Una propuesta de evolución flexible en el diseño de algoritmos evolutivos

G. Winter, B. Galván, S. Alonso y B. González

Resumen—**En este trabajo se presenta, en el campo de la Computación Evolutiva, una metodología que pretende ser un primer paso hacia un nuevo enfoque a la hora de afrontar la resolución de problemas de optimización, al que hemos denominado Evolución Flexible (EF). El procedimiento propuesto posee la característica de no tener que elegir a priori entre los posibles métodos de muestreo existentes, siendo el algoritmo implementado quien decide cuál usar en cada etapa de la evolución en función del aprendizaje conseguido durante la propia resolución del problema de optimización. Así los operadores están sujetos a un cierto juego de cooperación-competición entre sí, mientras el algoritmo como agente inteligente decide cuál y qué usar de éstos (mediante asignaciones numéricas a parámetros asociados a los operadores) para favorecer la eficiencia computacional del proceso global de optimización. Las principales características del algoritmo son las de poseer una Estructura dinámica de Operadores (EDO), la de Ampliar el Código Genético de cada solución (ACG) y la inclusión de Mecanismos de Control Probabilístico (MCP) en el proceso evolutivo. Estas actuaciones deben estar coordinadas de manera que produzcan un efecto cooperativo-competitivo, aumentando la eficiencia y robustez en la resolución del problema. Como ejemplo, la EF ha sido probada en un caso matemático bien conocido y en un problema de interés medio ambiental.**

Palabras clave—**Evolución Flexible (EF), Algoritmos Evolutivos, EDO, ACG, MCP.**

I. INTRODUCCIÓN

EN la actualidad, cuando un investigador quiere optimizar la solución a un determinado problema, debe decidir qué método o métodos va a emplear para afrontarlo. Esta elección suele estar basada en su formación y experiencia en la resolución de problemas de tipo similar y en los métodos utilizados con anterioridad para resolverlos. Resulta evidente que decidirse por un determinado método implica la aceptación de los inconvenientes que éste tenga, y siempre existirá una especie de "riesgo calculado" de no obtener resultados tan satisfactorios como se requieren. Obviamente, al escoger uno o varios métodos, también estamos renunciando a las características favorables que el resto de procedimientos podrían aportar para la resolución del problema tratado. Clásicamente, esta elección, así como la de los parámetros necesarios para la optimización, es el punto de partida obligatorio a la hora de resolver un problema, y los intentos de aumentar la probabilidad de éxito en la elección se basan fundamentalmente en tres tipos de actuaciones: Adaptación de Parámetros, métodos Libres de Parámetros y Reparación de soluciones. Estas acciones presentan buenos resultados, pero normalmente están limitadas a los métodos particulares a los que son aplicados. Como ejemplo de esta línea de trabajo se pueden consultar los trabajos de Toshine et al [1] ó Kee et al [2], donde se ajustan parámetros para mejorar los resultados.

En un intento claro de solventar estas limitaciones, se ha apostado por un nuevo enfoque para afrontar la resolución de un problema determinado sin necesidad de renunciar a ninguna de las características beneficiosas que presentan los diferentes métodos de muestreo. El algoritmo propuesto está basado en las nuevas posibilidades que abren los avances en las Ciencias de la Computación, Informática e Inteligencia Artificial en cuanto al potencial real de diseñar estructuras algorítmicas capaces de escoger lo mejor de cada método en cada momento particular de la optimización, liberando así al investigador de la, hasta hoy, obligada elección previa del método y sus parámetros. De esta forma, **un algoritmo flexible** nace con la premisa de que la mejor alternativa para diseñar estructuras algorítmicas evolutivas, en el momento de escoger qué método de optimización se debe emplear para un problema determinado, es **la de maximizar oportunidades para los diferentes métodos evolutivos**. Esto debe ser entendido en el contexto de la eficiencia global de la resolución del proceso evolutivo, puesto que en sus diferentes etapas los métodos y operadores asociados serán más o menos eficaces en la búsqueda de la solución o soluciones óptimas.

Los principios básicos que deben requerir son:

1. Presentar una **adaptabilidad total**, tanto en sus parámetros como del propio algoritmo.
2. **Ser independiente** tanto **respecto a los operadores** (permitiendo la inclusión de cualquier operador, aunque no esté clara su capacidad de producir beneficios en el proceso), como **a los parámetros** (a los que el algoritmo deberá asignar valores).
3. Considerar que **todas las decisiones son posibles**, concediendo igual probabilidad de elección a todas las variantes a priori, y, tras aprender durante el proceso cuáles ayudan en la optimización, dando más peso o chance a las decisiones más favorables.

II. CONCEPTOS BÁSICOS DE LA EVOLUCIÓN FLEXIBLE

Teniendo en cuenta el cambio de planteamiento en el enfoque de los Algoritmos Evolutivos al utilizar todos

Instituto de Sistemas Inteligentes y Aplicaciones Numéricas en Ingeniería, división CEANI. Edf. Central del Parque Científico y Tecnológico, Universidad de Las Palmas de Gran Canaria, 2ªPlanta, 35017 Las Palmas de Gran Canaria, gabw@step.es, bgalvan@step.es, oraami@yahoo.es, begolan@yahoo.es.

los métodos que se quieran incluir para el muestreo, es necesario introducir una serie de conceptos que ayudarán en gran manera a comprender mejor la Evolución Flexible. Es muy importante resaltar que en el caso de la EF, no sólo se produce adaptación de parámetros o de operadores, como otros autores ya han tratado con cierta profundidad, sino que también el algoritmo mismo se adapta al problema planteado, lo que introduce la verdadera novedad del método.

Si el algoritmo es capaz de decidir los valores de los parámetros a utilizar así como qué operadores se van a utilizar y el momento apropiado para ello, es necesario implementar una serie de mecanismos capaces de tomar esas decisiones. La forma en que la Naturaleza permite tomar o corregir decisiones de manera más eficaz es utilizar una dinámica de poblaciones, aplicar Selección y memorizar después, de alguna forma, los eventos que han sucedido para poder aprender de ellos. Por esto, tradicionalmente, los investigadores en el área de la Evolución Computacional, la Robótica y la IA han ingeniado diversas formas de implementar y usar ciertos tipos de memoria en sus algoritmos. Por lo tanto, a los principios básicos enunciados anteriormente de la EF se debe añadir otro:

4. El algoritmo **debe tener memoria**.

Para implementar estos puntos se han usado una Estructura Dinámica de Operadores, Mecanismos de Control Probabilístico además de Ampliar el Código Genético de las soluciones con información relevante, según se explica en los apartados siguientes y tal como ya han sido empleados en un algoritmo de EF por Winter et al en [3].

A. La Estructura Dinámica de Operadores (EDO).

Los actuales algoritmos en general no permiten variar su propia estructura de operadores durante la optimización, es decir, poseen una estructura fija. En arras de las últimas investigaciones, como la realizada por Edmons[4] en la que sólo los operadores tenían una implementación totalmente variable, se ha visto la necesidad de incluir en los algoritmos evolutivos algún tipo de estructura flexible, que ayude de manera notable al proceso de optimización. Es por esta razón que se ha considerado necesario la inclusión de un mecanismo que permita una autoadaptación de la secuencia de operadores dependiendo de las condiciones que se vayan presentando a lo largo del proceso de optimización, y que ha sido denominado **Estructura Dinámica de Operadores**. Por **EDO** se entiende una estructura algorítmica que favorezca la elección de un operador u otro durante cada etapa.

La forma más simple de diseñar un algoritmo flexible es subdividir los operadores en **dos** grupos ó **clases**: los de **Selección** y los de **Muestreo** (que incluye tanto los operadores de *Mutación* como los de *Cruce*). En ambos casos, se incluyen todo tipo de operadores, aún cuando no esté clara si su actuación en el proceso va a favorecer o no la obtención de buenos resultados. Debe ser el algoritmo el que decida, partiendo de que todos deben ser usados al menos una vez, qué operadores favorecer o no sobre el resto, teniendo en cuenta los resultados que se vayan obteniendo. Se ha implementado una subrutina que funciona como base de datos incluyendo todos los operadores y que ha sido denominado **'*motor de muestreo*'**. Controla la utilización de los operadores, permitiendo usar sólo uno de ellos para cada variable en cada ejecución.

Las clases están diseñadas tratando de mantener en todo momento un equilibrio entre la exploración y la explotación tradicionalmente buscadas en los algoritmos evolutivos, que aquí viene dada considerando dos características adicionales en los operadores: La Naturaleza y el Rango. La **Naturaleza** de un operador distingue a uno de otro y está contenida en su definición, marcando la forma en que el operador se comporta y en qué zonas va a actuar (su sesgo). Por su parte, el **Rango** se introduce para identificar el orden de magnitud de las posibles variaciones al muestrear alrededor de los valores actuales. De esta forma, dos operadores puede tener igual naturaleza pero distinto rango. Es muy importante incluir un número adecuado de operadores con igual naturaleza y diferente rango en cada clase, con el fin de obtener el equilibrio antes mencionado.

B. Ampliando el Código Genético de las soluciones (ACG).

Agrandar el código genético de las soluciones puede hacerse añadiendo información relevante al mismo, y haciéndola evolucionar con el individuo a lo largo del proceso de optimización y siguiendo las reglas del juego. De esta forma, la información se mantiene (o es memorizada) mientras el individuo sobrevive, y desaparece si no es así. En la EF, se guardan los métodos de Selección y Muestreo usados para obtener cada variable en el código de cada individuo, siendo ésta la clave que permite usar la EDO con los mecanismos de control Probabilístico que serán comentados a continuación.

C. Los Mecanismos de Control Probabilístico (MCP).

Los MCP se encargarán de dos cuestiones principalmente: decidirán los operadores que serán usados y corregirán las decisiones incorrectas, ambas actuaciones para cada variable a optimizar en cada paso del proceso. Por supuesto, existe una gran variedad de métodos que pueden ser usados para controlar un proceso en la EF, tales como Sistemas Expertos ó Lógica Borrosa, pero en esta primera aproximación se ha escogido un mecanismo muy sencillo. Al usar ACG para guardar los métodos de Muestreo y Selección usados para obtener cada variable, la regla implícita de los algoritmos evolutivos de 'supervivencia de los más adaptados al medio ' causará que, muy probablemente las mejores decisiones serán almacenadas en el código extendido de los mejores individuos. Posiblemente, la razón de esto es que los mejores individuos se obtienen cuando los métodos más adecuados en cada caso son escogidos, ya que al tratar con elitismo, los individuos elitistas memorizarán las mejores decisiones de operadores tomadas por el MCP, y no el resto.

Por supuesto, la toma de decisiones se ha de hacer de forma que se dé la oportunidad de explorar a todos los posibles operadores, como ya se ha comentado, pero favoreciendo el uso de aquellos memorizados por el ACG de los individuos elitistas. En este caso, utilizamos una simple decisión probabilística con una frontera denominada 'alfa', que actúa como sigue. Primero, se escoge un número aleatorio uniforme entre 0 y 1, que se compara con el valor preasignado de alfa(en este caso 0.6). Si el valor hallado es mayor que alfa se usará cualquier método, todos con igual probabilidad en la misma clase; por el contrario, si es menor o igual, se usará el método almacenado en uno de los individuos elitistas.

III. EL MOTOR DE MUESTREO DE LA EF.

Debido a la necesaria 'elasticidad' que el algoritmo de la EF debe poseer, su implementación se ha hecho de forma que se puedan coordinar las diversas actuaciones que se habrán de efectuar. Esto se ha conseguido dividiendo el algoritmo en varios 'motores', que son en realidad subrutinas que marcan los pasos a seguir dependiendo de lo se quiera hacer. El motor principal del método es el denominado 'Motor de Muestreo', que incluye los métodos de muestreo que se van a utilizar, tanto los de Cruce (Aritmético, Geométrico, Heurístico, etc) como los de Mutación (Uniforme, no Uniforme, Gaussiano, etc), casi todos ellos detallados en el artículo de Michalewicz et al [5] y el informe de la red INGENET de Bäck et al [6]. Además, se han desarrollado nuevas variaciones de muchos de los operadores conocidos con el fin de explorar y/o explotar más convenientemente el espacio de búsqueda. El motor actual contiene 39 métodos de muestreo, gran parte de los cuales se detallan a continuación dividiéndolos en sus dos clases.

Sean

U(0,1) = Número aleatorio uniforme entre [0 , 1]

sign = variable real con dos posibles valores: +1.0 ó -1.0

$\vec{x}^{old1}$ = mejor valor hallado

$\vec{x}^{old2}$ = uno de los mejores valores , escogido al azar

$\vec{x}^{old}$ = puede ser $\vec{x}^{old1}$ ó $\vec{x}^{old2}$

a) Métodos de distinta Naturaleza:

1.- Asignar un valor:

$$\vec{x}^{new} = \vec{x}^{old};$$

2.- Reducir ó aumentar un valor (dividiendo un valor por otro):

$$\vec{x}^{new} = (\vec{x}^{old1} / \vec{x}^{old2}) * \mathbf{U(0,1);}$$

3.- Reducir ó aumentar un valor (usando la suma):

$$\vec{x}^{new} = \vec{x}^{old} + \mathbf{sign} * \vec{x}^{old} * \mathbf{U(0,1);}$$

4.- Reducir ó aumentar un valor (usando la división + la suma):

$$\vec{x}^{new} = (\vec{x}^{old1} / \vec{x}^{old2}) + \mathbf{sign} * \vec{x}^{old} * \mathbf{U(0,1);}$$

5.- Reducir ó aumentar un valor, incluyendo un parámetro:

$$\vec{x}^{new} = \vec{x}^{old} * \mathbf{sign} + \mathbf{param1} * \mathbf{sign} * \vec{x}^{old} * \mathbf{U(0,1);}$$

6.- **Cruce Heurístico**:

$$\vec{x}^{new} = \vec{x}^{old2} + \mathbf{sign}*(\vec{x}^{old1} - \vec{x}^{old2})*\mathbf{U(0,1);}$$

7.- **Cruce Heurístico** de control variable**:**

$$\vec{x}^{new} = \vec{x}^{old2} + \mathbf{sign}*(\vec{x}^{old2} - \vec{x}^{old1})*\mathbf{U(0,1);}$$

8.- **Cruce Geométrico**(versión más simple)**:**

$$\vec{x}^{new} = \mathbf{sqrt}(\vec{x}^{old1} * \vec{x}^{old2});$$

9.- **Cruce Geométrico:**

Si α =U(0,1) y β = 1- α,

$$\vec{x}^{new} = \left(\vec{x}^{old1}\right)^{\alpha} \times \left(\vec{x}^{old2}\right)^{\beta};$$

10.- **Cruce Aritmético**(versión más simple) **:**

Si auxi =U(0,1),

$$\vec{x}^{new} = \vec{x}^{old1} * \mathbf{auxi} + \mathbf{sign}*(\mathbf{1} - \mathbf{auxi})* \vec{x}^{old2};$$

11.- Variante del **Cruce Aritnético**:

Si auxi =-0.5+U(0,1)*2,

$$\vec{x}^{new} = \vec{x}^{old1} * \mathbf{auxi} + \mathbf{sign}*(\mathbf{1} - \mathbf{auxi})* \vec{x}^{old2};$$

12.- **Cruce de Media Garantizada:**

$$\vec{x}^{new} = \mathbf{0.5} * \vec{x}^{old1} + \mathbf{0.5} * \vec{x}^{old2};$$

13.- Variante del **Cruce de Media Garantizada:**

Si auxi = U(0,1),

$$\vec{x}^{new} = \mathbf{0.5} * \mathbf{auxi} * \vec{x}^{old1} + \mathbf{0.5} * \vec{x}^{old2};$$

14.- **Mutación Gaussiana** (α = 0):

Siendo:

$N(\alpha, \vec{\sigma})$ un vector de números aleatorios independientes con media α y desviación estándar $\vec{\sigma}$.

Si α = 0 → N(0, σ) = (maximun-minimum) * sqrt(-2*ln(U(0,1)))*sin (2*π*U(0,1))

$$\vec{x}^{new} = \vec{x}^{old} + \mathbf{N(0, \sigma);}$$

15.- Variante de la **Mutación Gaussiana**(α = 0)**:**

Siendo:

$N(\alpha, \vec{\sigma})$ un vector de números aleatorios independientes con media α y desviación estándar $\vec{\sigma}$.

Si $\alpha = 0 \rightarrow N(0, \sigma)$ = (maximun-minimum) + sqrt(-2*ln(U(0,1)))*sin (2*π*U(0,1))

$\vec{x}^{new} = \vec{x}^{old}$ * **N(0, σ);**

16.- Variante de la **Mutación Gaussiana**($\alpha = 0, \sigma = 1$)**:**

Si nvartot= nº total de variables
q = {1,…,nvartot}
$\tau_0 = 1/\text{sqrt}(q)$,
N(0,1) = sqrt(-2*ln(U(0,1)))*sin (2*π*U(0,1))
y $\sigma = \vec{x}^{old}$ * exp (τ_0 * N(0,1))

$\vec{x}^{new} = \vec{x}^{old} + \sigma$ * **N(0,1);**

17.- Variante de la **Mutación Gaussiana**($\alpha = 0, \sigma = n$)**:**

Si nvartot= nº total de variables
q = {1,…,nvartot}
$\tau = 1/\text{sqrt}(2*q)$
$\tau' = 1/\text{sqrt}(2 * \text{sqrt}(q))$
N(0,1) = sqrt(-2*ln(U(0,1)))*sin (2*π*U(0,1))
y $\sigma = \vec{x}^{old}$ * exp (τ * N(0,1) + τ' * N(0,1))

$\vec{x}^{new} = \vec{x}^{old} + \sigma$ * **N(0,1);**

18.- Variante de la **Mutación Gaussiana** ($\alpha = n*(n-1)/2, \sigma = n$)**:**

Si nvartot= nº total de variables
$\beta = 0.0873$
q = {1,…,nvartot}
$\tau = 1/\text{sqrt}(2*q)$
$\tau' = 1/\text{sqrt}(2 * \text{sqrt}(q))$
N(0,1) = sqrt(-2*ln(U(0,1)))*sin (2*π*U(0,1))
$\alpha = \vec{x}^{old} + \beta$ * N(0,1)
y $\sigma = \vec{x}^{old}$ * exp (τ * N(0,1) + τ' * N(0,1))

$\vec{x}^{new} = \vec{x}^{old1} + \vec{x}^{old2}$ * α* σ**;**

b) Métodos con diferente Rango:

Los métodos mencionados a continuación son iguales a algunos de los descritos anteriormente, pero con rangos diferentes. El resto de variaciones hasta completar los 39 métodos de nuestro motor se han conseguido aplicando estas tres formas de variar del rango a algunos de los métodos anteriores. Por simplicidad, sólo se han incluido los métodos generales aplicados a tres casos concretos, y no todas las variaciones reales del motor.

19.- **Multiplicar por un número aleatorio→**
Reducir o aumentar un valor (usando la suma)

$\vec{x}^{new} = \vec{x}^{old}$ **+ sign*** $\vec{x}^{old}$ ***** **[U(0,1)]^3;**

- Método más usado para variar el rango-

20.- **Dividir por un parámetro→**
Reducir ó aumentar un valor, incluyendo dos parámetros:

$\vec{x}^{new} = \vec{x}^{old}$ **+(param1*sign*** $\vec{x}^{old}$ ***U(0,1))/param2;**

21.- **Dividir por un cierto valor→**
Por ej: Variante de la Mutación Gaussiana ($\alpha = 0, \sigma = 1$)**:**

Si nvartot= nº total de variables
q = {1,…,nvartot}
$\tau_0 = 1/\text{sqrt}(q)$,
N(0,1) = sqrt(-2*ln(U(0,1)))*sin (2*π*U(0,1))
$\sigma = \vec{x}^{old}$ * exp (τ_0 * N(0,1))
y **param3={10,100,1000,…}**

$\vec{x}^{new} = \vec{x}^{old} + \sigma$ * **N(0,1)/param3;**

Finalmente, se incluyen algunos procedimientos que han demostrado ser ineficaces en nuestro motor de muestreo.

1.- Reducir un valor:

$\vec{x}^{new} = \vec{x}^{old}$ * **U(0,1);**

2.- Obtener un nuevo valor en un rango predefinido(variante de la Mutación No-Uniforme):

$\vec{x}^{new}$ **= minimun + (maximum - minimum)* U(0,1);**

Para definir distancias cada vez más cortas entre el máximo y el mínimo a lo largo del proceso se ha usado un mecanismo incremental del tipo:

$$N = c\left(1 - \frac{t}{T}\right),$$

donde c= constante, t = nº de ejecución actual y T= nº máximo de ejecuciones

3.- Reducir o aumentar un valor en función de una dirección señalada por otro valor :

$r\vec{x}^{old1} = \vec{x}^{old1}$ + rv1sign* $\vec{x}^{old1}$ *U(0,1);

$r\vec{x}^{old2} = \vec{x}^{old2}$ + rv2sign* $\vec{x}^{old2}$ *U(0,1);

$\vec{x}^{new}$ **= r** $\vec{x}^{old2}$ **+ sign*(r** $\vec{x}^{old1}$ **- r** $\vec{x}^{old2}$ **)*U(0,1);**

IV. UN EJEMPLO MATEMÁTICO

Como aplicación práctica se ha escogido la función de Rastrigin, debido a su dificultad manifiesta para alcanzar el mínimo, y cuya expresión es:

$$f(\vec{x}) = nA + \sum_{i=1}^{n}\left(x_i^2 - A\cos(2\pi x_i)\right)$$

con las restricciones que siguen:
A = 10, n = 20 y $x_i \in [-5.12, 5.12]$, $\forall i \in \{1, \ldots, n\}$.

Los resultados obtenidos muestran una convergencia muy rápida, ya que se obtienen buenos resultados en sólo **18 generaciones**, o sea 1800 evaluaciones de la función objetivo.

TABLA I

RESULTADOS NUMÉRICOS

Runs #	Rastrigin
1	1,1937e-12
2	<1.0e-14
3	<1.0e-14
4	<1.0e-14
5	7,4201e-10
6	6,9065e-11
7	2,8422e-14
8	1,1369e-13
9	<1.0e-14
10	<1.0e-14
Media	**9,0268e-11**
Peor	7,4201e-10
Mejor	**2,8422e-14**

Los valores hallados son muy robustos, ya que son similares o incluso mejores que los publicados por otros autores, como por ejemplo los obtenidos por Bäck et al [7], del orden de 10^{-11}.En la Fig. 1, se observa la diferente convergencia de la EF si la comparamos con un algoritmo genético simple.

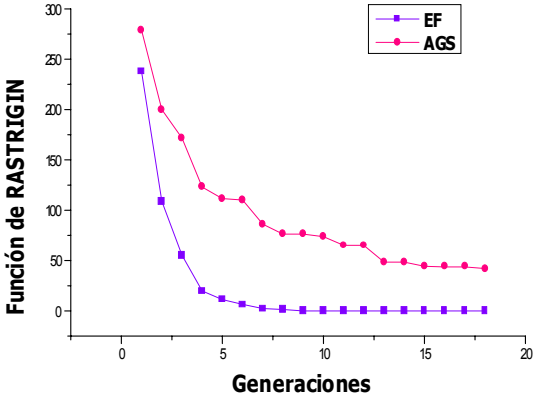

Fig. 1. Convergencia para la EF frente a un AGS.

V. UNA APLICACIÓN EN INGENIERÍA

En [8] se ha desarrollado una metodología para la resolución eficiente del problema de encontrar las localizaciones óptimas de fuentes puntuales de vertidos de aguas residuales en un medio acuático (lagos, estuarios o zonas marinas) mediante Algoritmos Evolutivos.

A partir de un punto de acumulación, se distribuye el vertido por uno o varios emisarios subacuáticos, al final de los cuales, por uno o varios difusores, se vierte o dispersa en el medio. Se trata de asegurar que la concentración de un contaminante determinado no exceda de un límite máximo establecido a priori, bien a una distancia prefijada de un contorno del dominio, bien en un subdominio del mismo y, simultáneamente, minimizar el coste económico asociado a la inversión en infraestructuras hidráulicas para la colocación in situ de los emisarios con sus respectivos difusores.

El problema de control que tenemos que resolver tiene por variables de control la localización física de cada difusor en el dominio acuático tridimensional considerado, siendo el estado una ecuación de convección-difusión-reacción en 3D, no estacionaria. Es un problema de control óptimo con restricción(es) de tipo desigualdad sobre el estado y minimización de una función coste.

La resolución de este problema o similares es muy dificultosa usando los usuales métodos deterministas, fundamentalmente por la complejidad del análisis matemático, debido a la presencia de fuentes puntuales en la ecuación de estado, modelizadas con deltas de Dirac, la existencia de restricciones sobre el estado de tipo desigualdad y, además, la función coste es no lineal por intervenir en la misma distancias geométricas. Así, la resolución de este problema de control óptimo se lleva a cabo en el contexto de la EF.

Como aplicación numérica vamos a considerar el problema de ubicar dos fuentes puntuales alineadas, Q_1 y Q_2, que vierten coliformes en el dominio bidimensional de la Fig. 2, de forma que la concentración de coliformes, en la recta $x = 500$, no exceda de una concentración máxima prefijada, y se minimice la suma de las distancias de ambas fuentes al punto de acumulación O, al tiempo que también se minimice la distancia de ambos puntos a la recta $x = 0$.

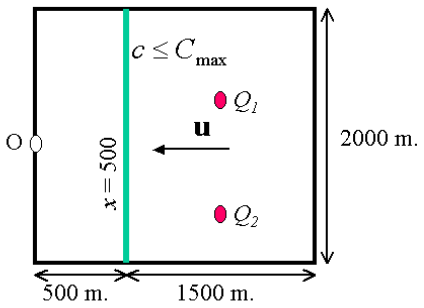

Fig. 2. Dominio bidimensional de estudio.

Datos utilizados en la aplicación del algoritmo flexible:

- El cromosoma consta de cinco genes:
  $(x_Q, y_{Q1}, y_{Q2}, q_1, q_2)$
  donde (x_Q, y_{Qi}) representa las coordenadas de la fuente puntual Qi y q_i el caudal de la misma.
  Obviamente se ha de verificar que la suma de los caudales que salen por ambas fuentes ha de coincidir con el caudal total Q_O que parte de O.
  Es decir: $Q_O = \sum_{i=1}^{2} q_i$.
- Número de individuos considerados: 20.
- Coordenadas de O: (0, 1000).
- Caudal de O: $Q_O = 10^{12}$ colis/s.
- Coeficientes de difusión: $Dx = Dy = 10$ m^2/s.
- Factor k = 6.39 10^{-4} s^{-1}.

- Velocidad: $\mathbf{u} = -0.07\ \mathbf{e_1}$ m/s.
- Concentración máxima de colis en la recta $x = 500$: $C_{max} = 3.089561\ 10^8$ colis/s.
- El espacio de búsqueda es:
  $E = [500, 1000] \times [500, 1500] \times [500, 1500] \times [0, Q_O] \times [0, Q_O]$
- La función objetivo propuesta es:

$$F(z) = \sum_{i=1}^{2} \left\| \mathbf{x}_i - \mathbf{x}_0 \right\|_2 + (x_i - x_0)^4 + \lambda_1 \left(Q_O - \sum_{i=1}^{2} q_i \right)^2 + \lambda_2 (c_{max} - C_{max})^2$$

  $\forall\ \mathbf{z} = (x, y_1, y_2, q_1, q_2) \in E$.
  donde c_{max}[1] es la concentración máxima calculada en la recta $x = 500$ para el vector $\mathbf{z}$ considerado, $\lambda_1 = 10^{-3}$, $\lambda_2 = 10^{-6}$, $\mathbf{x_0} = (x_0, y_0)$ y $\mathbf{x_i} = (x_i, y_i)$, i = 1, 2.

En la Fig. 3 se muestran los resultados obtenidos con la EF. La mejor solución hallada fue:
$F(x, y_1, y_2, q_1, q_2) = 2.401\ 10^{11}$ para $x = 700.82$, $y_1 = 1352.80$, $y_2 = 607.82$, $q_1 = 4.956583\ 10^{11}$, $q_2 = 5.043427\ 10^{11}$, siendo en este caso $c_{max} = 3.089520\ 10^8$ colis/s.

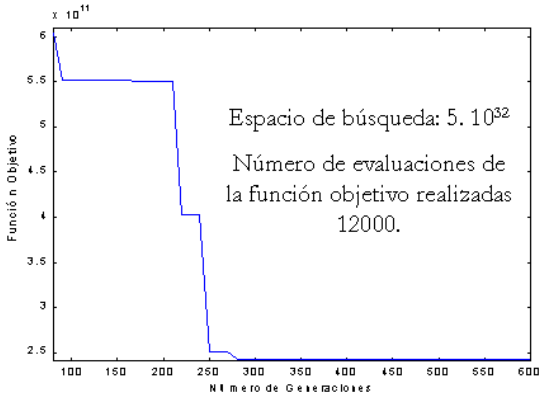

Fig. 3. Convergencia para la EF.

VI. CONCLUSIONES

En el presente trabajo se ha demostrado que es posible diseñar con éxito nuevos y más eficientes algoritmos evolutivos que los existentes hasta la fecha, sin más que considerar no sólo adaptabilidad en los operadores y los parámetros involucrados en la optimización, sino también en la propia estructura del algoritmo. Esos diseños pueden obtenerse flexibilizando dicha estructura de manera que sea capaz de adaptarse dinámicamente (y de adaptar sus operadores) dependiendo de las circunstancias actuales del proceso de búsqueda. Para implementar estas estructuras flexibles son necesarios algunos mecanismos para la toma de decisiones (o su corrección) y control del proceso. Como ejemplo de este nuevo concepto se ha presentado un algoritmo denominado de Evolución Flexible, que cumple con lo especificado anteriormente. Para comenzar, su elasticidad se debe principalmente a su Estructura dinámica de operadores (EDO), que no depende del usuario. Para favorecer la búsqueda, se han incluido Mecanismos de Control Probabilístico (MCP) así como otra útil herramienta para el diseño de algoritmos robustos como es Alargar el Código Genético (ACG) de las soluciones. La exitosa aplicación de la EF a varios casos, tanto matemáticos como de la vida real, da pie a un razonable optimismo sobre el futuro de este nuevo tipo de algoritmos evolutivos. Además, el trabajo ulterior en esta área no se circunscribe sólo a la mejora del 'motor de muestreo' mencionado y al desarrollo de nuevos motores como por ejemplo, el de 'aprendizaje', sino a la inclusión de nuevos mecanismos de control para la toma de decisiones, como puede ser el empleo de la Lógica Borrosa.

[1] La concentración de coliformes se calcula, por ejemplo, aplicando el método propuesto en [8] y [9].

VII. REFERENCIAS

[1] N. Toshine, N. Iwauchi, S. Wakabayashi, T. Koide and I. Nishimura. *A Parallel GA with Adaptative Adjustment of Genetic Parameters*. Proceedings of the Genetic and Evolutionary Computation Conference, publicado por Morgan Kaufmann Publishers, San Francisco, California. GECCO 2001, págs. 679-686.

[2] E. Kee, S. Airey and W. Cyre. *An Adaptative Genetic Algorithm*. Proceedings of the Genetic and Evolutionary Computation Conference, publicado por Morgan Kaufmann Publishers, San Francisco, California. GECCO 2001, págs. 391-397.

[3] G. Winter, B. Galván, P. Cuesta y S. Alonso. *"Flexible Evolution"*. Atenas, Grecia, Eurogen 2001.

[4] B. Edmons. *Meta-Genetic Programming: Co-evolving the Operators of Variation*. CPM Report Nº: 98-32. Centre for Policy Modelling, Manchester Metropolitan University. http://www.cpm.mmu.ac.uk/cpmrep32.htm

[5] Z. Michalewicz, G. Nazhiyath, and M. Michalewicz. *A note on usefulness of geometrical crossover for numerical optimization problems*. Proceedings of the fifth Annual Conference on Evolutionary Programming, publicado por L. J. Fogel, P. J. Angeline and Th. Bäck, editores. The MIT Press, Cambridge, MA, 1996, págs. 305-312.

[6] Th. Bäck y B. Naujoks. "*Innovative Methodologies in Evolution Strategies*". Informe D2.2 de la red INGENET. Junio 1998.

[7] Th. Bäck, W. Haase, B. Naujoks, L. Onesti y A. Turchet. *Evolutionary algorithms for academic and industrial cases*. "Evolutionary Algorithms in Engineering and computer Science", editado por K.Miettinen, M. M. Mäkelä, P. Neittaanmäki and J. Périaux. John Wiley & Sons, Ltd, Inglaterra , 1999, págs. 383-398.

[8] B. González. *Determinación de localizaciones óptimas de puntos múltiples de vertidos de aguas residuales en zonas costeras mediante algoritmos genéticos*. Tesis Doctoral,

Universidad de Las Palmas de Gran Canaria, Julio de 2001.

[9] B. González, B. Galván y G. Winter. *Optimal Placement of Wastewater Outfalls by a Flexible Evolution system and Evolutionary Computation*. Atenas, Grecia, Eurogen 2001.