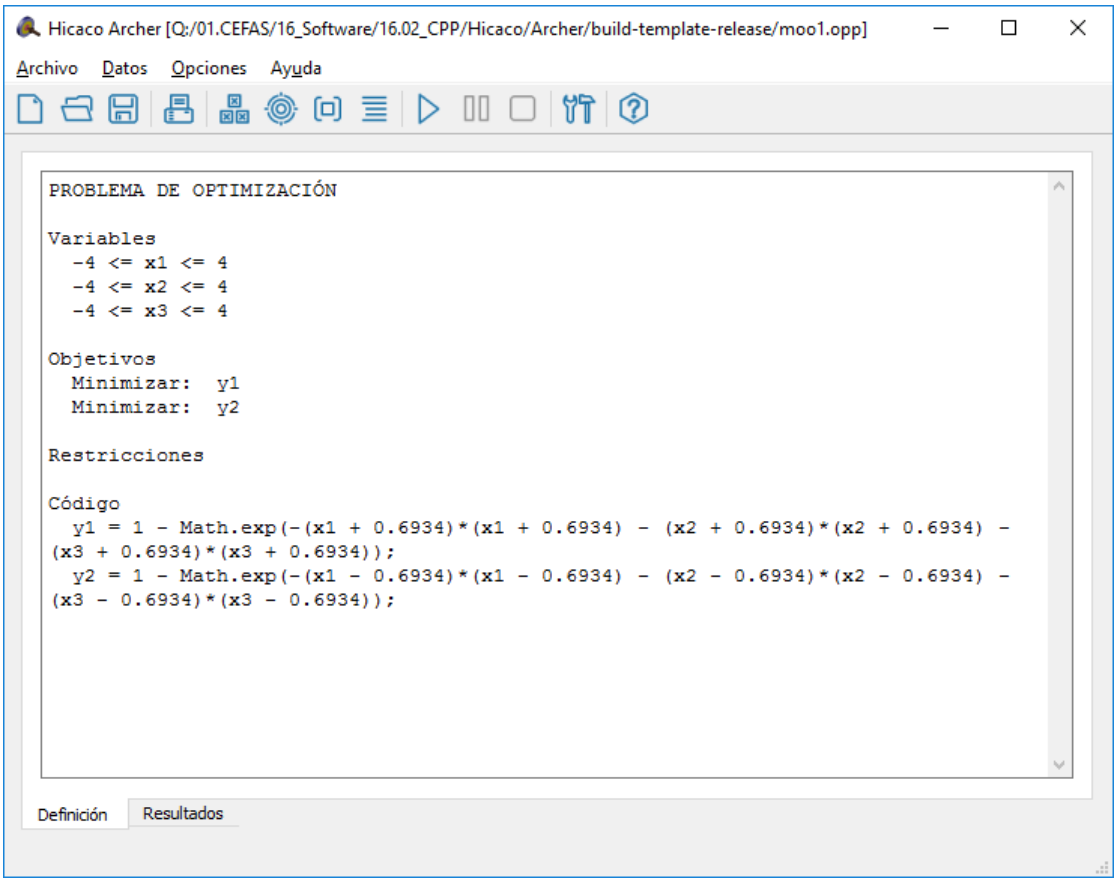

Figura 10 Ventana principal de Hicaco Archer

La aplicación permite, a través de cuadros de diálogo, especificar las componentes del problema de optimización: variables, objetivos, restricciones y código. Luego de ejecutado el proceso de optimización, presenta los resultados tanto en forma tabular como gráfica (representando la frontera de Pareto).

El código que relaciona las variables de decisión con las funciones objetivo y las restricciones, se implementa en Qt Script, el cual se basa en ECMAScript.

Resultados y discusión

Para analizar los resultados, se resolvieron tres problemas de optimización multiobjetivo, reflejados por la literatura (Huband et al., 2006). El primero de ellos (conocido en la literatura como MOP1), tiene dos objetivos a minimizar:

$$\begin{cases} y_1 = x^2 \\ y_2 = (x-2)^2 \end{cases} \tag{7}$$

con una única variable de decisión, $10^{-3} \leq x \leq 10^3$, sin restricciones.

Para la ejecución del mismo, se estableció una población de 500 soluciones, con una fracción de élite de 0,2; un número máximo de iteraciones de 100; y una cantidad de intervalos del histograma de 5. La Fig. 11 muestra la frontera de Pareto obtenida. La misma tiene un factor de hiperárea de 0.9981, lo cual significa que la frontera obtenida está muy cercana a la real.

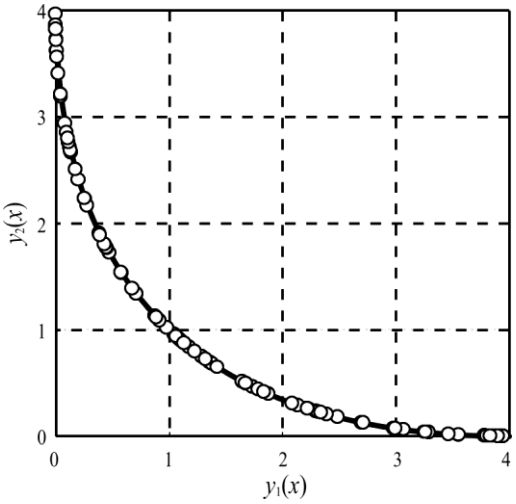

Figura 11 Frontera de Pareto obtenida para el problema MOP1

El segundo ejemplo considerado (denominado usualmente como MOP2), también tiene dos variables de decisión:

$$\begin{cases} y_1 = 1 - \exp\left(-\sum_{i=1}^{3} \left(x_i - 1/\sqrt{n} \right)^2 \right) \\ y_2 = 1 - \exp\left(-\sum_{i=1}^{3} \left(x_i + 1/\sqrt{n} \right)^2 \right) \end{cases} \quad (8)$$

con tres variables de decisión, $-4 \leq x_1, x_2, x_3 \leq 4$, y sin restricciones. Para el mismo, se establecieron como valores de los parámetros del algoritmo, los siguientes: tamaño de la población, 1000; factor de élite, 0,35; cantidad de iteraciones, 200; y cantidad de intervalos del histograma, 10. Los resultados obtenidos se muestran en la Fig. 12. La razón de hiperárea correspondiente fue de 0.9999, mostrando una coincidencia casi perfecta con la frontera de Pareto real.

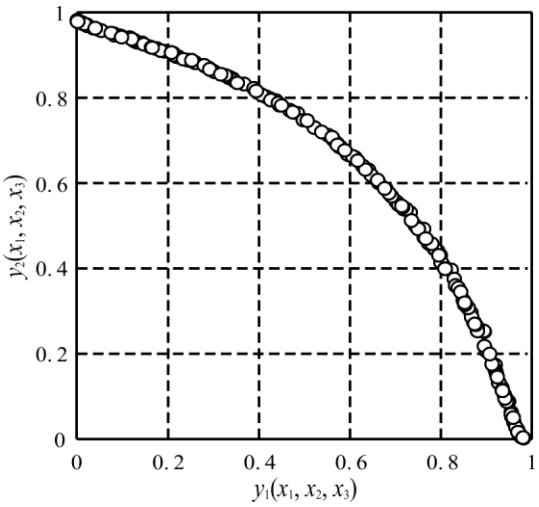

Figura 12 Frontera de Pareto obtenida para el problema MOP2

El tercer ejemplo (conocido como MOP4C), también considera dos objetivos:

$$\begin{cases} y_1 = x_1 \\ y_2 = x_2 \end{cases} \quad (9)$$

pero con dos variables de decisión, $-\pi \leq x1, x_2 \leq \pi$, y la restricción:

$$g(x_1, x_2) = 1 - x_1^2 - x_2^2 + 0{,}1\cos\left(16\tan^{-1}(x_1 / x_2)\right). \quad (10)$$

Para su ejecución, se estableció una población de trabajo de 5 000 soluciones, 200 iteraciones, 0,35 como factor de elitismo y 10 intervalos en el histograma. La Fig. 13 muestra los resultados obtenidos. La relación de hiperárea de la

frontera de Pareto obtenida fue de 0.9999, mostrando, al igual que en el ejemplo anterior, una coincidencia casi perfecta con la frontera real.

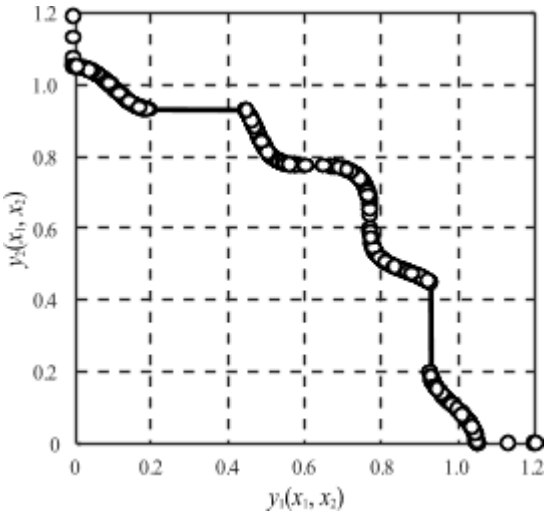

Figura 13 Frontera de Pareto obtenida para el problema MOPC4

Conclusiones

El principal resultado del presente trabajo ha sido la implementación de una biblioteca, en C++, para la optimización multiobjetivo utilizando el método de entropía cruzada. La misma se distribuirá como software libre y de código abierto, bajo la Licencia Pública General Reducida de GNU.

El funcionamiento de la biblioteca y su capacidad de integración a otros programas ha sido comprobado mediante el desarrollo de una aplicación con interfaz gráfica de usuario, para optimización multiobjetivo. Mediante la misma, se llevaron a cabo tres estudios de caso, basados en problemas de prueba para heurísticas de optimización multiobjetivo, tomados de la literatura especializada. En los tres casos, la frontera de Pareto obtenida mostró una excelente correspondencia con la frontera real, con valores de hiperárea superiores a 0,99.

Como desarrollos futuros del presente trabajo, se prevé la extensión de la biblioteca para incluir otras heurísticas no basadas en gradiente, tales como algoritmos genéticos, recocido simulado o algoritmo de hormiguero. También se prevé la implementación de facilidades de computación distribuida, que acelere el funcionamiento de los algoritmos, especialmente para problemas de alta complejidad.

Referencias

Bekker, J., & Aldrich, C. (2011). The cross-entropy method in multi-objective optimisation: An assessment. European *Journal of Operational Research*, 211(1), 112-121. doi: 10.1016/j.ejor.2010.10.028

Beruvides, G., Quiza, R., & Haber, R. E. (2016). Multiobjetive optimization based on an improved cross-entropy method. A case study of a micro-scale manufacturing process. *Information Sciences*, 334-335, 161-173. doi: 10.1016/j.ins.2015.11.040

Coello, C. A., Lamont, G. B., y Veldhuizen, D. A. (2007). *Evolutionary algorithms for solving multi-objective problems* (2nd ed.). New York: Springer.

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, 19-67. doi: 10.1007/s10479-005-5724-z

Giagkiozis, I., Purshouse, R. C., & Fleming, P. J. (2014). Generalized decomposition and cross entropy methods for many-objective optimization. *Information Sciences*, 282, 363-387. doi: 10.1016/j.ins.2014.05.045

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

Haber, R. E., Beruvides, G., Quiza, R., & Hernández, A. (2017). A simple multi-objective optimization based on the cross-entropy method. *IEEE Access*, 5(1), 22272-22281. doi: 10.1109/ACCESS.2017.2764047

Hauman, C. (2012). *The application of the cross-entropy method for multi-objective optimisation to combinatorial problems*. (Master Thesis), Stellenbosch University, Stellenbosch (South Africa).

Huband, S., P. Hingston, L. Barone, and L. While. 2006. "A review of multiobjective test problems and a scalable test problem toolkit." *IEEE Transactions on Evolutionary Computation* 10 (5): 477-506. doi: 10.1109/TEVC.2005.861417.

La Fé, I.; Beruvides, G.; Quiza, R.; Haber, R.E.; Rivas, M. (2018). Automatic selection of optimal parameters based on simple soft computing methods. A case study on micro-milling processes. *IEEE Transactions on Industrial Informatic*s, [accepted]. doi: 10.1109/TII.2018.2816971

Sebaa, K., Tlemçani, A., Bouhedda, M., & Henini, N. (2013). Multiobjective optimization using cross-entropy approach. *Journal of Optimization*, 9. doi: http://dx.doi.org/10.1155/2013/270623

Ünveren, A., & Acan, A. (2007). Multi-objective optimization with cross entropy method: Stochastic learning with clustered Pareto fronts. Paper presented at the *IEEE Congress on Evolutionary Computation (CEC 2007)*, Singapore

Xiong, N., Molina, D., Ortiz, M. L., y Herrera, F. (2015). A walk into metaheuristics for engineering optimization: Principles, methods and recent trends. *International Journal of Computational Intelligence Systems*, 8, 606-636. doi: 10.1080/18756891.2015.1046324

Yang, X. S., Koziel, S., y Leifsson, L. (2014). Computational optimization, modelling and simulation: Past, present and future. *Procedia Computer Science*, 29, 754-758. doi: 10.1016/j.procs.2014.05.067