Ciencias Matemáticas, Vol. 29, No. 1, Pag. 1-6, 2015
Recibido 04-2014

Mecanismos de aceleración para el algoritmo prototipo de optimización basado en análisis de intervalos
Acceleration mechanisms for the prototype algorithm of optimization based on interval analysis

Greter Domínguez Rodríguez[1]*, Aymée Marrero Severo[2], Jorge Luis Rodríguez Pérez[3], Gonzalo Joya Caparrós[4]

Resumen En la literatura especializada hay gran variedad de trabajos referentes al problema de optimización global que utilizan desde técnicas clásicas hasta heurísticas. Este trabajo presenta una técnica determinista usada para resolver el problema de la optimización global basada en Análisis de Intervalos. En primer lugar se abordan las características fundamentales del Análisis de Intervalo y se explica el uso de esta técnica en la confección del Algoritmo Prototipo de la Optimización Global (AP). Posteriormente se presentan 2 mecanismos implementados con el objetivo de acelerar la convergencia del AP y se realiza el análisis de los resultados obtenidos en la experimentación numérica. Por último se exponen las conclusiones del trabajo.

Abstract In the literature there are many results concerning the global optimization problem using techniques from classical to heuristics. This paper presents a deterministic technical used to solve the problem of global optimization based on Interval Analysis. In first place are addressed the basics features of Interval Analysis and explained the use of this technique in the construction of the Global Optimization Algorithm Prototype (AP). Subsequently are presented two mechanisms implemented in order to accelerate the convergence of the AP and are performed the analyzing of the results of the numerical experiments. Finally are exposed the conclusions.

Palabras Clave
Análisis de Intervalos, Optimización Global, Procesamiento Paralelo

[1] *Universidad de La Habana, Dirección Docente de Informatización, greter.dominguez@iris.uh.cu*
[2] *Universidad de La Habana, Facultad de Matemática y Computación, aymee@matcom.uh.cu*
[3] *Universidad de Ciencias Informáticas, Cuba, jlrp@uci.cu*
[4] *Universidad de Málaga, Dpto. Tecnología Electrónica, E.T.S. Ing. Telecomunicación, gjoya@uma.es*
***Autor para Correspondencia**

1. Introducción

Las posibilidades que ofrece el Análisis de Intervalos para afrontar el problema de la optimización global fueron explotadas por primera vez en el trabajo de Skelboe de 1974 [1], mejorado en 1976 por [2]. Desde entonces han sido numerosos los trabajos relacionados con el Análisis de Intervalos que han tratado problemas de optimización, principalmente sin restricciones. No obstante, no fue hasta la década de los 90 que el Análisis de Intervalos tuvo mayor difusión entre la comunidad científica. Esto fue posible al surgir lenguajes de programación y software con herramientas implementadas para el trabajo con intervalos. Algunos ejemplos son FORTRANXSC, también conocido como ACRITH-XSC [3] (una librería para FORTRAN 90), C-XSC [4] (una librería para C++), Pascal- XSC [5] (una extensión del Pascal estándar), e INTLAB [6] (Interval Laboratory - un toolbox de Matlab que se puede incorporar desde la versión 5.3), y que es el lenguaje utilizado en este trabajo para realizar estudios computacionales.

El análisis de intervalos ha sido aplicado con éxito en campos muy diversos, como en ingeniería química, diseño asistido por ordenadores, sistemas dinámicos y caos, sistemas expertos, control de calidad y mecánica de fluidos, entre otros. Sin embargo, a pesar de su extendido uso, el Algoritmo Prototipo para resolver problemas de optimización basado en Análisis de Intervalos tiene un alto tiempo de ejecución debido a que su orden es exponencial respecto al número de variables del problema a resolver.

En un intento por mejorar dicho tiempo de ejecución surge este trabajo, cuyo objetivo se centra en elaborar mecanismos de aceleración para el algoritmo prototipo. El trabajo aborda dos modificaciones al algoritmo, la primera incorpora técnicas

numéricas con el objetivo de mejorar los puntos críticos de su funcionamiento, y la segunda utiliza técnicas de programación en paralelo de manera que puedan explotarse mejor las potencialidades del hardware utilizado.

El trabajo está dividido en cuatro secciones, la primera dedicada a la Aritmética de Intervalos y las funciones de inclusión por ser este el concepto que sustenta toda la teoría del AP, la segunda a explicar el funcionamiento del AP, la tercera a presentar las técnicas de aceleración implementadas y en la cuarta se exponen los resultados computacionales obtenidos para problemas test de optimización global y el análisis realizado a partir de dichos resultados. El documento culmina con la presentación de las conclusiones donde se plantean además las posibles líneas futuras de investigación y finalmente se incluye la bibliografía consultada.

2. Aritmética de Intervalos y Funciones de Inclusión

El conjunto de los intervalos reales cerrados lo denotaremos por I, el conjunto de vectores n-dimensionales de intervalos, también llamados cajas, por I^n. Las letras en negrita denotarán intervalos, las minúsculas cantidades escalares y las mayúsculas vectores, también llamados cajas. Los corchetes [.] delimitarán intervalos, mientras que los paréntesis (.) delimitarán vectores. El subrayado inferior denotará los extremos inferiores de los intervalos y el subrayado superior los extremos superiores.

Con esta notación, si tenemos que $X = (x_1, \cdots, x_n)$ es una caja o vector de intervalos reales cerrados, siendo la componente i-ésima el intervalo $x_i = \left[\underline{x_i}, \overline{x_i}\right]$, podemos escribir que $X = \left[\underline{X}, \overline{X}\right]$, donde $\underline{X} = (\underline{x_1}, \cdots \underline{x_n})$ y $\overline{X} = (\overline{x_1}, \cdots, \overline{x_n})$.

2.1 Aritmética de intervalos

Debido a que los intervalos pueden ser interpretados como conjuntos, las operaciones aritméticas pueden ser definidas mediante la siguiente expresión:

$$x \odot y = \{x \odot y : x \in x, y \in y\}$$

Donde $\odot$ representa una de las cuatro operaciones binarias básicas $(+; -; *; /)$ y x/y está definido solo si $0 \notin y$.

Una manera de representar las operaciones aritméticas, más cómoda para realizar una implementación de las mismas, es expresarlas en función de los extremos de los intervalos que intervienen en la operación [7].

Sean $x = [\underline{x}, \overline{x}]$, $y = \left[\underline{y}, \overline{y}\right]$:

$$\begin{aligned}
x + y &= [\underline{x} + \underline{y}, \overline{x} + \overline{y}] \\
x - y &= [\underline{x} - \overline{y}, \overline{x} - \underline{y}] \\
x * y &= [\min\left\{\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}\right\}, \max\left\{\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}\right\}] \\
x/y &= x * \left[1/\overline{y}, 1/\underline{y}\right], \text{ si } 0 \notin y
\end{aligned}$$

Las operaciones aritméticas sobre cajas o vectores de intervalos son las extensiones naturales de las definidas sobre vectores reales.

2.2 Funciones de Inclusión

El concepto clave en el desarrollo de la teoría de la optimización basada en Análisis de Intervalos es el de función de inclusión [8].

Definición 1 *Sea $D \subset R^n$ un conjunto cualquiera, $f : D \to R$ una función definida sobre él. $I^n(D) = \{X : X \in I^n, X \subset D\}$ el conjunto de los vectores de intervalos contenidos en D . Una función $f : I^n(D) \to I$ se dice que es una función de inclusión de f si*

$$f^u(X) = \{f(X) : X \in X\} \subseteq f(X) \;\; \forall X \in I^n(D) \qquad (1)$$

La utilidad de las funciones de inclusión radica en que podemos obtener directamente cotas inferiores y superiores de f sobre cualquier caja X en el dominio de f tomando $\underline{f(X)}$ y $\overline{f(X)}$, respectivamente (véase figura 1).

Las funciones de inclusión de funciones vectoriales se definen de forma análoga. En tal caso, la condición (1) debe ser satisfecha en cada componente.

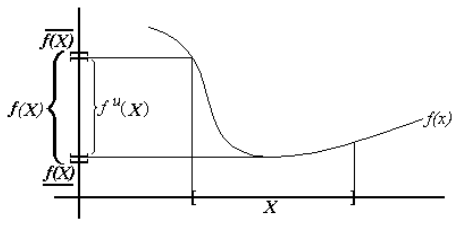

Figura 1. Obtención de cotas a través de funciones de inclusión.

Para las funciones h predeclaradas en los lenguajes de programación (como la exponencial, el seno, etc.), es fácil obtener funciones de inclusión h, ya que los intervalos de monotonía de dichas funciones son conocidos, y se puede tomar $h(x) = h^u(x)$ para cualquier $x \in I$ en el dominio de h . De hecho, cualquiera de los lenguajes de programación FORTRAN-XSC [3], C-XSC [4], Pascal-XSC [5], INTLAB [6] así como otros lenguajes preparados para el cálculo científico, disponen de funciones de inclusión predeclaradas h para cada una de las funciones predeclaradas h .

Para una función general $f(x)$, $x \in R^n$, la obtención de funciones de inclusión se puede hacer de forma automática con el uso de la aritmética de intervalos y de dichas funciones de inclusión predeclaradas.

3. Algoritmo Prototipo

Para la presentación del AP de la optimización global, consideraremos el problema de optimización dado por:

$$\begin{cases} \min f(x) \\ \text{s.a. } g_i(x) \leq 0, j = 1, \cdots, r \\ x \in X_0 \in I^n \end{cases} \qquad (2)$$

Donde $f, g_i : D \subset R^n \to R$, $j = 1, \cdots, r$, son funciones cualesquiera, y $X_0 \subseteq D$ es una caja o vector de intervalos que contiene al conjunto donde se quiere buscar el óptimo.

Este problema es necesario reformularlo de forma tal que se pueda usar la teoría de Análisis de Intervalos para hallar su solución, para ello transformamos la función objetivo y las restricciones del problema en sus respectivas funciones de inclusión, de manera que el problema a resolver por el AP quedaría planteado de la siguiente forma:

$$\begin{cases} \min f(X) \\ \text{s.a. } g_i(X) \leq 0, j = 1, \cdots, r \\ X \subseteq X_0 \in I^n \end{cases} \quad (3)$$

El AP sigue un esquema de ramificación y acotación. Comienza dividiendo la caja inicial X_0 en varias subcajas, de estas subcajas son eliminadas las que se tiene la certeza de que no contienen puntos de mínimos globales, mientras que las restantes son almacenadas en una lista de trabajo para una posterior subdivisión. Este proceso se repite hasta que las cajas en la lista de trabajo tienen un ancho menor que una cierta tolerancia prescrita, denotada por ε. Como en todos los algoritmos de ramificación y acotación, su costo computacional es de orden exponencial respecto a la cantidad de variables y está determinado por el análisis de las cajas resultantes del proceso de subdivisión en cada iteración.

En forma de pseudocódigo, el algoritmo quedaría como sigue:

Algorithm 1 Pseudocódigo del algoritmo prototipo

1: $Y \leftarrow X_0, L_t \leftarrow \emptyset, L_s \leftarrow \emptyset$;
2: Elegir las direcciones coordenadas a lo largo de las cuales realizar la subdivisión de Y;
3: Subdividir Y perpendicularmente a las direcciones elegidas, realizando un determinado número de cortes a lo largo de cada dirección;
4: Sean $Y_1, \cdots, Y_s$ las subcajas obtenidas;
5: **for** $i = 1$ to s **do**
6: **if** Y_i no tiene puntos óptimos **then**
7: Eliminar Y_i;
8: **else**
9: Almacenar Y_i en la lista de trabajo L_t, siguiendo algún criterio de ordenación;
10: **end if**
11: **end for**
12: **if** $L_t = \emptyset$ **then**
13: **return**
14: **end if**
15: Seleccionar una caja de L_t y eliminarla de dicha lista. Denotemos por Y dicha caja;
16: **if** $(\overline{Y} - \underline{Y}) < \varepsilon$ **then**
17: Adicionar Y en la lista L_s de las cajas solución e ir al paso 16;
18: **else**
19: Ir al paso 2
20: **end if**

Los métodos para rechazar una subcaja corresponden a pruebas de selección, entre ellas las más importantes son el test de factibilidad y el test de punto medio que se describen a continuación, aunque también existen otras tales como: test de monotonía y test de concavidad, para los cuales es necesario el uso de diferenciación automática.

3.1 Test de factibilidad [8]:

Definición 2 *Se dice que una caja* $Y \subseteq X_0$ *cumple (con certeza) la restricción* $g_j(Y) \leq 0$ *si* $\overline{g_j(Y)} < 0$ *y que la incumple (con certeza) si* $\underline{g_j(Y)} > 0$.

Esta definición se justifica gracias al concepto de función de inclusión (ver definición 1).

Definición 3 *Una caja* Y *se dice (ciertamente) factible si cumple (con certeza) todas las restricciones del conjunto factible, (ciertamente) infactible si incumple (con certeza) alguna de las restricciones que definen el conjunto factible, e indeterminada en otro caso.*

Así pues, las cajas Y para las que algún $\underline{g_j(Y)} > 0$, pueden ser eliminadas por ser infactibles.

3.2 Test de punto medio [8]:

En el transcurso del algoritmo, la función objetivo f es evaluada en distintos puntos factibles. El menor valor obtenido en dichas evaluaciones en un momento dado $\widetilde{f}$ es una cota superior para el valor óptimo f^* del problema. Si para una caja Y, $\underline{f(Y)} > \widetilde{f}$ entonces la caja Y puede ser eliminada, ya que ningún punto de dicha caja mejora el valor $\widetilde{f}$, esto está garantizado gracias al concepto de función de inclusión (ver definición 1).

Este test es conocido como el test de punto medio porque lo habitual para intentar mejorar el valor de $\widetilde{f}$ durante en algoritmo es evaluar la función f en el punto medio de la caja elegida para ser subdividida (si este punto es factible), y actualizar entonces $\widetilde{f}$ como $\widetilde{f} = min\left\{\widetilde{f}, \overline{f(m(Y))}\right\}$. No obstante, también sería válido utilizar cualquier otro punto factible $y \in Y$.

4. Técnicas de Aceleración

4.1 La optimización numérica y el test de punto medio

A lo largo de la ejecución del algoritmo, la función objetivo f es evaluada en diferentes cajas factibles. Como se había explicado anteriormente, el menor valor obtenido en dichas evaluaciones $\widetilde{f}$ constituye una cota superior para el valor optimo del problema y es empleado por el test del punto medio para decidir si una caja es descartable o no.

Como el valor $\widetilde{f}$ es un punto, y no un intervalo, resulta entonces válido plantearse el problema de emplear algún algoritmo de optimización numérica que permita obtener con rapidez valores de $\widetilde{f}$ tan bajos como sea posible y de esta manera hacer más efectivo el test del punto medio.

Para resolver este problema de optimización, seleccionamos dos métodos numéricos clásicos que convergen a un

mínimo local de la función objetivo, el Método de máximo descenso y el Método Cuasi-Newton con las correcciones DFP y BFGS [9].

4.2 Paralelización del algoritmo

El paso 4 del AP, se basa en el análisis de una colección de cajas, a las que deben aplicarse un conjunto de test, con el objetivo de descartarlas o preservarlas. Este paso puede reformularse de forma más detallada como sigue:

Algorithm 2 Reformulación del paso 4

1: **for** $i = 1$ to s **do**
2: Aplicar a la caja Y_i el test $T_1(X)$. Si la caja no pasa el test, eliminarla;
3: Aplicar a la caja Y_i el test $T_2(X)$. Si la caja no pasa el test, eliminarla;
4: ...
5: Aplicar a la caja Y_i el test $T_k(X)$. Si la caja no pasa el test, eliminarla;
6: Si la caja ha pasado satisfactoriamente todos los test. Guardarla en la lista de trabajo L_t y pasar a analizar la siguiente caja;
7: **end for**

Al analizar lo anterior, es claro que cada iteración se resume en la aplicación de un conjunto de k test (funciones) independientes unas de otra, sobre una misma caja. Por tanto, se decidió ejecutar concurrentemente cada uno de esos test.

Para realizar este proceso de optimización se implementó un esquema de paralelización Jefe/Trabajador (Manager/Worker) [10] que es utilizado cuando un problema puede ser dividido en un conjunto de tareas que los trabajadores (workers) pueden procesar sin necesidad de comunicarse entre sí, es decir; las tareas son independientes.

La característica más importante de este modelo es que cada trabajador se comunica exclusivamente con el proceso jefe (manager) y no tiene información acerca de lo que otros trabajadores hacen.

El esquema implementado es el siguiente:

Algorithm 3 Esquema Paralelo

1: Sea P una lista que contiene un número s de cajas, y sea un conjunto de funciones test $T_1(X), T_2(X), \ldots, T_k(X)$;
2: $P \leftarrow L_t$;
3: **for** $i = 1$ to s **do**
4: Crear k procesos subordinados, cada uno de los cuales encapsula una función test $T_i(X)$;
5: Asignar a cada proceso la caja analizada de la lista P;
6: Ejecutar todos los procesos de manera concurrente;
7: Recopilar los resultados devueltos por cada uno de los procesos;
8: Añadir la caja a la lista L_t si pasó satisfactoriamente todos los test;
9: **end for**

5. Resultados Computacionales

Se muestra una selección de los resultados obtenidos para una serie de problemas clásicos, con el objetivo de validar la eficacia de las diferentes técnicas de aceleración aplicadas al AP original.

5.1 Problema 1

$$\min f(x) = -\sum_{k=1}^{5} k \sin[(k+1)x+k]$$

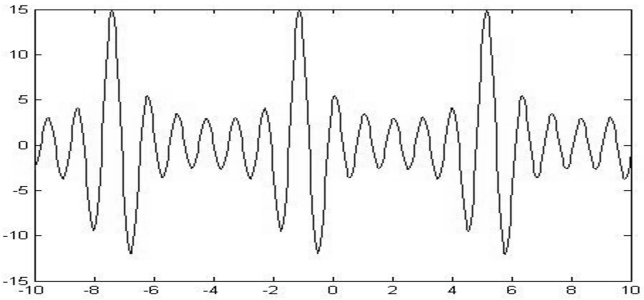

Figura 2. Representación gráfica de la función f dada por Schubert en [11]

Como intervalo inicial para el algoritmo se tomó $[-10;10]$ en el cual la función presenta 3 mínimos globales como se puede apreciar en la figura 2, y como cota para las condiciones de parada $\varepsilon = 10^{-4}$.

Con esta configuración los resultados alcanzados se muestran en la tabla 1.

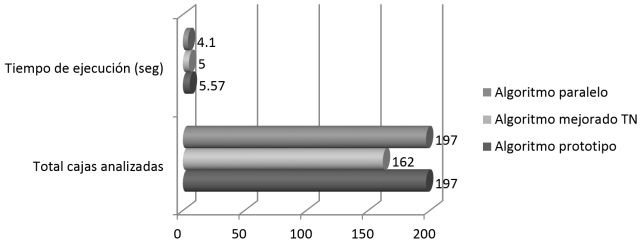

Figura 3. Representación gráfica del comportamiento del algoritmo original con respecto al algoritmo mejorado mediante técnicas numéricas y al algoritmo paralelo para el problema 1.

El algoritmo ofrece como resultado tres intervalos que contienen los puntos de mínimos globales y ofrecen una estimación del valor óptimo con una precisión menor que $\varepsilon = 10^{-4}$. En la figura 3 que ilustra su comportamiento se puede apreciar una ligera disminución en el tiempo de ejecución al aplicar técnicas de optimización numérica. Esta disminución está dada por el hecho de que el test del punto medio resulta más efectivo, por lo tanto se eliminan más cajas en las primeras etapas del algoritmo, y esto provoca una disminución apreciable en el total de cajas que se generan a lo largo de toda

Tabla 1. Resultados para el problema 1.

Puntos óptimos	Intervalos resultantes	Valor óptimo	Valor resultante
$x_1 = -6{,}7745$	$x \in [-6{,}7746; -6{,}7745\ ]$	$-12{,}0312$	$-12{,}0312$
$x_2 = -0{,}4914$	$x \in [-0{,}4914; -0{,}4913]$	$-12{,}0312$	$-12{,}0312$
$x_3 = 5{,}7918$	$x \in [5{,}7918; 5{,}7919]$	$-12{,}0312$	$-12{,}0312$

la ejecución. Sin embargo el algoritmo paralelo resulta ligeramente superior al mejorado mediante técnicas numéricas debido a la reducción en el tiempo de ejecución.

5.2 Problema 2

$$\text{mín} f(x) = 2x_1^2 - 1{,}05x_1^4 + \frac{1}{6} * x_1^6 - x_1x_2 + x_2^2$$

Esta función es conocida como *función camello de tres jorobas* [12]. La función tiene un mínimo global en el origen de coordenadas, dos mínimos locales en $[\pm 1{,}75, \pm 0{,}87]$ y dos puntos de ensilladura en $[\pm 1{,}07, \pm 0{,}535]$. En la figura 4 se muestra el gráfico de la función.

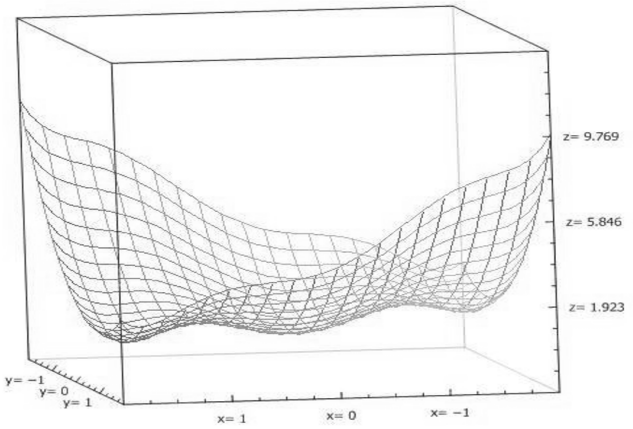

Figura 4. Representación gráfica de la función ¨camello de tres jorobas¨

Para este problema, se consideró la caja inicial $[[-10;10];[-10;10]]$ y como cota para las condiciones de parada $\varepsilon = 10^{-4}$ .Con esta configuración los resultados alcanzados se muestran en la tabla 2.

Tabla 2. Resultados para el problema 2.

Punto óptimo	Intervalos resultantes
$[0;0]$	$[0{,}01000; 0{,}01000]$
	$[-0{,}01000; -0{,}01000]$
	$[-0{,}01000; 0{,}01000]$
	$[0{,}01000; -0{,}01000]$

En este caso el algoritmo alcanza cuatro intervalos degenerados (Decimos que un intervalo x es degenerado si $\underline{x} = \overline{x}$. Tal intervalo contiene un solo número real x y lo podemos denotar por dicho número) que constituyen una aproximación

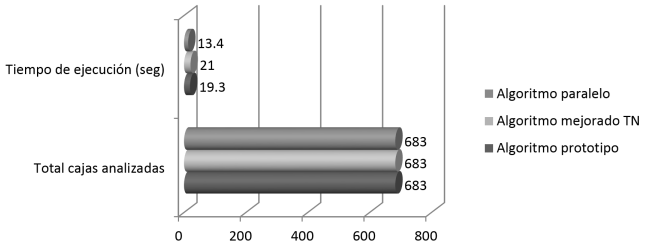

Figura 5. Representación gráfica del comportamiento del algoritmo original con respecto al algoritmo mejorado mediante técnicas numéricas y al algoritmo paralelo para el problema 2.

de los puntos óptimos con la precisión deseada. Este problema ilustra que el empleo de técnicas numéricas no siempre resulta beneficioso. En problemas que como este, el test de punto medio es poco efectivo, mejorar el valor de la cota superior del óptimo mediante métodos numéricos no produce ningún beneficio apreciable; al contrario, se incurre en un pequeño sobrecosto para el tiempo total de ejecución del algoritmo. En cambio el empleo de técnicas de programación en paralelo si conlleva una mejora en cuanto al tiempo de ejecución.

5.3 Problema 3

$$\begin{aligned} \text{mín} f(x) &= (x_1 - 10)^3 + (x_2 - 20)^3 \\ &\text{s.a} \\ g_1(x) &= (x_1 - 5)^2 + (x_2 - 5)^2 - 100 \geq 0 \\ g_2(x) &= -(x_1 - 6)^2 + (x_2 - 5)^2 + 82{,}81 \geq 0 \\ 13 &\leq x_1 \leq 100 \\ 0 &\leq x_2 \leq 100 \end{aligned}$$

La solución global conocida para este problema es el punto $x = [14{,}095; 0{,}84296]$ que al ser evaluado resulta en un valor óptimo $f(x) = -6961{,}81381$.

Para las pruebas con las diferentes variantes del algoritmo se empleó como caja inicial $[[13;100];[0;100]]$ y como cota para las condiciones de parada $\varepsilon = 10^{-4}$.

Con esta configuración los resultados alcanzados se muestran en la tabla 3.

El resultado alcanzado por el AP para este problema es igualmente preciso que los ejemplos anteriores, se obtuvieron cuatro intervalos degenerados que aproximan el óptimo de la función con una precisión menor que $\varepsilon = 10^{-4}$.

En la figura 6 se puede observar que en este problema, las optimizaciones numéricas no provocan ninguna mejoría. Esto

Tabla 3. Resultados para el problema 3.

Punto óptimo	Intervalos resultantes
[14,095; 0,84296]	[14,0952; 0,8432]
	[14,0940; 0,8409]
	[14,0942; 0,8414]
	[14,0943; 0,8416]

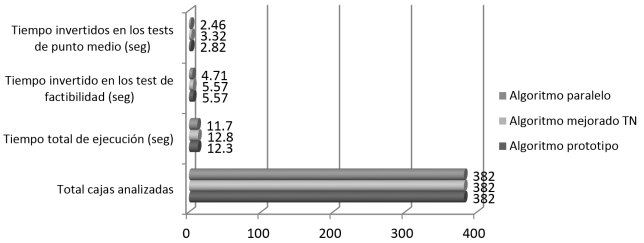

Figura 6. Representación gráfica del comportamiento del algoritmo original con respecto al algoritmo paralelo para el problema 3.

se debe a que dichas técnicas, por su naturaleza, no hacen ningún chequeo de factibilidad sobre los puntos de iteración, por lo que pueden convergen a un punto que es mínimo local del problema irrestricto, pero queda fuera del conjunto factible una vez aplicadas las restricciones.

El algoritmo paralelo resulta ligeramente superior al algoritmo original en cuanto a tiempo de ejecución, lo cual se evidencia los tiempos de ejecución de los test paralelos respecto a sus versiones seriales.

6. Conclusiones

Este trabajo muestra un algoritmo capaz de resolver problemas de optimización global con y sin restricciones, para los que se precisa encontrar un intervalo que contenga la solución óptima con una cierta precisión. El algoritmo fue modificado mediante la adición de un método numérico de minimización, así como mediante la aplicación de un esquema de ejecución paralela a muchas de sus operaciones. La incorporación de técnicas numéricas no resulta siempre beneficiosa, depende de las características del problema a resolver. Sin embargo la implementación usando técnicas de programación en paralelo logra disminuir el tiempo de ejecución para problemas de todo tipo. Aún así, el tiempo de ejecución del AP continúa siendo su punto crítico.

Nuestro interés de mejorar el Algoritmo Prototipo basado en Análisis de Intervalos para resolver problemas de optimización global está dado por la necesidad de resolver problemas de estimación de parámetros en modelos epidemiológicos definidos por Ecuaciones Diferenciales Ordinarias (EDO), donde no solo se necesitan hallar valores de los parámetros que caractericen la dinámica de dichos modelos de forma cercana a la realidad, sino para validar los intervalos de definición obtenidos para estos mediante técnicas estadísticas.

Con esta perspectiva futura recomendamos confeccionar un método híbrido entre el AP y alguna metaheurística, para obtener una combinación balanceada entre precisión y rapidez y concretar la idea de aplicar este algoritmo en la resolución del problema de estimación de parámetros para modelos epidemiológicos.

Agradecimientos

Esta investigación ha sido parcialmente financiada por los proyectos AECID D/017218/08 y D/9842/07 sobre ”Modelación Matemática de Procesos Biomédicos. Estimación de Parámetros”.

Referencias

[1] S. Skelboe. 1974. Computation of Rational Interval Functions. ed. BIT. Vol. 14.

[2] R.E. Moore. 1976. Computing the Range of Values of a Rational Function of n variables over Bounded Region. Computing. Vol. 16.

[3] W.V. Walter. 1993. A portable Fortran 90 module library for accurate and reliable scientific computing. Computing (Suppl). Vol. 9.

[4] W. Hofschuster, W. Kramer, S. Wedner, A. Wiethoff. 1993. CXSC 2.0, A C++ Class Library for Extended Scientific Computing. Springer-Verlag.

[5] R. Klatte, U. Kulisch, M. Neaga, D. Ratz y Ch. Ullrich. 1992. PASCAL-XSC Language Reference with Examples H. Springer.

[6] S.M. Rump. 1999. INTLAB- Interval Laboratory. N. Springer.

[7] R.B. Kearfott. 1996. Rigorous Global Search: Continuous Problems. ed. D. Kluwer Academic Publisher.

[8] J. Fernández. 1999. Nuevas técnicas para el diseño y resolución de modelos de localización continúa. Tesis doctoral. Universidad de Murcia. Inédita

[9] J.M. Otero, A. Kakes y A. Marrero. 2006. Modelos de Optimización Continuos, 54:149-232, 1998. ed. Félix Varela.

[10] J. Kepner. 2009. Parallel MATLAB for Multicore and Multinode Computers. SIAM Press.

[11] B. O. Schubert. 1972. A sequential Method Seeking the Global Maximum of a Function. SIAM Journal of Numerical Analysis.

[12] E. Hansen. 1980. Global optimization using interval analysis the multi-dimensional case. Numerische Mathematik. Vol. 34.