Uso de Auto-Adaptación para Manejar Restricciones con un Algoritmo Genético

Carlos A. Coello Coello **Efrén Mezura Montes**
{ccoello,emezura}@xalapa.lania.mx
Laboratorio Nacional de Informática Avanzada
Rébsamen No. 80
Xalapa, Veracruz 91090, México

Resumen

Este artículo propone el uso de la co-evolución para adaptar los factores de penalización de una función de aptitud incorporada a un algoritmo genético utilizado para optimización numérica. El enfoque propuesto produce soluciones mejores que otras previamente reportadas por otras técnicas (basadas en algoritmos genéticos y programación matemática), a pesar de que dichas técnicas han sido calibradas usando un cuidadoso proceso de ensayo y error para resolver un problema o conjunto de problemas específicos. La técnica propuesta es fácil de implementar y apropiada para paralelización, lo cual deberá resultar una alternativa viable para mejorar su desempeño actual.

1 Introduction

La importancia de los algoritmos genéticos (AGs) como una herramienta poderosa para optimización en ingeniería ha sido ampliamente mostrado en los últimos años a través de una gran cantidad de aplicaciones [1]. Sin embargo, aún cuando los AGs han sido exitosos en muchas aplicaciones prácticas, la calidad de la solución que producen dependen no sólo de la naturaleza estocástica de la técnica, sino también de la manera en la que la función objetivo se transforma en una "función de aptitud" que pueda "conducir" al AG a la región deseada en el espacio de búsqueda.

Uno de los problemas principales al utilizar AGs en aplicaciones prácticas es cómo implementar la función de aptitud. Una estimación comparativa de qué tan buena es una solución llega a ser suficiente en la mayoría de los casos, pero si se están manejando problemas con restricciones, se debe encontrar una manera de estimar también qué tan cerca está una solución no factible de la zona factible. Esta no es una tarea fácil puesto que la mayoría de los problemas del mundo real tienen un gran volumen de restricciones lineales y no-lineales. A la fecha, se han propuesto muchos enfoques en la literatura especializada para manejar toda esta gama de restricciones [2; 3; 4]. En dicha literatura, la función de penalización parece ser la técnica más popular para problemas de ingeniería, pero las dificultades intrínsecas para definir buenas funciones de penalización hacen aún más difícil el proceso de optimización usando un AG [5].

En este artículo, se propone el uso de una técnica basada en el concepto de co-evolución. Dicha técnica crea dos poblaciones que interactúan entre sí de manera que una población evoluciona los factores de penalización a ser utilizados por la función de aptitud de la otra población, la cual es responsable de optimizar la función objetivo. El enfoque es demostrado con varios problemas de optimización con un solo objetivo y con restricciones de desigualdad lineales y no-lineales y los resultados se comparan con los generados por otros enfoques (basados en AGs y programación matemática) reportados en la literatura especializada.

2 Auto-Adaptación

Michalewicz et al. [2] han reconocido la importancia de usar penalizaciones adaptativas en la optimización evolutiva y han considerado este enfoque como una dirección muy promisoria de investigación en optimización evolutiva. La técnica propuesta en este artículo pretende implementar esta idea usando el concepto de co-evolución, bajo el cual dos (o más) poblaciones evolucionan de manera concurrente, en un proceso durante el cual intercambian información entre sí. Paredis [6] ha utilizado previamente co-evolución para satisfacción de restricciones (en problemas de optimización combinatoria), aunque su técnica no ha sido extendida a optimización numérica. En la propuesta de Paredis, se hace evolucionar una población de soluciones potenciales paralelamente con una población de restricciones, de manera que se pudiesen mezclar los individuos factibles con los no factibles durante el proceso evolutivo.

El enfoque propuesto en este artículo utiliza una función de penalización convencional [1; 5] en lugar de intentar manejar restricciones en una manera completamente diferente (ver por ejemplo [3; 2]). La razón para ello es que las funciones de penalización son aun el enfoque más popular para manejar restricciones en aplicaciones

prácticas [2; 3], mientras que los enfoques más recientes han sido normalmente usados sólo para manejar restricciones en problemas muy específicos (y generalmente fuera del contexto de las aplicaciones del mundo real).

El problema que queremos resolver es:

$$\text{Optimizar} \quad f(\mathbf{X}) \tag{1}$$

Sujeto a :

$$g_i(\mathbf{X}) \leq \mathbf{0} \quad \mathbf{i = 1, \ldots, p} \tag{2}$$

Sólo se consideran restricciones de desigualdad en este artículo, puesto que las funciones de penalización no son muy apropiadas para manejar restricciones de igualdad como restricciones duras, debido a que resulta muy difícil en general utilizar una técnica estocástica para resolver de manera exacta (hasta el último decimal) una restricción. El procedimiento común es relajar las restricciones de igualdad usando:

$$|h_j(\mathbf{X})| - \epsilon \leq \mathbf{0} \tag{3}$$

donde ϵ es la tolerancia permitida (un valor muy pequeño).

En aplicaciones previas, varios investigadores [3] han determinado que una función de penalización que incluye información tanto del número de restricciones violadas como del grado de violación de cada una de ellas, resulta muy efectiva en optimización numérica.

La expresión utilizada para procesar el valor de aptitud de un individuo para los propósitos de este artículo (suponiendo maximización) es la siguiente:

$$\text{aptitud}_i = f_i(\mathbf{X}) - (\mathbf{coef} \times \mathbf{w_1} + \mathbf{viol} \times \mathbf{w_2}) \tag{4}$$

donde $f_i(\mathbf{X})$ es el valor de la función objetivo para el conjunto de valores variables codificados en el cromosoma i; w_1 y w_2 son dos factores de penalización (considerados como enteros en este artículo); $coef$ es la suma de la violación de las restricciones:

$$coef = \sum_{i=1}^{p} g_i(\mathbf{X}) \quad \forall \mathbf{g_i}(\mathbf{X}) > \mathbf{0} \tag{5}$$

y $viol$ es un factor entero, inicializado a cero e incrementado en uno por cada restricción del problema que es violada, sin importar la cantidad de violación.

En nuestro enfoque, la penalización es separada en dos valores ($coef$ y $viol$), a fin de que el AG tenga suficiente información para poder estimar la lejanía de una solución no factible de la frontera de factibilidad. Esto es en conformidad con las sugerencias de Richardson et al. [5].

Nuestra técnica usa 2 poblaciones diferentes, a las que llamaremos $P1$ y $P2$, cada una de las cuales tiene un tamaño correspondiente, a los que llamaremos $M1$ y $M2$. La segunda de esas poblaciones ($P2$) codifica el conjunto de combinaciones de pesos (w_1 y w_2), el cual será usado para calcular el valor de aptitud de los individuos en $P1$ (es decir, $P2$ contiene los factores de penalización que serán utilizados en la función de aptitud de $P1$). La idea es utilizar una población para evolucionar soluciones (como en un AG convencional) y otra para evolucionar los factores de penalización w_1 y w_2.

Puede verse una representación gráfica de este enfoque en la figura 1. Se puede observar que por cada individuo A_j en $P2$ hay una instancia de $P1$. Sin embargo, la población $P1$ es reutilizada para cada nuevo elemento A_j procesado desde $P2$. Cada individuo A_j ($1 \leq j \leq M2$) en $P2$ es decodificado y la combinación de pesos producida (es decir, los factores de penalización) se utiliza para evolucionar $P1$ durante un cierto número ($Gmax1$) de generaciones. La aptitud de cada individuo B_k ($1 \leq k \leq M1$) es procesada utilizando la ecuación (4), manteniendo los factores de penalización constantes para cada individuo en la instancia de $P1$ correspondiente al individuo A_j que esté siendo procesado.

Después de evolucionar cada $P1$ correspondiente a cada A_j en $P2$ (hay sólo una instancia de $P1$ por cada individuo en $P2$) se determina la mejor aptitud promedio producida utilizando:

$$\text{aptitud\_promedio}_j = \sum_{i=1}^{M1} \left(\frac{\text{aptitud}_i}{conteo\_factible} \right) + conteo\_factible \quad \forall \mathbf{X} \in \mathcal{F} \tag{6}$$

En la ecuación (6), se agregan las aptitudes de todas las soluciones factibles en $P1$ y se obtiene un promedio de ellas (el valor entero $conteo\_factible$ es un contador que indica cuántas soluciones factibles fueron encontradas en la población).

Debe hacerse notar que aunque la sumatoria cubre todo el rango de individuos en $P1$, sólo aquellos que sean factibles serán considerados para el cálculo de la aptitud promedio. La razón para ello es que si no se excluyen las soluciones no factibles de este proceso, el mecanismo de selección del AG podría llevar a la población a regiones en el espacio de búsqueda donde hay soluciones con pesos muy pequeños (w_1 y w_2), las cuales tendrían valores altos de aptitud, pero serían no factibles. Esto se debe a que valores pequeños de w_1 y w_2 pueden producir penalizaciones que no sean suficientemente grandes como para escalar adecuadamente el valor de la función objetivo. Nótese también el uso de $conteo\_factible$ para evitar el estancamiento en ciertas regiones en las cuales sólo unos cuantos individuos tendrían una buena aptitud o serían factibles. Agregando esta cantidad a la aptitud promedio de los individuos factibles en la población ayudará a que el AG se mueva a regiones en las cuales existan no sólo unas pocas soluciones factibles con buenos valores de aptitud, sino muchas de ellas.

En la práctica puede ser necesario aplicar un factor de escala para el promedio de la aptitud antes de agregar $conteo\_factible$, para evitar que el AG quede atrapado

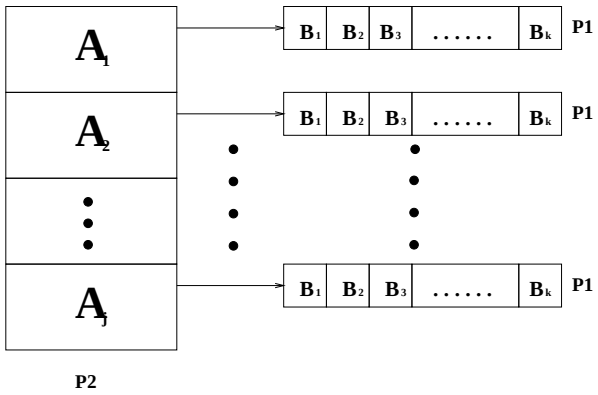

Figura 1: Representación gráfica del enfoque basado en co-evolución propuesto en este artículo.

en un óptimo local. De hecho, en los experimentos reportados en este artículo tuvimos que utilizar un factor de escala. Sin embargo, estos factores de escala no son muy difíciles de calcular debido a que suponemos tamaños de población constantes (ya que el tamaño debe ser definido antes de la ejecucuón del AG) y el rango de valores de aptitud puede ser fácilmente obtenido en cada generación porque se sabe cuáles son los valores de aptitud máximos y mínimos en la población en cada generación.

La ecuación (6) puede, obviamente, limitar la aplicación del enfoque propuesto a sólo aquellas situaciones en las cuales haya al menos una solución totalmente factible en la primera generación, pero debido a que se consideran varias combinaciones de pesos por cada instancia de $P1$, podemos lidiar con situaciones de este tipo con nuestra técnica. De hecho, en nuestros experimentos se presentó esta situación (la ausencia de soluciones factibles para una instancia de $P1$) y a pesar de eso nuestra técnica fue capaz de converger hacia el óptimo del problema.

El proceso arriba indicado se repite hasta que todos los individuos en $P2$ cuenten con un valor de aptitud (la mejor aptitud promedio de su instancia de $P1$ correspondiente). Posteriormente, $P2$ evolucióna una generación usando operadores genéticos convencionales (cruza y mutación) y la nueva $P2$ producida se utiliza para comenzar el mismo proceso una vez más. Es importante remarcar que la interacción entre $P1$ y $P2$ introduce diversidad en ambas poblaciones, la cual impide al AG converger a un óptimo local.

3 Ejemplos

Usaremos algunos ejemplos tomados de la literatura especializada para ilustrar el funcionamiento de nuestra técnica, comparando en cada caso nuestros resultados con los reportados por otros enfoques.

3.1 Ejemplo 1 : Diseño de un recipiente de presión

Un recipiente de presión está cubierto en ambos lados por tapas hemisféricas, tal y como se observa en la figura 2. El objetivo es minimizar el costo total del recipiente, incluyendo el costo del material, su fabricación y el proceso de soldado. Existen cuatro variables de diseño: $x_1(T_s)$ (espesor de la superficie), $x_2(T_h)$ (espesor de la tapa), $x_3(R)$ (radio interno) y $x_4(L)$ (longitud de la sección cilíndrica del recipiente, sin incluir la tapa). T_s y T_h son enteros múltiplos de 0.0625 pulgadas, y se refieren a los espesores disponibles de placas de acero. Finalmente, $R1$ y L son valores continuos. Usando la misma notación dada por Kannan y Kramer [7], el problema puede plantearse como sigue:

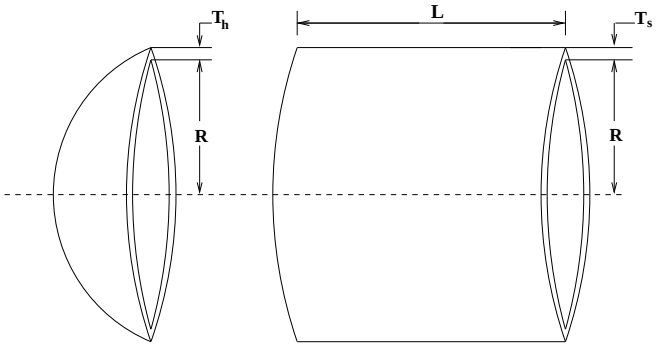

Figura 2: Sección central y final del recipiente de presión usado para el primer ejemplo.

Minimizar :

$$F(\mathbf{X}) = \mathbf{0.6224 x_1 x_3 x_4 + 1.7781 x_2 x_3^2} + \\ +3.1661 x_1^2 x_4 + 19.84 x_1^2 x_3 \qquad (7)$$

Sujeto a:

$$g_1(\mathbf{X}) = -\mathbf{x_1} + \mathbf{0.0193 x_3} \leq \mathbf{0} \qquad (8)$$

$$g_2(\mathbf{X}) = -\mathbf{x_2} + \mathbf{0.00954 x_3} \leq \mathbf{0} \qquad (9)$$

$$g_3(\mathbf{X}) = -\pi \mathbf{x_3^2 x_4} - \frac{\mathbf{4}}{\mathbf{3}} \pi \mathbf{x_3^3} + \mathbf{1,296,000} \leq \mathbf{0} \qquad (10)$$

$$g_4(\mathbf{X}) = \mathbf{x_4} - \mathbf{240} \leq \mathbf{0} \qquad (11)$$

Parámetro	Valor
M_1	60
M_2	30
$Gmax_1$	25
$Gmax_2$	20

Tabla 1: Parámetros del AG utilizado para resolver todos los ejemplos.

3.2 Ejemplo 2 : Problema de Optimización no lineal de Himmelblau

Este problema fue propuesto originalmente por Himmelblau [8], y fue seleccionado para probar el enfoque propuesto aqui ya que ha sido usado antes como ejemplo de comparación por varias otras técnicas basadas en AGs que usan penalizaciones [9]. En este problema hay cinco variables de diseño (x_1, x_2, x_3, x_4, x_5), 6 restricciones no lineales de desigualdad y 10 condiciones de frontera. El problema puede expresarse como sigue:

$$\text{Minimizar } f(\mathbf{X}) = \mathbf{5.3578547x_3^2} + \\ +0.8356891x_1x_5 + 37.293239x_1 - 40792.141 \quad (12)$$

Sujeto a:

$$g_1(\mathbf{X}) = \mathbf{85.334407 + 0.0056858x_2x_5} + \\ 0.00026x_1x_4 - 0.0022053x_3x_5 \quad (13)$$

$$g_2(\mathbf{X}) = \mathbf{80.51249 + 0.0071317x_2x_5} + \\ 0.0029955x_1x_2 + 0.0021813x_3^2 \quad (14)$$

$$g_3(\mathbf{X}) = \mathbf{9.300961 + 0.0047026x_3x_5} + \\ 0.0012547x_1x_3 + 0.0019085x_3x_4 \quad (15)$$

$$0 \leq g_1(\mathbf{X}) \leq \mathbf{92} \quad (16)$$

$$90 \leq g_2(\mathbf{X}) \leq \mathbf{110} \quad (17)$$

$$20 \leq g_3(\mathbf{X}) \leq \mathbf{25} \quad (18)$$

$$78 \leq x_1 \leq 102 \quad (19)$$

$$33 \leq x_2 \leq 45 \quad (20)$$

$$27 \leq x_3 \leq 45 \quad (21)$$

$$27 \leq x_4 \leq 45 \quad (22)$$

$$27 \leq x_5 \leq 45 \quad (23)$$

4 Comparación de Resultados

Para hacer una comparación justa, los siguientes ejemplos fueron resueltos utilizando el mismo conjunto de parámetros, los cuales se muestran en la Tabla 1. Nuestro AG usó representación de punto fijo [10]. En trabajos previos, hemos mostrado que esta representación resulta mejor que su contraparte binaria en problemas de optimización numérica [10]. Utilizamos también cruza uniforme con una probabilidad de 0.8 y mutación no uniforme [11] con un valor inicial de mutación de 0.1 que permite un alto comportamiento exploratorio del AG en las primeras generaciones, y una búsqueda más local en las últimas generaciones. La mejor solución fue retenida en todos los casos (elitismo).

4.1 Ejemplo 1

Este problema fue resuelto antes por Deb [12] usando GeneAS (*Genetic Adaptive Search*), por Kannan y Kramer usando Multiplicadores de Lagrange [7], por Fu et al. [13] usando Programación no lineal Entera-Discreta-Continua y por Cao y Wu [14] usando Programació Evolutiva. La comparación de estos resultados contra los nuestros se muestra en la Tabla 2. La solución reportada en esta tabla es la mejor producida después de 11 corridas, en las que se usó el siguiente rango de valores para las variables de diseño y los pesos de la función de penalización: $1 \leq x_1 \leq 99$, $1 \leq x_2 \leq 99$, $10.0000 \leq x_3 \leq 200.0000$, $10.0000 \leq x_4 \leq 200.0000$, $1 \leq w_1 \leq 999$, y $1 \leq w_2 \leq 999$. Los valores de x_1 y x_2 se consideraron como enteros múltiplos de 0.0625 y los valores de x_3 y x_4 se consideraron con una precisión de 4 cifras después del punto decimal. La media de las 11 corridas fue $f(\mathbf{X}) = \mathbf{6293.84323196}$, con una desviación estándar de 7.41328537. La peor solución hallada fue $f(\mathbf{X}) = \mathbf{6308.14965192}$, que es mejor que cualquiera de las soluciones previamente reportadas en la literatura para este problema. La solución en la mediana fue $f(\mathbf{X}) = \mathbf{6290.01873568}$ (que corresponde a $x_1 = 0.8125$, $x_2 = 0.4372$, $x_3 = 40.3302$ y $x_4 = 200.0000$), lo cual es aproximadamente 2% mejor que la mejor solución previamente reportada en la literatura.

4.2 Ejemplo 2

Este problema fue propuesto originalmente por Himmelblau [8], quien lo resolvió utilizando el método del Gradiente Reducido Generalizado (GRG por sus siglas en inglés). Gen y Cheng [9] resolvieron este problema usando un algoritmo genético basado en referencias locales y globales. Homaifar, Qi y Lai [15] resolvieron este problema usando un AG con un tamaño de población de 400 que usaba una función de penalización estática. La comparación de nuestros resultados con los de estas otras técnicas se muestran en la Tabla 3. La solución reportada en esta tabla es la mejor producida después de 11 corridas, en las que se usó el siguiente rango de valores para las

Diseño	Mejor solución				
Variables	Este trabajo	GeneAS [12]	Kannan [7]	Fu [13]	Cao [14]
$x_1(T_s)$	0.8125	0.9375	1.125	1.125	1.000
$x_2(T_h)$	0.4375	0.5000	0.625	0.625	0.625
$x_3(R)$	40.3239	48.3290	58.291	48.3807	51.1958
$x_4(L)$	200.0000	112.6790	43.690	111.7449	90.7821
$g_1(\mathbf{X})$	-0.034324	-0.004750	0.000016	-0.191252	-0.011921
$g_2(\mathbf{X})$	-0.052847	-0.038941	-0.068904	-0.163448	-0.136592
$g_3(\mathbf{X})$	-27.105845	-3652.876838	-21.220104	-72.970137	-13584.583968
$g_4(\mathbf{X})$	-40.00000	-127.321000	-196.310000	-128.25510	-149.2179
$f(\mathbf{X})$	**6288.7445**	**6410.3811**	**7198.0428**	**8049.3411**	**7108.6160**

Tabla 2: Comparación de los resultados del primer ejemplo (optimización de un recipiente de presión).

variables de diseño y los pesos de la función de penalización: $78.0000 \leq x_1 \leq 102.0000$, $33.0000 \leq x_2 \leq 45.0000$, $27.0000 \leq x_3 \leq 45.0000$, $27.0000 \leq x_4 \leq 45.0000$, $27.0000 \leq x_5 \leq 45.0000$, $1 \leq w_1 \leq 999$, y $1 \leq w_2 \leq 999$. Los valores de x_1 a x_5 fueron considerados con una precisión de 4 lugares después del punto decimal. La media de las 11 corridas fue $f(\mathbf{X}) = \mathbf{-30984.24070309}$, con una desviación estándar de 73.63353661. La peor solución encontrada fue $f(\mathbf{X}) = \mathbf{-30792.4077377525}$, que resulta mejor que cualquiera de las soluciones previamente reportadas en la literatura. La solución en la mediana fue $f(\mathbf{X}) = \mathbf{-31017.21369099}$ (correspondiente a $x_1 = 78.010$, $x_2 = 33.030$, $x_3 = 27.119$, $x_4 = 45.000$, y $x_5 = 44.872$).

5 Discusión

A pesar del hecho de que la técnica aquí propuesta requiere más evaluaciones de la función de aptitud que correr un AG con una sola población, se puede argumentar que en la práctica nuestra técnica puede resultar más eficiente, pues evita el ajuste tradicional de parámetros que suele acompañar a todo problema de optimización en el que se usa un AG.

El conjunto de parámetros propuestos al inicio de este artículo para las 2 poblaciones se derivaron en base a una serie de experimentos. Inicialmente, se determinó que en la mayor parte de los casos un tamaño de población bastante pequeño para $P2$ (≤ 20 cromosomas) era suficiente para encontrar soluciones razonables (dentro del 5% de la mejor solución previamente reportada en la literatura), pero el tamaño de $P1$ dependía mucho más en la naturaleza del problema, aunque en los ejemplos reportados aquí resultó suficiente usar valores pequeños (entre 30 y 60 individuos), aunque optamos por usar valores más grandes para hacer más justas nuestras comparaciones con otras técnicas. Similarmente, el efecto del máximo número de generaciones en los resultados parecía ser más significativo para $P1$ que para $P2$. Esto no debe sorprendernos, ya que $P1$ es realmente la población responsable de la optimización. Es interesante mencionar también que en nuestros experimentos descubrimos que el incrementar el número máximo de generaciones para $P1$ normalmente mejoraba la calidad de la solución, pero hasta cierto umbral después del cual los incrementos no tenían ya ningún efecto significativo. Por otra parte, el incrementar el número máximo de generaciones para $P2$ no resultó normalmente muy benéfico (es decir, el resultado obtenido, si bien no fue peor que los anteriores, tampoco mejoró al aumentar el número máximo de generaciones), por lo que preferimos usar valores más pequeños para $Gmax2$ que para $Gmax1$.

6 Conclusiones y Trabajo Futuro

En este artículo hemos propuesto una técnica co-evolutiva para el ajuste automático de una función de penalización en el contexto de optimización numérica con algoritmos genéticos. La técnica propuesta fue ilustrada con algunos ejemplos tomados de la literatura especializada, produciendo en todos los casos mejores resultados a los reportados anteriormente. El desempeño de la nueva técnica permanece, sin embargo, como su talón de Aquiles, ya que el número total de evaluaciones de la función de aptitud es alto $(900,000)$, por lo que es deseable paralelizar esta técnica a fin de hacerla más eficiente. También contemplamos extender este estudio a un número mayor de funciones de prueba (usamos varias más de las incluidas en este artículo, aunque no pudimos proporcionar los resultados correspondientes debido a limitaciones obvias de espacio, pero el lector interesado puede consultarlas en [16]) y consideramos comparar nuestra técnica contra un explorador local (*hill climber*) que opere únicamente en torno a los factores de penalizaciones. Los primeros experimentos a este respecto parecen confirmar que la co-evolución es una técnica superior.

Reconocimientos

Los autores agradecen el apoyo recibido por el CONACyT a través del proyecto de instalación número I-29870 A.

Diseño	Mejor solución			
Variables	Este trabajo	Gen [9]	Homaifar [15]	GRG [8]
x_1	78.0495	81.4900	78.0000	78.6200
x_2	33.0070	34.0900	33.0000	33.4400
x_3	27.0810	31.2400	29.9950	31.0700
x_4	45.0000	42.2000	45.0000	44.1800
x_5	44.9400	34.3700	36.7760	35.2200
$g_1(\mathbf{X})$	91.997635	90.522543	90.714681	90.520761
$g_2(\mathbf{X})$	100.407857	99.318806	98.840511	98.892933
$g_3(\mathbf{X})$	20.001911	20.060410	19.999935	20.131578
$f(\mathbf{X})$	**−31020.859**	**−30183.576**	**−30665.609**	**−30373.949**

Tabla 3: Comparación de resultados para el segundo ejemplo (la función de Himmelblau).

Referencias

[1] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing Co., Reading, Massachusetts, 1989.

[2] Zbigniew Michalewicz and Marc Schoenauer. Evolutionary Algorithms for Constrained Parameter Optimization Problems. *Evolutionary Computation*, 4(1):1–32, 1996.

[3] Dipankar Dasgupta and Zbigniew Michalewicz, editors. *Evolutionary Algorithms in Engineering Applications.* Springer-Verlag, Berlin, 1997.

[4] Carlos A. Coello Coello. A Survey of Constraint Handling Techniques used with Evolutionary Algorithms. Technical Report Lania-RI-99-04, Laboratorio Nacional de Informática Avanzada, Xalapa, Veracruz, México, 1999. (Disponible en: http://www.lania.mx/~ccoello/constraint.html).

[5] Jon T. Richardson, Mark R. Palmer, Gunar Liepins, and Mike Hilliard. Some guidelines for genetic algorithms with penalty functions. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 191–197, George Mason University, 1989. Morgan Kaufmann Publishers.

[6] J. Paredis. Co-evolutionary Constraint Satisfaction. In *Proceedings of the 3rd Conference on Parallel Problem Solving from Nature*, pages 46–55, New York, 1994. Springer Verlag.

[7] B. K. Kannan and S.Ñ. Kramer. An Augmented Lagrange Multiplier Based Method for Mixed Integer Discrete Continuous Optimization and Its Applications to Mechanical Design. *Journal of Mechanical Design. Transactions of the ASME*, 116:318–320, 1994.

[8] David M. Himmelblau. *Applied Nonlinear Programming.* McGraw-Hill, New York, 1972.

[9] Mitsuo Gen and Runwei Cheng. *Genetic Algorithms & Engineering Design.* John Wiley & Sons, Inc, New York, 1997.

[10] Carlos A. Coello Coello, Filiberto Santos Hernández, and Francisco Alonso Farrera. Optimal design of reinforced concrete beams using genetic algorithms. *Expert Systems with Applications : An International Journal*, 12(1):101–108, January 1997.

[11] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, second edition, 1992.

[12] Kalyanmoy Deb. GeneAS: A Robust Optimal Design Technique for Mechanical Component Design. In Dipankar Dasgupta and Zbigniew Michalewicz, editors, *Evolutionary Algorithms in Engineering Applications*, pages 497–514. Springer-Verlag, Berlin, 1997.

[13] J. F. Fu, R. G. Fenton, and W. L. Cleghorn. A mixed integer-discrete-continuous programming method and its application to engineering design optimization. *Engineering Optimization*, 17(3):263–280, 1991.

[14] Y. J. Cao and Q. H. Wu. Mechanical Design Optimization by Mixed-Variable Evolutionary Programming. In Thomas Bäck, Zbigniew Michalewicz, and Xin Yao, editors, *Proceedings of the 1997 International Conference on Evolutionary Computation*, pages 443–446, Indianapolis, Indiana, 1997. IEEE.

[15] A. Homaifar, S. H. Y. Lai, and X. Qi. Constrained Optimization via Genetic Algorithms. *Simulation*, 62(4):242–254, 1994.

[16] Carlos A. Coello Coello. Use of a Self-Adaptive Penalty Approach for Engineering Optimization Problems. *Computers in Industry*, 1999. (Accepted for publication).