Reencadenamiento de trayectorias para optimización global

Francisco Gortázar, Abraham Duarte
Dept. de Ciencias de la Computación
Universidad Rey Juan Carlos
ETS Ingeniería Informática
28933 Madrid
{francisco.gortazar, abraham.duarte}@urjc.es

Rafael Martí
Dept. de Estadística e Investigación Operativa
Facultad de Matemáticas
Universidad de Valencia
46100 Valencia
rafael.marti@uv.es

Resumen

En este trabajo presentamos un método basado en la metodología de reencadenamiento de trayectorias para problemas de optimización global sin restricciones. El método introduce ideas del diseño factorial de experimentos y reencadenamiento de trayectorias entre más de dos soluciones. Hemos realizado experimentos con 19 funciones multimodales, con dimensiones entre 50 y 1000 variables, comparando nuestros resultados con otros métodos del estado del arte.

1. Introducción

El reencadenamiento de trayectorias (PR, *Path Relinking*) es una estrategia de intensificación que explora las trayectorias que conectan soluciones de calidad que han sido obtenidas mediante métodos heurísticos [6]. Se puede considerar una extensión de los métodos de combinación que se encuentran en la mayoría de algoritmos evolutivos. Sin embargo, en lugar de producir una solución directamente a partir de un par de soluciones, el reencadenamiento explora el camino entre las soluciones seleccionadas.

PR parte de una solución, denominada solución inicial, y explora un camino en el espacio de vecindad que lleva a la otra solución, denominada solución guía. Este proceso se lleva a cabo introduciendo atributos de la solución guía en la solución inicial. En [9] se propuso un método de PR como forma de intensificación en el contexto de un método GRASP. Posteriormente, en [12] se introdujo PR evolutivo, donde un conjunto de soluciones élite se evolucionaba de una forma similar a como lo hace el conjunto de referencia en la métodología de búsqueda dispersa. En este artículo se explora la aplicación de PR y su variante PR evolutivo en el contexto de optimización global.

Los problemas de optimización global sin restricciones se pueden caracterizar de la siguiente forma:

$$\begin{gathered} Minimizar\ f(x) \\ l \le x \le u \\ x \in \mathbb{R} \end{gathered} \qquad (1)$$

Donde $f(x)$ es una función no lineal y x es un vector de variables continuas en el intervalo $[l,u]$. Hemos diseñado un PR evolutivo para problemas que pueden ser formulados como en (1), y realizado experimentación sobre un conjunto bien conocido de funciones de las que se conocen los óptimos en varias dimensiones.

En trabajos anteriores de optimización global se ha aplicado con éxito la metodología de búsqueda dispersa [10]. Este enfoque es extendido en [3] donde se presenta STS, un algoritmo que incorpora a la búsqueda dispersa un método tabú. En [5] se consideran tres métodos previos como el estado del arte: *Differential Evolution* [15], G-CMA-ES [1] y *Real coded* CHC [4]. En la experimentación realizada hemos incluido los cuatro métodos: STS, DE, G-CMA-ES y CHC.

En la siguiente sección se describe el método de reencadenamiento de trayectorias básico desarrollado. En la Sección 3 se describe un método basado en PR evolutivo, que se puede considerar una extensión del PR básico. En la sección 4 se describe la experimentación

realizada. Finalmente, exponemos nuestras conclusiones en la Sección 5.

2. Reencadenamiento de trayectorias

El Algoritmo 1 presenta un esquema general del método de reencadenamiento de trayectorias para un problema de minimización. El conjunto de soluciones élite (*EliteSet*) contiene *b* soluciones élite construidas previamente. Estas soluciones pueden generarse utilizando un generador de soluciones diversas como en la búsqueda dispersa, donde se construye un conjunto *D* relativamente grande de soluciones, y se seleccionan las *b* mejores, teniendo en cuenta tanto la calidad como la diversidad. Sin embargo, PR no está restringido a este enfoque, y puede comenzar desde un conjunto de soluciones élite obtenidas por cualquier otro método.

```
1. Crear un EliteSet de tamaño b.
2. Evaluar las soluciones en
   EliteSet y ordenarlas. Sea x^1
   la mejor.
3. Aplicar el método de mejora a
   x^1 y reemplazarla por la
   solución mejorada.
4. Generar el conjunto
   NewSubsets, con las ternas de
   soluciones del EliteSet de la
   forma (a, x, y).
mientras ( NewSubsets ≠ ∅ ) hacer
   5. Seleccionar el siguiente
      conjunto (a, x, y) de
      NewSubSets.
   6. Aplicar el método de
      reencadenamiento para
      producir la secuencia de a a
      x y a z.
   7. Aplicar el método de mejora
      a la mejor solución obtenida
      de dicha secuencia. Sea w la
      solución mejorada.
   si (f(w) < f(x^1) ) entonces
      8. x^1 = w
   fin if
   9. Eliminar (a, x, y) del
      conjunto NewSubsets
fin mientras
```

Algoritmo 1.

En el contexto de optimización global en el que se desarrolla este trabajo el número de evaluaciones está limitado, por tanto restringimos la aplicación del método de mejora a la mejor solución del *EliteSet* en el paso 3. En el paso 4 se construye el conjunto de ternas de soluciones para las que se llevará a cabo el proceso de reencadenamiento. En este sentido, no realizamos este proceso entre un par de soluciones, sino que lo llevamos a cabo entre tres soluciones. PR no está limitado a explorar el camino entre dos soluciones. Denotaremos estas ternas como (*a*, *x*, *y*).

Cada una de estas ternas es seleccionada en orden lexicográfico y se genera una secuencia de soluciones desde *a* hasta *x* y *z* (pasos 5 y 6). En el paso 7 se aplica el método de mejora a la mejor solución encontrada en el camino explorado. En este momento se comprueba si la solución mejorada es mejor que la mejor solución encontrada hasta el momento x^1, en cuyo caso la reemplaza. La búsqueda finaliza cuando todas las ternas del conjunto *NewSubsets* han sido exploradas.

Para generar las ternas hemos adaptado un método típico de la metodología de búsqueda dispersa [11] donde se generan ternas expandiendo pares en conjuntos de mayor tamaño. Concretamente, considerando el *EliteSet* como el conjunto de soluciones $x^1,\ldots,x^b$ donde las soluciones están ordenadas por calidad ($f(x^i) < f(x^{i+1})$) limitamos la aplicación de PR a las 3-tuplas formadas de la siguiente manera: (x^i, x^j, x^{j+1}), donde $i < j$.

2.1. Construcción del *EliteSet*

Para la construcción inicial del *EliteSet* hemos implementado un generador de soluciones basado en ideas del área de la estadística conocida como diseño de experimentos. Uno de los métodos más populares para el diseño de experimentos es el diseño factorial k^n, donde *n* es el número de factores (en nuestro caso variables) y *k* es el número de niveles (posibles valores de esas variables). Un diseño factorial completo consideraría todas las posibles combinaciones de factores y niveles. Este esquema se vuelve rápidamente impracticable, dado que el número de experimentos crece exponencialmente con el número de variables. En nuestro caso, hemos considerado un diseño fraccional en el que sacaremos conclusiones utilizando sólo una fracción de todos los experimentos posibles, seleccionando esa fracción de experimentos

estratégicamente. Genichi Taguchi [13] fue uno de los primeros en proponer el diseño fraccional de experimentos. Taguchi ideó un método basado en el concepto de *orthogonal array*. Utilizamos los *arrays* de Taguchi para generar soluciones diversas, un enfoque ya presentado en [10]. Los *arrays* de Taguchi son de tamaños muy particulares, por la forma en la que se construyen. Por ejemplo, el *array* $L_9(3^4)$ puede ser utilizado para generar 9 soluciones para un problema con 4 variables, cada una de las cuales toma 3 valores.

Experiment	Factors			
	1	2	3	4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	2	3
8	3	2	1	3
9	3	3	2	1

Tabla 1. $L_9(3^4)$ orthogonal array

Los valores de la Tabla 1 representan los niveles que deberían establecerse para los diferentes factores en cada experimento. Con el objeto de crear un método de generación de soluciones diversas basado en las tablas de Taguchi, hemos traducido cada valor de la siguiente forma:

$$midvalue = l + \frac{1}{2}(u - l)$$
$$lowervalue = l + \frac{1}{4}(u - l) \quad (2)$$
$$uppervalue = l + \frac{3}{4}(u - l)$$

En nuestro problema de optimización global tenemos funciones con dimensiones entre 50 y 1000, sin embargo, no tenemos tablas Taguchi para un número de variables tan grande. La tabla Taguchi más grande es de tamaño n=40. Por lo tanto partiremos las variables en conjuntos de tamaño 40 y completaremos el resto de variables con los valores asignados a los niveles 1, 2 y 3. La tabla con 40 variables y tres niveles contiene 81 experimentos. Por tanto podemos generar 81 soluciones asignando los valores de la tabla a las 40 primeras variables, y el valor asociado al nivel 1 (*mid value*) al resto de variables. Generamos otras 81 soluciones asignando los valores de la tabla a las mismas 40 primeras variables, y el valor asociado al nivel 2 (*lower value*) al resto de variables. De la misma forma generamos otras 81 soluciones asociando a las 40 primeras variables y el valor del nivel 3 al resto de variables. Por tanto generamos de esta manera 243 soluciones aplicando los valores de la tabla a las 40 primeras variables. A continuación nos movemos al siguiente conjunto de variables para asignar los niveles de Taguchi. Concretamente, aplicamos estos niveles a las variables 21 a 60. Hemos comprobado experimentalmente que este desplazamiento de 20 variables no consume demasiadas evaluaciones de la función objetivo, y produce mejores resultados. De esta forma obtenemos 81 soluciones aplicando los valores de la tabla Taguchi a las variables 21 a 60, y *mid value*, *lower value* y *upper value* al resto de variables, obteniendo otras 243 soluciones. Matemáticamente podemos determinar el número total de soluciones *DSize*:

$$DSize = 243\lceil n/20 \rceil \quad (3)$$

El método genera un conjunto D de *DSize* soluciones que están distribuidas uniformemente en el espacio de soluciones. Es importante notar que Taguchi nos proporciona soluciones diversas. El *EliteSet* es entonces construido escogiendo directamente las b mejores soluciones.

2.2. Método de mejora

Hemos implementado un método de mejora consistente en una búsqueda lineal en una variable seguida del método Simplex. La búsqueda lineal realiza una serie de movimientos en cada variable, utilizando una rejilla, por lo que la precisión del resultado obtenido depende de lo cercano que esté el óptimo a un punto de la rejilla. El método Simplex nos permite salirnos de la rejilla con el objeto de acercarnos más al valor del óptimo local. La combinación de estos dos métodos fue aplicada satisfactoriamente en [10], donde además se incluían estructuras de memoria. En nuestro método de mejora hemos considerado un tamaño de rejilla $h = (u\text{-}l) / 100$.

El proceso, por tanto, está dividido en dos fases. En una primera fase el método de mejora aplica sucesivas búsquedas lineales cada una de las cuales consiste en modificar una variable i en la dirección de un vector unitario e_i. Dada una solución x, en primer lugar evaluamos la contribución de cada variable a la mejora. Para cada variable i=1,…,n, evaluamos dos soluciones: $x + he_i$ y $x - he_i$, quedándonos con la mejor de las dos soluciones. A continuación ordenamos las variables en función del valor de la función objetivo de la solución seleccionada anteriormente para cada variable, de forma que la variable i con mejor valor es seleccionada en primer lugar. Seleccionamos entonces n/2 variables de forma ordenada y aplicamos una búsqueda lineal a cada una de ellas.

La búsqueda lineal consiste en evaluar las soluciones factibles de la forma $x+khe_i$, siendo k un valor entero en el rango [-20, 20]. Dado que sólo se imponen restricciones en los valores máximo y mínimo de las variables, el único requisito que hay que verificar es que $l \leq x + khe_i \leq u$. Con el objetivo de reducir el número de evaluaciones de la función objetivo aplicamos una estrategia de tipo *first improvement*, evaluando en orden aleatorio las soluciones de la forma $x+khe_i$, y aplicando el primer movimiento que mejora la mejor solución que encontrada hasta el momento. Entonces seleccionamos la siguiente variable, partiendo de la mejor solución encontrada en la búsqueda lineal anterior. Una vez realizadas las n/2 búsquedas lineales, se vuelve a evaluar la contribución de cada variable. De la nueva ordenación obtenida, escogemos de nuevo n/2 variables y realizamos una búsqueda lineal sobre cada una de ellas. Este proceso se repite hasta que no es posible encontrar una solución que mejore la solución actual, o hasta que se alcancen 10 iteraciones. De esta forma se realizan un máximo de 10n/2 búsquedas lineales.

En la segunda fase del método de mejora, aplicamos el método Simplex a la mejor solución x=(x_1,x_2,…,x_3) encontrada en la primera fase. En primer lugar se perturba el valor de cada variable en una cantidad α. Concretamente, obtenemos n soluciones x_1,x_2,…,x_n, de la forma x_i = (x_1,…,x_i+α,…,x_n). El valor de α se selecciona con una distribución uniforme en el intervalo [-1,1]. En este momento se aplica el método Simplex [2]. Este método termina cuando no es posible mejorar la solución.

2.3. Reencadenamiento de trayectorias

Hemos considerado dos procedimientos de reencadenamiento diferentes. En el primero de ellos, denominado reencadenamiento ortogonal, exploramos el camino entre una solución inicial a y dos soluciones guía x e y. Este procedimiento comienza con la solución a = (a_1,...,a_n) y va reemplazando alternativamente bloques de variables de tamaño m de a por las de x = (x_1,…,x_n) e y = (y_1,…,y_n), obteniendo una secuencia de soluciones a(1), a (2), …, tal que:

$$\begin{aligned} a(1) &= (x_1, \dots, x_m, a_{m+1}, \dots, a_n), \\ a(2) &= (x_1, \dots, x_m, y_{m+1}, \dots, y_{2m}, \\ &\quad a_{2m+1}, \dots, a_n) \\ a(3) &= (x_1, \dots, x_m, y_{m+1}, \dots, y_{2m}, \\ &\quad x_{2m+1}, \dots, x_{3m}, a_{3m+1}, \dots, a_n) \end{aligned} \tag{4}$$

De nuevo, para limitar el número de evaluaciones, calculamos $m = n/k$, donde k es un valor pequeño que esablecemos a partir de la experimentación preliminar.

El segundo procedimiento de reencadenamiento, denominado reencadenamiento en línea recta, consiste en comenzar en la solución inicial y moverse en el camino establecido por un vector de a a x, en primer lugar. Se exploran las soluciones a(1), a(2), …, a(k-1) obtenidas como la combinación de a y x en la mitad del segmento que las une, como se puede ver en (5).

$$\begin{aligned} a(1) &= a + \frac{1}{k}(x - a) \\ a(2) &= a + \frac{1}{k-1}(x - a) \\ &\dots \\ a(k-1) &= a + \frac{1}{2}(x - a) \end{aligned} \tag{5}$$

De estas k-1 soluciones intermedias escogemos la mejor, $a(j)$, y repetimos el proceso moviéndonos en el camino de $a(j)$ a y como puede verse en (6).

$$\begin{aligned} a(k) &= a(j) + \frac{1}{k}\big(y - a(j)\big) \\ a(k+1) &= a(j) + \frac{1}{k-1}\big(y - a(j)\big) \\ &\dots \\ a(2k-2) &= a(j) + \frac{1}{2}(y - a(j)) \end{aligned} \tag{6}$$

3. Reencadenamiento de trayectorias evolutivo

El Algoritmo 2 muestra el seudocódigo del método basado en reencadenamiento de trayectorias evolutivo que hemos desarrollado para problemas de optimización global. El procedimiento comienza seleccionando *b* soluciones de élite del conjunto inicial *D* generado con el método constructivo descrito en la Sección 2.1. Estas soluciones se insertan en el conjunto *EliteSet*, que se ordena por calidad, de forma que la solución x^1 es la de mayor calidad. La búsqueda comienza asignando el valor *TRUE* a *NewSolutions* y estableciendo a cero el número de iteraciones globales. En el paso 3 se construye el conjunto *NewSubsets* que contiene las ternas (*a*,*x*,*y*) sobre las que se realizará el reencadenamiento de trayectorias, y *NewSolutions* toma el valor *FALSE*.

Las ternas en *NewSubsets* son seleccionadas en orden lexicográfico, y se les aplica el método de reencadenamiento. La mejor solución obtenida es mejorada con el procedimiento de mejora descrito en la Sección 2.2. Esta solución mejorada, *w*, es añadida a un conjunto *Pool* que utilizaremos posteriormente para actualizar el conjunto de soluciones de élite. Cuando todas las ternas (*a*,*x*,*y*) han sido exploradas, en los pasos 8 a 10 comprobamos si las soluciones del conjunto *Pool* pueden entrar en el conjunto *EliteSet*. Realizamos estos dos pasos, ejecutando el reencadenamiento de soluciones para las ternas seleccionadas y actualizando el *EliteSet* hasta que se alcanza el número máximo de evaluaciones.

```
1. Construir el EliteSet con b
   soluciones élite.
2. Ordenar las soluciones por
   calidad de forma que x^1 sea la
   mejor solución y x^b la peor.
   Hacer NewSolutions = TRUE y
   GlobalIter=0.
mientras ( NumEvaluations <
   MaxEvaluations ) hacer
   3. Generar NewSubsets, con las
      ternas (a, x, y) de
      soluciones en el EliteSet
      que incluyen al menos una
      solución nueva. Hacer
      NewSolutions = FALSE y Pool
      = ∅.
   mientras ( NewSubsets ≠ ∅ )
      hacer
      4. Seleccionar (a, x, y) de
         NewSubSets.
      5. Aplicar el método de
         reencadenamiento.
      6. Aplicar el método de
         mejora a la mejor
         solución de la secuencia.
         Sea w la solución
         mejorada. Añadir w a
         Pool.
      7. Borrar (a, x, y) de
         NewSubsets
   fin mientras
   para cada solution w ∈ Pool
      8. Sea x^w la solución más
         cercana a w en EliteSet
      si ( f(w) < f(x^1) o ( f(w) <
         f(x^b) & d(w, x^w)>dthresh)
         entonces
        9. Hacer x^w = w y reordenar
           EliteSet
        10. Hacer NewSolutions =
            TRUE
        fin si
   fin para
   11. GlobalIter = GlobalIter +1
   si ( GlobalIter = MaxIter o
      NewSolutions= FALSE)
      12. Reconstruir el EliteSet.
         GlobalIter =0
   fin si
fin mientras
```

Algoritmo 2.

Si en algún momento de este proceso, no se encuentra ninguna solución que pueda entrar en el *EliteSet*, o el número de iteraciones globales alcanza su valor máximo, *MaxIter*, el conjunto *EliteSet* es reconstruido en el paso 12, y el proceso continúa restableciendo *GlobalIter* a su valor inicial. La idea de reconstruir el *EliteSet* al alcanzar un número máximo de iteraciones es evitar que el método quede atrapado muy cerca de un óptimo local, realizando mejoras muy pequeñas.

Es importante notar que para que una solución entre en el conjunto de soluciones de élite se tiene en cuenta tanto su calidad como su diversidad. Dada una solución *w* del conjunto *Pool*, se considera la solución x^w del conjunto *EliteSet* más cercana a *w* y con un valor peor que *w*. Sólo introducimos *w* en el conjunto *EliteSet* si mejora la mejor solución, x^1, o bien si mejora la peor y su

distancia a x^w es mayor que un umbral *dthresh*. Esto se comprueba entre los pasos 8 y 9.

Cuando no es posible introducir ninguna nueva solución en el *EliteSet*, o se alcanza el número máximo de reencadenamientos+mejoras, *MaxIter*, reconstruimos el *EliteSet*. Para ello, seleccionamos la siguiente solución del conjunto *D* obtenido durante la fase constructiva. De este conjunto, ordenado por calidad, ya escogimos las *b* mejores soluciones para formar el conjunto *EliteSet* inicial. Ahora escogemos la solución *b*+1 y realizamos el reencadenamiento entre esta solución y dos soluciones del *EliteSet*. Estas dos soluciones se escogen probabilísticamente atendiendo a la calidad de las soluciones. A la mejor solución obtenida del reencadenamiento se le aplica el método de mejora y se comprueba directamente si puede entrar a formar parte del *EliteSet*. Si entra en el *EliteSet*, sustituye a una solución que ya estaba en él y será utilizada como solución guía. Este proceso se repite *b* veces. Una vez consideradas las *b* siguientes soluciones de *D*, se vuelve a evolucionar el *EliteSet*.

4. Evaluación experimental

Para comprobar la eficiencia de nuestro método lo hemos ejecutado utilizando 11 funciones simples, denominadas F1 a F11, y 8 funciones híbridas, denominadas F12 a F19, y hemos comparado los resultados obtenidos con los del estado del arte. Las funciones F1 a F19 están descritas en detalle en [5] y de todas ellas se conoce el óptimo. Las funciones híbridas se obtienen por composición de las funciones F9 o F10 con otra función simple (F1, F3, F4 o F7). En todos los experimentos indicamos el error respecto al óptimo. Dada una solución *x*, el error se define como $f(x) - f(op)$, siendo *op* el óptimo de la función.

Primero realizamos una serie de experimentos preliminares para establecer los parámetros del método y sus diferentes componentes. Finalmente, comparamos nuestro método final con los métodos del estado del arte.

En nuestro primer experimento preliminar, comprobamos el comportamiento del método constructivo Taguchi frente al método constructivo definido en [3], un constructivo por frecuencias que denominamos *Frequency*. Realizamos 1000 construcciones con cada método e indicamos la mejor solución obtenida. Para este experimento utilizamos las funciones F3, F8 y F13, cada una de ellas con las dimensiones 50, 100, 200, 500 y 1000. La Tabla 2 muestra la media del error sobre las tres instancias obtenida con cada constructivo para cada una de las dimensiones.

	50	100	200	500	1000
Frequency	5,3E10	1,1E11	2,7E11	7,3E11	1,5E12
Taguchi	3,7E10	6,0E10	1,3E11	3,7E11	7,6E11

Tabla 2. Métodos constructivos

En base a los resultados de la Tabla 2, escogemos el método basado en Taguchi como método constructivo.

En el segundo experimento preliminar comprobamos varios métodos de búsqueda local del estado del arte y los comparamos con el método en dos fases descrito en la Sección 2.2. De acuerdo con [7] algunos de los mejores métodos de mejora para optimización global son: *Compass Search* (CS) [8], Solis and Wets (SW) [14] y *Tabu line search* (TLS) [3]. Dado que nuestro método en dos fases (TSLS) incluye la aplicación del método Simplex tras la aplicación de la búsqueda lineal, hemos aplicado Simplex también al resto de métodos del experimento, de forma que la comparativa sea más justa. En este experimento utilizamos las funciones F3, F13 y F17 e indicamos la media del error para las tres instancias en cada dimensión. Sustituimos F8 por F17 dado que los tres métodos encuentran fácilmente el óptimo para F8.

	50	100	200	500	1000
CS	8,2E13	2,8E14	8,0E14	1,7E15	2,1E16
SW	3,2E10	6,2E10	1,4E11	1,9E11	8,4E11
TLS	1,7E9	5,2E9	1,3E10	2,3E11	9,7E10
TSLS	1,8E3	4,1E3	1,2E4	2,2E10	4,6E4

Tabla 3. Métodos de mejora

Los resultados de la Tabla 3 muestran que nuestro método de mejora en dos fases supera en media, en todas las dimensiones, al resto de métodos considerados. Utilizaremos este método como método de mejora en los experimentos sucesivos.

El tercer experimento preliminar comprueba el efecto de los dos procedimientos de reencadenamiento de trayectorias descritos en la Sección 2.3. Comparamos tres versiones de cada uno de ellos, con los valores de *k* 1, 2 y 3.

Utilizamos las mismas funciones F3, F13 y F17 del anterior experimento preliminar.

	Línea recta			Ortogonal		
	2	3	4	2	3	4
50	2,6E2	2,0E2	2,5E2	1,7E2	3,5E2	2,0E2
100	1,5E3	1,1E3	1,1E3	1,5E3	1,3E3	1,2E3
200	4,2E3	4,0E3	4,2E3	6,3E3	5,6E3	5,1E3
500	1,3E4	1,2E4	1,3E4	1,7E4	1,9E4	1,8E4
1000	2,6E4	2,4E4	2,4E4	3,7E4	3,8E4	3,6E4
Media	8,8E3	8,4E3	8,4E3	1,2E4	1,3E4	1,2E4

Tabla 4. Métodos de reencadenamiento

Los resultados de la Tabla 4 indican que el reencadenamiento en línea recta es un enfoque más apropiado que el ortogonal. En concreto, un valor de k=3 parece la mejor elección. Seleccionamos un reencadenamiento en línea recta con k=3 para el resto de nuestros experimentos.

En el último de los experimentos preliminares comparamos el método de reencadenamiento de trayectorias descrito en el Algoritmo 1 con el método de reencadenamiento de trayectorias evolutivo descrito en el Algoritmo 2. Establecemos el tamaño b del *EliteSet* a 10 para el primero, y para el segundo probamos tres valores diferentes: 4, 8 y 12.

	PR	EvoPR		
|ES|	10	4	8	12
50	6,2E1	7,6E1	5,2E1	7,9E1
100	1,3E2	1,8E2	2,5E2	6,1E2
200	5,2E2	4,8E2	9,8E2	1,7E3
500	2,3E3	1,6E3	2,7E3	4,3E3
1000	5,8E3	3,9E3	6,7E3	8,9E3
Media	1,7E3	1,3E3	2,1E3	3,1E3

Tabla 5. PR y PR evolutivo

En la Tabla 5 se puede observar que el método PR mejora al método PR evolutivo cuando el tamaño del *EliteSet* es relativamente grande (8 y 12), sin embargo con un valor bajo (4), el método PR evolutivo supera al método PR básico. Utilizaremos este método en nuestro experimento final.

En nuestra experimentación final establecemos el valor de *MaxIter* a 40. Comparamos nuestro método, EvoPR, con cuatro métodos del estado del arte: STS [3], DE [15], G-CMA-ES [1] y CHC [4].

Siguiendo las indicaciones de [5], realizamos esta experimentación final de la siguiente forma:

- Cada algoritmo es ejecutado 25 veces para cada función
- El método se detiene al alcanzar el número máximo de evaluaciones: 5000n, siendo n el número de dimensiones

	DE	CHC	G-CMA-ES	STS	EvoPR
50	3,1E0	2,4E5	1,0E2	1,3E2	1,4E1
100	3,0E1	5,8E6	2,3E2	6,2E2	6,3E1
200	3,5E2	1,4E8	5,4E2	2,8E3	1,9E2
500	3,7E3	3,4E9	2,2E255	1,9E4	3,7E2
1000	1,5E4	2,0E10	-	1,4E4	1.2E3
Media	3,7E3	4,8E9	5,6E254	7,2E3	3,7E2

Tabla 6. Error medio sobre 25 ejecuciones

Los datos de la Tabla 6 muestran la media del error sobre las 25 ejecuciones. En esta tabla se puede apreciar que para las dimensiones más bajas (50 y 100), el método DE es el que obtiene un menor error medio. Sin embargo, en las dimensiones más altas (200, 500 y 1000), nuestro método, EvoPR, obtiene el error medio más bajo. En la última fila de la table se muestra la media sobre todas las dimensiones. Nuestro método presenta una media inferior a los otros cuatro métodos del estado del arte.

5. Conclusiones

Hemos descrito un método de reencadenamiento de trayectorias para la optimización de funciones sin restricciones. Hemos descrito un método constructivo basado en el diseño factorial de experimentos, dos métodos de reencadenamiento de trayectorias entre una solución inicial y dos soluciones guía, y un método de mejora en dos fases.

Los experimentos realizados muestran que nuestro método es competitivo con otros métodos del estado del arte para problemas de optimización global sin restricciones, especialmente en las dimensiones más altas, cuando se consideran funciones con 200, 500 ó 1000 dimensiones.

Agradecimientos

Este trabajo ha sido financiado parcialmente con fondos del Ministerio de Educación y Ciencia (TIN2009 – 07516).

Referencias

[1] Auger. A.. N. Hansen. 2005. A Restart CMA Evolution Strategy With Increasing Population Size. In Procs. of 2005 IEEE Congress on Evol. Comput. (CEC'2005). 1769 - 1776.

[2] Avriel. M. 1976. Nonlinear Programming. Analysis and Methods. Prentice - Hall. Englewood Cliffs. New Jersey.

[3] Duarte. A.. R. Martí. and F. Glover. F. Gortazar. Hybrid Scatter Tabu Search for Unconstrained Global Optimization. Annals of Operations Research. to appear. 2010.

[4] Eshelman. L. J. and J. D. Schaffer. Real - coded genetic algorithm and interval schemata. Foundation of Genetic Algorithms. pages 187–202. 1993.

[5] Herrera. F. M. Lozano and D. Molina. Test Suite for the Special Issue of Soft Computing on Scalability of Evolutionary Algorithms and other Metaheuristics for Large Scale Continuous Optimization Problems. Technical report. University of Granada. 2010.

[6] Glover. F. and M Laguna. 1997. *Tabu Search*. Kluwer Academic Publishers. Boston.

[7] Hvattum. L.M.. A. Duarte. R. Martí and F. Glover. The Improvement Method in Scatter Search: An experimental study on global optimization. Technical report. University of Valencia. 2010.

[8] Kolda. T.G.. R.M. Lewis and V.J. Torczon. Optimization by Direct Search: New perspectives on some Classical and Modern Methods. SIAM Review 45. 385 - 482. 2003.

[9] Laguna. M. and Martí. R.. 1999. GRASP and Path Relinking for 2 - Layer Straight Line Crossing Minimization. INFORMS Journal on Computing 11(1). 44 - 52.

[10] Laguna. M. and R. Martí. 2005. Experimental Testing of Advanced Scatter Search Designs for Global Optimization of Multimodal Functions. Journal of Global Optimization 33. 235 - 255.

[11] Martí. R.. F. Glover and M. Laguna. 2006. Principles of Scatter Search. European Journal of Operational Research 169. pp. 359 - 372 (2006)

[12] Resende. M.G.C. and R.F. Werneck. 2004. A hybrid heuristic for the p - median problem. Journal of Heuristics 10. 59–88.

[13] Roy. R.K. (1990) A Primer on the Taguchi Method. Van Nostrand Reinhold. New York.

[14] Solis. F.J. and R.J. - B. Wets. Minimization by random search techniques. Mathematical Operations Research 6. 19 - 30. 1981.

[15] Storn. R. and K. Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization. 11:341–359. 1997.