Optimización Continua mediante la Evolución de Funciones de Densidad de Probabilidad con un Modelo de Dos Islas

Alicia D. Benítez y Jorge Casillas
Dpto. de Ciencias de la Computación e Inteligencia Artificial
Universidad de Granada. España, 18071.
E-mail: aliciades@gmail.com, casillas@decsai.ugr.es

Resumen— **El trabajo presenta un nuevo algoritmo evolutivo diseñado para la optimización continua. Se basa en la evolución de funciones de densidad de probabilidad, que se centran en las zonas del espacio de búsqueda más prometedoras del dominio de cada variable. Se incluyen varios mecanismos para la auto-adaptación del algoritmo a las características del problema. Gracias a los estudios empíricos, hemos observado que nuestro algoritmo obtiene buenos resultados de precisión con respecto a otros algoritmos del estado del arte.**

Palabras clave— **algoritmos evolutivos, optimización continua, función de distribución de probabilidad**

I. Introducción

La optimización continua es un problema importante para la ingeniería, ya que es difícil encontrar métodos que encuentren óptimos globales en determinados problemas. Metaheurísticas tales como los Algoritmos Genéticos (AG) [8], Estrategias Evolutivas (EE) [4] y Algoritmos Meméticos (AG hibridados con técnicas de búsqueda local) [13], son los que actualmente se están aplicando. Existen otros algoritmos diseñados originalmente para este tipo de optimización, los Algoritmos de Estimación de Distribución (para referirnos a estos algoritmos, usaremos su acrónimo anglosajón EDA) [9], que no usan operadores de cruce o de mutación, a diferencia de los AG y las EE. Para permitir la evolución de la población, usa un mecanismo que consiste en el muestreo de individuos a partir de una densidad de probabilidad. En la línea de este algoritmo, encontramos el PBIL Continuo [19], con una densidad de probabilidad normal para cada variable a partir de la cual, se genera una población que actualiza las normales. También incluye mecanismos para adaptar las desviaciones típicas de las normales.

En muchos casos, un conjunto de poblaciones que trabajan en paralelo [1] puede producir bastantes beneficios. Esta clase de algoritmos se conocen con el nombre de Algoritmos Evolutivos Paralelos (AEP) o modelo de islas. Consisten en un conjunto de subpoblaciones que evolucionan independientemente y en ocasiones intercambian información entre ellas.

En este trabajo, se propone un nuevo algoritmo que surge como resultado de la selección de las ideas principales de varias metaheurísticas: AG, EE, Algoritmos Meméticos, EDA, y AEP. El algoritmo básico consiste en la evolución de una mixtura de distribuciones normales, manteniendo las mejores soluciones obtenidas. Dicho algoritmo está contenido en otro que considera dos islas de diferentes características para auto-adaptarse al problema. De acuerdo a esto, el trabajo se organiza en las siguientes secciones: la *sección II*, describe el algoritmo que se propone en este trabajo; la *sección III*, relata brevemente las similitudes del algoritmo propuesto con otras metaheurísticas; la *sección IV*, muestra el estudio empírico del algoritmo con otros algoritmos propuestos; la *Sección V*, muestra algunas conclusiones y trabajos futuros.

II. Descripción del Algoritmo EvolPDF-2

A continuación, describimos el algoritmo propuesto. Para un mejor entendimiento, hemos distinguido entre *algoritmo básico* y el final que se propone (llamado EvolPDF-2). El algoritmo básico, que se puede considerar como el núcleo del proceso de búsqueda, se describe en las subsecciones II-A, II-B y II-C (en la figura 1, se muestra la estructura del algoritmo básico). Este algoritmo básico puede funcionar incorrectamente debido a su alto riesgo de caer en óptimos locales y su alta dependencia de los valores de los parámetros. Para hacer frente a esto, hemos diseñado un algoritmo más complejo basado en el básico, que se describirá en las subsecciones II-D y II-E.

A. Componentes Básicos: representación, Inicialización, Muestreo y Reemplazamiento

El algoritmo básico tiene cuatro componentes principales: *representación*, que describe cómo se codifican las soluciones; *inicialización*, que crea un conjunto inicial de soluciones; *muestreo*, que genera nuevas soluciones con funciones de densidad de probabilidad (para referirnos a este tipo de funciones, usaremos su acrónimo anglosajón PDF) estimadas de las mejores soluciones actuales; y *reemplazamiento*, que actualiza el conjunto de soluciones consideradas para generar nuevas soluciones. Los componentes mencionados, están diseñados de la siguiente forma:

- *Representación*: el algoritmo mantiene un conjun-

Entrada:
- Parámetros: σ_0, σ_f, P (número de conjuntos de PDF)
- Sea n el número de variables y $T^p = \{S_1^p, \ldots, S_k^p\}$ el p-ésimo conjunto de PDF, $p \in \{1, \ldots, P\}$

Algoritmo:
1. Inicialización($S_i^p \in T^p$), para $i \in \{1, \ldots, k\}$ y $p \in \{1, \ldots, P\}$
2. Mientras (*no se cumpla la condición de parada*)
   a) Para $p \in \{1, \ldots, P\}$:
      $R^p \leftarrow$ Muestreo(T^p)
      if (mejor(mejor(R^p), peor(T^p)))
         $T^p \leftarrow (T^p - \{\text{peor}(T^p)\}) \cup \{\text{mejor}(R^p)\}$
   b) if (*condición de recombinación*)
         Recombinación($T^1, \ldots, T^p$)

Fig. 1. Algoritmo EvolPDF básico

to de k soluciones durante el proceso de búsqueda. Cada solución se representa de la siguiente forma: $S_i = \left(\mu_i^1, ..., \mu_i^v, ..., \mu_i^n\right)$, siendo n el número de variables continuas del problema.

- *Inicialización*: el algoritmo comienza generando aleatoriamente k soluciones según una distribución uniforme.
- *Muestreo*: en cada iteración, se genera un número de soluciones del conjunto actual. Para ello, consideramos que el conjunto actual de soluciones constituye una PDF para cada variable.

Para generar, es decir, muestrear nuevas soluciones, el proceso se realiza de la siguiente forma: para cada variable, se escoge el 20 % de soluciones mejores que constituyen la PDF de esa variable, y a partir de él, se elige una solución aleatoriamente, según una distribución uniforme. Aunque este tipo de muestreo no sea completamente proporcional al peso de cada normal, permite que el algoritmo pueda trabajar con maximización o minimización y sin necesidad de conocer el dominio de la función. Seguidamente, se genera un número aleatorio con una densidad normal, centrada en la media de la normal elegida en el paso anterior. El valor obtenido será el asignado a la correspondiente variable de la solución generada.

En caso de generar un valor fuera del permitido para cada variable, el valor se establecerá de acuerdo al extremo correspondiente del intervalo. Consecuentemente, existirá una saturación en los extremos para compensar la falta de acumulación de densidad en esas regiones. No hemos probado otros parámetros, pero sí queremos destacar que gracias a la consideración de este porcentaje, podemos incrementar y disminuir la presión selectiva, es decir, con un alto porcentaje de selección de mejores soluciones, la presión selectiva será menor; sin embargo, si disminuimos dicho porcentaje, tendremos una mayor presión selectiva, ya que existiría menor diversidad.

Destacar que la desviación típica es común a todas las variables.

- *Reemplazamiento*: entre las soluciones generadas en la iteración actual, se elige la mejor según el fitness $f(S_i)$ y reemplaza a la peor de las k mejores soluciones actuales mantenidas por el algoritmo en caso de que las mejore. Este reemplazamiento, se puede considerar como una clase de estimación de la PDF.

B. Convergencia de la Desviación Típica

En todas las metaheurísticas, es necesario que exista tanto exploración como explotación. Al comienzo del algoritmo, nos interesa que haya exploración, con la finalidad de favorecer la diversificación en el espacio de búsqueda. A medida que nos vamos acercando al final del algoritmo, es más conveniente una alta explotación, para alcanzar un óptimo local o global de la zona, que sea más prometedora. De acuerdo a esto, nuestro algoritmo regula la desviación típica de las distribuciones, intentando hacerla más alta al comienzo del algoritmo y más baja al final. Para conseguirlo, hemos definido un esquema de convergencia para la desviación típica, según la siguiente fórmula (la figura 2 muestra un ejemplo):

$$C(t) = \begin{cases} \sigma_0 & \text{si} \quad t \leq \alpha \\ (\sigma_0 - \sigma_f)(1 - 2(\frac{t-\alpha}{\beta-\alpha})^2) & \text{si} \quad \alpha < t \leq \frac{\alpha+\beta}{2} \\ (\sigma_0 - \sigma_f)(2(\frac{t-\beta}{\beta-\alpha})^2) + \sigma_f & \text{si} \quad \frac{\alpha+\beta}{2} < t \leq \beta \\ \sigma_f & \text{si} \quad t > \beta \end{cases} \tag{1}$$

siendo t el número de evaluación, σ_0 y σ_f los valores inicial y final deseados para la desviación típica respectivamente, y α y β dos constates fijadas dependiendo del número de evaluaciones: $\alpha = evaluaciones \cdot 0{,}1$, $\beta = evaluaciones \cdot 0{,}8$. Con esta fórmula, se regula el decremento de la desviación típica, distinguiendo tres estados durante la evolución de las PDF:

1. *Inicialización*: al principio (10 % de evaluaciones), la desviación típica se mantiene constante y se utiliza el algoritmo para configurar un conjunto de PDF iniciales.
2. *Consolidación*: posteriormente, se produce la reducción paulatina de la desviación típica para favorecer gradualmente la explotación frente a la exploración.
3. *Refinamiento*: finalmente, durante el último 20 % de evaluaciones, se realiza un ajuste final de las PDF, manteniendo una desviación típica baja y constante.

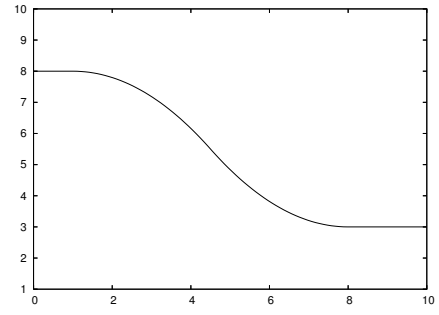

Fig. 2. Ejemplo de la convergencia de la desviación típica (10 evaluaciones, $\alpha = 1$, $\beta = 8$, $\sigma_0 = 8$, $\sigma_f = 3$)

Los valores que se van a usar para σ_0 y σ_f son: σ_0 = 10e-01 y σ_f = 10e-04. Estos valores han sido pro-

bados en distintos problemas.

C. Recombinación

El reemplazamiento elitista de nuestro algoritmo permite obtener con rapidez buenas soluciones, pero con un alto riesgo de convergencia prematura. Para hacer frente a esto, proponemos usar varios conjuntos de PDF que, después de un cierto número de iteraciones, se combinen para de esta manera, alejarnos de óptimos locales. Esta combinación se realiza de la siguiente forma. Primero, se recogen todas las soluciones de los distintos conjuntos de PDF, y entonces, cada solución se distribuye aleatoriamente entre los diferentes conjuntos. Este proceso se puede considerar como una clase de operador de cruce. El hecho de combinar los conjuntos de PDF hace que si un conjunto cae en un óptimo local, otro conjunto puede trasladarlo y conducirlo a otra zona más prometedora del espacio de búsqueda.

Se ha fijado el número inicial de conjuntos de PDF a cinco y el número de recombinaciones sigue una proporcionalidad con respecto al número de evaluaciones de 0,01.

D. Modelo de Islas

Una vez definido el algoritmo básico en las tres secciones previas, vamos a estudiar su comportamiento en distintas situaciones. Está claro que dos de los parámetros más importantes del algoritmo básico son el número de muestreos y de normales. En la Tabla I, se muestran los resultados obtenidos con distintas combinaciones de estos parámetros para los problemas Schwefel (unimodal) y Rastrigin (multimodal), ambos con 25 variables. Se muestra la media y la desviación típica de 10 ejecuciones del algoritmo básico (figura 1). En esta tabla, se puede observar cómo en el problema unimodal Schwefel, con un número bajo de muestreos y de normales, conduce a buenos resultados. Con valores bajos de número de normales y muestreos, aparece una alta intensificación, lo cual es bueno para el comportamiento del algoritmo de cara a problemas unimodales. Si prestamos atención a Rastrigin (problema multimodal), se puede observar que ocurre justo lo contrario que en Schwefel. Es decir, las soluciones mejoran a medida que el número de muestreos y normales se incrementa. Esto se debe al hecho de que, el incremento del valor de estos parámetros produce más diversidad, y esto significa un funcionamiento apropiado para problemas multimodales.

Dadas estas premisas, decidimos especializar el algoritmo en ambos comportamientos, fijando ambos parámetros *a priori*. Para ello, consideramos un modelo paralelo basado en islas, como se muestra en la figura 3.

De esta forma, cada isla está asignada a una característica diferente. Básicamente, cada isla implementa el algoritmo básico, pero con diferentes parámetros. Cuando transcurre un determinado número de iteraciones, las mejores soluciones actuales de cada isla migran de una a otra. En cada isla, sus conjuntos de PDF evolucionan independientemente y en paralelo. El intercambio de información entre ellas favorece la cooperación entre islas para alcanzar el óptimo global. Una ventaja de este modelo basado en islas es que no es obligatorio usar el mismo algoritmo en ambas islas, lo cual indica que podemos usar cada isla para una clase de aproximación, y de esta forma usamos una isla para problemas que requieren más intensificación que diversificación y otra para problemas que requieren más diversificación que intensificación. Esto hace que el algoritmo sea capaz de auto-adaptarse a la aproximación más prometedora para resolver el problema. Cada isla tiene una serie de características propias: número de normales que constituyen cada PDF para un valor fijo de k, número de muestreos, e implementación de Búsqueda Local (BL).

- *Isla 1 (I_1)*: el propósito de esta isla es intensificar la búsqueda en el espacio, manteniendo las solu-

TABLA I
RESULTADOS DE LA EXPERIMENTACIÓN DEL ALGORITMO BÁSICO CON LAS FUNCIONES SCHWEFEL Y RASTRIGIN (10 EJECUCIONES).

Función	*Muestreos*	*Normales*	$\bar{x}$	σ
Schwefel	5	1	2.42e-2	1.15e-1
		5	2.34e-3	6.41e-3
		10	3.21e-1	1.06e+0
		20	5.15e+1	1.17e+2
		30	2.53e+2	4.49e+2
	10	1	1.05e-1	2.81e-1
		5	6.15e-3	2,41e-2
		10	7.20e-1	2.22e+0
		20	1.33e+2	4.23e+2
		30	2.19e+2	4.86e+2
	15	1	2.22e-1	6.78e-1
		5	2.62e-2	1.32e-1
		10	2.08e+0	1.03e+1
		20	9.05e+1	2.93e+2
		30	1.83e+2	2.99e+2
	20	1	1.33e+0	4.17e+0
		5	1.24e-1	5.93e-1
		10	2.38e+0	4.56e+0
		20	7.33e+1	1.46e+2
		30	1.32e+2	2.20e+2
Rastrigin	5	1	6.40e+1	4.31e+1
		5	4.06e+1	1.68e+1
		10	1.09e+1	1.06e+1
		20	8.55e+0	7.20e+0
		30	4.47e+0	4.04e+0
	10	1	7.32e+1	4.71e+1
		5	4.42e+1	4.28e+1
		10	1.59e+1	1.10e+1
		20	6.76e+0	7.41e+0
		30	4.57e+0	7.47e+0
	15	1	5.95e+1	4.71e+1
		5	3.93e+1	3.74e+1
		10	1.41e+1	1.07e+1
		20	2.56e+0	7.98e+0
		30	4.97e+0	5.26e+0
	20	1	6.81e+1	4.46e+1
		5	4.99e+1	4.02e+1
		10	1.39e+1	8.44e+0
		20	5.87e+0	6.36e+0
		30	4.67e+0	4.88e+0

Entrada:
- Parámetros: σ_0, σ_f, P_l (número de conjuntos de PDF para la l-ésima isla)
- Sea n número de variables y $T^{(l,p_l)} = \{S_1^{(l,p_l)}, \ldots, S_{k_l}^{(l,p_l)}\}$ el p_l-ésimo conjunto de PDF para la l-ésima isla, $l \in \{1,2\}$, $p_L \in \{1, \ldots, P_l\}$.

Algoritmo:

1. Inicialización($S_i^{(l,p_l)} \in T^{(l,p_l)}$, para $l \in \{1,2\}$, $i \in \{1, \ldots, k_l\}$ y $p_l \in \{1, \ldots, P_l\}$
2. Repetir durante el 90 % de evaluaciones
   - *a*) Para $p_1 \in \{1, \ldots, P_1\}$:
     $R_t^{(1,p_1)} \leftarrow$ Muestreo($T^{(1,p_1)}$)
     if (mejor(mejor($R_t^{(1,p_1)}$), peor($T^{(1,p_1)}$)))
       $r \leftarrow$ Simplex(mejor($R_t^{(1,p_1)}$))
       $T^{(1,p_1)} \leftarrow (T^{(1,p_1)} - \{\text{peor}(T^{(1,p_1)})\}) \cup \{r\}$
   - *b*) Para $p_2 \in \{1, \ldots, P_2\}$:
     $R_t^{(2,p_2)} \leftarrow$ Muestreo($T^{(2,p_2)}$)
     if (mejor(mejor($R_t^{(2,p_2)}$), peor($T^{(2,p_2)}$)))
       $T^{(2,p_2)} \leftarrow (T^{(2,p_2)} - \{\text{peor}(T^{(2,p_2)})\}) \cup \{\text{mejor}(R_t^{(2,p_2)})\}$
   - *c*) if (*condición de recombinación*)
     Recombination($T^{(1,1)}, \ldots, T^{(1,p_1)}$)
     Recombination($T^{(2,1)}, \ldots, T^{(2,p_2)}$)
   - *d*) if (*condición de migración*)
     Migrar(mejor(T^2), T^1)
     Migrar($mejor(T^1)$, T^2)
3. Durante el restante 10 % de evaluaciones, aplicar algoritmo Simplex a la mejor solución encontrada.

Fig. 3. Algoritmo EvolPDF-2

ciones más prometedoras. Para ello, empleamos los siguientes parámetros del algoritmo básico:

1. Número de normales $k = 1$.
2. Número de muestreos por iteración = 5.
3. La BL es aplicada a la mejor solución muestreada, tan pronto como sea actualizada.

- *Isla 2 (I_2)*: el objetivo de esta segunda isla es proporcionar la diversidad necesaria al algoritmo, y así no caer en un óptimo local. Se usan los siguientes parámetros y componentes:

1. Número de normales $k = 30$.
2. Número de muestreos por iteración = 20.

Con un rápido vistazo, podemos ver la configuración de cada isla. El número de normales es mucho más alto en I_2 que en I_1. Las migraciones se producen según un número dado de iteraciones de la siguiente forma: se elige la mejor solución de cada isla para migrar a la otra isla. El mejor individuo de I_2 se convierte en miembro de la isla I_1 si mejora a su mejor solución actual de cada población de I_1. La solución proveniente de I_1 se convertirá en solución de I_2, reemplazando la peor solución de cada población de I_2. El número de migraciones sigue una proporcionalidad con respecto al número de evaluaciones de 0,02. Queremos destacar que esta migración permite al algoritmo auto-adaptarse según el problema que estemos considerando. Para problemas multimodales, la I_2 trabaja mejor debido a sus características; si se produce un estancamiento en la I_1, y la migración de una solución de I_2 a I_1 tiene lugar, esto hace que I_1 abandone una zona con un óptimo local, porque saltará a otra zona prometedora del espacio de búsqueda dada por I_2. De la misma forma, para problemas unimodales, la solución de I_1 migra a I_2 con el objetivo de paralelizar la búsqueda y dar más velocidad al proceso.

E. Búsqueda Local

La BL es una metaheurística que consiste en la búsqueda de buenas soluciones en un vecindario. Ocasionalmente, esta técnica puede hacernos caer en un óptimo local, pero puede ser una herramienta eficaz en la búsqueda de soluciones. En nuestro algoritmo, usaremos BL como una forma de refinar las soluciones. En I_1, aplicaremos BL a la mejor solución muestreada, si es mejor que la mejor elitista de su isla. Esto refuerza más la intensificación del espacio de búsqueda. Además también se aplica BL en el último 10 % de evaluaciones para así refinar la mejor solución encontrada. La BL usada en este trabajo, es el algoritmo Simplex, disponible en [14].

III. Relación del Algoritmo con otras Metaheurísticas

Nuestro algoritmo presenta una serie de semejanzas con otras metaheurísticas existentes. A continuación analizamos brevemente algunas de ellas.

Nuestro algoritmo tiene claras similaridades con los EDA [10]. Podemos considerar nuestro algoritmo como un EDA en el que se mantiene una población élite de las mejores soluciones obtenidas hasta el momento, y en cada iteración, se realiza una estimación de las PDF según un modelo incremental construido mediante normales univariantes. Este proceso simplifica nuestro algoritmo, evitando la necesidad de emplear otras técnicas de estimación de densidades de probabilidad más complejas.

Tanto nuestro algoritmo como el PBIL Continuo [19] y el algoritmo SHCLVND [18], utilizan funciones de densidad de probabilidad para encontrar nuevas soluciones. Sin embargo, el modo de construir y modificar las PDF difiere en nuestro algoritmo del resto.

En el PBIL Continuo y SHCLVND, cada PDF está construida por una única distribución normal, por lo que dichos algoritmos no se pueden adaptar a problemas multimodales, ya que la media de la normal sólo puede estar centrada en un único punto. En nuestro algoritmo, ese problema no existe, ya que nuestra PDF está constituida por un conjunto de normales que sumadas constituyen una PDF moldeable. De esta forma, nos podemos adaptar perfectamente a la función que estemos resolviendo, incluso en casos de asimetría y multimodalidad.

El PBIL Continuo y el SHCLVND utilizan las soluciones generadas para desplazar las PDF, lo cual ralentiza el proceso de adaptación. Sin embargo, nuestro algoritmo utiliza las soluciones generadas para formar parte de las PDF. Es decir, presenta una mejor perspectiva, ya que en el primer caso la influencia de las buenas soluciones tiene una caducidad que se pierde después de algunas iteraciones, mientras que en nuestro caso, permanecen mientras no

se encuentren soluciones mejores. Nuestro algoritmo coincide con SHCLVND en el uso de una única desviación típica global que va reduciéndose conforme avanza en algoritmo. Sin embargo, mientras que el SHCLVND usa un enfriamiento geométrico (producto por una constante inferior a 1), nuestro proceso de reducción de la desviación típica distingue entre las fases de *inicialización*, *consolidación* y *refinamiento* para una mejor regulación del equilibrio entre exploración/explotación.

Nuestro algoritmo también tiene algunas similaridades con los AG. Podemos considerar cada solución como un cromosoma, y cada PDF como un gen. La mutación genética correspondería a cuando generamos un número aleatorio procedente de la densidad normal escogida.Análogamente, podríamos relacionarlo con las Estrategias Evolutivas (1,1) [4], cuando empleamos una única normal por PDF. En nuestro caso, trabajamos con varias PDF por variables en paralelo, sobre las que se hacen recombinaciones periódicas para intentar salir de los óptimos locales. Esta idea está muy relacionada con el operador de cruce de los AG, ya que este operador recombina dos o más cromosomas que hereden las propiedades de los padres.

También podemos considerar nuestro algoritmo como un Algoritmo Memético [13], ya que se hibrida con una técnica de búsqueda local, que se aplica a algunas soluciones durante la evolución del algoritmo y al final del algoritmo para refinar la mejor solución encontrada.

IV. Estudio Empírico

Esta sección presenta los resultados empíricos de los experimentos llevados a cabo, donde nuestro algoritmo se compara con los resultados obtenidos por algunos algoritmos evolutivos para optimización global del estado del arte.

A. Experimentación

Hemos considerado las 25 funciones propuestas en la Sesión Especial "Real-parameter optimization" del IEEE CEC 2005 [6], [21], con las especificaciones establecidas en esta sesión: las experimentaciones se han realizado con 10 y 30 variables para cada función, con un número de 25 ejecuciones cada una; y un máximo de 100.000 evaluaciones para el caso de 10 variables y 300.000 para el caso de 30. Hay tres clases de funciones: unimodales (de la función 1 a la 5), multimodales (de la 6 a la 14), e hibridaciones de funciones multimodales (de la función 15 a la 25). Hay que destacar el hecho de que todas las funciones se encuentran desplazadas para evitar la simetría y la localización centrada del óptimo. El máximo error permitido en la función es 10e-8. En la tabla II, se muestra una relación de las funciones consideradas, incluyendo el valor óptimo de cada una de ellas. La cuarta columna indica si para una determinada función se ha alcanzado el valor óptimo por alguno de los algoritmos de comparación (sin incluir nuestro algoritmo) en 10 y 30 variables. Los resultados de los métodos con los que comparamos se han extraido de [6].

Hemos considerado once algoritmos, pertenecientes a diferentes paradigmas de la metaheurística. A continuación describimos brevemente cada uno de ellos. BLX-GL50 [7] es un AG en donde se pone de manifiesto dos clases de cromosomas: masculino y femenino. El algoritmo BLX-MA [12] es un Algoritmo Evolutivo Memético, que usa Solis-Wets como Búsqueda Local, y que divide la población en tres grupos para aplicar la Búsqueda Local: los más prometedores, prometedores y los menos prometedores. El algoritmo DE [17] está basado en la Evolución Diferencial. L-SaDE [16] es un variante autoadaptativa de la Evolución Diferencial. Elige una estrategia según una probabilidad, la cuál se adapta de acuerdo a la proporción de reemplazamiento de cada uno. DMS-L-PSO [11] actúa de forma que cada n iteraciones, los grupos se reorganizan. Cada solución solo afecta a su grupo, pero al final, afecta a todo el grupo. EDA [22] emplea una distribución Gausiana multivariante, siendo capaz de representar la correlación entre variables en los individuos seleccionados mediante una matriz de covarianza. L-CMA-ES [3] introduce una estrategia de reinicialización de un algoritmo de búsqueda local avanzado: el algoritmo CMA-ES con un paso inicial pequeño. G-CMA-ES [2] se basa en L-CMA-ES, donde la población se incrementa en cada reinicialización. K-PCX [20] es un algoritmo estacionario con búsqueda basada en población. CoEVO [15] es un Algoritmo Evolutivo que usa una evolución cooperativa para crear y producir mutaciones con mayor éxito. En SPC-PNX [5], se seleccionan dos padres y se generan λ hijos, que se producen a través del operador de cruce. Entonces, los descendientes se combinan de manera que la población permanece con tamaño constante, por medio del operador de reemplazamiento.

Usaremos las siguientes métricas para analizar los resultados obtenidos:
1. *Ranking*: recoge el número de veces que un algoritmo ha quedado en primera, segunda, tercera, cuarta y quinta posición, según los resultados obtenidos. Esta métrica se calcula de la siguiente forma. Para cada función, los algoritmos se ordenan por su error medio. Entonces, asignamos una posición a cada algoritmo (primera, segunda, tercera, cuarta o quinta). Si varios algoritmos tienen el mismo valor de media (o han alcanzado el valor óptimo), éstos tendrán el mismo valor de posición. Por lo tanto, si dos algoritmos alcanzan el valor óptimo, y el tercero no, asignaremos la posición 1 a los dos primeros algoritmos, y la posición 3 al tercero. Para cada algoritmo, añadimos el número de veces que aparece en cada posición para todas la funciones consideradas.
2. *Error medio total*: este método consiste en el cálculo del error medio obtenido por cada algoritmo normalizado por el mejor resultado obtenido para

TABLA II

FUNCIONES DEL IEEE CEC 2005

Num.	*Nombre*	*Valor* Optimo	*Resuelta*	
			10 var	*30 var*
	Unimodal			
1	Función Esfera Desplazada	-450	Sí	Sí
2	Problema de Schwefel Desplazado 1.2	-450	Sí	Sí
3	Función Elíptica Desplazada Condicionada a alta Rotación	-450	Sí	Sí
4	Problema Desplazado de Schwefel con Ruido en Fitness 1.2	-450	Sí	Sí
5	Problema de Schwefel con óptimo global en límites 2.6	-310	Sí	Sí
	Multimodal			
6	Función de Rosenbrock Desplazada	390	Sí	Sí
7	Función de Griewank Rotada Desplazada sin límites	-180	Sí	Sí
8	Función de Ackley Desplazada Rotada con óptimo global en límites	-140	Sí	Sí
9	Función de Rastrigin Desplazada	-330	Sí	No
10	Función de Rastrigin Desplazada y Rotada	-330	No	No
11	Función de Weierstrass Desplazada y Rotada	90	No	No
12	Problema de Schwefel 2.13	-460	No	No
13	Función de Rosenbrock con Griewank Extendida Expandida(F8F2)	-130	No	No
14	Función de Scaffer Expandida Rotada Desplazada F6	-300	No	No
	Hibridación de Multimodales			
15	Función de Composición Híbrida	120	No	No
16	Función de Composición Híbrida Rotada	120	No	No
17	Función de Composición Híbrida Rotada con Ruido en Fitness	120	No	No
18	Función de Composición Híbrida	10	No	No
19	Función de Composición Híbrida con una curvatura estrecha para el óptimo global	10	No	No
20	Función de Composición Híbrida Rotada con óptimo global en el límite	10	No	No
21	Función de Composición Híbrida Rotada	360	No	No
22	Función de Composición Híbrida Rotada con Matriz de alto número de condición	360	No	No
23	Función de Composición Híbrida Rotada no Continua	360	No	No
24	Función de Composición Híbrida Rotada	260	No	No
25	Función de Composición Híbrida sin límites	260	No	No

cada función. Se calcula de la siguiente forma:

$$EMT_{alg} = \sum_{i \in \text{Funciones}} \frac{f_i(S_{alg}) - \theta_i}{\underset{j \in \text{Algoritmos}}{\text{máx}} (f_i(S_j)) - \theta_i} \quad (2)$$

donde θ_i es el umbral de error de la i-ésima función (10e-8 en nuestro caso).
Además del ranking de algoritmos, consideramos el error medio total para tener una mejor idea de la precisión del algoritmo. Esta métrica es muy útil cuando los algoritmos no alcanzan el óptimo global o cuando todos los algoritmos obtienen tasas de error muy similares.

B. Resultados Obtenidos

A partir de los resultados obtenidos, se han construido las tablas III y IV para resumir la posición de cada algoritmo (de acuerdo con el ranking descrito en la Sección IV-A), así como el error medio total (eq. 2). En estas tablas, los algoritmos están ordenados según el ranking excepto nuestro algoritmo, que se muestra en la última fila. Los mejores resultados para la primera posición del ranking y el error medio total se muestran en negrita. Finalmente, mostramos el ranking de los algoritmos en una gráfica de barras para su mejor visualización (Figuras 4 y 5). Los algoritmos están ordenados de acuerdo al ranking.

TABLA III

RANKING Y ERROR MEDIO TOTAL OBTENIDO PARA LA DIMENSIÓN 10

	Ranking					
Algoritmo	*1º*	*2º*	*3º*	*4º*	*5º*	*EMT*
G-CMA-ES	**12**	4	2	0	3	**7,1**
L-CMA-ES	7	1	1	2	1	13,21
L-SaDE	6	2	5	2	1	8,46
DMS-L-PSO	5	4	4	4	2	8,58
BLX-GL50	5	3	3	2	3	8,96
DE	5	3	2	1	4	9,18
SPC-PNX	4	2	2	2	0	10,01
EDA	4	1	0	3	4	11,51
CoEVO	4	1	0	3	1	13,78
K-PCX	3	4	2	2	2	12,58
BLX-MA	3	1	3	2	1	10,75
EvolPDF-2	11	2	0	0	1	8,45

TABLA IV

RANKING Y ERROR MEDIO TOTAL PARA 30 DIMENSIONES.

	Ranking					
Algoritmo	*1º*	*2º*	*3º*	*4º*	*5º*	*EMT*
G-CMA-ES	7	7	1	2	1	8,32
L-CMA-ES	7	2	3	5	0	12,94
EDA	5	1	0	0	0	n/d
K-PCX	4	5	2	3	2	10,82
BLX-GL50	4	2	3	4	6	8,96
DMS-L-PSO	4	1	4	1	2	n/d
L-SaDE	3	2	3	2	2	n/d
SPC-PNX	2	3	2	1	4	9,74
DE	2	2	2	2	1	10,4
BLX-MA	2	0	6	0	5	10,46
CoEVO	0	1	0	2	0	19,26
EvolPDF-2	**10**	0	2	2	1	**7,07**

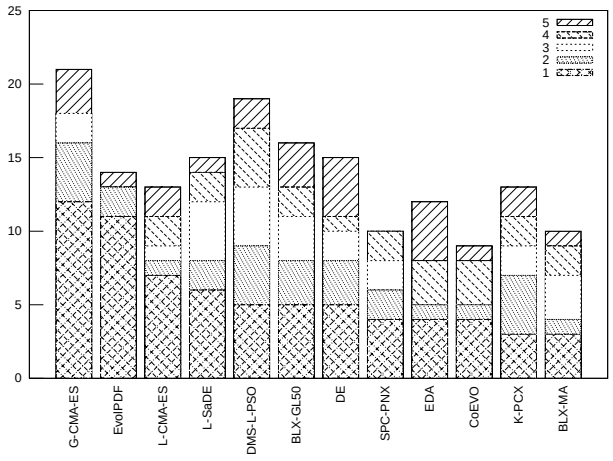

Fig. 4. Ranking de algoritmos analizados para dimension 10

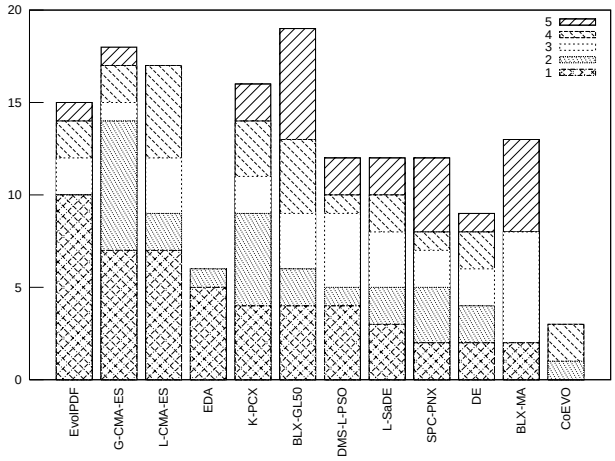

Fig. 5. Ranking de algoritmos analizados para dimension 30

Como podemos observar para la dimensión 10, nuestro algoritmo es el segundo mejor con respecto al número de primeras posiciones. Nuestro algoritmo es capaz de encontrar soluciones óptimas (un valor de función por debajo de 10e-8) en 7 funciones y el mejor resultado de todos los algoritmos analizados en 4 funciones. Esto sitúa a nuestro algoritmo en segunda posición y sólo es superado por G-CMA-ES. Si analizamos los resultados desde un punto de vista global de la primera a la quinta posición, nuestro algoritmo junto con G-CMA-ES, L-CMA-ES, DMS-L-PSO y L-SaDE son los que aparecen como mejores.

Para dimensión 30, nuestro algoritmo es el que más aparece en primera posición (10 veces). Los algoritmos G-CMA-ES y L-CMA-ES muestran también buenos resultados, con 7 primeras posiciones en ambos casos. Si analizamos los resultados desde un punto de vista global de la primera a la quinta posición, podemos decir que los cinco algoritmos que tienen mejores posiciones son: BLX-GL50 con 19 apariciones y G-CMA-ES 18 , L-CMA-ES con 17, K-PCX con 16 y el nuestro con 15.

Con respecto al error medio total, nosotros obtenemos el error más bajo en 30 variables y el segundo error más bajo en 10 variables. Esto muestra el buen alcance global de nuestro algoritmo.

El comportamiento de nuestro algoritmo se puede apreciar en la Figura 6. Es una representación gráfica que muestra la evolución del fitness (eje de ordenadas de la izquierda) y del número de migraciones (eje de ordenadas de la derecha) de la función número 18 para 30 dimensiones, que es una hibridación de funciones multimodales.

Esta representación es la media de 25 ejecuciones. En ella se puede apreciar cómo los valores del muestreo de la I_2 son mayores que los valores del muestreo de la I_1; esto es debido a que en cada población de la I_2, mantenemos 30 soluciones, lo cual proporciona más diversidad al algoritmo, que es lo que conviene siempre al principio. El muestreo de la I_1, es menor por su definición, ya que mantenemos 1 solución por población, lo cual implica que refine más y proporcione más intensificación a lo largo de la ejecución del algoritmo. Por ello, la I_1 exporta con mayor frecuencia a la I_2 más soluciones, intentando que haya un equilibrio entre la exploración y la explotación del espacio de búsqueda.

Si atendemos al diseño del algoritmo, es obvio que el radio de acción de la I_1 es menor que el radio de acción de la I_2 en el espacio de búsqueda. Al principio de las evaluaciones, la I_1 se dedicará a buscar buenas soluciones cercanas a la mejor encontrada hasta el momento, mientras que la I_2 explora el espacio de búsqueda. Cuando se produce una migración de la I_2 a la I_1, la I_1 comienza a realizar una búsqueda en el entorno de I_2, lo cual contribuye a mejorar las soluciones mantenidas por la I_2. Como se aprecia en la Figura 6, la I_1 (muestreo representado por un +) explota, pero no mejora con frecuencia las soluciones, de ahí que exista un trasvase de soluciones de I_2 (muestreo representado por un punto hueco) a I_1; una vez que la I_1 se encuentra en el espacio de búsqueda de I_2 (en las evaluaciones intermedias), comienza a explotar dicho espacio. La explotación se acentúa más en las evaluaciones finales, en donde la desviación típica es más pequeña. Esto permite disminuir la diversidad proporcionada por la I_2 e incrementar más la explotación de la I_1. De esta forma, el algoritmo en su comienzo es capaz de detectar qué zonas del espacio de búsqueda son más prometedoras, para después centrarse en la mejor y tratar de explotarla para así obtener una solución óptima.

Resumiendo, podemos decir que nuestro algoritmo obtiene muy buenos resultados entre los algoritmos analizados, siendo el segundo mejor en 10 variables y el mejor en 30 variables. Además, se puede observar que las funciones en donde nuestro algoritmo encuentra la solución óptima o de menor error están entre las número 15 y 23 (todas ellas son hibridación de funciones multimodales). Nuestro algoritmo es capaz de obtener soluciones óptimas para algunos problemas que no son capaces de resolver otros algoritmos (funciones 15, 21 y 23 en 30 variables y funciones 16, 21, 22 y 23 en 10 variables). El algorit-

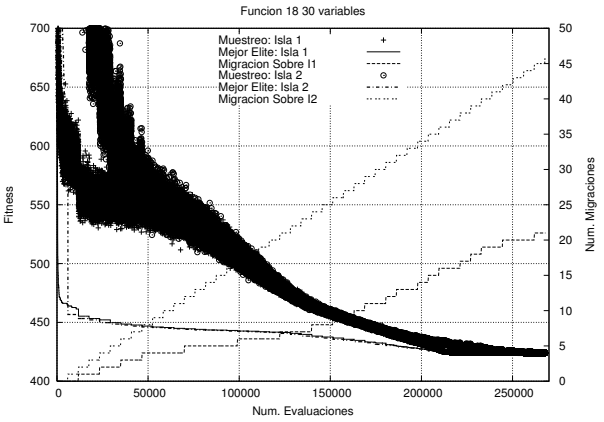

Fig. 6. Convergencia de la Función 18. Muestreo y solución elitista en función del número de evaluaciones.

mo se mantiene en buenas posiciones en las funciones multimodales de la 7 a la 14. Estos resultados nos sugieren que nuestro algoritmo, gracias al mecanismo incluido para prevenir óptimos locales, funciona mejor en problemas multimodales.

Destacar que un pequeño inconveniente que presenta el algoritmo es la inicialización de los parámetros al comienzo del algoritmo. Pero también hay que tener en cuenta, que un ajuste adecuado de los mismos, nos permitirá encontrar buenas soluciones. Otro inconveniente, es que para problemas unimodales no funciona tan bien como en los problemas multimodales.

V. Conclusiones y Trabajos Futuros

Hemos propuesto un algoritmo simple para optimización continua. Este algoritmo implica el uso de varias PDF para modelar la región que interesa a cada variable y las evoluciona hacia el centro de la región de búsqueda. Se incluyen algunos mecanismos para prevenir la convergencia prematura y los óptimos locales. El algoritmo no pertenece a ningún paradigma clásico de metaheurística, pero toma ideas de algunos de ellos como AG y EDA. Los resultados empíricos obtenidos nos permiten pensar que nuestro algoritmo tiene una gran capacidad, obteniendo los mejores resultados en problemas multimodales. Como trabajos futuros, nos proponemos la autoadaptación de más parámetros (por ejemplo, σ_0 y σ_f) y además realizar una caracterización más profunda del problema en el que el algoritmo tiene una ventaja competitiva.

Agradecimientos

Los autores quieren agradecer a Carlos García y Daniel Molina (ambos de la Universidad de Granada, España) su ayuda con las herramientas usadas en la comparación empírica. Este trabajo está soportado parcialmente por el proyecto del Ministerio de Ciencia y Tecnología num. TIC2003-00877 y por fondos FEDER.

Referencias

[1] E. Alba, J.M. Troya. *A survey of parallel distributed genetic algorithms.* Complexity, 4:303-346, 1999.

[2] A. Auger, S. Kern, N. Hansen. *A Restart CMA Evolution Strategy with Increasing Population Size.* In [6], pp. 1769-1776.

[3] A. Auger, S. Kern, N. Hansen. *Performance Evaluation of an Advanced Local Search Evolutionary Algorithm.* In [6], pp. 1777-1784.

[4] T. Bäck, M. Schütz, S. Khuri (1996). *Evolution Strategies: an Alternative Evolution Computation Method.* In J.M. Alliot, E. Lutton, E. Ronald, M. Schoenhauer, D. Snyers (Eds.), Artificial Evolution, pp. 3-20, Springer, Berlin.

[5] P.J. Ballester, J. Stephenson, J.N. Carter, K. Gallagher. *Real-Parameter Optimization Performance Study on the CEC-2005 benchmark with SPC-PNX.* In [6], pp. 498-505.

[6] K. Deb, D. Corne, Z. Michalewicz. *Special Session on Real-parameter Optimization.* IEEE Congress on Evolutionary Computation 2005, Edimburgh, UK, September 2005.

[7] C. García-Martínez, M. Lozano. *Hybrid Real-Coded Genetic Algorithms with Female and Male Differentiation.* In [6], pp. 896-903.

[8] F. Herrera, M. Lozano, J.L. Verdegay (1998). *Tackling real-coded genetic algorithms: operators and tools for the behavioural analysis.* Artificial Intelligence Reviews 12(4):265-319.

[9] P. Larrañaga, R. Etxeberria, J.A. Lozano, J. Peña (2000). *Optimization in continuous domains by learning and simulation of Gaussian networks.* In Proceedings of the Workshop in Optimization by Building and using Probabilistic Models, Genetic and Evolutionary Computation Conference, pp. 201-204, Las Vegas, Nevada, USA.

[10] P. Larrañaga, J.A. Lozano (Eds.) (2001). *Estimation of distribution algorithms. A new tool for evolutionary computation.* Kluwer Academic Publishers.

[11] J.J. Liang, P.N. Suganthan. *Dynamic Multi-Swarm Particle Swarm Optimizer with Local Search.* In [6], pp. 522-528.

[12] D. Molina, F. Herrera, M. Lozano. *Adaptive Local Search Parameters for Real-Coded Memetic Algorithms.* In [6], pp. 888-895.

[13] P.A. Moscato (1999). *Memetic algorithms: a short introduction.* In D. Corne, M. Dorigo, F. Glower, New Ideas in Optimization, pp. 219-234, McGraw-Hill, London.

[14] *Numerical Recipes in C. The Art of Scientific Computing. Second Edition.* William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery. Cambridge University Press.

[15] P. Posik. *Real Parameter Optimisation Using Mutation Step Co-evolution.* In [6], pp 872-879.

[16] A.K. Qin, P.N. Suganthan. *Self-adaptive Differential Evolution Algorithm for Numerical Optimization.* In [6], pp. 1785-1791.

[17] J. Rönkkönen, S. Kukkonen, K.V. Price. *Real-Parameter Optimization with Differential Evolution.* In [6], pp. 506-513.

[18] S. Rudolf, M. Köppen (1996). *Stochastic hill climbing with learning by vectors of normal distributions.* In 1st Online Workshop on Soft Computing, URL: http://www.bioele.nuee.nagoya-u.ac.jp/wscl/.

[19] M. Sebag, A. Ducoulombier (1998). *Extending population-based incremental learning to continuous search spaces.* Lecture Notes in Computer Science 1498: 418-427.

[20] A. Sinha, S. Tiwari, K. Deb. *A Population-Based, Steady-State Procedure for Real-Parameter Optimization.* In [6], pp. 514-521.

[21] P.N. Suganthan, N. Hansen, J.J. Liang, K. Deb, T.K. Chen, A. Auger, S. Tiwari (2005). *Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization.* Technical report, Nanyang Tecnological University, Singapore, May 2005, http://www.ntu.rdu.sg/home/EPNSugan.

[22] B. Yuan, M. Gallagher. *Experimental Results for the Special Session on Real-Parameter Optimization at CEC 2005: a Simple, Continuous EDA.* In [6], pp. 1792-1799.