Evolución de Funciones de Densidad de Probabilidad para Optimización Continua*

Jorge Casillas y Alejandro Rodríguez

Dpto. Ciencias de la Computación e Inteligencia Artificial,
ETS Ingeniería Informática, Universidad de Granada,
18071 – Granada
e-mail: casillas@decsai.ugr.es, vignote@yahoo.es

Resumen — En este artículo presentamos un nuevo algoritmo diseñado para la optimización en espacios continuos. Dicho algoritmo está basado en la evolución de funciones de densidad de probabilidad para adaptarse a las zonas más prometedoras del dominio de cada variable. Realizando estudios experimentales hemos observado como nuestro algoritmo obtiene buenos resultados de precisión y eficiencia en comparación con otros métodos.

Palabras clave — optimización continua, funciones de densidad de probabilidad, estimación de distribuciones, algoritmos evolutivos

I. INTRODUCCIÓN

La optimización en espacios continuos constituye una parte importante dentro de la ingeniería ya que a menudo no existen métodos formales para obtener diseños óptimos. Por ejemplo, supongamos que queremos diseñar un filtro dentro de un circuito eléctrico, la estructura general ya estará establecida, pero necesitaremos saber los valores de los componentes que pueden variar para así mejorar nuestro circuito. Por este motivo, se utilizan metaheurísticas de optimización para encontrar los mejores parámetros de un diseño. Así pues, un problema de optimización continua consiste en la búsqueda de los parámetros reales con los que obtenemos el mejor valor de la función objetivo que define la bondad de las soluciones al problema.

Además de numerosas técnicas de búsqueda local [12], existen algunas metaheurísticas que se aplican actualmente a optimización continua. Quizás las más destacadas por los buenos resultados obtenidos sean las Estrategias de Evolución [1] y los Algoritmos Genéticos (AGs) con Codificación Real [5]. Esta codificación parece particularmente natural cuando se abordan problemas de optimización donde el espacio de búsqueda es continuo. Un cromosoma es un vector de números reales cuyo tamaño es el mismo que la longitud del vector, el cual es la solución al problema. Su hibridación con otras técnicas generalmente mejoran la eficacia de la búsqueda. Entre estas, destaca por los buenos resultados obtenidos los algoritmos meméticos [9], donde los nuevos individuos generados son optimizados por una búsqueda local.

Otros algoritmos diseñados originalmente para optimización combinatoria pero que se aplican actualmente a optimización continua son los Algoritmos de Estimación de Distribuciones (en inglés *Estimation of Distribution Algorithms*, EDAs) [6]. En ellos no hay ni operadores de cruce ni de mutación. En lugar de esto, las nuevas poblaciones de individuos son muestreados a partir de una densidad de probabilidad. Esta función es estimada a partir de un conjunto de individuos generados en la iteración anterior, generalmente empleando redes bayesianas.

Relacionado con este tipo de algoritmo también encontramos al PBIL Continuo [11]. El algoritmo comienza con una densidad de probabilidad normal por variable. A partir de esta densidad, se genera un conjunto de soluciones que constituyen la población que es usada para actualizar las normales. En cada iteración, las medias de las normales son desplazadas mediante una combinación lineal de las mejores y peores soluciones de la población actual. Igualmente, se incluyen mecanismos para adaptar las desviaciones típicas de las normales.

En este trabajo proponemos una metaheurística basada en funciones de densidad de probabilidad (en inglés *probability density functions*, PDFs) que presenta ciertas relaciones con los algoritmos mencionados. La idea principal del algoritmo consiste en tener PDFs que representen las zonas prometedoras del espacio de búsqueda y que vayan evolucionando conforme avanza el algoritmo en función de las mejores soluciones exploradas.

Para ello, el artículo se organiza de la siguiente manera: en la sección II presentamos el algoritmo propuesto; en la sección III se relaciona el algoritmo propuesto con algunas de las metaheurísticas existentes; la sección IV presenta un análisis empírico del algoritmo; y, finalmente, la sección V muestra algunas conclusiones y trabajos futuros.

* Trabajo soportado por la Red HEUR TIC2002-10866-E y el Proyecto TIC2003-00877

II. DESCRIPCIÓN DEL ALGORITMO PROPUESTO

La idea principal de nuestro algoritmo consiste en tener una PDF para cada una de las variables del problema. Cada PDF nos indicará las zonas más prometedoras para cada variable dentro de su dominio, de forma que podremos utilizarlas como generadoras de soluciones prósperas. El esquema inicial de nuestro algoritmo es el siguiente:

1. Inicializar las PDFs con densidad uniforme
2. Repetir mientras (No condición_parada) {
   a. Muestrear las PDFs para generar N soluciones
   b. Evolucionar las PDFs a partir de las soluciones generadas

}

En las siguientes secciones vemos detalladamente el diseño del algoritmo.

A. *Representación de la PDF*

Para representar las PDFs necesitamos cumplir dos requisitos: representar cada PDF de tal manera que fácilmente podamos ir moldeándola a las nuevas zonas exploradas que nos han dado buenas soluciones, y poder muestrear fácilmente las PDFs, para así obtener nuevas soluciones.

Con este fin, representamos cada PDF mediante una mezcla de densidades normales (nótese que la desviación típica es común a todas ellas):

$$PDF_v = \sum_{i=1}^{k} \alpha_i^v \cdot \mathrm{N}\left(\mu_i^v, \sigma\right), \quad \sum_{i=1}^{k} \alpha_i^v = 1$$

$$\mathrm{N}(\mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

La ponderación de normales que definen la mezcla de la PDF de cada variable están normalizadas de tal manera que la integral de la PDF es igual a 1, y su peso se relaciona directamente con el valor objetivo de la solución. Para ello, realizamos el siguiente cálculo:

$$\alpha_i = \frac{f(S_i)}{\sum_{h=1}^{k} f(S_h)}$$

donde $f(S_i)$ es la función objetivo de la solución de la que procede la normal *i*. Suponemos que se maximiza dicha función.

B. *Inicialización*

Al principio de nuestro algoritmo cada PDF es inicializada como una mezcla de *k* densidades normales. La media de cada normal se define aleatoriamente a partir de una distribución uniforme. De este modo suponemos que no tenemos ninguna información del problema y, por tanto, cualquier valor del dominio de la variable tiene la misma probabilidad de contener una buena solución.

C. *Evolución de las PDFs*

Para evolucionar la PDF de cada variable a partir de una nueva solución, incluiremos una nueva normal en cada PDF con media igual al valor de la variable de esa solución y ponderación proporcional a la función objetivo. De esta forma, conseguimos dar mayor densidad a la PDF en torno al valor prometedor obtenido para la variable.

Sería completamente ineficiente incluir en la mezcla una nueva normal para cada una de las soluciones obtenidas. Por una parte, tendríamos que trabajar con un gran número de normales lo cual es bastante costoso computacionalmente y por otra parte el conservar más normales no hace que nuestro algoritmo funcione mejor, ya que zonas que obtuvimos hace tiempo y que no han sido tan prometedoras como otras obtenidas recientemente deberán desecharse.

Por tanto, limitaremos el número de normales que constituyen cada PDF a un valor fijo ***k***. Estas *k* normales procederán de las *k* mejores soluciones encontradas hasta el momento. De este modo, cada PDF representará siempre las regiones donde se obtuvieron las mejores soluciones. Cuando se incluye una nueva normal, se elimina previamente la peor contenida en la mezcla. Valores usuales de *k* rondarán entre 1 y 30. A mayor valor de *k*, más resolución de la PDF y, por tanto, mejor adaptación a densidades con formas complejas multimodales. Sin embargo, un valor alto de *k* provoca que la PDF evolucione más lentamente y ralentice la convergencia, aparte de hacer el algoritmo más costoso computacionalmente.

D. *Muestreo de las PDFs para Generar Nuevas Soluciones*

En cada iteración se generan ***N*** nuevas soluciones a partir de las PDFs. Aunque cualquier valor del dominio es posible, aquellas zonas con mayor acumulación de normales o mejor valor objetivo correspondiente tendrán mayor probabilidad de ser escogidas para construir la nueva solución.

Para implementar esta idea, se realiza el siguiente proceso para cada variable:

1. En primer lugar, escogeremos probabilísticamente una de las normales que constituyen la PDF correspondiente (cuanto mayor sea la ponderación de una normal, mayor probabilidad tendrá de ser escogida). Esto se

puede implementar construyendo una ruleta con los valores de ponderación y girándola una vez.

2. A continuación, generaremos un número aleatorio con densidad normal centrada en la media de la normal escogida en el paso anterior y con desviación típica la correspondiente en la iteración actual (ver siguiente sección). El valor obtenido será el valor que le demos a la variable correspondiente.

En caso de generar un valor fuera del rango permitido para cada variable, se iguala este valor al extremo correspondiente del intervalo. Por tanto, se realiza una saturación en los extremos para compensar la falta de acumulación de densidad en esas regiones.

E. Convergencia de la Desviación Típica

En toda metaheurística nos interesa la existencia de dos factores: exploración y explotación. Al principio de un algoritmo siempre nos interesará que haya mayor exploración para favorecer la diversificación en el espacio de búsqueda. Conforme vamos alcanzando el final del algoritmo, nos interesa que haya mayor explotación para así acercarnos al óptimo local o global de la zona que haya resultado más prometedora.

En nuestro caso, también regularemos este equilibrio entre diversificación e intensificación. Teniendo en cuenta que los valores que obtenemos para asignar a las variables de las nuevas soluciones proceden de números aleatorios de densidades normales, nos interesará que al principio del algoritmo el valor de la desviación típica de estas distribuciones sea mayor, para así permitir explorar nuevos valores más alejados de la media (cuya zona ya fue explorada). Sin embargo, al final del algoritmo nos interesará que este valor sea menor para que así los números aleatorios obtenidos estén más próximos a las medias de las normales, y así centrarnos tan solo en los valores más próximos a éstas.

Hay muchas formas de reducir el valor de la desviación típica a lo largo del algoritmo: lineal, exponencial, esquema de Cauchy, etc. En nuestro problema, es deseable que se den dos propiedades: que el valor inicial de la desviación típica no se reduzca durante un cierto número de iteraciones con el objetivo de permitir configurar adecuadamente las PDFs iniciales y que podamos controlar el valor inicial y final de la desviación típica para así regular mejor las fases de inicialización y refinamiento de las PDFs.

Para ello, hemos definido un esquema de convergencia de la desviación típica propio atendiendo a la siguiente fórmula:

$$C(x)=\begin{cases}\sigma_0 & si\ \ x\le\alpha\\ (\sigma_0-\sigma_f)\left(1-2\left(\frac{x-\alpha}{\beta-\alpha}\right)^2\right)+\sigma_f & si\ \ \alpha<x\le\frac{\alpha+\beta}{2}\\ (\sigma_o-\sigma_f)\left(2\left(\frac{x-\beta}{\beta-\alpha}\right)^2\right)+\sigma_f & si\ \ \frac{\alpha+\beta}{2}<x\le\beta\\ \sigma_f & si\ \ x>\beta\end{cases}$$

siendo σ_0 y σ_f los valores inicial y final de la desviación típica, y α y β dos constantes fijadas en función del número de evaluaciones a los siguientes valores:

α = Evaluaciones · 0,2
β = Evaluaciones · 0,8

Con esta fórmula, regulamos el decremento de la desviación típica para distinguir entre tres etapas durante la evolución de las PDFs:

1. *Inicialización*: En un primer momento (20% de evaluaciones) se mantiene la desviación típica y se utiliza el algoritmo para configurar un conjunto de PDFs 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 el ajuste final de las PDFs manteniendo una desviación típica baja y constante.

La figura 1 muestra un ejemplo gráfico del proceso de convergencia de la desviación típica.

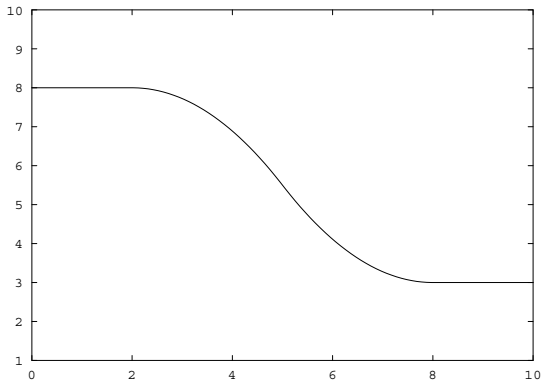

Figura 1. Ejemplo de convergencia de la desviación típica (10 evaluaciones, σ_0=8, σ_f=3)

F. Evolución de varias PDFs en paralelo

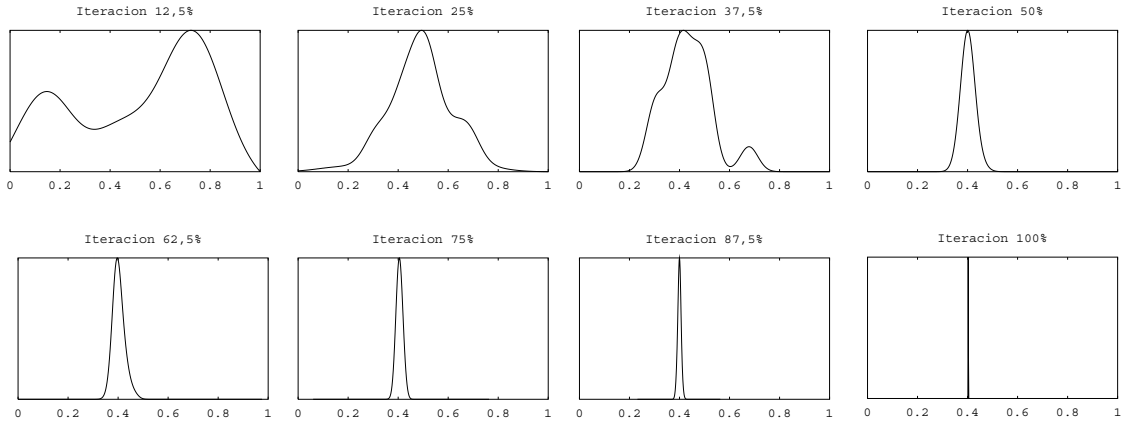

Figura 2. Ejemplo de caída en óptimo local durante la evolución de la PDF de una variable

Realizando experimentaciones preliminares comprobamos que nuestro algoritmo era capaz de encontrar regiones de soluciones prometedoras rápidamente. Sin embargo, sufría un riesgo de caer en óptimos locales en problemas multimodales. Por ejemplo, en la figura 2 mostramos la evolución de la PDF de una determinada variable en el problema Rastrigin (ver sección IV para una descripción del problema). Como podemos observar, el algoritmo converge rápidamente a un óptimo local situado en 0,4 en lugar de converger al óptimo global situado en 0,5.

Para resolver este problema, y dado que nuestro algoritmo es rápido en su convergencia, nos propusimos evolucionar varias PDFs en paralelo. Periódicamente, recombinamos las distintas PDFs de cada variable mediante un operador de cruce consistente en recoger todas las normales y distribuirlas aleatoriamente entre las diferentes PDFs. En este artículo, la experimentación se realizó con recombinaciones en el 50%, 66% y 83% de las evaluaciones totales.

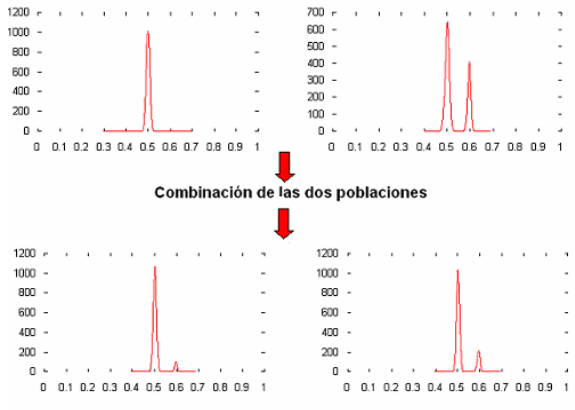

Figura 3. Combinación de dos poblaciones

La figura 3 muestra un ejemplo real de su funcionamiento para el caso de dos poblaciones en el problema Rastrigin. Podemos observar cómo en la PDF de la segunda población, en el cuadrante superior derecho de la figura 3, la mezcla presenta alta densidad alrededor del valor 0,6 (óptimo local). Si esta PDF continuara evolucionando, existe un alto riesgo de converger a ese valor. Sin embargo, al recombinarla con otra PDF cuya densidad sí se centra alrededor de 0,5 (óptimo global), cuadrante superior izquierdo de la figura 3, la influencia del óptimo local se diluye en las PDFs descendientes y se evita derivar a esa región.

G. Aplicación Secuencial de una Búsqueda Local

Por último, empleamos una búsqueda local al final del algoritmo (secuencialmente) para refinar la mejor solución encontrada. En nuestro caso, no nos preocupa que la búsqueda local converja rápidamente a un óptimo local dado que con la fase anterior se logra focalizar la región de búsqueda alrededor de un óptimo prometedor. La búsqueda local utilizada es la propuesta por Solis y Wets [12]. En la experimentación de este artículo aplicamos la búsqueda local durante el último 10% de evaluaciones.

El comportamiento de esta búsqueda local es la de proyectar la mejor solución multivariable sobre cada eje, para entonces buscar en dichas direcciones valores próximos al actual. Se emplea una densidad normal cuya desviación típica va reduciéndose conforme se mejoran los resultados, o ampliándose en caso contrario. Igualmente, se va desplazando la media en cada eje atendiendo a si se mejora o no la solución. Hemos incluido esta búsqueda local y no otra debido a que la filosofía de proyectar en los ejes y buscar en las direcciones prometedoras coincide con el algoritmo propuesto, como veremos en el siguiente apartado.

H. Análisis del Algoritmo

En la figura 4 se representa en pseudo-código la estructura final del algoritmo propuesto. Podemos

1. Inicializar k (número de normales en las PDF), σ_0 (valor inicial de la desviación típica), σ_f (valor final), N (número de soluciones generadas en cada iteración) y *P* (número de poblaciones)
2. Sea $f(\cdot)$ la función objetivo (suponemos maximización) y *n* el número de variables del problema
3. Sea $PDF_{(p,v)} = \sum_{i=1}^{k} \alpha_i^{(p,v)} \cdot \mathrm{N}\left(\mu_i^{(p,v)}, \sigma\right)$
4. $\sigma \leftarrow \sigma_0$
5. Para *p* = 1 hasta *P*
   a. Generar *k* soluciones $S_i^p = (x_i^{(p,1)}, \ldots, x_i^{(p,n)})$ aleatoriamente con densidad uniforme
   b. Inicializar cada $PDF_{(p,v)}$ con $\mu_i^{(p,v)} = x_i^{(p,v)}$ y $\alpha_i^{(p,v)} = \dfrac{f(S_i^p)}{\sum_{j=1}^{k} f(S_j^p)}$
6. $iter \leftarrow 1$
7. Repetir durante el 90% de las evaluaciones
   a. Para *j* = 1 hasta *N*
      Para *p* = 1 hasta *P*
      Para *v* = 1 hasta *n*
      $y_j^{(p,v)}$ = Muestrear ($PDF_{(p,v)}$)
      Sea $R_j^p = (y_j^{(p,1)}, \ldots, y_j^{(p,n)})$
   b. Sea R_{mejor}^p *t.q.* $f(R_{mejor}^p) \geq f(R_j^p), \forall j \in \{1, \ldots, k\},\ j \neq mejor$
   c. Sea S_{peor}^p *t.q.* $f(S_{peor}^p) \leq f(S_i^p), \forall i \in \{1, \ldots, k\}, i \neq peor$
   d. Si $f(R_{mejor}^p) > f(S_{peor}^p)$
      $S_{peor}^p \leftarrow R_{mejor}^p$
   e. Evolucionar cada $PDF_{(p,v)}$ con $\mu_{peor}^{(p,v)} = y_{mejor}^{(p,v)}$ y $\alpha_i^{(p,v)} = \dfrac{f(S_i^p)}{\sum_{j=1}^{k} f(S_j^p)}, \forall i \in \{1, \ldots, k\}$
   f. $\sigma \leftarrow C(iter)$
   g. $iter \leftarrow iter + 1$
   h. Si (condición de recombinación)
      i. Sea $\mathrm{M}_v = \left\{\mu_i^{(p,v)},\ \forall p \in \{1, \ldots, P\}, \forall i \in \{1, \ldots, k\}\right\},\ \forall v \in \{1, \ldots, n\}$
      ii. Para *p* = 1 hasta *P*
      Para v = 1 hasta *n*
      Para *i* = 1 hasta *k*
      $\mu_i^{(p,v)} \leftarrow \mu_v \in \mathrm{M}_v$
      $\mathrm{M}_v \leftarrow \mathrm{M}_v - \{\mu_v\}$
8. Durante el restante 10% de evaluaciones, aplicar la búsqueda local de Solis y Wets sobre la mejor solución encontrada

Figura 4. Pseudo-código del algoritmo propuesto

hacer algunas reflexiones sobre el comportamiento del algoritmo que ayudarán a comprender mejor su funcionamiento y cualidades.

El hecho de considerar en cada mezcla de normales (una PDF por variable) aquellas que han obtenido la mejor calificación de la función objetivo provoca que, en realidad, se trabaje siempre con un conjunto élite de soluciones multivariable. En efecto, dado que se considera un criterio global multivariable para mantener las *k* mejores soluciones en la PDF de cada variable, todas las normales provienen de las *k* mejores soluciones encontradas.

Sin embargo, la elección probabilística de una densidad de probabilidad concreta se hace independiente para cada variable. Es decir, no se escogen las *n* normales (una por variable) correspondiente a una de las *k* mejores soluciones. Por tanto, nuestro algoritmo considera las *k* mejores soluciones exploradas como indicadores de trayectorias de búsqueda, dado que se realiza una proyección por eje de dichas soluciones. De este modo, la exploración del espacio continuo se realiza en direcciones alineadas con los ejes de las variables, en las zonas en las que se van obteniendo las mejores soluciones.

Esta simplificación en la búsqueda favorece a aquellos problemas en los cuales el gradiente quede alineado a los ejes, mientras que no será tan efectiva cuando suceda lo contrario.

III. Semejanzas y Diferencias con Otros Algoritmos

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

A. *Algoritmos de Estimación de Distribuciones*

Nuestro algoritmo presenta claras semejanzas a los EDA [7]. En efecto, 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 mediante *kernels* normales univariantes. Este proceso simplifica a nuestro algoritmo evitando la necesidad de emplear otras técnicas de estimación de densidades de probabilidad más complejas. Existen otros métodos de uso de estimadores de densidad de probabilidad basados en *kernels* [3] o en un modelo de mezcla adaptativo [4].

El concepto de cruce de PDFs para abordar la multimodalidad no parece encuadrarse dentro de esta familia de metaheurísticas.

B. *Otros Modelos Probabilísticos*

Tanto nuestro algoritmo como el PBIL Continuo [11] como el algoritmo SHCLVND [10] utilizan funciones de densidad de probabilidad para encontrar nuevas soluciones. Sin embargo, el modo de construir y modificar las PDFs difiere en nuestro algoritmo del resto.

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

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

Otro de los inconvenientes del PBIL Continuo viene provocado por el desplazamiento de la densidad de probabilidad, en la cual se provoca un acercamiento hacia las soluciones más prometedoras y alejamiento de las soluciones peores. Aunque este criterio pueda tener sentido en optimización combinatoria (que es el origen del algoritmo PBIL), no se comporta bien en problemas de optimización continua multimodales, donde el óptimo global suele estar rodeado de soluciones mediocres. Así, al alejarse de las peores soluciones según su función de desplazamiento de la media de la normal, podría estar alejándose del óptimo global.

Nuestro algoritmo coincide con SHCLVND en el uso de una única desviación típica global que va reduciéndose conforme avanza el algoritmo. Sin embargo, mientras que 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 exploración-explotación.

C. *Algoritmos Genéticos y Estrategias de Evolución*

Podemos pensar que nuestro algoritmo tiene algunas similitudes con los AGs. En estos, existe una población inicial (conjunto de cromosomas) y a partir de la realización de operaciones de selección, cruce y mutación vamos generando las siguientes poblaciones.

Podemos considerar a cada una de las normales de cada PDF como un cromosoma, y a cada PDF como una población (de esta manera tendríamos una población para cada variable). La selección del AG correspondería con la primera parte del muestreo de la PDF, es decir, cuando escogemos probabilísticamente en función del valor objetivo una normal de la mezcla que compone la PDF. La mutación del AG correspondería a la segunda parte del muestreo de la PDF, es decir, cuando generamos un número aleatorio procedente de la densidad normal escogida. Análogamente, podríamos relacionarlo con las Estrategias de Evolución (1,1) [1] cuando empleamos una única normal por PDF.

En nuestro caso, para algunos problemas (aquellos con alta probabilidad de caer en óptimos locales) podemos trabajar con varias PDFs por variable 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 para así obtener nuevos cromosomas que hereden las propiedades de los padres. Por tanto, cada una de las PDF de nuestro algoritmo podría considerarse como un cromosoma, y cada normal que la constituye un gen del cromosoma. De esta manera cruzamos varios cromosomas para que sus descendientes hereden sus genes y así formen nuevas PDFs.

En el caso de evolucionar varias PDFs por variable y emplear recombinación periódica, también podríamos considerar nuestro algoritmo como un algoritmo memético [9]. En efecto, cada cromosoma sería una PDF y cada uno de estos cromosomas se optimizaría mediante nuestro algoritmo de evolución de PDFs. Tras la optimización, se realiza el cruce del meta-AG con probabilidad 1 y reemplazo directo.

IV. ANÁLISIS EMPÍRICO

En los dos apartados siguientes presentamos algunos resultados empíricos; primero analizando las diferentes componentes de nuestro algoritmo y los valores de los parámetros, y posteriormente comparando los resultados de nuestra propuesta con los obtenidos por otros algoritmos.

Los problemas sobre los que hemos realizado la experimentación son:

- Sphere: Unimodal y separable

$$f(\vec{x}) = \sum_{i=1}^{p} x_i^2 \qquad x_i \in [-5{,}12,\ 5{,}12]$$

- Rosenbrock: Unimodal y no separable

$$f(\vec{x}) = \sum_{i=1}^{p-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$$
$$x_i \in [-2{,}048,\ 2{,}048]$$

- Schwefel: Unimodal y no separable

$$f(\vec{x}) = \sum_{i=1}^{p} \left(\sum_{j=1}^{i} x_j\right)^2$$
$$x_i \in [-65{,}536,\ 65{,}536]$$

- Rastrigin: Multimodal y separable

$$f(\vec{x}) = 10p + \sum_{i=1}^{p} (x_i^2 - 10\cos(2\pi x_i))$$
$$x_i \in [-5{,}12,\ 5{,}12]$$

- Ackley: Multimodal y separable

$$f(\vec{x}) = 20 + e - 20e^{-0{,}2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)}$$
$$x_i \in [-30,\ 30]$$

- Griewangk: Multimodal y separable

$$f(\vec{x}) = 1 + \sum_{i=1}^{p} \frac{x_i^2}{400p} - \prod_{i=1}^{p} \cos\left(\frac{x_i}{\sqrt{i}}\right)$$
$$x_i \in [-600,\ 600]$$

En nuestro algoritmo, hemos fijado el número de soluciones muestreadas en cada iteración a N=20 para toda la experimentación. Igualmente, fijamos los valores inicial y final de la desviación típica (σ_0,σ_f) según el problema: (10^{-2},10^{-8}) en Sphere; (10^{-3},10^{-4}) en Rosenbrock; (10^{-3},10^{-8}) en Schwefel; (10^{-1},10^{-4}) en Rastrigin; (10^{-2},10^{-8}) en Ackley; y (10^{-2},10^{-4}) en Griewangk.

A. *Justificación del diseño del algoritmo*

En esta sección justificamos experimentalmente dos de las componentes de nuestro algoritmo: el uso de una mezcla de densidades normales y el uso de recombinación de PDFs. Igualmente, comprobaremos su efecto al variar el parámetro k, que define el número de normales a considerar en la mezcla, y el parámetro P, que define el número de PDFs por variable que evolucionan en paralelo.

En las tablas 1 y 2 vemos los resultados medios para 40 ejecuciones obtenidos en dos problemas distintos, Rosenbrock (unimodal) y Rastrigin (multimodal), con 25 variables y 100.000 evaluaciones.

Como era de esperar, el hecho de introducir varias normales permite diversificar más y así en problemas donde hay múltiples óptimos locales, como por ejemplo en Rastrigin, se pueda adaptar mejor a las distintas zonas prometedoras. Sin embargo, en problemas unimodales vemos como hay que reducir la diversificación para obtener mejores soluciones.

Tabla 1. Resultados en Rosenbrock para distintas combinaciones de los parámetros *k* y *P*

	P				
k	**1**	**2**	**3**	**4**	**5**
1	**1,58e+1**	1,60e+1	2,98e+1	2,58e+1	3,34e+1
5	2,10e+1	2,72e+1	5,05e+1	6,29e+1	5,53e+1
10	2,67e+1	2,90e+1	4,68e+1	6,43e+1	5,52e+1
20	2,67e+1	3,99e+1	6,47e+1	3,85e+1	4,94e+1
30	4,85e+1	4,68e+1	3,35e+1	4,58e+1	3,80e+1

Tabla 2. Resultados en Rastrigin para distintas combinaciones de los parámetros *k* y *P*

	P				
k	**1**	**2**	**3**	**4**	**5**
1	8,03e+1	6,75e+1	6,74e+1	6,72e+1	6,24e+1
5	2,83e+1	1,31e+1	9,03e+0	6,87e+0	4,75e+0
10	1,67e+1	7,98e+0	4,53e+0	2,64e+0	1,59e+0
20	1,05e+1	3,73e+0	1,19e+0	3,73e-1	2,24e-1
30	6,17e+0	1,89e+0	3,23e-1	1,74e-1	**4,97e-2**

Por otro lado, el uso de varias PDFs por variables que periódicamente se recombinan también permite favorecer la exploración en espacios multimodales, dado que se reduce el riesgo de caer en óptimos locales. Igualmente, en problemas multimodales interesará evolucionar varias PDFs en paralelo mientras que en problemas unimodales bastará con un única PDF por variable. Por tanto, para el resto de la experimentación fijamos los parámetros de la siguiente forma: k=1 y P=1 en Sphere, Rosenbrock y Schwefel; y k=30 y P=5 en Rastrigin, Ackley y Griewangk.

Finalmente, presentamos dos gráficas de convergencia del algoritmo en los problemas Rosenbrock (figura 4) y Rastrigin (figura 5) donde se puede observar la mejora paulatina de la solución durante el primer 20% de evaluaciones (donde la desviación típica permanece constante), la mejora significativa durante el 60% restante (donde se reduce progresivamente la desviación típica) y la última fase de optimización local en el último 10%, donde se mejoran ligeramente los resultados obtenidos. Nuevamente, las gráficas presentan los

Tabla 3. Comparativa frente a AGs y Algoritmos Meméticos [8] (25 variables y 100.000 evaluaciones)

	Sphere		Rosenbrock		Schwefel		Rastrigin		Griewangk	
Algoritmo	$\overline{x}$	**Mejor**	$\overline{x}$	**Mejor**	$\overline{x}$	**Mejor**	$\overline{x}$	**Mejor**	$\overline{x}$	**Mejor**
CHC	5,8e-031	3,1e-031	1,9e+1	1,7e+1	2,0e-02	1,5e-03	1,6e+01	7,0e+00	6,5e-03	42%
CHC-SW	8,5e-063	3,1e-065	1,5e+1	7,3e+0	1,2e-01	1,6e-02	4,5e+01	2,9e+01	3,4e-03	6%
AG-SW	2,9e-120	1,4e-322	4,3e+0	2,4e-3	4,1e-09	1,4e-26	7,6e+00	**40%**	4,9e-04	5,6e-16
G3	1,0e-099	**7,6e-111**	8,2e+0	**3,4e-5**	4,6e-02	1,3e-05	4,4e+01	2,0e+01	1,7e-02	20%
FC	1,5e-006	3,7e-007	2,3e+1	2,1e+1	1,1e+02	4,5e+01	5,5e+00	2,2e+00	3,5e-04	2,2e-05
RCMA-XHC	**5,2e-119**	1,2e-124	**1,4e+0**	8,6e-7	1,2e-10	1,2e-13	8,9e-01	40%	5,6e-03	**58%**
$PBIL_C$	1,0e-011	5,4e-012	2,1e+1	1,8e+1	1,4e+02	9,8e-01	1,0e+01	4,9e+00	4,1e-04	1,6e-08
Evol-PDF	6,2e-025	1,1e-025	1,6e+01	9,9e+0	**1,1e-10**	6,6e-10	**5,0e-02**	1,4e+00	**1,0e-14**	2,6e-15

resultados medios en cada evaluación para 40 ejecuciones distintas, aplicado sobre 25 variables y 100.000 evaluaciones.

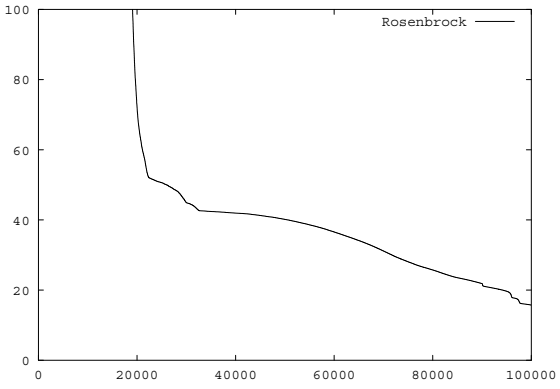

Figura 4. Gráfica de convergencia en Rosenbrock

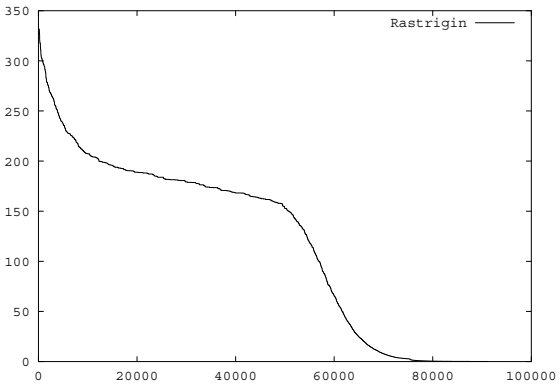

Figura 5. Gráfica de convergencia en Rastrigin

B. Comparativa con otros métodos

Hemos realizado un estudio comparativo de nuestro algoritmo frente a otros que han demostrado un buen comportamiento en optimización continua. Se incluyen varios AGs con codificación real (extraídos de la experimentación realizada en [8]), varios EDAs aplicados a optimización continua (extraídos de la experimentación realizada en [2]) y el algoritmo PBIL continuo [11].

1) Comparativa con Algoritmos Genéticos con Codificación Rea y PBIL Continuo

En esta experimentación consideramos cada problema con 25 variables y un número máximo de evaluaciones de 100.000. Hemos realizado 40 ejecuciones distintas en cada problema.

En la tabla 3 podemos ver los resultados de los experimentos. Nombramos como Evol-PDF al algoritmo presentado en este trabajo. Para cada problema se incluyen dos columnas. En la primera columna se indica la media de la función objetivo para las soluciones obtenidas en las distintas ejecuciones, mientras que la segunda columna muestra la mejor solución encontrada. En el caso de que se llegue a encontrar el óptimo global, en lugar de mostrar dicho valor, indicaremos el porcentaje de veces que se ha encontrado el óptimo. En negrita aparece el mejor resultado encontrado en cada problema.

Como podemos comprobar en la tabla de resultados, nuestro algoritmo obtiene la mejor media entre los algoritmos de comparación considerados en los problemas Schwefel, Rastrigin y Griewangk. Por el contrario, Rosenbrock es el problema más difícil de resolver para nuestro algoritmo, aunque aún así mejoramos a cuatro de los algoritmos de comparación (CHC, CHC-SW, FC y $PBIL_C$). Quizás la dificultad de este problema para nuestro algoritmo sea el hecho de que el gradiente de la función no esté alineado con los ejes de las variables. En el problema Sphere, aunque la solución obtenida con nuestro algoritmo no es tan próxima a la mejor solución encontrada, este caso no es de gran importancia. Démonos cuenta que este problema es muy simple ya que es unimodal y separable. Por tal motivo, este problema no refleja la calidad de los algoritmos para localizar la zona del óptimo, sino que muestra únicamente si el algoritmo converge apropiadamente o no.

Tabla 4. Comparativa frente a EDAs [2]
(10 variables, 301.850 evaluaciones, parada si la distancia al óptimo es menor que 1e-6)

	Sphere		Rosenbrock		Ackley		Griewangk	
Algoritmo	$\overline{x} \pm \sigma$	**Eval**	$\overline{x} \pm \sigma$	**Eval**	$\overline{x} \pm \sigma$	**Eval**	$\overline{x} \pm \sigma$	**Eval**
$UMDA_C$	6,7e-06 ± 1,3e-06	74164	8,7e+0 ± 3,8e-2	301850	7,9e-6 ± 1,2e-6	114944	6,2e-2 ± 1,9e-2	301850
$MIMIC_C$	7,3e-06 ± 2,1e-06	74964	8,7e+0 ± **1,7e-2**	301850	8,8e-6 ± 9,0e-7	114744	7,4e-2 ± 2,9e-2	301850
$EGNA_{BIC}$	2,6e-05 ± 3,7e-05	77162	8,8e+0 ± 0,2e+0	268067	5,2e+0 ± 4,5e+0	229086	3,9e-2 ± 2,4e-2	301850
$EGNA_{BGe}$	7,2e-06 ± 1,8e-06	74764	8,7e+0 ± 5,9e-2	164519	7,9e-6 ± 1,4e-6	113944	7,6e-2 ± 2,9e-2	301850
$EGNA_{EE}$	7,4e-06 ± 2,0e-06	**73964**	8,7e+0 ± 2,2e-2	301850	7,5e-6 ± 1,7e-6	118542	5,7e-2 ± 3,8e-2	301850
$EMNA_{GLOBAL}$	7,3e-06 ± 2,2e-06	94354	8,7e+0 ± 4,3e-2	289056	8,9e-6 ± 6,9e-7	119141	5,1e-2 ± 1,7e-2	301850
Evol-PDF	**8,7e-13 ± 6,9e-13**	90287	**7,7e-1** ± 9,6e+0	**99999**	**1,2e-6 ± 4,1e-7**	**90447**	**1,8e-4 ± 7,3e-3**	**90896**

Tabla 5. Comparativa frente a EDAs [2]
(50 variables, 301.850 evaluaciones, parada si la distancia al óptimo es menor que 1e-6)

	Sphere		Rosenbrock		Ackley		Griewangk	
Algoritmo	$\overline{x} \pm \sigma$	**Eval**	$\overline{x} \pm \sigma$	**Eval**	$\overline{x} \pm \sigma$	**Eval**	$\overline{x} \pm \sigma$	**Eval**
$UMDA_C$	8,9e-06 ± 8,4e-07	211495	4,9e+1 ± **4,0e-3**	301850	9,1e-6 ± 3,1e-7	296853	9,0e-6 ± 9,4e-7	177912
$MIMIC_C$	9,9e-06 ± 9,6e-07	211695	4,9e+1 ± 1,1e-2	301850	9,6e-6 ± 3,8e-7	295653	9,1e-6 ± 8,8e-7	177912
$EGNA_{BIC}$	1,2e-03 ± 7,7e-04	263869	5,0e+1 ± 2,3e+0	301850	2,0e-2 ± 7,5e-3	288257	1,7e-4 ± 6,8e-5	250475
$EGNA_{BGe}$	8,7-06 ± 1,3e-06	204299	4,9e+1 ± 0,1e+0	275663	8,7e-6 ± 3,8e-7	282060	**8,7e-6 ± 7,7e-7**	**173514**
$EGNA_{EE}$	8,3e-06 ± 1,0e-06	209496	4,9e+1 ± 1,1e-2	301850	6,8e+0 ± 0,3e+0	301850	9,2e-6 ± 5,9e-7	175313
$EMNA_{GLOBAL}$	8,5e-06 ± 1,4e-06	247477	5,0e+1 ± 0,5e+0	296253	9,6e-6 ± 2,4e-7	291255	8,8e-6 ± 1,0e-6	216292
Evol-PDF	**9,8e-13 ± 9,4e-14**	**181293**	**4,3e+1** ± 9,9e+1	**199999**	**5,6e-7 ± 3,5e-8**	**182309**	3,7e-5 ± 1,5e-3	186869

Por otro lado, también podemos comprobar que nuestro algoritmo se caracteriza por ser robusto frente a la inicialización de la secuencia pseudo-aleatoria. En efecto, en cada problema, el valor objetivo de la mejor solución de las distintas ejecuciones se aproxima mucho a la media obtenida. Esto hace al algoritmo ser más fiable con pocas ejecuciones y tener menos riesgo de caer en óptimos locales en una ejecución concreta.

2) Comparativa con Algoritmos de Estimación de Distribuciones

En este segundo estudio comparativo frente a varios EDAs consideramos 10 y 50 variables a optimizar.

En las tablas 4 y 5 se ven los resultados obtenidos frente a otros algoritmos EDA. Presentamos dos columnas para cada problema. En la primera aparece la media de los valores objetivo obtenidos y la varianza correspondiente. La segunda columna indica el número medio de evaluaciones realizadas. En negrita aparece el mejor resultado encontrado en cada problema.

El criterio de parada de los experimentos es ejecutar el algoritmo durante 301.850 evaluaciones o bien hasta que la distancia de la mejor solución encontrada a la solución óptima sea inferior a 1e-06, tal como se describe en [2]. Sin embargo, dado que nuestro algoritmo se caracteriza por obtener buenas soluciones con rapidez, consideraremos un número máximo de evaluaciones de 100.000 (para 10 variables) y 200.000 (para 50 variables) en nuestro caso. Nuevamente, hemos realizado 40 ejecuciones distintas de nuestro algoritmo en cada problema. El resto de resultados, extraídos de [2], se obtienen a partir de 10 ejecuciones por algoritmo y problema.

Como podemos observar, en general nuestro algoritmo obtiene mejores resultados de eficiencia que el resto. Además, hemos de tener en cuenta que nuestros resultados se obtienen, en general, en menor número de evaluaciones.

V. Conclusiones y Trabajos Futuros

Las características más notables del algoritmo de optimización continua propuesto se centran en ser un algoritmo simple, fácilmente implementable y con cálculos sencillos computacionalmente económicos. El algoritmo no se encuadra en ningún paradigma clásico de metaheurística, aunque sí está relacionado con algunos de ellos.

Por su diseño, parece adaptarse mejor a problemas donde el gradiente que lleva al óptimo global esté alineado con los ejes de las variables. Igualmente, incluye mecanismos eficaces para abordar problemas altamente multimodales. Los resultados obtenidos sobre algunos problemas de referencia parecen prometedores y animan a continuar la investigación.

Como trabajo futuro, nos proponemos adaptar el algoritmo para seguir direcciones de búsqueda no alineadas, mejorando bien el proceso de muestreo de la PDF o bien el proceso de búsqueda local posterior. Igualmente, se hace necesario incluir mecanismos de auto-adaptación de los parámetros más relevantes para hacer al algoritmo más autónomo y robusto frente a problemas desconocidos.

REFERENCIAS

[1] T. Bäck, M. Schütz, S. Khuri (1996). Evolution strategies: an alternative evolution computation method. En J.M. Alliot, E. Lutton, E. Ronald, M. Schoenhauer, D. Snyers (Eds.), Artificial Evolution, pp. 3-20, Springer, Berlin.

[2] E. Bengoetxea, T. Miquélez, P. Larrrañaga, J.A. Lozano (2002). Experimental results in function optimization with EDAs in continuous domain. En [7], págs. 181-194.

[3] P.A.N. Bosman, D. Thierens (2000). IDEAs based on the normal kernels probability density function. Utrecht University Technical Report UU-CS-2000-11.

[4] M.R. Gallagher, M. Frean, T. Downs (1999). Real-valued evolutionary optimization using a flexible probability density estimator. Proceedings of Genetic and Evolutionary Computation Conference (GECCO 99), pp. 840-846. Morgan Kaufmann.

[5] 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.

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

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

[8] M. Lozano, F. Herrera, N. Krasnogor, D. Molina (2003). Real-coded memetic algorithms with crossover hill-climbing. Evolutionary Computation, por aparecer.

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

[10] S. Rudlof, M. Köppen (1996). Stochastic hill climbing with learning by vectors of normal distributions. En 1st Online Workshop on Soft Computing, http://www.bioele.nuee.nagoya-u.ac.jp/wsc1/.

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

[12] F.J. Solis, R.J.-B. Wets (1981). Minimization by random search techniques. Mathematical Operations Research, 6: 19-30.