Departamento de Arquitectura de
Computadoras y Electrónica.
Universidad de Almería.

Métodos de Acotación en Algoritmos de Optimización Global Intervalar. Paralelización.

José Antonio Martínez García
e-mail: jmartine@ual.es
Almería, Febrero 2007

Director/es de la tesis:

Dr. D. Leocadio González Casado Dr. Dña. Inmaculada García Fernández

Doctorando:

D. José Antonio Martínez García

Agradecimientos

- En primer lugar me gustaría mostrar todo agradecimiento a mis directores de tesis Inmaculada García Fernández y Leocadio González Casado, por la paciencia que han demostrado tener conmigo y por su apoyo constante. Sin su ayuda nunca habría logrado terminar este trabajo.
- A mi mujer Toñi y a mi hija Marta porque sin todo su cariño este trabajo no habría sido posible.
- A mi madre y a mi padre; el primer esfuerzo para llegar hasta aquí fue de ellos. A mis hermanos y demás familia, por su constante apoyo.
- A todos los miembros del grupo de investigación Supercomputación: Algoritmos. En especial a José Román Bilbao y Sebastian García por haberme ayudado en numerosas ocasiones en el mantenimiento y reparación de las averías de los cluster. A Boglárka Tóth por su ayuda y colaboración en numerosas ocasiones.
- A los miembros del departamento de Arquitectura de Computadores y Electrónica de la Universidad de Almería.
- Al CESCA por haberme abierto cuenta muy rápidamente en *obacs*. Con ella pude seguir trabajando mientras nuestro Altix estaba averiado.
- A mis amigos, por esos ratos de ocio a veces tan necesarios.

Prólogo

En este trabajo de tesis se aborda el problema de Optimización Global desde la perspectiva de los métodos determinísticos. Específicamente, esta tesis se desarrolla en el ámbito de los métodos de Branch and Bound basados en Aritmética de Intervalos. En este entorno se han introducido y evaluado nuevas y eficientes técnicas de acotación y eliminación, que han dado como resultado unos algoritmos capaces de abordar problemas computacionalmente muy costosos. Desde esta perspectiva, en esta tesis también se aborda el estudio de versiones paralelas de nuestro modelo de computación, que son proyectadas sobre una arquitectura de memoria compartida.

La tesis se ha organizado en cinco capítulos, cuyos contenidos se resumen a continuación.

En el Capitulo 1 se hace una definición de la notación para la Aritmética Real y para la Aritmética de Intervalos usada en el desarrollo de esta tesis. Se define el problema de Optimización Global y los tipos de métodos que permiten resolverlos: métodos estocásticos y métodos determinísticos.

En el Capítulo 2 se describe la estructura general de los algoritmos de Branch and Bound, así como sus reglas básicas. Se hace una introducción a la Aritmética de Intervalos, como una herramienta para evitar los problemas derivados del uso de la Aritmética Computacional. Al final del capítulo se hace una breve introducción a la diferenciación automática.

El Capítulo 3 está dedicado al análisis de un nuevo algoritmo de Optimización Global Intervalar sólo para funciones unidimensionales (TIAM). Se describe una nueva regla de acotación y la correspondiente regla de eliminación, que son incorporadas al algoritmo TIAM, obteniéndose un nuevo algoritmo denominado IAG.

El capítulo termina con una evaluación experimental de la propuesta, suministrando resultados de la ganancia en velocidad de este nuevo algoritmo con respecto al básico.

El Capítulo 4 extiende los resultados obtenidos en el Capítulo 3 al caso de funciones multidimensionales, cuyo resultado son los algoritmos denominados MTIAM (Multidimensional TIAM) y MIAG (Multidimensional IAG). Sobre esta propuesta se estudian nuevos refinamientos que incorporan el uso de las formas centradas; como resultado de estos refinamientos se obtienen los algoritmos denominados MIGO y AMIGO.

El Capitulo 5 está dedicado a los aspectos computacionales relacionados con el paralelismo. Tras una revisión del estado del arte, se evalúan distintas posibilidades de implementación paralela sobre una arquitectura de memoria compartida y finalmente se proponen dos modelos de algoritmos paralelos para el algoritmo AMIGO (Local-PAMIGO y Global-PAMIGO). Local-PAMIGO y Global-PAMIGO son evaluados para un conjunto de funciones de prueba muy costosas y usando una precisión muy elevada, para las que nunca antes se había podido obtener una solución.

Índice general

Índice de figuras

Índice de Tablas

Capítulo 1

Introducción

Este capítulo introduce el problema de la Optimización Global de forma general y presenta el problema particular de Optimización Global. Para ello primero se muestra la notación de la que se hará uso en esta tesis, haciendo especial énfasis en la notación para la Aritmética de Intervalos que es una de las herramientas básicas que ayudan a resolver el tipo de problemas en los que estamos interesados.

1.1. Notaciones de la Aritmética Real

En este trabajo se usarán las siguientes notaciones:

$\mathbb{R}$	El conjunto de los números reales.
$\mathbb{R}^n$	El conjunto de los vectores de reales de dimensión n.
$\mathbb{R}^{n\times m}$	El conjunto de las matrices $n \times m$ de reales.

Las componentes de un vector $x \in \mathbb{R}$ o de una matriz $M \in \mathbb{R}^{n\times m}$ se enumerarán por subíndices. Se usarán superíndices para enumerar un vector o una matriz dentro de una secuencia. Por ejemplo:

x_i	$\in$	$\mathbb{R}$	Elemento i del vector x.
M_{i*}	$\in$	$\mathbb{R}^n$	Fila i de la matriz M.
M_{*j}	$\in$	$\mathbb{R}^m$	Columna j de la matriz M.
$M_{i,j} = M_{ij}$	$\in$	$\mathbb{R}$	Elemento i de la columna j de la matriz M.
x^k	$\in$	$\mathbb{R}^n$	Vector k dentro de una enumeración o secuencia.
M^k	$\in$	$\mathbb{R}^n$	Matriz k dentro de una enumeración o secuencia.

Se usarán minúsculas para definir números reales y vectores de números reales, y mayúsculas para definir matrices. Para la multiplicación de números reales, vectores y matrices se usará siempre el símbolo "$\cdot$"(se puede omitir). De este modo, el producto escalar de dos vectores $x, y \in \mathbb{R}^n$ se define como:

$$x \cdot y = xy = \sum_{i=1}^{n} x_i \cdot y_i \tag{1.1}$$

Para una función $f : D \subseteq \mathbb{R}^n \to \mathbb{R}$ con $f \in C^2(D)$, siendo $C^n(D)$ la clase de funciones diferenciables de grado n en el dominio D, se define:

$$\nabla f(x) = f'(x) = \begin{pmatrix} \frac{\partial f}{\partial x_1}(x) \\ \frac{\partial f}{\partial x_2}(x) \\ \vdots \\ \frac{\partial f}{\partial x_n}(x) \end{pmatrix} \tag{1.2}$$

como el gradiente de la función f y

$$\nabla^2 f(x) = f''(x) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2}(x) & \frac{\partial^2 f}{\partial x_1 \partial x_2}(x) & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n}(x) \\ \frac{\partial^2 f}{\partial x_2 \partial x_1}(x) & \frac{\partial^2 f}{\partial x_2^2}(x) & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n}(x) \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1}(x) & \frac{\partial^2 f}{\partial x_n \partial x_2}(x) & \cdots & \frac{\partial^2 f}{\partial x_n^2}(x) \end{pmatrix} \tag{1.3}$$

como la matriz Hessiana de f. Para $g : D \subseteq \mathbb{R}^n \to \mathbb{R}^n$ con $g \in C^1(D)$, se define

$$J_g(x) = \begin{pmatrix} \frac{\partial g_1}{\partial x_1}(x) & \frac{\partial g_1}{\partial x_2}(x) & \cdots & \frac{\partial g_1}{\partial x_n}(x) \\ \frac{\partial g_2}{\partial x_1}(x) & \frac{\partial g_2}{\partial x_2}(x) & \frac{\partial g_2}{\partial x_n}(x) & \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial g_n}{\partial x_1}(x) & \frac{\partial g_n}{\partial x_2}(x) & \cdots & \frac{\partial g_n}{\partial x_n}(x) \end{pmatrix}$$

como el Jacobiano de g. Además se cumple que si $g(x) = \nabla f(x)$, entonces $J_g(x) = \nabla^2 f(x)$.

1.2. Notaciones de la Aritmética de Intervalos

Se usarán letras mayúsculas para los intervalos. Así, el intervalo X se define como: $X = [\underline{x}, \overline{x}]$, $\underline{x} \leq \overline{x} \in \mathbb{R}$, donde $\underline{x}$ es límite inferior de X y $\overline{x}$ es el límite superior. Un número real x es análogo al intervalo $[x, x]$. Se usarán las siguientes notaciones:

$\mathbb{I}$	El conjunto de los intervalos.
$\mathbb{I}^n$	El conjunto de los vectores de intervalos de dimensión n.
$\mathbb{I}^{n\times m}$	El conjunto de las matrices $n \times m$ de intervalos.

Los elementos de $\mathbb{I}^n$ son llamados también cajas (n-dimensionales). La notación usada para indexar intervalos será análoga a la usada para indexar números reales. Para $X \in \mathbb{I}$ e $Y \in \mathbb{I}^n$ se define:

$w(X)$	$= \overline{x} - \underline{x}$	Anchura de X.
$w(Y)$	$= \max_{1\leq i\leq n} w(Y_i)$	Anchura de la caja Y.
$m(X)$	$= (\underline{x} + \overline{x})/2$	Punto medio de X.
$m(Y)$	$= (m(Y_1), m(Y_2), \ldots, m(Y_n))^T$	Punto medio de la caja Y.

Cuando se exprese un número real x, como un intervalo $[x, x]$, se usará normalmente la notación real debido a que es más simple, pudiéndose deducir del contexto cuándo se debe tratar como un intervalo y cuándo como un número real.

1.3. El problema de la Optimización Global

El campo emergente de la Optimización Global se ocupa de problemas de programación matemática, con la (posible) presencia de múltiples mínimos locales y/o globales. Habitualmente, el número de mínimos locales y/o globales para un problema particular es desconocido a priori y en muchas ocasiones puede ser bastante elevado. Además, las diferencias entre los valores de la función objetivo en los mínimos locales y globales pueden ser muy grandes. Debido a esto, las soluciones locales no son válidas en la mayoría de los casos. Incluso para diferencias pequeñas entre los valores en los mínimos locales y globales, el no encontrar el mínimo global puede traducirse, especialmente en aplicaciones económicas, en millones de euros de pérdidas. Este es uno de los campos en los que el uso de buenos métodos de Optimización Global puede llegar a ser extremadamente importante. No obstante, debemos destacar que los problemas de Optimización Global pueden aparecer en una amplia variedad de disciplinas científicas tales como Matemáticas, Física, Química y Biología, así como en cualquier campo de la ingeniería.

Existen muchos problemas reales en los que la mayoría de las aproximaciones clásicas no pueden aplicarse directamente para resolverlos. Como ejemplo, la Figura 1.1 muestra una función relativamente simple, que es la composición de varias funciones trigonométricas con argumentos polinomiales, sobre un espacio de búsqueda bidimensional. Este ejemplo muestra la necesidad de usar estrategias de búsqueda global apropiadas.

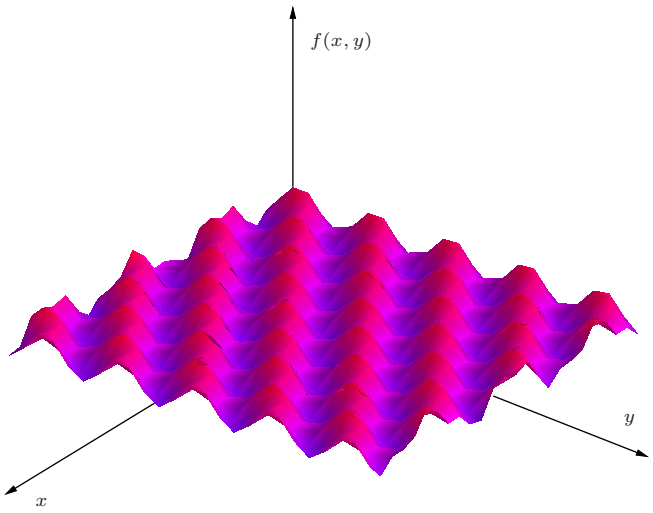


Figura 1.1: Representación gráfica de la función $f(x) = 0.2(\sin(x + 4y) - 2\cos(2x - 3y))$ con múltiples mínimos locales.

Las primeras referencias sobre Optimización Global aparecen a finales de los cincuenta. Desde entonces han surgido muchas aportaciones, tanto en aspectos teóricos como en propuestas prácticas. El *estado del arte* en la actualidad se caracteriza por varias decenas de monográficos y varios miles de artículos de investigación dedicados exclusivamente a este tema.

Resolver un problema de optimización consiste básicamente en encontrar un punto o un conjunto de puntos, a menudo expresados como vectores de reales, en los que el valor de una función objetivo es mínimo (o máximo). Posiblemente la solución está sujeta a un conjunto de restricciones sobre los posibles candidatos. La descripción general del problema es:

$$\begin{aligned} &minimizar && f(x) \\ &sujeto\ a: && g(x) \leq 0, \\ & && h(x) = 0, \\ & && \underline{x}_i \leq x_i \leq \overline{x}_i,\ i = 1, \ldots, n \\ & && x \in \mathbb{R}^n. \end{aligned} \tag{1.4}$$

donde $f(x) : \mathbb{R}^n \to \mathbb{R}$ es la *función objetivo* y $g(x) : \mathbb{R}^n \to \mathbb{R}^m$ y $h(x) : \mathbb{R}^n \to \mathbb{R}^k$ son funciones que restringen la región de búsqueda de la(s) solución(es).

Se define la región de búsqueda de los posibles candidatos, o dominio del Problema (1.4) como:

$$S = \{x \in \mathbb{R}^n : g(x) \leq 0,\ h(x) = 0,\ \underline{x}_i \leq x_i \leq \overline{x}_i,\ i = 1, \ldots, n\} \tag{1.5}$$

En el Problema (1.4), $\underline{x}$ y $\overline{x}$ son límites (finitos) implícitos. Cuando las restricciones están definidas únicamente mediante límites implícitos de la región de búsqueda, el problema de optimización se suele incluir en el grupo de problemas denominados sin restricciones. Nosotros queremos resolver el problema de Optimización Global sin restricciones, es decir,

$$\begin{aligned} &minimizar && f(x) \\ &sujeto\ a: && \underline{x}_i \leq x_i \leq \overline{x}_i,\ i = 1, \ldots, n \\ & && x \in \mathbb{R}^n, \end{aligned} \tag{1.6}$$

donde el espacio de búsqueda viene determinado por

$$S = \{x \in \mathbb{R}^n :\ \underline{x}_i \leq x_i \leq \overline{x}_i,\ i = 1, \ldots, n\} \tag{1.7}$$

Una solución global del Problema (1.6) viene dada por $x^* \in S$ de forma que $f^* = f(x^*) \leq f(x),\ \forall x \in S$. Una solución local del Problema (1.6), se define como $\tilde{f} = f(\tilde{x}) \leq f(x), \forall x \in S \cap N_\epsilon(\tilde{x})$ con $\epsilon > 0$, donde $N_\epsilon(\tilde{x})$ son los ϵ-vecinos del punto $\tilde{x}$, definidos por $N_\epsilon(\tilde{x}) = \{x : \parallel x - \tilde{x} \parallel < \epsilon\}$. Un mínimo local no se considera una solución óptima para el Problema (1.6) hasta que no se demuestra que es una solución global. Sólo se considera el problema de minimización, ya que el problema de maximización puede reescribirse como un problema de minimización:

$$\max_{x \in S}\{f(x)\} = \min_{x \in S}\{-f(x)\} \tag{1.8}$$

Muchas de las investigaciones realizadas se han enfocado al caso especial en el que la estructura del modelo matemático tiene una serie de características, con lo que se

puede establecer que sólo existe una única solución y por lo tanto, un único mínimo, que es a la vez local y global. Esto se cumple, por ejemplo, si f y S son convexas. Los métodos desarrollados para problemas convexos usan solamente información local. Con la información de uno o más puntos, se construye una aproximación de la solución del problema original, la cuál se usa para calcular un nuevo punto de prueba en la siguiente iteración. Estos métodos garantizan la convergencia al mínimo global sólo cuando se verifica la propiedad de convexidad. En cualquier otro caso, dicha convergencia no está asegurada.

Aunque existen muchos problemas pertenecientes a la clase anterior, también existe una amplia variedad de problemas donde la existencia de un solo mínimo no puede ser postulada o verificada, con lo que su resolución se vuelve más difícil. Ejemplos de esos problemas pueden encontrarse en [6, 38, 39, 48, 103, 107, 133].

1.3.1. Algoritmos de Optimización Global

De manera generalizada, los algoritmos de Optimización Global, suelen clasificarse en dos grandes grupos: algoritmos estocásticos y algoritmos determinísticos. Los métodos estocásticos se caracterizan por hacer uso de algún factor aleatorio y porque la demostración de su convergencia depende de argumentaciones estadísticas [120, 133]. Normalmente, los métodos estocásticos se aplican a problemas sin restricciones y no necesitan ningún conocimiento sobre la función objetivo, dando a menudo resultados óptimos o cercanos a los óptimos. Los métodos estocásticos son especialmente útiles para problemas donde la evaluación de la función objetivo no se obtiene de forma analítica sino que puede ser el resultado de un conjunto de experimentos o simulaciones. Los métodos estocásticos de Optimización Global se basan en la evaluación de la función objetivo en puntos de la región de búsqueda elegidos aleatoriamente. Son el recurso a utilizar cuando el tamaño del problema, dimensionalmente hablando, es grande o el problema no está definido analíticamente. Las desventajas de estos métodos es que no garantizan que la solución pueda ser encontrada en un número finito de pasos, ni que el mínimo encontrado sea el global. Dicho de otra forma, la convergencia de los métodos estocásticos hacia el óptimo global esta demostrada sólo cuando el número de evaluaciones de la función objetivo tiende a infinito.

Al contrario que los métodos estocásticos, los métodos determinísticos no hacen uso de ninguna variable aleatoria en su estructura y las demostraciones de su convergencia no son de tipo probabilístico. Algunos de estos métodos ofrecen un tiempo de terminación finito para problemas específicos mientras que otros convergen cuando el número de iteraciones se aproxima a infinito. La clase de los algoritmos determinísticos, donde se incluyen los métodos de Branch and Bound, de cobertura, basados en intervalos, de tunelado y enumerativos, puede dividirse a su vez en dos categorías:

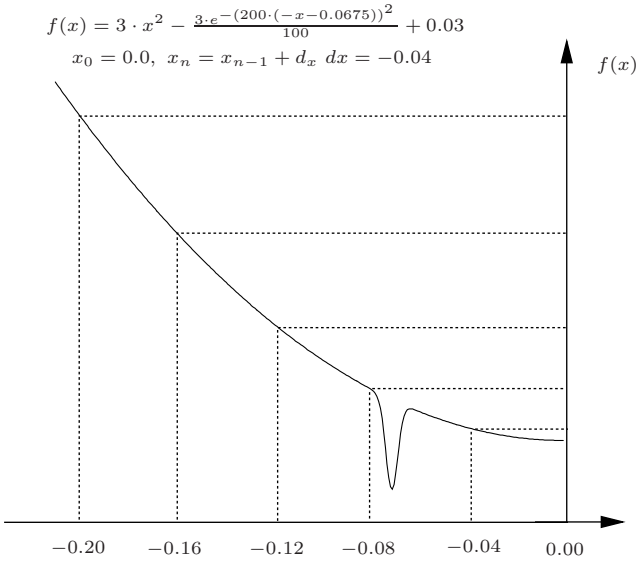


Figura 1.2: Un ejemplo donde algunos métodos de optimización clásicos podrían alcanzar únicamente un mínimo local.

- Los métodos que computan el valor en puntos de la función objetivo.
- Los métodos que computan límites de la función sobre conjuntos compactos (intervalos n-dimensionales cerrados y acotados, también llamados cajas).

Esta división separa además los métodos rigurosos de los no rigurosos. Los métodos que toman valores puntuales de la función objetivo son inherentemente incapaces de dar una información rigurosa para los problemas de Optimización Global ya que no se puede asegurar que la solución alcanzada sea realmente un óptimo global. Esto se debe a que en la aplicación del método, solamente son evaluados un número finito de puntos del espacio de búsqueda y el punto donde la función alcanza el mínimo global puede no estar contenido en dicho conjunto. En la Figura 1.2 se muestra la función:

$$f(x) = 3 \cdot x^2 - \frac{3 \cdot e^{-(200 \cdot (-x-0.0675))^2}}{100} + 0.03$$

que es evaluada solamente en un conjunto discreto de puntos. El punto donde se encuentra el mínimo global está fuera de dicho conjunto.

Por otro lado, los métodos basados en el cálculo de los límites de la función

objetivo pueden producir soluciones globales rigurosas, si se implementan adecuadamente, teniendo en cuenta los errores de redondeo producidos por la Aritmética Computacional. Los métodos determinísticos basados en el cálculo de los límites de la función objetivo que se estudiarán en esta tesis están basados en algoritmos de Branch and Bound y en el uso de la Aritmética de Intervalos. El principal inconveniente de estos métodos es que siempre se necesario disponer de una descripción analítica de la función objetivo asociada al problema que se desea resolver.

En el siguiente capítulo se desarrollarán los conceptos que dan soporte a los métodos determinísticos basados en técnicas de Branch and Bound y en Aritmética de Intervalos.

Capítulo 2

Herramientas para los algoritmos de Optimización Global Intervalar

Este capítulo introduce de forma general las herramientas necesarias para el desarrollo de algoritmos de Optimización Global Intervalar. La principal característica de este tipo de algoritmos es que garantizan que la solución encontrada es la óptima y que no existen más soluciones; es decir, encuentra todos los mínimos globales. Estos algoritmos garantizan que toda la región de búsqueda es examinada, debido a que se usan técnicas de Branch and Bound. La Aritmética de Intervalos es la herramienta que nos permitirá determinar de forma rigurosa en qué subregiones de búsqueda no existe un mínimo global, permitiendo de este modo acelerar la búsqueda. Algunos de los métodos que determinan la no existencia de solución se basan además en información de la derivada de la función objetivo. Por este motivo se introduce aquí una herramienta, la Diferenciación Automática, que nos permitirá evaluar la derivada sin tener que definirla analíticamente.

Todas estas herramientas se introducen aquí de forma general. Los Capítulos 3 y 4 describen el uso de estas herramientas en los algoritmos de Optimización Global Intervalar.

2.1. Algoritmos de Branch and Bound

Los primeros algoritmos de Branch and Bound aparecieron en artículos de los cincuenta y principios de los sesenta, donde los investigadores describieron esquemas

enumerativos para resolver, lo que posteriormente se denominaron, problemas NP-Completos [36, 117, 74, 44]. Debido a la generalidad del método y la efectividad que aportaba, fue ampliamente usado. De hecho, todavía es una de las mejores herramientas para resolver problemas difíciles. Los primeros que dieron el nombre de Branch and Bound a este método, fueron Little, Murty, Sweeny y Karel [80] (1963), en su artículo innovador sobre el problema del viajante de comercio. Lawer y Wood [75] estudiaron los algoritmos de Branch and Bound, obteniendo una descripción independiente del problema, siendo así el primer artículo que presenta un modelo general de Branch and Bound.

Los métodos de Branch and Bound son algoritmos basados en árboles de búsqueda. Su principio radica en una descomposición sucesiva del problema original en subproblemas más pequeños y disjuntos, hasta encontrar la solución óptima. Un árbol de búsqueda, $T = (V, \beta)$, describe estos subproblemas (el conjunto de vértices V, cuya raíz es el problema completo S) y el proceso de descomposición (el conjunto de arcos β). El algoritmo consiste en una búsqueda heurística iterativa en T, que evita visitar subproblemas que no contienen una solución óptima. Los algoritmos de Backtracking, Programación Dinámica y las búsquedas en árboles A^* y AND-OR pueden verse como variaciones de algoritmos de Branch and Bound [45, 69, 96, 98].

Una descripción de los modelos de algoritmos de Branch and Bound que aparecen en los artículos [75, 89, 57, 58, 59, 70, 98], hecha por Bruin, Kindervater y Trienekens puede encontrarse en [28]. Estos autores no discuten modelos más recientes debido a que sólo se diferencian de los anteriores en pequeñas variaciones. Esta revisión de los algoritmos de Branch and Bound está motivada para establecer un esquema común para todas las aplicaciones, mediante la descripción de unas reglas u operadores generales [57, 89]. Para especificar estas reglas nos basaremos en los formalismos utilizados por Corrêa y Ferreira [23], centrándonos en problemas donde se intenta encontrar el valor mínimo de una función objetivo.

2.1.1. Reglas básicas de los algoritmos de Branch and Bound

Según Mitten e Ibaraki [89, 57] los algoritmos de Branch and Bound pueden ser caracterizados por cuatro reglas:

- *Regla de Acotación*: Calcula un *límite inferior* de la solución óptima de un nodo.
- *Regla de Selección*: Define qué nodo será el siguiente a procesar.
- *Regla de División*: Establece cómo se descomponen los nodos del árbol de búsqueda.

- *Regla de Eliminación*: Establece cómo reconocer y eliminar nodos donde no se encuentra la solución del problema original.

Dependiendo del tipo del problema, se añade la siguiente regla:

- *Regla de Terminación*: Determina cuándo un nodo pertenece a la solución final. Esta regla aparece, por ejemplo, en problemas donde la solución está determinada por la precisión deseada. Por otro lado, en algunos problemas tradicionales de Optimización Combinatoria, el criterio de terminación es inherente a obtener una solución factible. Por ejemplo, en el problema del Viajante de Comercio [29, 80, 117, 136] [Apéndice B], donde hay que encontrar el camino mínimo para visitar todas las ciudades.

Llamaremos a estas reglas las *reglas básicas* de los algoritmos de Branch and Bound. Una buena comprensión de la estructura del problema a resolver, es decir, hacer una buena elección de las reglas básicas, da lugar a una reducción del tiempo de ejecución y a poder resolver problemas más complejos.

Los algoritmos de Branch and Bound consisten en una secuencia de iteraciones en las que se aplican las reglas básicas a una estructura de datos D, que puede verse como una lista ordenada de subproblemas [18], y a los subproblemas que se extraen de esta estructura. Estas reglas seleccionan un subproblema de D, lo descomponen y eventualmente insertan los subproblemas creados en la estructura de datos D. A cada subproblema se le asocia una solución factible, resolviendo el subproblema si es suficientemente simple o asignándole una solución, elegida entre las posibles dentro del subproblema (no necesariamente la mejor). La mejor solución factible encontrada durante la ejecución del algoritmo de Branch and Bound, será un límite superior de la solución final y se utilizará para eliminar aquellos subproblemas generados que no pueden contener una solución factible mejor que la que ya se ha encontrado [57, 59, 89, 135].

Se llaman subproblemas abiertos o activos, a aquellos que contienen una solución factible y que, en cualquier punto de la ejecución del algoritmo, fueron generados pero no descompuestos ni eliminados. En cualquier iteración, la estructura de datos está compuesta por una colección de subproblemas abiertos llamada conjunto abierto. La ejecución del algoritmo comienza con la estructura de datos en su estado inicial, $D_0 = (\{S\}, \overline{f^*})$, donde $\overline{f^*} \geq f^*$ representa el límite superior inicial de la solución óptima (posiblemente infinito), y se termina en el estado final $(\{\emptyset\}, f^*)$.

Definición 2.1.1
Regla de Acotación: Es una función, li (límite inferior), que asocia a un problema abierto $v \in V$ un valor menor o igual que la mejor solución factible en v. Se deben cumplir las siguientes condiciones:

1. $li(v) \leq \min_{x \in v}\{f(x)\}$
2. $li(v) = f(v)$, *si* $v = \{x\}$ *y* x *es una solución factible.*
3. $li(v) \geq li(t)$, *si* v *es generado a partir de una subdivisión de* t.

Definición 2.1.2
Regla de Selección: Sea V *el conjunto de los subproblemas abiertos al inicio de una iteración. La función de selección, aplicada a* V*, es cualquier* δ *tal que si* $V \neq \{\emptyset\}$ *entonces* $\delta(V) \neq \{\emptyset\}$ *y* $\delta(\{\emptyset\}) = \{\emptyset\}$.

Al problema elegido se le llama *nodo*-E [71]. A cada problema seleccionado se le calcula un límite inferior mediante la *Regla de Acotación*. Un árbol $T = (V, \beta)$ en el dominio S, con una función objetivo f y una función *límite inferior*, li, se llama árbol-BB (Branch and Bound) [72]. Las hojas de T son los nodos solución. Los subproblemas críticos son aquellos $v \in V$ tal que $li(v) < f^*$ [109]. Los nodos críticos de un árbol-BB, T, forman el subárbol crítico T^* de T [72]. Es fácil ver que todos los subproblemas críticos deben seleccionarse durante la ejecución de un algoritmo de Branch and Bound. El *camino crítico* es el camino desde el nodo raíz al nodo solución que contiene la solución óptima. El camino crítico con menor longitud es el *camino critico mínimo*.

Normalmente, la función de selección δ está determinada por una función de prioridad heurística h, de forma que los subproblemas se almacenan en la estructura de datos D en un orden parcial o total, no decreciente de h. La función de selección δ no afecta a la convergencia del algoritmo, pero sí, a su eficiencia y al espacio de memoria requerido [41]. A continuación se describen las reglas de selección heurísticas más comúnmente utilizadas:

Búsqueda en Profundidad (Depth-First): La función de selección δ escoge, de entre aquellos subproblemas que estén a más profundidad en el árbol de búsqueda T, el que tenga un menor valor de la función de prioridad heurística h. En este método se usa una estructura de almacenamiento LIFO (Last-In-First-Out) o pila. El árbol asociado a este tipo de selección es aquel en el que uno de los nodos hijo se expande hasta obtener una solución factible del problema o un subproblema no factible. La complejidad espacial de la pila, $O(lw)$, se incrementa linealmente con el nivel del árbol búsqueda alcanzado, l, y el factor de división w [81]. Al descomponer el subproblema elegido, los subproblemas generados a partir de él se introducen en la estructura LIFO, siguiendo un orden decreciente de h.

Desventajas:

- El número de subproblemas inspeccionados es normalmente mayor que el realizado en otros tipos de selecciones, como el de *Primero el Mejor*.

- Si se entra en una rama del árbol de búsqueda que no lleva a la solución óptima, se necesita mucho tiempo para salir de esa rama.

Ventajas:

- Es el método de selección más económico desde el punto de vista del espacio de memoria requerido.
- Conduce más rápidamente a la obtención de una mejor cota superior de f, $\overline{f^*}$.

Búsqueda en Anchura (Breadth-First): En la búsqueda en anchura, al contrario que en la búsqueda en profundidad, la función de selección δ elige, de entre aquellos subproblemas que estén a menor profundidad en el árbol de búsqueda T, el que tenga un menor valor de la función de prioridad heurística h. Este método usa una estructura de almacenamiento FIFO (First-In-First-Out) o cola.

Desventajas:

- Este tipo de estrategia no es recomendable ni desde el punto de vista del tiempo de computación ni desde el del ahorro de memoria.

Ventajas:

- Admite la aplicación de un test de dominancia entre los subproblemas que se encuentran a la misma profundidad del árbol de búsqueda (Definición 2.1.5).

Primero el Mejor (Best-First): La función de selección δ elige aquel subproblema en D, con un menor valor de la función de prioridad heurística, independientemente del nivel de profundidad del árbol de búsqueda (T) en el que se encuentre. La estructura más simple de almacenamiento es una lista ordenada por el valor de $li(v),\ v \in D$.

Desventajas:

- La mejor cota superior, $\overline{f^*}$, suele obtenerse en las fases finales del algoritmo.
- Crecimiento exponencial en el espacio de memoria necesario cuando se incrementa la profundidad del árbol de búsqueda. La complejidad espacial es $O(w^l)$.

Ventajas:

- Difícilmente un subproblema es descompuesto innecesariamente. En [43] se demuestra que usando *Best-First* se exploran menos nodos que usando otra estrategia, cuando hay que encontrar todas las soluciones óptimas, no siendo cierto cuando sólo se requiere encontrar una.

Modelo Híbrido: El modelo híbrido es una combinación de la *Búsqueda en Profundidad* y de la *Primero el Mejor*. Se basa en aplicar alternativamente cada una de las estrategias anteriores. Primero se realiza una búsqueda en profundidad (hasta que no se pueda proseguir la búsqueda por la rama del árbol generada). De entre los nodos activos generados en la búsqueda en profundidad, se selecciona uno, siguiendo una estrategia *Primero el Mejor* y a partir de él se realiza una nueva búsqueda en profundidad, repitiendo el proceso hasta la finalización del algoritmo. Esta regla de selección trata de obtener las ventajas de las reglas en las que está basada [124].

Además de los tipos de selección descritos, existen los llamados *heurísticos* que son análogos al *Primero el Mejor*, donde la función heurística h puede ser diferente de la función de acotación li [60].

Después de la selección de un subproblema v, la *Regla de División* crea un conjunto de nuevos subproblemas a partir de v.

Definición 2.1.3
La Regla de División de un nodo-E, v, en un algoritmo de Branch and Bound corresponde a:

- *Una partición de v, si v no es un nodo solución, es decir, no es lo suficientemente pequeño para ser resuelto.*
- *La solución de v, en otro caso.*

Sea $f^*(v^i)$ el valor óptimo de la solución del nodo v^i, que será descompuesto en los nodos $v^{i1}, v^{i2}, \ldots, v^{im}$ por la regla de división. Entonces, se tiene que:

$$f^*(v^i) = \min_{k=1,\ldots,m} f^*(v^{ik})$$

Es decir, el valor óptimo de la solución de un nodo puede obtenerse mediante la evaluación de sus nodos hijos. En la práctica, las reglas de división cumplen que cada solución factible de un nodo padre es también solución de al menos uno de sus hijos. Estas reglas de división dependerán del problema concreto al que se aplique el algoritmo de Branch and Bound.

En cada iteración se puede generar un nuevo límite superior de la mejor solución, $\overline{f^*}$, de dos formas: porque el nodo generado por descomposición es un nodo solución, o porque se encuentra una solución factible mejor durante la computación de una partición en una descomposición.

La *Regla de Eliminación* permite hacer una búsqueda inteligente en S que evite considerar subproblemas que se sabe que no conducirán a una solución óptima del problema original.

Definición 2.1.4
La Regla de Eliminación, en cada iteración, elimina todos los subproblemas activos y solución, v, de forma que $li(v) > \overline{f^*}$.

La definición anterior está soportada por el siguiente lema:

Lema 2.1.1
En una iteración, si $li(v) > \overline{f^*}$, entonces v no puede conducir a una solución factible que es mejor que la mejor solución factible conocida [22, 57].

La *Regla de Eliminación* puede generalizarse mediante un *Test de dominancia*:

Definición 2.1.5
Test de dominancia: Se dice que un nodo v^i domina a un nodo v^j si se puede probar que la solución óptima de v^i es menor que la solución óptima de v^j. Si v^i tiene una solución factible, v^i debe producir una solución por lo menos tan buena como v^j. Si v^i no es factible, v^j tampoco debe ser factible. En ambos casos v^j puede ser eliminado.

El *Test de dominancia* se basa en una relación de dominancia $\Re$. Denotaremos $v^i \Re v^j$ si v^i domina a v^j. De acuerdo con Ibaraki [59], la relación de dominancia debe satisfacer las siguientes propiedades:

- $\Re$ es parcialmente ordenada (cumple las propiedades reflexiva, antisimétrica y reflexiva).
- $v^i \Re v^j \Rightarrow v^j$ no es un descendiente propio de v^i
- ($v^i \Re v^j$ y $v^i \neq v^j$)$\Rightarrow v^i \Re v^{j*}$, para todos los descendientes v^{j*} de v^j.

Definición 2.1.6
Se dice que una relación de dominancia $\Re$ es consistente con la función g si $v^i \Re v^j \Rightarrow g(v^i) \leq g(v^j),\ \forall v^i, v^j \in V$.

Normalmente, $\Re$ es consistente con la función de prioridad heurística h, la cual establece la ordenación de los subproblemas en la estructura de datos D para su procesamiento. De esta forma, se seleccionarán primero los subproblemas más prometedores, ya que aportarán mejoras en el valor del límite superior $\overline{f^*}$, permitiendo reducir la búsqueda mediante la *Regla de Eliminación*.

Como puede observarse, no existe una regla más importante que las demás, ya que todas ellas cooperan conjuntamente para la resolución del problema. El objetivo principal es reducir el árbol de búsqueda para obtener rápidamente la mejor solución. Aunque la *Regla de Eliminación* es la que realiza la poda del árbol en última

Algorithm 2.1: Algoritmo genérico de Branch and Bound.

```
Entrada: S, f
Salida  : Q
Iniciar la lista de trabajo L := {S};
Iniciar la lista final Q := {};
while ( L ≠ {} ) do
    Seleccionar un problema v de L;                  /* Selección */
    Evaluar v;                                       /* Acotación */
    if v no puede ser eliminado then                 /* Eliminación */
        Dividir v generando v^i, i = 1...m;          /* División */
    if v^i cumple el criterio de parada then         /* Terminación */
        Almacenar v^i en Q;
    else
        Almacenar v^i en L;
```

instancia, esta será mas o menos efectiva, dependiendo de las demás reglas. Una *Regla de Acotación* que obtenga un buen límite inferior de la posible solución de un subproblema, permitirá caracterizar mejor a los subproblemas que no contienen la mejor solución, para eliminarlos. Para poder eliminar subproblemas, también hay que encontrar un buen límite superior de la solución. Este límite superior se consigue inspeccionando primero los nodos más prometedores, ya que son los que pueden aportar mejores soluciones. Este orden se establece mediante la *Regla de Selección*, de la que también dependen los requerimientos de memoria del problema. Hay que distinguir entre el número total de subproblemas inspeccionados y el número máximo de subproblemas almacenados en la estructura de datos D. La gestión de esta estructura de datos ordenada será más costosa cuanto mayor sea el número de subproblemas que contenga. La *Regla de División* también juega un papel importante ya que de ella dependerá el número de ramas generadas a partir de un nodo del árbol de búsqueda. Por un lado, generar muchas ramas permite obtener información más precisa debido a que los subproblemas generados son menores en tamaño, pero por otro lado, hay que inspeccionar más nodos del árbol en cada división. No sólo es importante el nivel de división realizado, sino también cómo se realiza la división. Por lo tanto, la *Regla de División* debe estar orientada a reducir el espacio de búsqueda pero sin generar un número excesivo de nodos en cada división.

El Algoritmo 2.1 muestra la estructura general de un algoritmo de Branch and Bound, donde se especifica el orden de uso de las reglas mencionadas anteriormente.

2.2. Errores de la Aritmética Computacional

Como se verá en la Sección 2.3, la Aritmética de Intervalos debe implementarse en el computador de forma que se eviten los errores de la Aritmética Computacional para que los resultados obtenidos sean rigurosos.

El problema de la Aritmética Computacional es una consecuencia de que la representación interna de los números reales en un ordenador utiliza un número finito de bits y por lo tanto no todos los números reales pueden ser representados en el ordenador. Actualmente la representación más común de los números reales se lleva a cabo usando el estándar IEEE 754 de punto flotante. Este estándar ha sido ampliamente aceptado y se utiliza prácticamente en todos los procesadores y coprocesadores numéricos actuales.

La representación de un número real en formato de punto flotante de longitud fija tiene la forma:

$$x = m \cdot b^e \tag{2.1}$$

donde m es la mantisa, b es la base y e es el exponente. Los números son representados internamente en base $b = 2$ y una mantisa normalizada con $\frac{1}{2} \leq m < 1$ [104].

El conjunto de números que el ordenador puede representar lo denotaremos como $\mathbb{R}_M$. Debido a esta representación finita de los números reales, la computación numérica puede sufrir de varios tipos errores:

- Errores en los datos. En el contexto de la evaluación de expresiones, sabemos que los valores de las constantes, parámetros y variables no son expresados de forma exacta. Por lo tanto, el ordenador tiene un valor aproximado de ellos.
- Errores de redondeo. La aritmética del ordenador no tiene acarreos exactos. La mayoría de los resultados de las operaciones aritméticas pueden ser matemáticamente inexactos ya que todos los resultados tienen que ser expresados como un número perteneciente a $\mathbb{R}_M$. En el cálculo, el resultado obtenido es una aproximación del verdadero resultado. La diferencia entre el resultado obtenido y el resultado verdadero se denomina error de redondeo. El estándar IEEE 754-1985 [129], sobre aritmética de punto flotante, establece cuatro modos de redondeo: al *más cercano*, *superior*, *inferior* y *al cero*, siendo el redondeo al *más cercano* el establecido por defecto.
- Errores de truncamiento. Muchas cantidades matemáticas, como integrales y derivadas, son definidas como el límite de secuencias infinitas de operaciones. Como en el caso concreto de la diferenciación de funciones, donde los límites están definidos explícitamente. La computación puede llegar hasta un punto

finito de la secuencia. El error resultante de la aproximación es el denominado error de truncamiento.

Normalmente, un resultado producido por la ejecución de un programa de ordenador puede verse afectado por uno o más errores. No es posible establecer los límites exactos de distinción entre los tipos de errores y obviamente, no podemos eliminarlos por completo. Por ejemplo, números como π o $\frac{1}{3}$, que aparecen repetidamente en las fórmulas, son imposibles de representar exactamente.

Como ejemplo de error de redondeo, podemos encontrar en la bibliografía la descripción del fallo de un misil Patriot en la guerra del golfo y la explosión del Arian 5, desastres supuestamente debidos a errores de la Aritmética Computacional se describen en el Apéndice B. Para enfatizar más sobre el problema, se mostrará a continuación un ejemplo debido a Rump [119]. La siguiente ecuación:

$$f(x,y) = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + \frac{x}{2y} \qquad (2.2)$$

fue evaluada para $x = 77617$ e $y = 33096$. El programa para su cálculo fue escrito en Fortran. Este fue compilado y ejecutado sobre una SparcStation SLC. La exactitud de las tres precisiones probadas, simple, doble y extendida fue de 6, 14 y 35 dígitos decimales, respectivamente. Sólo se usaron las operaciones básicas, por lo que las potencias se realizaron mediante multiplicaciones. Los resultados obtenidos fueron los siguientes:

$$\begin{aligned} \text{(precisión simple) } f &= 6.33825 \cdot 10^{29} \\ \text{(precisión doble) } f &= 1.1726039400532 \\ \text{(precisión extendida) } f &= 1.17260394005317863185883490452018 38 \end{aligned}$$

Observando los resultados, uno puede concluir que el resultado en simple precisión es erróneo. Además, uno puede (erróneamente) concluir que el resultado en doble precisión es preciso, ya que concuerda en 13 dígitos con el resultado en precisión extendida. Sorprendentemente para muchos, todos los resultados son incorrectos, incluso en el primer dígito ya que el signo es incorrecto. En [76] se habla también de este ejemplo, mostrando el resultado exacto, que fue obtenido mediante la Aritmética de Intervalos implementada en el software VPI [37], usando una precisión variable con 40 dígitos de exactitud. El resultado está incluido en el intervalo:

[-0.827396059946821368114116509547981629 2005,
-0.827396059946821368114116509547981629 1986]

Si el resultado se hubiera calculado usando Aritmética de Intervalos con poca precisión, se hubiera obtenido un intervalo menos preciso, pero que contiene el resultado correcto. Un área del análisis de intervalos, donde todavía se está trabajando

bastante, es el desarrollo de algoritmos de intervalos que produzcan resultados que se ciñan lo más posible a la solución exacta de los problemas [83, 84].

2.3. Aritmética de Intervalos

La Aritmética de Intervalos apareció como una forma de evitar los errores de redondeo producidos por el computador. La Aritmética de Intervalos es una generalización de la Aritmética Real donde los intervalos reemplazan a los números reales y la Aritmética de Intervalos reemplaza a la Aritmética Real. Los resultados teóricos obtenidos mediante el análisis numérico están basados en números exactos y en una aritmética exacta, mientras que en casos prácticos aparece la necesidad del redondeo cuando los números no son representables en la computadora (véase Sección 2.2). El propósito de la Aritmética de Intervalos es obtener unos límites superior e inferior de los errores cometidos en la Aritmética Computacional. Varios autores trabajaron sobre la idea de limitar los errores de la Aritmética Computacional mediante intervalos [35, 132]. La Aritmética de Intervalos se atribuye a Moore con la aparición de su libro *Interval Analysis* en 1966 [92]. Moore transformó esta simple idea en una herramienta para el análisis del error. En vez de tratar sólo errores de redondeo, Moore extendió el uso de la Aritmética de Intervalos para limitar el efecto de los errores de todas clases, incluyendo errores de aproximación y errores en los datos (véase la Sección 2.2). A partir del trabajo de Moore han aparecido miles de artículos y decenas de libros dedicados a este tema [50].

Nosotros nos centraremos en el análisis real de intervalos. Por este motivo cuando se haga uso de la Aritmética de Intervalos se estará referenciando la Aritmética de Intervalos de Reales. Se puede encontrar una amplia bibliografía sobre el análisis de intervalos complejos en [50] y una descripción del mismo en [64].

2.3.1. Extensión Natural a Intervalos

Para una operación binaria, "$\circ$", sobre intervalos, con $\circ \in \{+, -, \cdot, /\}$, se pretende que:

$$X \circ Y = [\min_{\substack{x \in X \\ y \in Y}} x \circ y, \max_{\substack{x \in X \\ y \in Y}} x \circ y] \tag{2.3}$$

Análogamente, para una función unaria ϕ sobre un intervalo:

$$\phi(X) = [\min_{x \in X} \phi(x), \max_{x \in X} \phi(x)] \tag{2.4}$$

Obteniéndose así el rango exacto de las correspondientes operaciones. Sin embargo, esta exactitud no se obtiene siempre debido a las operaciones definidas sobre intervalos:

$$X + Y = [\underline{x} + \underline{y}, \overline{x} + \overline{y}] \tag{2.5}$$

$$X - Y = [\underline{x} - \overline{y}, \overline{x} - \underline{y}] \tag{2.6}$$

$$X \cdot Y = [a, b] \tag{2.7}$$

$$a = \min\{\underline{x}\underline{y}, \overline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\overline{y}\}$$

$$b = \max\{\underline{x}\underline{y}, \overline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\overline{y}\}$$

$$\frac{1}{Y} = [\frac{1}{\overline{y}}, \frac{1}{\underline{y}}],\ 0 \notin Y \tag{2.8}$$

Definición 2.3.1
Si una función $f : \mathbb{R}^n \to \mathbb{R}$ es computable como una expresión, algoritmo o programa de computador, utilizando las cuatro operaciones básicas sobre intervalos, entonces se define la extensión natural de la Aritmética Real a la Aritmética de Intervalos de f, como aquella en la que los operandos reales se sustituyen por intervalos de reales y las operaciones reales por operaciones sobre intervalos.

Para las funciones $f : D \subset \mathbb{R}^n \to \mathbb{R}^m$ se define:

$f(X)$	El rango de f en X.
$F(X)$	Extensión de f a intervalos.
$\overline{F}(X)$	El límite superior de $F(X)$.
$\underline{F}(X)$	El límite inferior de $F(X)$.

Como se puede observar, la resta y la división definidas en $\mathbb{I}$ no son las inversas de la suma y la multiplicación. Por ejemplo:

$$\text{Si } X = [1, 2] \text{ entonces } X - X = [-1, 1] \text{ y } \tfrac{X}{X} = [\tfrac{1}{2}, 2].$$

Aunque las operaciones suma y multiplicación de intervalos son conmutativas y asociativas, no verifican la ley distributiva, definiéndose la nueva ley como subdistributiva:

$$\text{Para } A, B, C \in \mathbb{I},\ A \cdot (B + C) \subseteq A \cdot B + A \cdot C$$

Ejemplo 2.3.1
La función real $f_1(x) = x^2 - x$ también se puede escribir como $f_2(x) = x(x - 1)$. Realizando su extensión natural sobre el intervalo $X = [0, 1]$ se obtiene que:

$$\begin{array}{lllllll} F_1(X) & = & [0,1]^2 - [0,1] & = & [0,1] - [0,1] & = & [-1,1] \ \ y \\ F_2(X) & = & [0,1]([0,1]-1) & = & [0,1][-1,0] & = & [-1,0] \end{array}$$

Además, el rango real exacto de la función en el intervalo $[0,1]$ es $f([0,1]) = [-1/4, 0]$, ya que la Aritmética de Intervalos suele sobreestimar el rango de la función real en el intervalo X. Esto es debido a que se asume implícitamente que la variable x varía independientemente en el término x^2 y en el término $-x$. Este fenómeno se conoce como *dependencia de intervalos.*

2.3.2. Funciones de Inclusión

En el tratamiento de los problemas de Optimización usando Aritmética de Intervalos, la principal herramienta es el concepto y la aplicación de las *funciones de inclusión.*

Definición 2.3.2
Sea $f : D \subseteq \mathbb{R}^n \to \mathbb{R}$. Una función $F : \mathbb{I}^n(D) \to \mathbb{I}$ se llama función de inclusión de f si:

$$f(X) = \{f(x) : x \in X\} \subseteq F(X), \ \forall X \subseteq D. \tag{2.9}$$

Teorema 2.3.1
Si $F : \mathbb{I}^n \to \mathbb{I}$ es una extensión natural de intervalos de $f : \mathbb{R}^n \to \mathbb{R}$ y f está definida en $X \in \mathbb{I}^n$, entonces F es una función de inclusión de f en X [64].

Por lo tanto la Aritmética de Intervalos es una herramienta que nos permite obtener *funciones de inclusión* [92, 110]. Por ejemplo, en la Figura 2.1 se ha representado una función $f : \mathbb{R} \to \mathbb{R}$, el intervalo $X = [\underline{x}, \overline{x}]$, así como $f(X)$ y $F(X) = [\underline{F}(X), \overline{F}(X)]$, obtenida con las rutinas BIAS [66].

2.3.3. Extensión de la Aritmética de Intervalos

En las operaciones básicas definidas anteriormente (Ecuaciones (2.5) y (2.8)) se ha excluido la operación de división cuando el intervalo del denominador contiene el cero. Para soportar esta característica se realizó una extensión de la Aritmética de Intervalos llamada Kahan-Novoa-Ratz, ya que fueron estos autores los que presentaron las primeras versiones [63] y las mejoras de las mismas debidas a correcciones de inconsistencias [101, 114]. En la Aritmética de Kahan-Novoa-Ratz la operación de división de dos intervalos X e Y, con $0 \in Y$, se define de la siguiente forma:

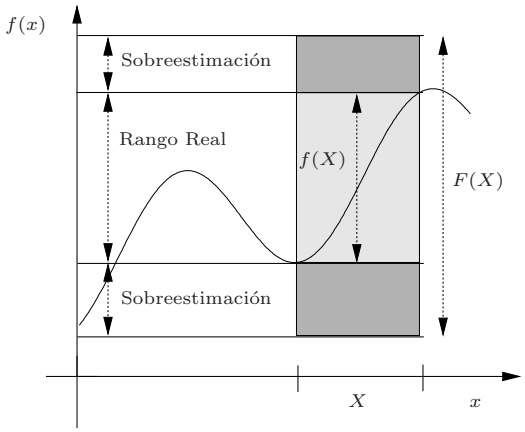


Figura 2.1: Función de inclusión de la función $f(x) = x + sin(5x)$ en el intervalo $X = [3.11, 3.76]$, obtenida con las rutinas BIAS [66].

$$\frac{[\underline{x},\overline{x}]}{[\underline{y},\overline{y}]} = \begin{cases} X[1/\overline{y}, 1/\underline{y}] & \text{si } 0 \notin Y, \\ [-\infty,\infty] & \text{si } 0 \in X \text{ y } 0 \in Y, \\ [\overline{x}/\underline{y},\infty] & \text{si } \overline{x} < 0 \text{ e } \underline{y} < \overline{y} = 0, \\ [-\infty,\overline{x}/\overline{y}] \cup [\overline{x}/\underline{y},\infty] & \text{si } \overline{x} < 0 \text{ e } \underline{y} < 0 < \overline{y}, \\ [-\infty,\overline{x},\overline{y}] & \text{si } \overline{x} < 0 \text{ y } 0 = \underline{y} < \overline{y}, \\ [-\infty,\underline{x}/\underline{y}] & \text{si } 0 < \underline{x} \text{ e } \underline{y} < \overline{y} = 0, \\ [-\infty,\underline{x}/\underline{y}] \cup [\underline{x}/\overline{y},\infty] & \text{si } 0 < \underline{x} \text{ e } \underline{y} < 0 < \overline{y}, \\ [\underline{x}/\overline{y}],\infty] & \text{si } \underline{x} < 0 \text{ y } 0 = \underline{y} < \overline{y}, \\ 0 & \text{si } 0 \notin X \text{ y } 0 = Y. \end{cases} \tag{2.10}$$

Por ejemplo, de acuerdo con la Ecuación (2.10), [2,3]/[-3,4] = [-∞,-2/3] ∪ [1/2,∞], donde [-∞,-2/3] ∪ [1/2,∞] representa el rango real de $x/y,\ x \in [2,3],\ y \in [-3,4]$. No hay que confundir la Aritmética de Kahan-Novoa-Ratz con otras extensiones de la Aritmética de Intervalos diseñadas para otros propósitos. Por ejemplo, la Aritmética de Kaucher se usó para obtener estimadores internos y externos para el conjunto de soluciones de un sistema lineal basado en intervalos [123]. Otro ejemplo es el debido a Markov [83, 84], donde el autor propone una aritmética que reduce la sobreestimación de los rangos obtenidos con la Aritmética de Intervalos tradicional. Estos tipos de aritméticas son a veces llamadas *Aritméticas de Intervalos Extendidas*.

2.3.4. Extensiones de funciones estándar

Para funciones monótonas es fácil definir su extensión a intervalos. Por ejemplo:

$$\sqrt{X} = [\sqrt{\underline{x}}, \sqrt{\overline{x}}],\ X \geq 0 \tag{2.11}$$

$$ln(X) = [ln(\underline{x}), ln(\overline{x})],\ X \geq 0 \tag{2.12}$$

$$e^X = [e^{\underline{x}}, e^{\overline{x}}] \tag{2.13}$$

Funciones no monótonas, como la función seno, se pueden tratar haciendo uso de la periodicidad de las mismas. Incluso existen estudios de hardware específico para su computación [55]. De hecho, se pueden obtener extensiones naturales de intervalos para cualquier función que pueda ser representada con un programa de computador. Siempre se pretende que los límites obtenidos por la Aritmética de Intervalos sean lo más cercanos posibles a los límites reales de la función $f(X)$. Estos límites pueden ser computados mediante cualquier expansión (racional, series de Taylor, etc.) que tenga una fórmula explícita del término de error. Así, los rangos de funciones estándar pueden ser aproximados con la precisión deseada dentro de los límites del sistema de punto flotante.

Teorema 2.3.2
Supóngase una extensión de intervalos, $F(X)$, de una función real $f : \mathbb{R}^n \to \mathbb{R}^m$, obtenida mediante una secuencia de computaciones con límites rigurosos de los rangos $\sigma_j(X)$, de funciones estándar $\{\sigma_j\}_{j=1}^{p}$, entremezcladas con las cuatro operaciones básicas sobre intervalos. Entonces, la extensión natural de intervalos F es una función de inclusión de f [64] .

2.3.5. Propiedades de las extensiones de intervalos

En esta sección vamos a describir algunas propiedades (adicionales a las vistas anteriormente) para la formulación y codificación de funciones objetivo con Aritmética de Intervalos.

Una característica deseable de las funciones de inclusión es que sean monótonas.

Definición 2.3.3
Una extensión de intervalos, F, es una función de inclusión monótona o isótona, si para $Y \subseteq X$ se cumple que $F(Y) \subseteq F(X)$

Teorema 2.3.3
La extensiones naturales de intervalos, tal como se definieron en el Teorema 2.3.2 son funciones de inclusión monótonas [64].

La principal consecuencia del Teorema 2.3.3 es que permite resolver problemas de Optimización Global, como se verá más adelante.

Las primeras críticas a la Aritmética de Intervalos fueron debidas a que los límites de los rangos obtenidos eran demasiado grandes o pesimistas para ser útiles. Esto ocurre frecuentemente cuando la Aritmética de Intervalos se usa de una forma directa para extender funciones reales a intervalos, pero una mejor comprensión de sus propiedades conduce a algoritmos apropiados y efectivos.

Teorema 2.3.4
Supóngase que $f : \mathbb{R}^n \to \mathbb{R}$ se escribe como una secuencia de funciones estándar $\{\sigma_j\}_{j=1}^{p}$, entremezcladas con las cuatro operaciones básicas sobre intervalos. Supóngase que en esta expresión, cada una de las variables $\{x_i\}_{i=1}^{n}$ ocurren una sola vez. Si F es la extensión de intervalos de f y es calculada con Aritmética de Intervalos exacta y rangos exactos para cada σ_j, entonces $F(X) = f(X)$, es decir, el intervalo de inclusión resultante es igual al rango exacto de la función real [64].

Ejemplo 2.3.2
Para $f(x) = x - x^2$, $F([0,1]) = [0,1] - [0,1]^2 = [-1,1]$ *y*
para $f(x) = x(1-x)$, $F([0,1]) = [0,1](1-[0,1]) = [0,1]$.

El teorema anterior y la ley subdistributiva, sugieren que para maximizar la exactitud de una extensión de intervalos, las expresiones o algoritmos definidos deberían reescribirse de forma que se minimice el número de ocurrencias de cada variable o subexpresión. De hecho, ésta es una heurística razonable en muchos casos.

Computar el rango exacto de una función arbitraria f, sobre una caja X, es similar a optimizar f sobre X. De cualquier forma se desean propiedades asintóticas de los límites de los intervalos del rango de f, conforme el ancho de la caja X tiende a cero.

Definición 2.3.4
Sea $F : \mathbb{I}^n \to \mathbb{I}^m$ una extensión de intervalos de $f : \mathbb{R}^n \to \mathbb{R}^m$, evaluada en la caja X. Si existe una constante K, independiente de la caja X, tal que $w(F(X)) - w(f(X)) \leq Kw(X)^\alpha$, para todas las cajas X con $w(X)$ suficientemente pequeño y un $\alpha > 0$ fijo, entonces se dice que F es una función de inclusión de orden α o α-convergente. Cuando α es uno o dos, entonces la función de inclusión es de primer o segundo orden, respectivamente.

Teorema 2.3.5
Las extensiones naturales de Intervalos de una función, donde se usan los cuatro operadores básicos de la Aritmética de Intervalos y rangos exactos de funciones estándar, son de primer orden [1, 64, 94, 110].

Extensiones de segundo orden se pueden obtener mediante extensiones de intervalos de series de Taylor, también llamadas *formas centradas*, y obteniendo límites del rango de las derivadas.

Definición 2.3.5
Supóngase que $f : D \subseteq \mathbb{R}^n \to \mathbb{R}$, con derivadas continuas, $X \subseteq D$ y $c \in X$. Entonces, la extensión del Teorema del valor medio de f a intervalos sobre X y centrada en c se define por: $F_c(X, c) = f(c) + (X - c)^T \nabla F(X)$, donde $\nabla F(X)$ es el intervalo de inclusión del rango de ∇f, componente a componente, sobre X.

Del teorema del valor medio y de las propiedades de la Aritmética de Intervalos se deduce que $F_c(X, c)$ es una función de inclusión del rango de f sobre X [64, 99].

Teorema 2.3.6
Supóngase que los componentes de ∇F son extensiones a intervalos de ∇f de orden al menos uno. Entonces F_c es una extensión de intervalos de f de orden dos [67, 110].

Desafortunadamente, aunque sea deseable en algoritmos de Optimización Global, parece difícil obtener extensiones de intervalos de orden mayor de dos para funciones arbitrarias [19, 34].

El Teorema del valor medio es un caso especial de las *formas centradas* para funciones de inclusión basadas en intervalos. Otras *formas centradas*, así como su discusión matemática, aparecen en [110]. Dichas formas deben ser usadas para obtener límites más precisos en casos específicos.

Krawczyk y Neumaier [99] introdujeron las formas basadas en la pendiente (*Slope Forms*), mostrando que:

Definición 2.3.6
Si para un intervalo $S \in \mathbb{I}$ tal que, $\forall x \in X$ se tiene que

$$f(x) = f(c) + s \cdot (x - c) \text{ para algún } s \in S, \tag{2.14}$$

entonces el intervalo $F_s(X) = f(c) + S \cdot (X - c)$ contiene el rango de f sobre X, es decir, $f(X) \subseteq F_s(X)$. El intervalo S puede calcularse por medio de la extensión de intervalos de la pendiente y no sólo por la extensión de intervalos de la derivada. Si se usa la extensión de intervalos de la pendiente, entonces se expande f respecto a un punto arbitrario pero fijo $c \in X$.

Definición 2.3.7
Sea f una función diferenciable. La función $S_f : \mathbb{R}^m \to \mathbb{R}^n$ con $f(x) = f(c) + s_f(c, x)(x - c)$ se llama pendiente de $f : \mathbb{R}^n \to \mathbb{R}$ (entre c y x). En el caso unidimensional se tiene que:

$$s_f(c,x) = \begin{cases} \frac{f(x)-f(c)}{x-c}, & if\ x \neq c \\ \tilde{s}, & if\ x = c \end{cases}$$

donde $\tilde{s} \in \mathbb{R}$ puede elegirse arbitrariamente. Si se asume que f es diferenciable y la pendiente continua, se puede definir $\tilde{s} = \nabla f(c)$.

Además, se define la extensión a intervalos de la pendiente de f sobre el intervalo X como:

$$s_f(c,X) = \{s_f(c,x) : x \in X,\ x \neq c\},$$

donde f no necesita ser diferenciable.

Desde un punto de vista práctico deben tenerse en cuenta las siguientes consideraciones:

1. Es fácil ver que $S = s_f(c,X)$ satisface la Ecuación (2.14) y
$$f(x) \in f(c) + S \cdot (x-c) \subseteq f(c) + S \cdot (X-c).$$
2. A menudo se usa $c = m(X)$ para computar la extensión de intervalos de la pendiente.
3. Si se supone que f es continuamente diferenciable se tiene que [99]:
$$\nabla f(X) = \bigcup_{c \in X} s_f(c,X)$$

Ejemplo 2.3.3
Sea $f(x) = x^2 - 4x + 2$. Para el intervalo $X = [1,7]$ y $c = 4$ se tiene que:

$$\begin{aligned} &s_f(c,X) = X \\ &F_s(c,X) = f(c) + s_f(c,X) \cdot (X-c) = [-19,23] \end{aligned}$$

mientras que la extensión natural y la del Teorema del valor medio a intervalos dan como resultado:

$$\begin{array}{lll} F(X) & = X^2 - 4X + 2 & = [-25,47] \\ \nabla F(X) & = 2X - 4 & = [-2,10] \\ F_c(X,c) & = f(c) + \nabla F(X) \cdot (X-c) & = [-28,32] \end{array}$$

que son mayores que el obtenido por la forma basada en la pendiente. En realidad, la forma basada en la pendiente da generalmente resultados más precisos que la aplicación del Teorema del valor medio generalizado, usando el mismo valor de c [99].

En [111] se muestra que no es siempre bueno usar la extensión de intervalos del Teorema del valor medio, especialmente para cajas grandes, ya que la sobreestimación tiende cuadráticamente a infinito con la anchura de la caja, mientras que la sobreestimación de la extensión natural de intervalos es lineal. Un problema todavía abierto en el análisis de intervalos es saber cual debe ser la anchura de un intervalo para que la extensión del Teorema del valor medio proporcione un resultado al menos tan bueno como el obtenido por la fórmula original [50, 137, 138]. Estudios más extensos sobre la Aritmética de Intervalos se encuentran en [1, 50, 94, 99]

2.3.6. Aritmética de Intervalos Computacional

En el Teorema 2.3.4 se mostró que existen situaciones en las que la Aritmética de Intervalos puede computar límites rigurosos de los rangos de las funciones, asumiendo computaciones de precisión infinita. Si las operaciones sobre intervalos se realizan en un computador, puede ocurrir que el redondeo usado por defecto dé lugar a resultados erróneos [104]. Por ejemplo, el rango de la expresión [0.123,0.456]+[0.0116,0.0214] es [0.1346,0.4774]. Si se usan sólo tres dígitos decimales de precisión y un tipo de redondeo al *más cercano*, el resultado es [0.135,0.477] $\not\supseteq$ [0.1346,0.4774]. Si en vez de usar el redondeo al *más cercano* se usa un redondeo *directo*, donde para un intervalo X, $\underline{x}$ se redondea al mayor número representable en el computador, menor que $\underline{x}$ y $\overline{x}$ al menor número representable por el computador, mayor que $\overline{x}$, entonces el intervalo computado contiene forzosamente el rango exacto. En el ejemplo anterior el resultado usando redondeo *directo* sería [0.134,0.478] $\supset$ [0.1346,0.4774].

Así, con el redondeo *directo*, la Aritmética de Intervalos Computacional puede ser definida de forma que el resultado de las operaciones básicas contengan el resultado que sería obtenido con la Aritmética de Intervalos Real. El desarrollo matemático de este concepto puede encontrarse en [68]. Para simplificar la notación, no se hará una distinción explicita entre el espacio $\mathbb{I}$ y el correspondiente del conjunto de intervalos computacionales, pero debe ser inferido del contexto.

Para que la Aritmética de Intervalos Computacional siga teniendo la propiedad de generar funciones de inclusión (ver Sección 2.3.2), hay que hacer uso de los modos de redondeo *superior* e *inferior*, establecidos en el estándar IEEE 754-1985 [129]. Cuando se computen los límites superiores e inferiores de cada operación sobre intervalos, hay que establecer el modo de redondeo *superior* e *inferior*, respectivamente. Mediante este estándar, los modos de redondeo necesarios para la Aritmética de

Intervalos están disponibles en una amplia gama de computadores.

La propiedad de monotonía de la *función de inclusión* (Definición 2.3.3) ha sido sugerida como criterio de parada en algoritmos iterativos basados en intervalos. No cumplir esta propiedad implica que predominan los errores por redondeo *directo*, es decir, como para $Y \subseteq X$ se cumple que $F(Y) \subseteq F(X)$, conforme se reduce el ancho de X también se reduce el ancho de $F(X)$. Cuando esto deja de ocurrir durante un número suficientemente grande de iteraciones, se puede concluir que predominan los errores debidos al redondeo, imposibilitando el progreso del algoritmo. Por lo tanto, en un computador cuando $w(X) \to 0 \not\Rightarrow w(F(X)) \to 0$ [111].

2.4. Diferenciación Automática

Cuando los problemas donde se quieren aplicar algoritmos de Optimización Global son diferenciables existen herramientas que hacen uso de las derivadas para acelerar dichos algoritmos, como por ejemplo, el test de monotonía o el método de Newton, que trataremos con más profundidad en el siguiente capítulo. Para calcular las derivadas de una función se pueden usar los siguientes métodos:

- Diferenciación numérica.
- Diferenciación simbólica.
- Diferenciación automática.

La diferenciación numérica fue la primera aproximación utilizada donde los valores de las derivadas se obtienen mediante aproximaciones por diferencias finitas. De todas formas con esta solución no se pueden calcular inclusiones de las derivadas. Además, los resultados calculados con diferenciación numérica son frecuentemente más inexactos que los calculados con diferenciación automática, por lo que no es conveniente su uso. La diferenciación simbólica aplica las reglas de diferenciación a la fórmula analítica de una función y determina una forma analítica para la derivada. Esto puede ser resuelto incluso de forma automática y ha sido implementado en sistemas algebraicos como Maple o Mathematica. Otros paquetes, como LANCELOT [17], tienen programas hechos a medida para interpretar formatos de entrada estándar, como el formato MPS [97], SIF [17] o el propuesto por Neumaier [99], escribiendo subrutinas en un lenguaje destino, como FORTRAN, para cada computación requerida. El problema de la diferenciación simbólica es que frecuentemente las expresiones analíticas obtenidas para la derivada son muy extensas y complejas. Además, el usuario debe aprender formatos de entrada y paquetes especiales que no son universales.

La alternativa a estas aproximaciones es la diferenciación automática. En ella se usan los valores de los argumentos mientras se aplican las reglas de diferenciación, evitando el aumento del número de fórmulas, como ocurre con la diferenciación simbólica y se efectúa a la vez el cálculo de las derivadas y del valor de la función.

En la diferenciación automática existen dos modos de operar: avance y retroceso. El método de retroceso [47] es más eficiente que el de avance pero tiene muchos más requerimientos de memoria y es más complejo de implementar. El modo de diferenciación automática de avance es análogo a la Aritmética de Intervalos, en el sentido de que se define una aritmética sobre un conjunto extendido de objetos: intervalos cuando se usa Aritmética de Intervalos y reales en el caso de la diferenciación automática real. Un objeto para la diferenciación automática es un par ordenado de la forma (u, u'), si estamos interesados en el gradiente, o un trío ordenado (u, u', u''), si estamos interesados en la matriz Hessiana. Los elementos de u, u' y u'' pueden ser números reales o intervalos. Generalizando a intervalos, podemos decir que $u \in \mathbb{I}$, $u' \in \mathbb{I}^n$ y $u'' \in \mathbb{I}^{n\times n}$, donde u representa el valor de la función $F(X)$, u' el valor del gradiente $\nabla F(X)$ y u'' el valor de la matriz Hessiana $\nabla^2 F(X)$, siendo $F : \mathbb{I}^n \to \mathbb{I}$ la extensión a intervalos de la función objetivo $f : \mathbb{R}^n \to \mathbb{R}$ sobre $X \in \mathbb{I}^n$. A continuación se van a definir las operaciones para obtener los valores de la matriz Hessiana. Para obtener el gradiente sólo hay que omitir el tercer componente del trio ordenado (u, u', u''). Aplicando las reglas de diferenciación se obtienen las cuatro operaciones básicas:

$$(u, u', u'') + (v, v', v'') = (u + v, u' + v', u'' + v'') \tag{2.15}$$

$$(u, u', u'') - (v, v', v'') = (u - v, u' - v', u'' - v'') \tag{2.16}$$

$$(u, u', u'') \cdot (v, v', v'') = (uv, uv' + u'v, uv'' + u'(v')^T + v'(u')^T + u''v) \tag{2.17}$$

$$\begin{aligned}(u, u', u'') \,/\, (v, v', v'') = {} & u/v, (u'v - uv')/v^2, \\ & (u'' - w'(v')^T - v'(w')^T - wv'')/v), \\ & v \neq 0 \ (0 \notin v)\end{aligned} \tag{2.18}$$

Para las funciones estándar $s : \mathbb{I} \to \mathbb{I}$ se tiene que :

$$s((u, u', u'')) = (s(u), s'(u)u', s'(u)u'' + s''(u)u'(u')^T) \tag{2.19}$$

En expresiones aritméticas, una constante c se identifica con (c,0,0), mientras que una variable independiente x se identifica con (x,1,0).

Ejemplo 2.4.1
Supóngase que se quiere conocer el valor de la siguiente función y de su derivada, en el intervalo $X = [0.99, 1.01]$*:*

$$f(x) = x^4 + x^3 + x$$

La evaluación de esta función puede ser descompuesta en las siguientes operaciones:

$$\begin{aligned} x_1 &= x \\ x_2 &\to x_1^4 \\ x_3 &\to x_1^3 \\ x_4 &\to x_2 + x_3 \\ x_5 &\to x_4 + x_1 \end{aligned}$$

y usando una aritmética con redondeo de cuatro dígitos, se tiene que :

$$\begin{aligned} (x_2, x_2') \to (x,1)^4 &= ([.99, 1.01]^4, 4[.99, 1.01]^3) \\ &\subset ([.9606, 1.041], [3.881, 4.122]), \\ (x_3, x_3') \to (x,1)^3 &= ([.99, 1.01]^3, 3[.99, 1.01]^2) \\ &\subset ([.9702, 1.031], [2.940, 3.061]) \\ (x_4, x_4') \to (x_2, x_2') + (x_3, x_3') &\subset ([1.930, 2.072], [6.821, 7.183]) \\ (x_5, x_5') \to (x_4, x_4') + (x,1) &\subset ([2.920, 3.082], [7.821, 8.183]). \end{aligned}$$

Así, el intervalo de inclusión para $f(X)$ es $F(X) = [2.920, 3.082]$ y el de $f'(X)$ es $F'(X) = [7.821, 8.183]$.

Usualmente, todos los paquetes de software que resuelven el problema de Optimización Global tienen rutinas donde se implementa la diferenciación automática. Un estudio más detallado de la diferenciación automática y su implementación en el paquete software FADBAD/TADIF puede encontrarse en [128]. Enlaces a otros paquetes e información sobre diferenciación automática pueden encontrarse en el Apéndice [B].

Capítulo 3

Optimización Global Intervalar unidimensional

Tal como se mostró en el Capítulo 1, estamos interesados en resolver problemas de Optimización Global sin restricciones, mediante métodos que obtienen todas las soluciones y garantizan que las soluciones encontradas son las globales. Haciendo uso de las herramientas descritas en el Capítulo 2, este capítulo muestra los distintos tipos de técnicas usadas para reducir el espacio de búsqueda en problemas unidimensionales mediante algoritmos de Optimización Global Intervalar. A partir de estas técnicas se presenta un algoritmo básico de Optimización Global Intervalar y, posteriormente, se presentan las aportaciones teóricas desarrolladas en este trabajo de tesis, que permiten proponer un nuevo algoritmo de Optimización Global Intervalar que mejora los anteriores.

3.1. Reglas de Branch and Bound

Los algoritmos de Optimización Global Intervalar hacen uso de técnicas de Branch and Bound. Tal como se muestra en la Sección 2.1.1 los algoritmos de Branch and Bound están caracterizados por cinco reglas: Acotación, Eliminación, Selección, División y Terminación. Aquí se presentan las reglas comúnmente usadas en los algoritmos de Optimización Global Intervalar unidimensionales.

3.1.1. Regla de Acotación

El teorema fundamental de la Aritmética de Intervalos establece un modo natural y riguroso de obtener una *función de inclusión* (véase la Sección 2.3.2). Por lo tanto, dada la función objetivo f, su extensión a intervalos F y un intervalo $X \subseteq S$ podemos concluir que $f(X) \subseteq F(X) = [\underline{F}(X), \overline{F}(X)]$.

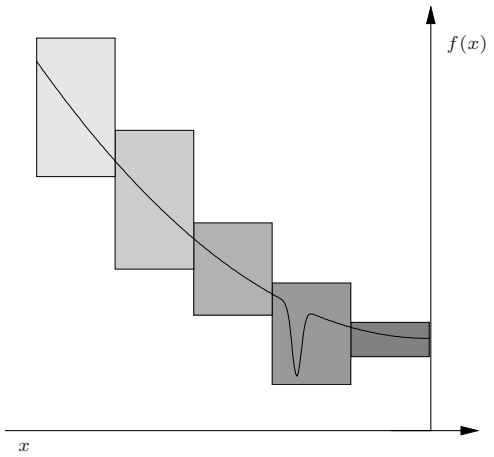


Figura 3.1: Representación esquemática de la regla de acotación.

Tomando como ejemplo la función mostrada en la Figura 1.2, donde se mostró que los métodos de optimización que se basan en evaluaciones puntuales de la región de búsqueda no garantizan una solución global, la Figura 3.1 muestra que la *función de inclusión* obtenida con Aritmética de Intervalos permite tener información global sobre todos los puntos de un intervalo, lo que permite localizar los mínimos globales.

3.1.2. Regla de Eliminación

Las reglas de eliminación que normalmente se usan en Algoritmos de Optimización Intervalar están basadas en los test del punto medio, monotonía, concavidad y de Newton, los cuales se detallan a continuación.

Test del punto medio

Tal como se explicó en la Definición 2.1.4, $\overline{f^*}$ es un límite superior del mínimo global (f^*) de $f(x)$, dentro de la región de búsqueda S. Dado $X \subset S$, si $\overline{f^*} < \underline{F}(X)$,

entonces no puede existir un mínimo global en X [62]. Una forma simple de obtener y actualizar $\overline{f^*}$ es con el menor valor $\overline{F}(X)$ de los intervalos evaluados.

Un mejor valor de $\overline{f^*}$ puede obtenerse mediante la evaluación de f en algún punto x de S. El nombre de este test es debido a que normalmente se usa el punto medio de algún subespacio de S para obtener $\overline{f^*}$.

Ejemplo 3.1.1
Considérese la función $f(x) = x(x-1)$. Inspeccionando la función, se puede observar que $f(0) = 0 = \overline{f^*}$. La evaluación de $F(X)$ sobre $X = [1.001, 100]$ da como resultado el intervalo $F(X) = X(X-1) = [0.001001, 9900]$ y como $\overline{f^*} = 0 < \underline{F}(X) = 0.001001$, no puede existir un mínimo global en $X = [1.001, 100]$.

Para obtener un valor riguroso de $\overline{f^*}$, en los algoritmos de Optimización Global basados en intervalos se usa $\overline{F}(x),\ x \in S$, en lugar de calcular $f(x)$.

Test de corte

Teniendo en cuenta que los intervalos procesados y no eliminados se almacenan en una lista de trabajo L y los que alcanzan el criterio de terminación se almacenan en una lista final Q (ver el Algoritmo 2.1), se puede recorrer L y Q para ver si existen intervalos almacenados que puedan ser eliminados por el *Test del punto medio*. De esta forma se reducen los requerimientos de memoria del algoritmo. La aplicación del *Test del punto medio* a los intervalos de la lista de trabajo y final recibe el nombre de *Test de corte*.

Test de monotonía

El gradiente de la función objetivo es cero en cualquier máximo, mínimo o punto de inflexión de la función. Si $0 \notin F'(X)$, entonces no existe un punto estacionario en X. En particular, el mínimo global sólo podría estar en los extremos de X [111].

Ejemplo 3.1.2
Usando la función anterior: $f(x) = x(x-1)$ y evaluando la derivada de su función de inclusión sobre $X = [0.75, 100]$ se obtiene que $F'(X) = 2X - 1 = [0.5, 199]$. Este resultado implica que $0 \notin f'([0.75, 100])$ y por lo tanto no existe un punto estacionario en este intervalo.

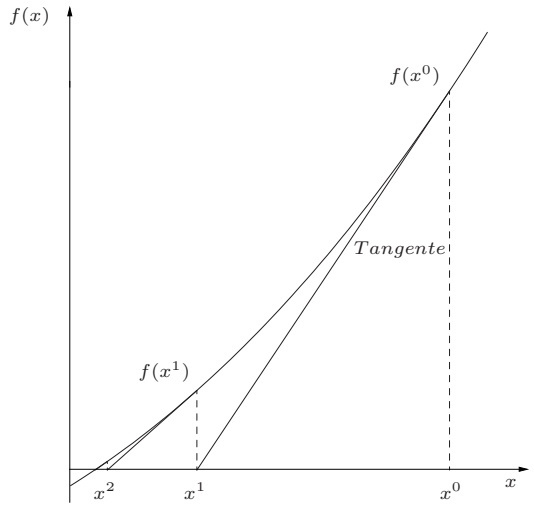


Figura 3.2: Descripción gráfica del método clásico de Newton.

Test de concavidad

Este test separa los mínimos de los máximos. Si $f(x)$ tiene un mínimo en x^*, entonces $f(x)$ debe ser convexa alrededor de x^*. Una condición necesaria para que exista un mínimo en X, es que $F''(X)$ sea positiva, i.e. $\underline{F''}(X) \geq 0$. Por lo tanto, si $\overline{F''}(X) < 0$ el intervalo X puede eliminarse de la región de búsqueda.

El método de Newton unidimensional clásico

En 1669, Newton presentó un nuevo método para la resolución de ecuaciones, utilizándolo para encontrar las raíces de la ecuación cúbica: $x^3 - 2x - 5 = 0$. El método de Newton asume que f es diferenciable y puede ser descrito gráfica o algebraicamente.

En la descripción gráfica de la Figura 3.2 puede observarse que:

$$f'(x^0) = \frac{f(x^0)}{x^0 - x^1} \tag{3.1}$$

por lo tanto,

$$x^1 = x^0 - \frac{f(x^0)}{f'(x^0)} \tag{3.2}$$

Así, a partir de un punto inicial x^0 se produce una nueva aproximación, x^1, de la raíz de f. Si este proceso se repite, se genera una secuencia de puntos, $x^0, x^1, x^2, \ldots, x^n$, con

$$x^{i+1} = x^i - \frac{f(x^i)}{f'(x^i)},\ i = 0, \ldots, n \tag{3.3}$$

que converge al punto $x^* : f(x^*) = 0$, cuando se elige un buen candidato para el punto inicial x^0.

Una interpretación algebraica del método de Newton puede obtenerse aproximando el valor de $f(x)$ en x^0, mediante las series de Taylor. Dado que se pretende que $f(x) = 0$, entonces:

$$0 = f(x^0) + f'(x^0)(x - x^0) + \frac{f''(x^0)(x - x^0)^2}{2!} + \cdots + \frac{f^{(n+1)}(\xi)(x - x^0)^{(n+1)}}{(n+1)!} \tag{3.4}$$

Usando sólo los dos primeros términos de la Ecuación (3.4) se obtiene que:

$$0 \simeq f(x^0) + f'(x - 0)(x - x^0),\ x \approx x^0 - \frac{f(x^0)}{f'(x^0)} \tag{3.5}$$

Sustituyendo x por x^1, se obtiene la Ecuación (3.2). Si $f \in C^2$ y $f' \neq 0$ en la raíz x^* (i.e. x^* es un cero simple), entonces el método de Newton convergerá cuadráticamente a la raíz x^*, si el punto inicial x^0 está lo suficientemente cerca, es decir:

$$\|x^{n+1} - x^*\| \leq c\|x^n - x^*\|^2 \tag{3.6}$$

Para evitar el uso de derivadas, puede usarse el método de la secante, ya que la derivada puede aproximarse por:

$$\nabla f(x^i) \simeq \frac{f(x^i) - f(x^{i-1})}{x^i - x^{i-1}} \tag{3.7}$$

Obteniéndose la siguiente secuencia de puntos que converge a una raíz de f, siempre que se elija un buen punto inicial x^0:

$$x^{i+1} = x^i - f(x^i)\frac{x^i - x^{i-1}}{f(x^i) - f(x^{i-1})} \tag{3.8}$$

La principal desventaja del método de Newton es que puede que no converja y si lo hace, es difícil predecir a qué raíz lo hará. Además, el método de Newton no detecta los casos en los que la función no tiene raíces. El siguiente ejemplo muestra una función para la cual el *método de Newton clásico* diverge.

Ejemplo 3.1.3
Considérese la función $f(x) = x^{\frac{1}{3}}$. *El método de Newton clásico establece que:*

$$x^{n+1} = x^n - \frac{f(x^n)}{f'(x^n)} = -2x^n \tag{3.9}$$

Por lo que la secuencia diverge para cualquier punto inicial distinto de $x^0 = 0$.

Método de Newton de intervalos uni-dimensional

Con el uso de la Aritmética de Intervalos, el método de Newton se puede extender a intervalos. El método de Newton de intervalos fue introducido por Moore [92] y carece de los errores debidos a las posibles divergencias del método clásico.

Teorema 3.1.1
(Del valor medio de Lagrange) Se considera una función $f : [a, b] \to \mathbb{R}$ *definida en un intervalo cerrado y acotado* $[a, b]$. *Si* f *es continua en* $[a, b]$ *y derivable en* $]a, b[$, *entonces existe al menos un punto* $\xi \in]a, b[$ *tal que*

$$f(b) - f(a) = f'(\xi)(b - a) \tag{3.10}$$

La conclusión, desde el punto de vista geométrico, del Teorema 3.1.1 es: existe al menos un punto de la gráfica de f, distinto de sus extremos a y b, en el que la tangente a la gráfica es paralela a la cuerda ab.

A partir del Teorema 3.1.1, suponiendo que $0 = f(b)$:

$$0 = f(b) = f(a) + f'(\xi)(b - a) \tag{3.11}$$

entonces,

$$b = a - \frac{f(a)}{f'(\xi)} \Rightarrow b \in N(X) = a - \frac{f(a)}{F'(X)},\ si\ a, b \in X, \tag{3.12}$$

siendo $N(X)$ el operador de Newton de intervalos. La extensión a intervalos del método de Newton se define de la siguiente forma:

$$X^{n+1} = N(X^n) \cap X^n, \text{ hasta que } X^{n+1} = X^n \text{ ó } X^{n+1} = \{\emptyset\} \tag{3.13}$$

Normalmente $a = m(X)$. Si $0 \in F'(X)$, entonces $N(X^n) \cap X^n$ dará como resultado dos intervalos semi-infinitos.

Este método tiene las siguientes propiedades [50, 92, 111]:

1. Cualquier raíz $b \in X^n$ de f cumple que $b \in N(X^n)$.
2. Si $N(X^n) \cap X^n = \{\emptyset\}$, entonces no existe ninguna raíz de f en X.
3. Si $N(X^n) \subset X^n$, entonces existe una sola raíz de f en $N(X^n)$.
4. Si existe una raíz de f en X, entonces la convergencia, en términos de la anchura del intervalo, es cuadrática, $w(X^{n+1}) \leq k \cdot w(X^n)^2$, siempre que los errores de redondeo computacionales sean menores que la precisión final requerida.

Ejemplo 3.1.4
Supóngase la función $f(x) = x^2 - 2$. *Tomando* $X^0 = [-3, 2]$ *se obtiene que:* $x^0 = -0.5$, $f(x^0) = -1.75$ *y* $F'(X^0) = [-6, 4]$. *El operador de Newton unidimensional* $N(X^0) = -0.5 - (-1.75/([-6, 0] \cup [0, 4])) = [-\infty, -079166] \cup [0.625, \infty]$. *Obteniendo que* $X^1 = [-3, -0.79166] \cup [0.4375, 2]$. *Ahora se almacena uno de los intervalos, por ejemplo* $[0.4375, 2]$, *para su posterior inspección y se continua el método sobre el intervalo* $[-3, -0.79166]$. *La secuencia de intervalos que se obtiene es la siguiente:*

$$\begin{aligned} X^2 &= [-1.63\ldots, -0.88\ldots] \\ X^3 &= [-1.492\ldots, -1.386\ldots] \\ X^4 &= [-1.4153\ldots, -1.41348\ldots] \\ X^5 &= [-1.4142137\ldots, -1.4142135\ldots] \\ &\vdots \end{aligned}$$

Cuando se aplica el método de Newton de intervalos para encontrar los ceros de f' en vez de f, entonces:

1. Si $N(X^n) \cap X^n = \{\emptyset\}$, X no contiene ningún máximo o mínimo.
2. Si $N(X^n) \subset X^n$, sólo existe un máximo o mínimo en $N(X^n) \cap X^n$.
3. Si $N(X^n) \cap X^n \neq \{\emptyset\}$ y $N(X^n) \not\subset X^n$, el intervalo contiene más de un máximo o mínimo.

3.1.3. Regla de Selección

Como se explicó en la Definición 2.1.2, la regla de selección determinará los requerimientos de memoria del algoritmo y el número de subproblemas evaluados. Tradicionalmente se ha usado una regla de selección *Primero el Mejor*, donde se selecciona el intervalo X con menor valor de $\underline{F}(X)$. Esta regla de selección viene motivada por el hecho de que la evaluación de $F(m(X))$ en los intervalos con menor valor de $\underline{F}(X)$ tiene mayor probabilidad de actualizar el valor de $\overline{f^*}$ por lo que el *Test del Punto Medio* podría eliminar más intervalos.

3.1.4. Regla de División

Para intervalos unidimensionales la regla de división tradicionalmente usada es la de dividir el intervalo por su punto medio, es decir, usar bisección [111]. También se puede usar multisección. Normalmente la multisección provoca que el número de intervalos evaluados sea mayor.

3.1.5. Regla de Terminación

Generalmente la *Regla de Terminación* está basada en los valores de $w(X)$ y/o $w(F(X))$. La selección de los valores mínimos de $w(X)$ y/o $w(F(X))$ dependerá de la precisión en la que estamos interesados para localizar la posición del mínimo global y/o para acotar el valor óptimo de la función objetivo, respectivamente. Una descripción más detallada de estos criterios de terminación puede encontrarse en [111].

3.2. Algoritmo TIAM

Una implementación concreta de un algoritmo de Branch and Bound genérico, como el presentado en el Algoritmo 2.1, dependerá de la información disponible a cerca de la función objetivo $f(x)$. Esta sección presenta las reglas de Branch and Bound que usa un algoritmo de Optimización Global Intervalar cuando podemos evaluar $f(x)$ y su derivada $f'(x)$ en X, es decir, la información que se puede obtener de la función objetivo durante la búsqueda es $F(x)$, $F(X)$ y $F'(X)$.

Las reglas de un algoritmo tradicional de Optimización Global con test de monotonía (TIAM: Traditional Interval analysis global minimization Algorithm with Monotonicity test [14]) son las siguientes:

Selección: Entre los intervalos X^i almacenados en la lista de trabajo L seleccionar el intervalo X tal que $\underline{F}(X) = \min\{\underline{F}(X^i) \ : \ X^i \in L\}$.

Acotación: Tal como se explicó en la sección anterior la Aritmética de Intervalos permite obtener de forma rigurosa una *función de inclusión*.

Eliminación: Las reglas de eliminación usadas son el *Test del punto medio*, el *Test de corte* y el *Test de monotonía*, que fueron presentados en la sección anterior.

División: Bisección.

Terminación: El parámetro ϵ determina la precisión deseada de la solución al problema. De esta forma los intervalos X con una anchura $w(X)$ menor que ϵ; i.e. $w(X) \leq \epsilon$, son almacenados en la lista final Q (ver al Algoritmo 2.1).

El algoritmo TIAM nos va a servir como referencia para evaluar las mejoras que se presentan en la Sección 3.3, incorporadas en el algoritmo de Optimización Global Intervalar propuesto en la Sección 3.4, que usa la misma información que TIAM.

3.3. Aportaciones teóricas unidimensionales

En esta sección se verá que los algoritmos tradicionales de Optimización Global basados en Aritmética de Intervalos hacen un uso limitado de la información extraída en cada instante. Las reglas de eliminación descritas hasta ahora, hacen un uso muy restrictivo de la información obtenida de las funciones de inclusión de f y f'. Así, podemos observar que los valores de $F'(X)$ son usados por el test de monotonía, que realmente está completamente desconectado de la información contenida en $F(m(X))$ y $F(X)$. Desde nuestro punto de vista, el Algoritmo TIAM, presentado en la sección anterior, no aprovecha toda la información disponible en cualquier instante para reducir el espacio de búsqueda de las posibles soluciones del problema. En esta sección vamos a exponer un conjunto de consideraciones teóricas que nos permitirán concretar una nueva regla de eliminación. Dicha regla de eliminación será posteriormente incorporada al algoritmo de Branch and Bound y su evaluación sobre un conjunto de funciones de prueba nos permitirá valorar la eficiencia de la nueva regla de eliminación propuesta en esta sección.

La descripción de las bases teóricas que sustentan la nueva regla de eliminación comienzan con el Lema 3.3.1. Una representación gráfica de dicho lema es la que aparece en la Figura 3.3. Ideas similares a las contenidas en el Lema 3.3.1 han sido previamente desarrolladas para funciones no diferenciables, tanto en optimización Lipschitz (ver [52, 56, 90, 106, 107, 122, 130, 131]) como en las aproximaciones basadas en *slopes* presentadas en [115].

Lema 3.3.1
Dados los siguientes elementos: una función continua y diferenciable $f : S \to \mathbb{R}$, donde S es un intervalo cerrado de $\mathbb{R}$, un intervalo $X \subseteq S$, un intervalo de inclusión $F(c)$ para $f(c)$, $c \in X$, y un intervalo de inclusión $F'(X)$ para $f'(x)$, $x \in X$, es posible definir los siguientes limites para $f(x), x \in X$:

$$\underline{F}(c) + \min \left\{ \begin{array}{c} \overline{F'}(X) \cdot (x-c) \\ \underline{F'}(X) \cdot (x-c) \end{array} \right\} \leq f(x) \leq \overline{F}(c) + \max \left\{ \begin{array}{c} \overline{F'}(X) \cdot (x-c) \\ \underline{F'}(X) \cdot (x-c) \end{array} \right\}$$

Demostración: El teorema del valor medio establece que existe un punto $\xi \in [x, c]$ tal que

$$f(x) = f(c) + f'(\xi) \cdot (x - c). \tag{3.14}$$

Extendiendo la Ecuación (3.14) a Aritmética de Intervalos, se puede obtener la siguiente relación de inclusión:

$$f(x) \subseteq F(c) + F'(X) \cdot (x - c), \qquad x, c \in X, \tag{3.15}$$

Tomando un punto genérico $x \in X$ y dependiendo de la posición mutua de los puntos c y x en X, tres posibles resultados pueden ser deducidos de la Ecuación (3.15):

- $x = c$:
$$f(x) \subseteq F(c).$$
- $x > c$:
$$f(x) \leq \overline{F}(c) + \overline{F'}(X) \cdot (x - c),$$
$$f(x) \geq \underline{F}(c) + \underline{F'}(X) \cdot (x - c).$$
- $x < c$:
$$f(x) \leq \overline{F}(c) - \underline{F'}(X) \cdot (c - x) = \overline{F}(c) + \underline{F'}(X) \cdot (x - c),$$
$$f(x) \geq \underline{F}(c) - \overline{F'}(X) \cdot (c - x) = \underline{F}(c) + \overline{F'}(X) \cdot (x - c).$$

con lo que el lema queda demostrado. □

Este lema nos da la posibilidad de construir nuevas funciones de acotación, basadas en el análisis de intervalos, para $f(x)$. Podemos observar en la Figura 3.3 que esta propuesta de acotación es similar a las construidas en Optimización Global Lipschitz, ver por ejemplo [51, 106, 107, 121, 131]. Las funciones soporte Lipschitz son lineales por tramos. Las derivadas de cada tramo lineal son L o $-L$, donde L es la constante Lipschitz. En la aproximación mostrada aquí, para cada intervalo X las pendientes de las funciones soporte son:

- $\overline{F'}(X)$ para todo $x \leq c$.
- $\underline{F'}(X)$ para todo $x \geq c$.

que gráficamente se han representado en la Figura 3.3. Los siguientes resultados son las bases de las nuevas funciones soporte y explican como calcular nuevos límites inferiores para acotar la función $f(x)$.

Teorema 3.3.1
Sean X y S dos intervalos cerrados que cumplen que $X \subseteq S \subset \mathbb{R}$, y $f : S \to \mathbb{R}$ una función continua y diferenciable. Supongamos que para un punto $c \in X$ se determina un limite inferior de la función $\underline{lb}(c)$ de $f(c)$ y se obtiene una inclusión $F'(X)$ de $f'(X)$. Para un determinado limite superior actualizado $\overline{f^{\sim}}$ de la solución f^*, existe un conjunto $V \subseteq X$, donde están incluidos todos los mínimos globales de X, si existen.

Demostración: Dado que para un punto minimizador $x^* \in S^*$, se verifica que $f(x^*) \leq \overline{f^{\sim}}$, y teniendo en cuenta el Lema 3.3.1, un minimizador $x^* \in X \cap S^*$ tiene que cumplir que:

$$\underline{lb}(c) + \min \left\{ \begin{array}{l} \overline{F'}(X) \cdot (x^* - c) \\ \underline{F'}(X) \cdot (x^* - c) \end{array} \right\} \leq f(x^*) \leq \overline{f^{\sim}}$$

y por lo tanto, únicamente puede estar localizado en el conjunto:

$$V = \left\{ x \in X : \underline{lb}(c) + \min \left\{ \begin{array}{l} \overline{F'}(X) \cdot (x - c) \\ \underline{F'}(X) \cdot (x - c) \end{array} \right\} \leq \overline{f^{\sim}} \right\}$$

□

Del Teorema 3.3.1 podemos deducir que si $\overline{f^{\sim}} < \underline{lb}(c)$, entonces $c \notin S^*$ y se puede construir un intervalo $V = V^1 \cup V^2$, donde V^1 y V^2 se definen como:

$$V^1 = \left\{ \begin{array}{lll} \{x \in X : x \leq c - \frac{\underline{lb}(c) - \overline{f^{\sim}}}{\overline{F'}(X)}\} & , & \textit{cuando } \overline{F'}(X) > 0 \\ \emptyset & , & \textit{en otro caso} \end{array} \right. \tag{3.16}$$

$$V^2 = \left\{ \begin{array}{lll} \{x \in X : x \leq c - \frac{\underline{lb}(c) - \overline{f^{\sim}}}{\underline{F'}(X)}\} & , & \textit{cuando } \underline{F'}(X) < 0 \\ \emptyset & , & \textit{en otro caso} \end{array} \right. \tag{3.17}$$

Como un ejemplo, en la Figura 3.3 se han representado los subintervalos V^1 y V^2 para el caso concreto de $\overline{F'}(X) > 0$ y $\underline{F'}(X) < 0$.

Nótese que V puede también ser calculado, de una manera similar, aplicando el operador de Newton para encontrar las raíces de la función $f(x) - \overline{f^{\sim}}$ en X, bajo las condiciones del Teorema 3.3.1 [50, 92, 99].

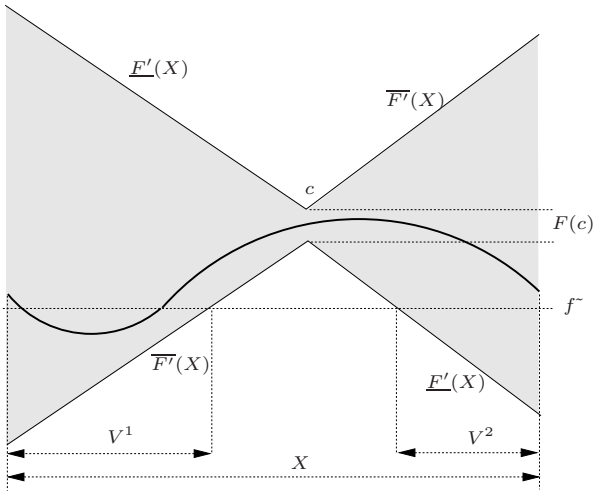


Figura 3.3: Ejemplo gráfico del Lema 3.3.1 y el Teorema 3.3.1

Teorema 3.3.2
Sea $f : S \to \mathbb{R}$, donde S es un intervalo cerrado de $\mathbb{R}$, una función continua y diferenciable y sean X, Y, dos intervalos tales que $X \subseteq Y \subseteq S$. Si:

1. *los limites inferiores $\underline{lb}(\underline{x})$ y $\underline{lb}(\overline{x})$ de $f(\underline{x})$ y $f(\overline{x})$, respectivamente, pueden ser evaluados;*
2. *un limite superior actualizado de f^*, $\overline{f^{\sim}}$, verifica que:*
$$\overline{f^{\sim}} \leq \min\{\underline{lb}(\underline{x}), \underline{lb}(\overline{x})\};$$
3. *y se han obtenido límites de $F'(Y)$, $\underline{G} = \underline{F'}(Y) < 0$ y $\overline{G} = \overline{F'}(Y) > 0$.*

Entonces, solamente el intervalo definido como:

$$V = [\underline{x} - \frac{\underline{lb}(\underline{x}) - \overline{f^{\sim}}}{\underline{G}}, \overline{x} - \frac{\underline{lb}(\overline{x}) - \overline{f^{\sim}}}{\overline{G}}], \quad V \subseteq X, \tag{3.18}$$

puede contener minimizadores globales y un nuevo limite inferior:

$$\underline{z}(X, \underline{lb}(\underline{x}), \underline{lb}(\overline{x}), G)$$

de $f(x)$ sobre el intervalo X puede ser calculado como sigue:

$$z(X, \underline{lb}(\underline{x}), \underline{lb}(\overline{x}), G) = \frac{\underline{lb}(\underline{x}) \cdot \overline{G} - \underline{lb}(\overline{x}) \cdot \underline{G}}{w(G)} + w(X) \cdot \frac{\underline{G} \cdot \overline{G}}{w(G)}, \tag{3.19}$$

donde $w(G) = \overline{G} - \underline{G}$ *(ver Figura 3.4).*

Demostración: Aplicando el Teorema 3.3.1, para $c = \overline{x}$, de la Ecuación (3.17), se obtiene el intervalo V^2. Aplicando la misma operación para el punto $c = \underline{x}$, de la Ecuación (3.16), se genera el intervalo V^1. Entonces, de la Ecuación (3.18), se obtiene el intervalo $V = V^1 \cap V^2$.

Probemos ahora la Ecuación (3.19). Dado que $X \subseteq Y$, $F'(X) \subseteq F'(Y)$ y, aplicando el teorema del valor medio, tenemos que :

$$f(x) \geq \underline{lb}(\underline{x}) + \underline{F'}(X) \cdot (x - \underline{x}) \geq \underline{lb}(\underline{x}) + \underline{G} \cdot (x - \underline{x}),\ x \in X,$$

y

$$f(x) \geq \underline{lb}(\overline{x}) + \overline{F'}(X) \cdot (x - \overline{x}) \geq \underline{lb}(\overline{x}) + \overline{G} \cdot (x - \overline{x}),\ x \in X.$$

De estas dos inecuaciones se obtiene que:

$$f(x) \geq d(x) = \max \left\{ \begin{array}{l} \underline{lb}(\underline{x}) + \underline{G} \cdot (x - \underline{x}) \\ \underline{lb}(\overline{x}) + \overline{G} \cdot (x - \overline{x}) \end{array} \right\},\ x \in X. \tag{3.20}$$

por lo tanto

$$z(X, \underline{lb}(\underline{x}), \underline{lb}(\overline{x}), G) = \min d(x),\ x \in X \tag{3.21}$$

y

$$\underline{f}(X) \geq z(X, \underline{lb}(\underline{x}), \underline{lb}(\overline{x}), G)$$

lo que demuestra el teorema. □

Corolario 3.3.1
Si para un intervalo X *se cumple la inecuación* $\underline{z}(X, \underline{lb}(\underline{x}), \underline{lb}(\overline{x}), G) > \overline{f^{\sim}}$*, entonces podemos concluir que* X *no contiene el mínimo global.*

Demostración: La demostración es evidente y puede ser omitida. □

Una forma de resolver de forma más eficiente el problema de Optimización Global, consiste en usar toda la información existente sobre los valores de $F(x)$, $F(X)$

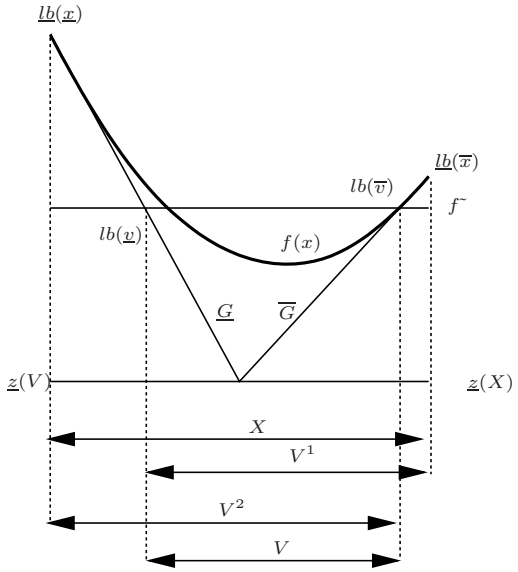


Figura 3.4: El intervalo V es la región que puede contener mínimos globales. El conjunto $X \backslash V$ no contiene ningún mínimo global.

y $F'(X)$ obtenidos durante el proceso de búsqueda. Así pues, usando $\underline{F}(X)$ junto con la función $d(x)$ de la Ecuación (3.20), podemos construir una nueva función soporte $D(x)$ para $f(x)$ sobre el intervalo X:

$$D(x) = \max\{\underline{F}(X), d(x)\}, x \in X. \tag{3.22}$$

El correspondiente nuevo limite inferior $\underline{Fz}(X)$ para $f(x)$ sobre el intervalo X se calcula de la siguiente manera:

$$\underline{Fz}(X) = \max\{\underline{F}(X),\ \underline{z}(X, \underline{lb}(\underline{x}), \underline{lb}(\overline{x}), G)\} = \min D(x),\ x \in X. \tag{3.23}$$

La parte fundamental de este estudio teórico, es que para $V = [\underline{v}, \overline{v}]$ obtenido para el intervalo X de acuerdo con la Ecuación (3.18), el valor actualizado de $\underline{f^{\tilde{}}}$ es un limite inferior de $f(x)$ en $\underline{v}$ y $\overline{v}$; es decir, $\underline{f^{\tilde{}}} \leq f(\underline{v})$ y $\underline{f^{\tilde{}}} \leq f(\overline{v})$, y por lo tanto $\underline{lb}(\underline{v}) = \underline{lb}(\overline{v}) = \underline{f^{\tilde{}}}$ son límites que pueden ser obtenidos fácilmente.

3.4. IAG: Nuevo Algoritmo de Optimización Global Intervalar

Los resultados teóricos vistos en la sección anterior nos permiten mejorar las reglas de Branch and Bound usadas en el algoritmo TIAM, que fue presentado en la Sección 3.2. El nuevo algoritmo de Optimización Global basado en Aritmética de Intervalos (Algoritmo IAG: Interval Analysis global minimization Algorithm with Gradient information [14]) hace un uso más eficiente de la información de la derivada. A continuación se detallan las reglas de Branch and Bound usadas por el Algoritmo IAG:

- **Regla de selección:** Se selecciona un intervalo X tal que:

$$\underline{Fz}(X) = \min\{\underline{Fz}(X^i) : X^i \in L\}$$

  donde L es la lista de trabajo ordenada de manera no decreciente de los valores de $\underline{Fz}(X^i)$ como primer criterio de ordenación y en orden no creciente con respecto a la antigüedad de los intervalos como segundo criterio de ordenación. Por lo tanto el intervalo seleccionado será el que está situado en la cabecera de la lista de trabajo L.

- **Regla de acotación:** Se usará el limite inferior $\underline{Fz}(X)$ obtenido a partir de la Ecuación (3.23).

- **Regla de eliminación:** Para la eliminación usaremos cuatro reglas:

  - **Test RangeUp:** Un intervalo X es rechazado si se cumple que $\overline{f^{\sim}} < \underline{Fz}(X)$.
  - **Test de corte:** El test de corte usado en el Algoritmo IAG es mejor que el usado en el Algoritmo TIAM porque en vez de eliminar de la lista de trabajo o de la lista final todos los intervalos X que cumplan la condición $\underline{F}(X) > \overline{f^{\sim}}$, el algoritmo IAG elimina todos los intervalos X que cumplan la condición $\underline{Fz}(X) > \overline{f^{\sim}}$.
  - **Test de monotonía:** Tal como se presentó en la Sección 3.1.2 si para un intervalo X se cumple la condición $0 \notin F'(X)$, entonces sabemos que el intervalo X no contiene ningún mínimo por lo tanto podemos eliminarlo.
  - **Test Gradient:** La subregión $\{X \backslash V\}$, donde V está definido por la Ecuación (3.18), es rechazada. El intervalo completo puede ser eliminado cuando $V = \emptyset$.

- **Regla de división:** Aunque los Algoritmos TIAM e IAG usan bisección, el punto de división puede diferir de un algoritmo a otro debido a que el Algoritmo IAG usa el *Test Gradient*. Es decir, si un intervalo X ha sido obtenido como resultado de aplicar el *Test Gradient* al intervalo Y y el intervalo X es

elegido para ser dividido, se usa el punto $m(X)$ como punto de subdivisión y normalmente $m(X) \neq m(Y)$.

- **Regla de terminación:** Usaremos el mismo que el Algoritmo TIAM.

El Algoritmo 3.1 muestra el pseudo-código del Algoritmo IAG. El Algoritmo 3.1 comienza evaluando $F(\underline{s})$ y $F(\overline{s})$ (línea 3) e inicializando $x^{\check{}}$ y $f^{\check{}} = F(x^{\check{}})$ (líneas 4 y 5). Si se cumple el test de monotonía (línea 6), el algoritmo termina y la solución viene dada por $\{f^{\check{}}, x^{\check{}}\}$. En caso contrario, inicializamos $lb(\underline{s})$ y $lb(\overline{s})$ (línea 8) y aplicamos el *Test Gradient* (linea 9) sobre el intervalo inicial S. El Test Gradient, descrito más detalladamente en el Algoritmo 3.2, aplica los Teoremas 3.3.1 y 3.3.2 sobre un intervalo S, usando $c = \underline{s}$ y $c = \overline{s}$ y devuelve un intervalo $X \subseteq S$, tal que el conjunto de minimizadores globales de S están también en X. En la línea 10 inicializamos los límites inferiores de $f(\underline{x})$ y $f(\overline{x})$ ($lb(\underline{x})$ y $lb(\overline{x})$, respectivamente), así como $f^{\hat{}}(X)$ al valor de $f^{\check{}}$; con estos valores calculamos el valor de $\underline{Fz}(X)$ (línea 11). Dependiendo del valor de $w(X)$, el intervalo X se almacena en la lista final Q o en la lista de trabajo L (líneas 13 y 14). Téngase en cuenta que cuando un intervalo llega a ser suficientemente pequeño, el algoritmo lo almacena en la lista final Q y por lo tanto no vuelve a ser evaluado.

Esta etapa inicial del algoritmo se ha representado gráficamente en la parte superior de la Figura 3.5. Después de esta etapa de inicialización y mientras la lista de trabajo L contenga intervalos (línea 15), el Algoritmo 3.1 seleccionará el intervalo que se encuentra en la cabeza de la lista L (línea 16) para ser a continuación procesado. Si $F'(X)$ no cumple el test de monotonía (línea 17), $F(m(X))$ es calculado (línea 18). Si $F(m(X))$ es un mejor límite superior de f^* que $\overline{f^{\check{}}}$, $f^{\check{}}$ se actualiza a $F(m(X))$ (línea 19) y el *Test de Corte* se aplica a los intervalos de la lista de trabajo L y de la lista final Q (línea 20). Generamos ahora dos subintervalos X_1 y X_2 (líneas 21 y 22). Estos dos subintervalos X^1 y X^2 heredaran de X el límite inferior de $f(x)$ en uno de sus extremos y en el otro se actualizara al valor $F(m(X))$ (líneas 23 y 24). Ahora, sobre cada subintervalo generado $X^i,\ i = \{1, 2\}$, se aplica el *Test Gradient* usando la información de la derivada del intervalo inicial (antes de dividir) X (línea 26), en lugar de usar $F'(X^i)$, que todavía no ha sido evaluado. La evaluación de esos valores será realizada solo para aquellos intervalos que son elegidos posteriormente para la subdivisión. Usando la derivada de F en el intervalo X ($F'(X)$) no añadimos computación innecesaria para el calculo de $F'(X^i),\ i = \{1, 2\}$, ya que puede ocurrir que los intervalos $X^i,\ i = \{1, 2\}$, nunca sean considerados para ser subdivididos y podrían ser eliminados mediante el *Test de Corte*. Si un intervalo $X^i,\ i = \{1, 2\}$ no es rechazado en la línea 28, $\underline{Fz}(X^i)$ se evalúa usando también los valores de $F'(X)$ (línea 29). Únicamente cuando el *Test RangeUp* no se satisface (línea 30), el subintervalo X^i será almacenado en la lista de trabajo L o en la lista final Q, dependiendo del criterio de parada (líneas 31, 32 y 33).

Algorithm 3.1: Algoritmo IAG.

Entrada: S, F, ϵ
Salida : Q,$f^{\sim}$

```
1  L := {S} ;                                   /* Iniciamos la lista de trabajo */
2  Q := {} ;                                    /* Iniciamos la lista de final */
3  if ( F̄(s̲) < F̄(s̄) ) then
4  |  x˜ := s̲; f˜ := F(s̲);
5  else x˜ := s̄; f˜ := F(s̄);
6  if (0 ∉ F'(S)) then                          /* La función es monótona */
7  |  Salida(f* := f˜ ; s* := x˜);
8  lb(s̲) := F(s̲); lb(s̄) := F(s̄) ;                          /* Iniciamos lb */
9  X := TestGradient (S, lb(s̲), lb(s̄), f˜, F'(S)) ;        /* Recorte inicial */
10 lb(x̲) := lb(x̄) := f^(X) := f˜ ;                        /* Actualizamos lb */
11 F̲z(X) := max{F̲(X), z(X, lb(x̲), lb(x̄), F'(S))};
12 if ( w(X) ≤ ε ) then                          /* Crit. de parada */
13 |  Almacenar {X, f^(X), F̲z(X)} en Q;
14 else Almacenar {X, f^(X), F̲z(X)} in L;
15 while ( L ≠ {} ) do                          /* Mientras haya intervalos */
16 |  {X, f^(X), F̲z(X)} := Head(L) ;                    /* Primero de la lista */
17 |  if ( 0 ∈ F'(X) ) then                       /* Test de monotonía */
18 |  |  if (F̄(m(X)) < f˜‾) then
19 |  |  |  f˜ := F(m(X)) ;                      /* Act. el mejor límite sup. */
20 |  |  |  TestCorte ( f˜‾, L, Q );
21 |  |  X^1 = [x̲, m(X)];                              /* Subintervalo 1 */
22 |  |  X^2 = [m(X), x̄];                              /* Subintervalo 2 */
23 |  |  lb(x̲^1) := lb(x̄^2) := f^(X);                  /* Actualizamos lb */
24 |  |  lb(x̄^1) := lb(x̲^2) := F(m(X));                /* Actualizamos lb */
25 |  |  for (i := 1,2) do                       /* Para cada subintervalo */
26 |  |  |  X^i := TestGradient ( X^i, lb(x̲^i), lb(x̄^i), f˜, F'(X) );
27 |  |  |  lb(x̲^i) := lb(x̄^i) := f^(X^i) := f˜;
28 |  |  |  if (w(X^i) > 0) then                 /* Si no se ha eliminado */
29 |  |  |  |  F̲z(X^i) := max{F̲(X^i), z(X^i, lb(x̲^i), lb(x̄^i), F'(X))};
30 |  |  |  |  if F̲z(X^i) ≤ f˜‾ then                    /* RangeUp Test */
31 |  |  |  |  |  if ( w(X^i) ≤ ε ) then          /* Crit. de parada */
32 |  |  |  |  |  |  Almacenar {X, f^(X^i), F̲z(X^i)} in Q;
33 |  |  |  |  |  else Almacenar {X, f^(X^i), F̲z(X^i)} in L;
```

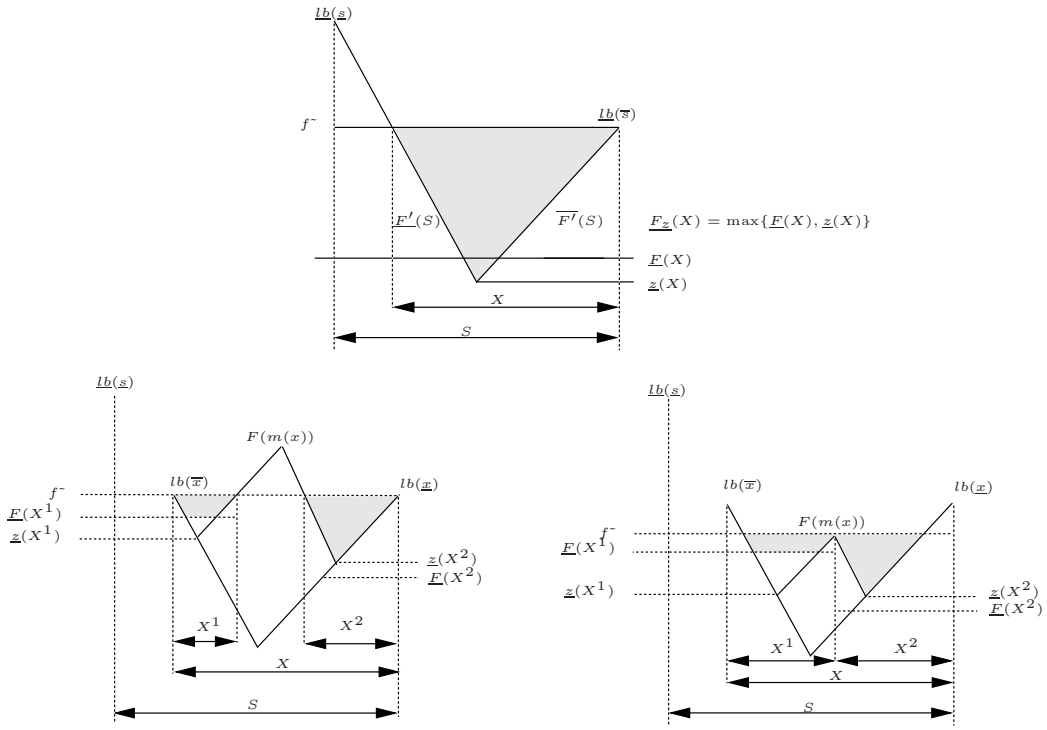


Figura 3.5: El gráfico de la parte superior es un ejemplo de la fase inicial del algoritmo IAG. Los gráficos de la parte inferior son ejemplos que muestran como se construyen los intervalos X^1 y X^2 a partir del intervalo X.

En la parte inferior de la Figura 3.5 se ha representado la forma en la que se construyen los subintervalos X^1 y X^2 a partir de X. Para el caso en el que se verifica que $\underline{F}(m(X)) > \overline{f^{\sim}}$ (mostrado en la parte izquierda) los subintervalos X^1 y X^2 se han obtenido como resultado de aplicar el *Test Gradient*. Si actualizamos $f^{\sim} = F(m(X))$, ocurre que $\overline{f^{\sim}} < \underline{lb}(\underline{x})$ y $\overline{f^{\sim}} < \underline{lb}(\overline{x})$ (ver la parte derecha de la figura), por lo que X_1 y X_2 pueden ser reducidos por el *Test Gradient*. Debe tenerse en cuenta que en el Algoritmo 3.1, cuando se verifica que $\overline{f^{\sim}} < \underline{lb}(\underline{x})$, también se verifica que $\overline{f^{\sim}} < \underline{lb}(\overline{x})$, y viceversa.

Algorithm 3.2: Algoritmo que implementa Test Gradient.

Entrada: $X, lb(\underline{x}), lb(\overline{x}), f^{\sim}, G$
Salida : X reducido
if ($\underline{lb}(\underline{x}) > \overline{f^{\sim}}$) **then** /* Recortamos por la derecha */
  $V^1 = [\underline{x} - \frac{\underline{lb}(\underline{x}) - \overline{f^{\sim}}}{\underline{G}},\ \infty)$;
  $X := V^1 \cap X$
if ($w(X) > 0$ & $\underline{lb}(\overline{x}) > \overline{f^{\sim}}$) **then** /* Recortamos por la izquierda */
  $V^2 = (-\infty,\ \overline{x} - \frac{\underline{lb}(\overline{x}) - \overline{f^{\sim}}}{\overline{G}}]$;
  $X := V^2 \cap X$;

3.5. Evaluación del algoritmo IAG

En esta sección presentamos los resultados numéricos de la evaluación del Algoritmo TIAM, y del nuevo algoritmo presentado en este trabajo de tesis (Algoritmo IAG). Comparando los requerimientos computacionales se obtendrá información de la eficiencia aportada por la incorporación de los nuevos criterios de acotación y eliminación definidos en este capítulo. Ambos algoritmos han sido evaluados usando un conjunto de cuarenta funciones test. Este conjunto de funciones test se describe en la Tabla 3.1 [1] y también puede ser consultado en [25, 26, 115] de donde ha sido tomado. La región de búsqueda y el número de mínimos globales y locales se muestran para todas las funciones. Para ambos algoritmos, el criterio de parada fue:

$$w(X) \leq \epsilon = 10^{-6}$$

La Tabla 3.2 muestra una comparación numérica entre los algoritmos TIAM e IAG. La columna NFE muestra el número de evaluaciones de la función; es decir, el número de evaluaciones de $F(X)$ más el número de evaluaciones de $F(x)$. La columna NDE muestra el número evaluaciones de las derivadas de la función realizadas $F'(X)$. La columna $w(f^*)$ muestra la anchura del intervalo que contiene el óptimo global (medido como $[\min\{\underline{F}(X^i) : X^i \in Q\}, \overline{f^{\sim}}]$ y $[\min\{\underline{F_z}(X^i) : X^i \in Q\}, \overline{f^{\sim}}]$) para los algoritmos TIAM y IAG, respectivamente.

Si denotamos por Esf_1 y Esf_2 a los valores del esfuerzo computacional, medido como $NFE + NDE$, para los algoritmos TIAM e IAG, respectivamente, la columna

[1] Para las funciones 31 y 35 los valores de a_i, k_i y c_i son:
$a = (3.040, 1.098, 0.674, 3.537, 6.173, 8.679, 4.503, 3.328, 6.937, 0.700)$,
$k = (2.983, 2.378, 2.439, 1.168, 2.406, 1.236, 2.868, 1.378, 2.348, 2.268)$,
$c = (0.192, 0.140, 0.127, 0.132, 0.125, 0.189, 0.187, 0.171, 0.188, 0.176)$ y
$a = (4.696, 4.885, 0.800, 4.986, 3.901, 2.395, 0.945, 8.371, 6.181, 5.713)$,
$k = (2.871, 2.328, 1.111, 1.263, 2.399, 2.629, 2.853, 2.344, 2.592, 2.929)$,
$c = (0.149, 0.166, 0.175, 0.183, 0.128, 0.117, 0.115, 0.148, 0.188, 0.198)$, respectivamente.

Tabla 3.1: Descripción de las funciones test. N: Identificación de la función, S: Intervalo inicial de búsqueda, NL: Número de mínimos locales, NG: Número de mínimos globales y f^*: Valor del mínimo global redondeado a 6 dígitos decimales.

N	Función $f(x)$	S	NL	NG
1	$e^{-3x} - \sin^3 x$	$[0, 20]$	4	1
2	$\sum_{k=1}^{5} -\cos[(k+1)x] + 4$	$[0.2, 7.0]$	7	1
3	$(x - x^2)^2 + (x+1)^2$	$[-10, 10]$	1	1
4	$(3x - 1.4)\sin(18x) + 1.7$	$[0.2, 7.0]$	21	1
5	$2x^2 - 3/100e^{-(200(x-0.0675))^2}$	$[-10, 10]$	1	1
6	$\cos(x) - \sin(5x) + 1$	$[0.2, 7.0]$	6	1
7	$-x - \sin(3x) + 1.6$	$[0.2, 7.0]$	4	1
8	$x + \sin(5x)$	$[0.2, 7.0]$	7	1
9	$-e^{-x}\sin(2\pi x) + 1$	$[0.2, 7.0]$	7	1
10	$e^{-x}\sin(2\pi x)$	$[0.2, 7.0]$	7	1
11	$-x + \sin(3x) + 1$	$[0.2, 7.0]$	5	1
12	$x\sin(x) + \sin(10x/3) + \ln(x) - 0.84x + 1.3$	$[0.2, 7.0]$	4	1
13	$\sin(x) + \sin(10x/3) + \ln(x) - 0.84x$	$[2.7, 7.5]$	3	1
14	$\ln(3x)\ln(2x) - 0.1$	$[0.2, 7.0]$	1	1
15	$\sum_{k=0}^{5} k\cos[(k+1)x + k] + 12$	$[0.2, 7.0]$	8	1
16	$-\sum_{k=1}^{5} k\sin[(k+1)x + k] + 3$	$[0.2, 7.0]$	7	1
17	$\sin^2(1 + (x-1)/4) + ((x-1)/4)^2$	$[-10, 10]$	1	1
18	$\sqrt{x}\sin^2(x)$	$[0.2, 7.0]$	3	2
19	$x^2 - \cos(18x)$	$[-5, 5]$	29	1
20	e^{x^2}	$[-10, 10]$	1	1
21	$(x^2/20) - \cos(x) + 2$	$[-20, 20]$	7	1
22	$\cos(x) + 2\cos(2x)e^{-x}$	$[0.2, 7.0]$	2	1
23	$(x + \sin(x))e^{-x^2}$	$[-10, 10]$	1	1
24	$2\sin(x)e^{-x}$	$[0.2, 7.0]$	2	1
25	$2\cos(x) + \cos(2x) + 5$	$[0.2, 7.0]$	3	2
26	$e^{\sin(3x)}$	$[0.2, 7.0]$	5	3
27	$\sin(x)\cos(x) - 1.5\sin^2(x) + 1.2$	$[0.2, 7.0]$	3	2
28	$\sin(x)$	$[0, 20]$	4	3
29	$2(x-3)^2 - e^{x/2} + 5$	$[0.2, 7.0]$	1	1
30	$-e^{\sin(3x)} + 2$	$[0.2, 7.0]$	4	4
31	$-\sum_{k=1}^{5} 1/((k_i(x - a_i))^2 + c_i)$	$[0, 10]$	8	1
32	$\sin(1/x)$	$[0.02, 1]$	6	6
33	$-\sum_{k=1}^{5} k\sin((k+1)x + k)$	$[-10, 10]$	20	3
34	$(x^2 - 5x + 6)/(x^2 + 1) - 0.5$	$[0.2, 7.0]$	1	1
35	$-\sum_{k=1}^{5} 1/((k_i(x - a_i))^2 + c_i)$	$[0, 10]$	7	1
36	$(x+1)^3/x^2 - 7.1$	$[0.2, 7.0]$	1	1
37	$x^4 - 12x^3 + 47x^2 - 60x - 20e^{-x}$	$[-1, 7]$	1	1
38	$x^6 - 15x^4 + 27x^2 + 250$	$[-4, 4]$	2	2
39	$x^4 - 10x^3 + 35x^2 - 50x + 24$	$[-10, 20]$	2	2
40	$24x^4 - 142x^3 + 303x^2 - 276x + 3$	$[0, 3]$	1	1

Esf_1/Esf_2 muestra la relación entre los valores de esfuerzo de los algoritmos TIAM e IAG, proporcionando información relativa a la ganancia en velocidad (*speedup*) del algoritmo IAG comparado con TIAM. Ambos algoritmos proporcionan los mismos resultados relativos a los óptimos encontrados, que en todos los casos coinciden con los suministrados en la Tabla 3.1. Es decir, los intervalos solución de ambos algoritmos contienen el óptimo global y como puede observarse de la Tabla 3.2, el algoritmo IAG ofrece resultados más precisos puesto que la anchura del intervalo $w(f^*)$ es menor para IAG que para TIAM.

Tabla 3.2: Comparación numérica entre TIAM e IAG.

	TIAM			IAG			
N	NFE	NDE	$w(f^*)$	NFE	NDE	$w(f^*)$	Esf_1/Esf_2
1	104	27	6.2e-14	75	20	2.9e-14	1.38
2	104	27	2.9e-13	72	21	4.7e-12	1.41
3	107	27	1.1e-13	88	22	3.0e-13	1.22
4	105	34	2.4e-06	79	25	1.3e-09	1.34
5	112	35	1.6e-07	88	26	9.0e-12	1.29
6	110	39	2.4e-07	75	25	1.4e-11	1.49
7	112	41	8.1e-07	69	25	8.9e-14	1.63
8	112	41	8.1e-07	66	22	1.1e-11	1.74
9	115	41	6.4e-07	76	25	4.0e-13	1.54
10	116	42	3.9e-07	83	28	5.3e-12	1.42
11	118	44	8.1e-07	79	28	1.4e-12	1.51
12	118	44	1.9e-06	74	26	3.5e-12	1.62
13	132	49	4.8e-07	81	30	5.0e-12	1.63
14	139	50	4.0e-07	91	31	7.4e-12	1.55
15	144	49	9.7e-06	102	34	2.3e-11	1.42
16	152	51	1.1e-05	113	35	3.0e-11	1.37
17	167	63	1.3e-07	114	39	3.7e-14	1.50
18	184	47	5.5e-15	140	38	1.4e-13	1.30
19	191	48	4.4e-16	77	23	5.5e-16	2.39
20	199	50	1.1e-15	116	38	1.1e-15	1.62
21	207	52	4.4e-16	114	36	6.6e-16	1.73
22	209	75	8.9e-08	132	46	4.0e-13	1.60
23	218	81	6.7e-07	146	50	9.6e-13	1.53
24	220	83	2.3e-08	152	50	4.7e-14	1.50
25	231	88	1.4e-06	167	61	5.3e-13	1.40
26	268	68	2.6e-14	214	57	4.3e-14	1.24
27	247	92	7.4e-07	191	67	1.3e-12	1.31
28	292	74	7.2e-15	206	55	2.8e-15	1.40
29	301	113	2.8e-06	174	60	1.6e-12	1.77
30	352	89	4.9e-15	270	71	9.1e-14	1.29
31	139	47	5.1e-06	113	35	2.8e-10	1.26
32	476	120	2.6e-11	384	101	2.2e-12	1.23
33	459	154	8.2e-06	333	108	2.8e-10	1.39
34	460	176	5.9e-07	259	91	7.4e-13	1.82
35	599	204	1.2e-05	293	96	5.3e-10	1.50
36	711	271	2.7e-06	331	114	2.4e-12	2.21
37	824	276	7.5e-05	288	101	5.9e-11	2.83
38	807	310	1.5e-03	498	176	1.2e-08	1.66
39	6041	2265	4.0e-04	1364	456	2.3e-09	4.56
40	6952	2534	1.4e-03	1020	340	2.9e-09	6.98
Av. Val	566.35	200.53		210.18	68.3		1.78
$\sum$	22654	8021		8407	2732		2.75

Podemos ver en la Tabla 3.2, que el ratio Esf_1/Esf_2 es siempre mayor que uno, lo que significa que IAG mejora los resultados de TIAM para todas las funciones. Para este conjunto de funciones la ganancia en velocidad Esf_1/Esf_2 tiene un rango entre $[1.22, 6.98]$ y una ganancia media de $1,78$. Podemos ver también en la Tabla 3.2 que para las funciones donde TIAM necesita una enorme cantidad de evaluaciones, se obtienen los mejores los valores de Esf_1/Esf_2 (para las funciones $N = 39$ y $N = 40$, la ganancia en velocidad es de 4.56 y 6.98, respectivamente).

Considerando todas las funciones a la vez, la relación entre Esf_1 y Esf_2 es de 2.75 (ver la última fila de la Tabla 3.2), por lo que teniendo en cuenta los datos de

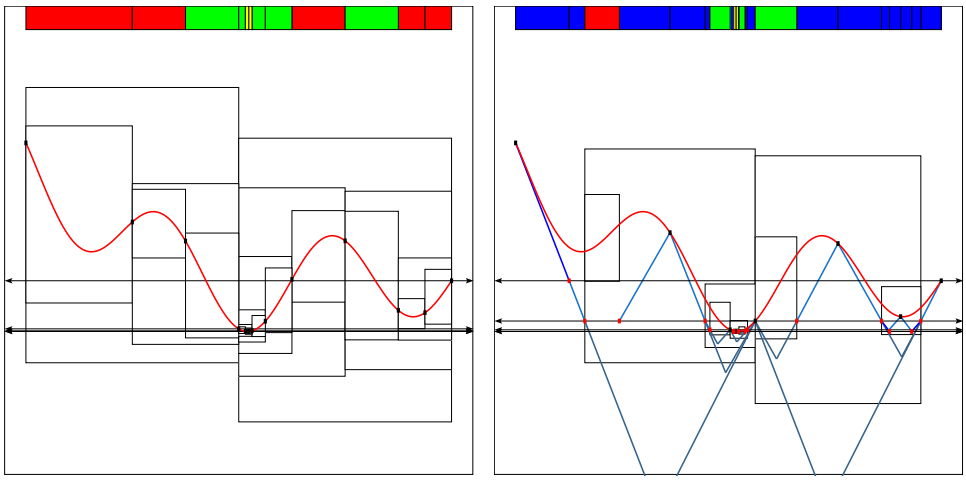


Figura 3.6: Representación gráfica de las ejecuciones de TIAM (gráfico de la izquierda) e IAG (a la derecha) para la función $N = 13$.

forma global, la ganancia en velocidad sería aun mayor.

Las Figuras 3.6 y 3.7, muestran un ejemplo gráfico de como trabajan los algoritmos TIAM e IAG. La función $N = 13$ mostrada en la Figura 3.6 tiene un único mínimo global mientras que la función $N = 25$, mostrada en la Figura 3.7, tiene dos mínimos globales. En ambas figuras, la gráfica presentada en la parte izquierda se refiere al algoritmo TIAM mientras que la mostrada en la parte derecha se refiere al algoritmo IAG. Para las dos figuras el criterio de parada fue de $w(X) \leq 0.05$. Las líneas horizontales representan las actualizaciones de los valores $\overline{f^{\sim}}$ durante la ejecución. Las cajas representan los limites superiores e inferiores de $F(X)$ de los intervalos evaluados.

Para el algoritmo IAG, también se muestran las nuevas funciones soporte $d(x)$. En la parte superior de los gráficos, aparecen cajas de colores que representan el conjunto de intervalos rechazados por los distintos test. El color de la caja, especifica el test responsable del rechazo del intervalo (Azul=*Gradient*, Verde=*Monotonía*, Rojo=*Punto Medio* y *Test de Corte* para el algoritmo TIAM, y *RangeUp* y *Test de Corte* para el algoritmo IAG, Amarillo = cajas en la lista final). Como vemos en ambas figuras, el nuevo test de rechazo es muy eficiente.

Las Figuras 3.6 y 3.7 muestran también que para estos ejemplos, más del 50 % del intervalo inicial fue rechazado por el *Test Gradient*. También muestra claramente que el algoritmo TIAM ha evaluado mas intervalos que IAG. La Figura 3.7 muestra además algunos intervalos donde hemos obtenido un mejor límite inferior de $f(X)$, calculado a partir de $z(X, \underline{lb}(\underline{x}), \underline{lb}(\overline{x}), F'(X))$ (ver Ecuación (3.21)) en

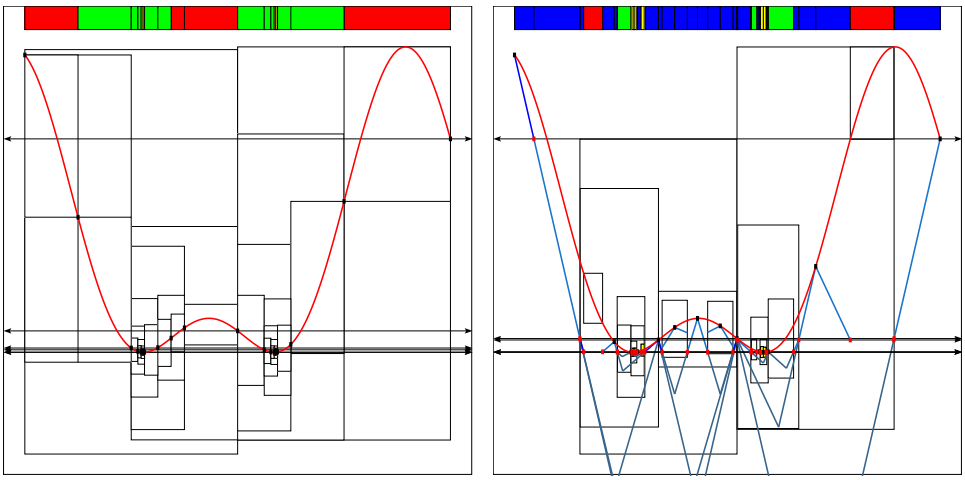


Figura 3.7: Representación gráfica de las ejecuciones de TIAM (gráfico de la izquierda) e IAG (a la derecha) para la función $N = 25$.

lugar de $\underline{F}(X)$, es decir, $\underline{Fz}(X) = z(X, \underline{lb}(\underline{x}), \underline{lb}(\overline{x}), F'(X))$ en vez de $\underline{F}(X)$. Debe también destacarse que el algoritmo TIAM es incapaz de aprovechar la información suministrada por la evaluación de $F(m(X))$ cuando $F(m(X)) > \overline{f^*}$. En contraposición, IAG permite reducir el intervalo incluso en este caso, lo que queda claramente mostrado en las Figuras 3.6 y 3.7.

En este capítulo únicamente hemos abordado problemas de Optimización Global del espacio real unidimensional. El siguiente capítulo esta dedicado a extender al espacio multidimensional todas las aportaciones que se han visto aquí. La primera parte de este capítulo ha estado dedicada a exponer el estado del arte de los métodos de Optimización Global basados en técnicas de Branch and Bound. Hemos descrito los métodos tradicionales, los cuales nos han servido de base para posteriormente exponer nuestras aportaciones teóricas dentro de este campo de investigación. Desde esta perspectiva teórica las aportaciones de este capítulo pueden resumirse de la siguiente forma. Hemos definido una nueva forma de calcular funciones soporte para problemas unidimensionales que presentan uno o múltiples óptimos. Las nuevas funciones soporte se basan en la obtención de la información que habitualmente se usa en los métodos tradicionales, es decir, en la información obtenida de la evaluación de $F(x)$, $F(X)$ y $F'(X)$. La ventaja de nuestra propuesta es que a diferencia de los métodos tradicionales, que usan toda esta información de forma separada, la nueva propuesta que se ha hecho en este capítulo usa toda la información disponible en cada instante de forma conjunta, lo que da como resultado la construcción de funciones soporte que se ajustan mejor a la función que se pretende acotar. Basándonos en estas nuevas funciones soporte, hemos construido nuevas reglas de eliminación

y acotación que dan como resultado un nuevo algoritmo de Optimización Global, que acelera significativamente la búsqueda de la solución. De hecho, como se ha explicado en la última sección de este capítulo, el nuevo algoritmo (IAG) es casi dos veces mas rápido que otros métodos tradicionales. Estos resultados son empíricos y se han llevado a cabo sobre un amplio conjunto de funciones test.

Capítulo 4

Optimización Global Intervalar multidimensional

Este capítulo extiende los contenidos del Capítulo 3 a problemas multidimensionales. Se presentan las reglas de Branch and Bound normalmente usadas en algoritmos de Optimización Global Intervalar multidimensional, haciendo más énfasis en aquellas que se ven más afectadas por este cambio. A partir de estas reglas, se extiende el algoritmo básico unidimensional presentado en el capítulo anterior al caso multidimensional. También se resuelven los problemas de extender al caso multidimensional las aportaciones teóricas unidimensionales presentadas en el capítulo anterior. Se presenta un algoritmo que incorpora las nuevas mejoras. Finalmente, se evalúan las mejoras propuestas sobre un conjunto de funciones de prueba.

4.1. Reglas de Branch and Bound

Esta sección presenta las reglas de Branch and Bound que normalmente se usan en algoritmos de Optimización Global Intervalar para problemas multidimensionales.

4.1.1. Regla de Acotación

Esta regla es la misma que la presentada en la Sección 3.1.1, ya que la Aritmética de Intervalos permite obtener una *función de inclusión* para funciones reales de

cualquier dimensión.

4.1.2. Regla de Eliminación

Aquí se muestran las reglas de eliminación descritas en la Sección 3.1.2, pero con las particularidades de su aplicación en espacios multidimensionales.

Tanto el *Test del Punto Medio* como el *Test de Corte* no sufren variaciones ya que, al igual que en el caso unidimensional, los intervalos multidimensionales X con $\overline{f^*} < \underline{F}(X)$ son eliminados. Ahora la actualización de $\overline{f^*}$ se hace mediante la evaluación del punto medio de un intervalo multidimensional, es decir, la evaluación de $F(m(X))$ con $m(X) = (m(X_1), m(X_2), \ldots, m(X_n))$.

Test de monotonía

Si $0 \notin F_i'(X)$, para algún $i \in \{1, \ldots, n\}$, entonces no existe un punto estacionario en X. En particular, el mínimo global sólo podría estar en los extremos que X comparte con S.

Test de concavidad

Sea H la matriz Hessiana de F ($\nabla^2 F$). Si $f(x)$ tiene un mínimo en x^*, entonces $f(x)$ debe ser convexa alrededor de x^*. Una condición necesaria para que exista un mínimo en X, es que la diagonal principal de H sea positiva, i.e. $\underline{H}_{ii}(X) \geq 0, \forall i = 1, \ldots, n$. Por lo tanto, si $\overline{H}_{ii}(X) < 0$ para algún $i \in \{1, \ldots, n\}$, la caja X puede eliminarse de la región de búsqueda [49, 100].

Método de Newton de intervalos multidimensional

El método de Newton de intervalos multidimensional varía dependiendo del método usado para resolver el sistema lineal de ecuaciones, precondicionamientos, etc.

La Ecuación (3.12) puede generalizarse al caso multidimensional, estableciendo el vector $x^n = m(X^n)$ y $F'(X)$ como la extensión a intervalos del Jacobiano de f. Así :

$$N(X^{n+1}) = x^n - [F'(X^n)]^{-1} f(x^n),\ x^n \in X^n \tag{4.1}$$

Para obtener $N(X^{n+1})$, hay que resolver el siguiente sistema de ecuaciones:

$$F'(X^n)(N(X^{n+1}) - x^n) = -f(x^n) \tag{4.2}$$

Nótese que si en vez de encontrar las raíces de f, se quiere encontrar los mínimos de f, hay que utilizar f' en vez de f y la matriz Hessiana de f (Jacobiano de f'), en vez de el Jacobiano de f.

La resolución de la Ecuación (4.2), puede ser complicada debido a la necesidad de acotar el conjunto solución de un sistema de ecuaciones lineales, siendo los coeficientes intervalos. Los principales métodos para su resolución son: el método de Krawczyk [93, 95] y el método de Gauss-Seidel extendido a intervalos [65], aunque también se puede usar la extensión del método de eliminación de Gauss a intervalos [99]. Los libros [50, 64, 99] y [111] son buenas referencias para encontrar estudios de los diferentes métodos de Newton de intervalos multidimensionales, los cuales se escapan del ámbito de esta tesis.

En [11, 13, 15] se presentan algunas reglas heurísticas de eliminación que permiten resolver problemas computacionalmente muy costosos, aunque al coste de perder la garantía de que la solución encontrada sea la óptima.

4.1.3. Regla de Selección

Esta regla es la misma en el caso uni y multidimensional ya que se basa en el valor del límite inferior de la función de inclusión del rango real de f en X que fue obtenido por la Regla de acotación. (Véase la Sección 3.1.3).

4.1.4. Regla de División

Podría plantearse en este caso la división simultanea en varias dimensiones, no obstante un simple análisis del problema demuestra que el resultado es una explosión en el número de cajas (intervalos) generadas que desborda rápidamente la capacidad de memoria de cualquier computador. Varios autores han analizado este problema, en concreto, Berner [4] determinó que es más eficiente realizar una bisección de dos coordenadas, simultáneamente. Para los problemas de prueba que utilizó, Berner obtuvo una mejora de un 20 % respecto a la bisección de una sola coordenada. Hansen [50], propuso la división de tres coordenadas, pero sin mostrar resultados. Un número mayor de coordenadas a dividir no es aconsejable debido al gran número de subcajas generadas en cada división. En nuestro trabajo hemos optado por la regla más simple, realizar una única división del elemento seleccionado. Adicionalmente, para el caso multidimensional, es necesario añadir una nueva regla de división que

determine la componente (una entre n) a dividir. Csendes y Ratz [24, 113] estudiaron cuatro estrategias para determinar cual es la coordenada de la caja seleccionada sobre la que se debe realizar la bisección. En estos trabajos, la coordenada, k, que determina la dirección de división, viene dada por la siguiente ecuación:

$$k = \min\left\{ j : j \in \{1, \ldots, n\} \mid D(j) = \max_{i=1..n}\{D(i)\} \right\} \tag{4.3}$$

donde $D(i)$ depende de la estrategia elegida:

Estrategia A: Está justificada por la idea de dividir la región de búsqueda uniformemente, para que la anchura de los intervalos converja a cero [92, 111, 125]:

$$D(i) = w(X_i) \tag{4.4}$$

Estrategia B: Hansen [50] propuso una estrategia heurística de forma que la coordenada elegida fuera aquella en la que f varía mas:

$$D(i) = w(F'(X_i)) \cdot w(X_i) \tag{4.5}$$

Estrategia C: En esta estrategia, Ratz [112] intenta disminuir la anchura de la *función de inclusión*:

$$D(i) = w(F'(X_i) \cdot (X_i - m(X_i))) \tag{4.6}$$

Estrategia D: Esta estrategia intenta disminuir las sobreestimaciones de la *función de inclusión*, i.e. $w(F(X) - w(f(X))$, debidas a la representación en punto flotante del computador [49]:

$$D(i) = \begin{cases} w(X_i) & , 0 \in X_i \\ w(X_i)/\min\{|\, x_i \mid : x_i \in X_i\} & , \text{en otro caso} \end{cases} \tag{4.7}$$

Estrategia E: Esta estrategia es análoga a la **C**, pero usando información de la segunda derivada [113]:

$$D(i) = w\left((X_i - m(X_i)) \cdot \left(F'(m(X)) + \frac{1}{2} \sum_{j=1}^{n} (H_{ij}(X) \cdot (X_i - m(X_i)) \right) \right) \tag{4.8}$$

En [10] se ha estudiado el caso de una regla de división adaptativa basada en una estimación de la proximidad de un intervalo a un minimizador global. El grado de división es mayor cuanto más próximo esté el intervalo al optimizador global.

4.1.5. Regla de Terminación

Al igual que en el caso unidimensional (Sección 3.1.5) la regla de terminación esta basada en el valor de $w(X)$ y/o $w(F(X)$. Tal como se especificó en la Sección 1.2, la anchura de un intervalo multidimensional viene dada por la longitud de la componente con mayor anchura.

4.2. MTIAM: multidimensional TIAM

La extensión al caso multidimensional del algoritmo TIAM (Sección 3.2) la denominaremos MTIAM (Multidimensional Traditional Interval analysis global optimization Algorithm with Monotonicity test)[86]. MTIAM será el algoritmo de referencia que nos permitirá evaluar la extensión al caso multidimensional de las aportaciones teóricas mostradas en la Sección 3.3. Tal como se ha descrito en la sección anterior las reglas de Acotación, Selección, Eliminación y Terminación del MTIAM son las mismas que en el Algoritmo TIAM, teniendo en cuenta para algunas de ellas el carácter multidimensional del problema a resolver. En la regla de División usada por MTIAM, al igual que TIAM, se hace una bisección de la caja seleccionada. Ahora hay que determinar la coordenada k por la que se realiza la bisección. MTIAM selecciona la coordenada a dividir k que verifica que $w(X_k) = \max_{i=1,\ldots,n} w(X_i)$, lo que se corresponde con la estrategia A mostrada en la Sección 4.1.4.

4.3. Aportaciones teóricas multidimensionales

Esta sección extenderá al caso multidimensional las aportaciones teóricas que, para el caso unidimensional, se mostraron en la Sección 3.3.

4.3.1. Extensión natural de la regla de acotación al caso multidimensional

En principio, se pueden aplicar los resultados teóricos obtenidos en el Lema 3.3.1, para el caso de funciones multidimensionales. Dada una función continua y diferenciable $f : S \to \mathbb{R}$, donde S es un vector de intervalos de $\mathbb{R}^n$, un vector de intervalos $X \subseteq S$, una inclusión $F(c)$ de $f(c)$, $c = (c_1, \ldots, c_n) \in X$, y una inclusión $F'(X)$ de $f'(x)$, $x \in X$, entonces se pueden definir los siguientes límites para la función $f(x), x \in X$:

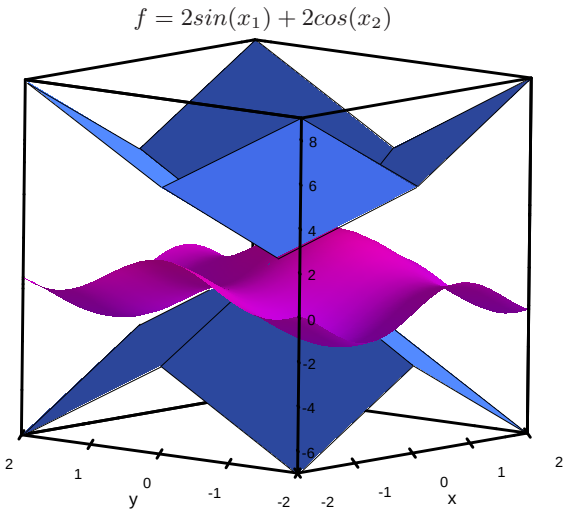


Figura 4.1: Límites piramidales de la función.

$$\underline{F}(c) + \min \left\{ \begin{array}{c} \overline{F'}(X) \cdot (x-c) \\ \underline{F'}(X) \cdot (x-c) \end{array} \right\} \leq f(x) \leq \overline{F}(c) + \max \left\{ \begin{array}{c} \overline{F'}(X) \cdot (x-c) \\ \underline{F'}(X) \cdot (x-c) \end{array} \right\}$$

Para mostrar con más claridad las acotaciones de la función, en la Figura 4.1 se ha representado la aplicación del Lema 3.3.1 para una función bidimensional $f = sin(2 * x) + cos(2 * y)$. Como puede observarse, los valores de la función se encuentran incluidos en el espacio definido por dos pirámides, una superior y otra inferior.

Igualmente, los resultados obtenidos en el Teorema 3.3.1, pueden aplicarse al caso de funciones multidimensionales. Sean X y S dos intervalos multidimesionales, cerrados y acotados que cumplen que $X \subseteq S \subset \mathbb{I}^n$, y sea $f : S \to \mathbb{R}$ una función continua y diferenciable. Supongamos que para un punto $c = (c_1, \ldots, c_n) \in X$ se determina un límite inferior de la función $\underline{lb}(c)$ de $f(c)$ y se obtiene una inclusión $F'(X)$ de $f'(X)$. Para un determinado límite superior actualizado $\overline{f^*}$ de la solución f^*, existe un conjunto $V \subseteq X$, donde están incluidos todos los mínimos globales de X, si existen.

Si además, como se suponía en el Teorema 3.3.1, se conoce un límite superior de la solución $\overline{f^*} < \underline{lb}(c)$, entonces es posible obtener la intersección del valor de $\overline{f^*}$

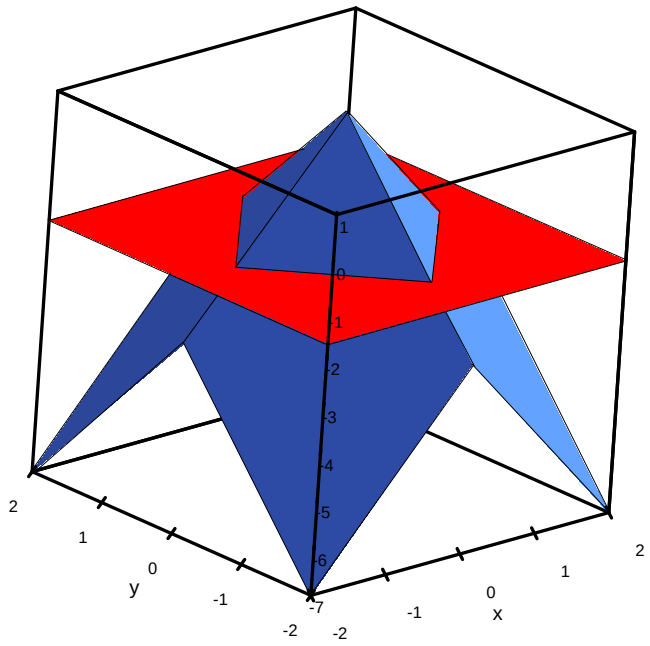


Figura 4.2: Corte de los planos con $\overline{f^{\sim}}$.

con los límites inferiores deducidos de la aplicación Lema 3.3.1 al caso de funciones multidimensionales, obteniendo nuevamente una región donde se puede asegurar que no está el mínimo global de la función.

En la Figura 4.2 se ha representado una aplicación del Teorema 3.3.1 para el caso concreto de la función bidimensional $f = \sin(2x)+\cos(2y)$. El área definida por la base de la pirámide es aquella que no contiene una solución, es decir el mínimo global de la función.

Para este simple caso bidimensional, el problema de reducir el espacio de búsqueda de la solución en el vector de intervalos X, equivale a determinar las líneas de intersección. En general, para funciones multidimensionales el problema es mucho más complejo y habría que determinar 2^n posibles intersecciones. Incluso para el caso bidimensional el problema es extremadamente complejo. El problema adicional, que aparece como consecuencia de la aplicación de esta extensión natural, es que la geometría del espacio que no contiene una solución presenta unos límites que no son paralelos a los ejes de coordenadas. Una solución podría ser eliminar el paralelogramo inscrito dentro del área rechazable. Pero para extraer este paralelo-

gramo de X, el número de divisiones realizadas sobre X da lugar a que el número de subintervalos generados sea relativamente grande dependiendo de n.

En definitiva, esta extensión natural para el caso multidimensional, podría ser muy costosa y la eficiencia del correspondiente algoritmo podría verse afectada. Por esta razón, se debe buscar un método alternativo que determine las regiones que pueden ser eliminadas y que sean compatibles con la geometría de división de los intervalos de búsqueda. En esa línea apuntan las discusiones de las siguientes secciones.

4.3.2. Extensión por componentes de la regla de acotación al caso multidimensional

Como se ha visto en la sección anterior, es complicado aplicar directamente los resultados teóricos vistos en el capitulo anterior al caso de funciones multidimensionales. Por lo tanto, nos hemos planteado la obtención de nuevos resultados teóricos que nos permitan generar nuevos test de rechazo o de reducción de los intervalos para funciones multidimensionales haciendo uso de las ideas presentadas en [115]. Antes de definir los nuevos test de rechazo, ampliaremos la nomenclatura.

Definición 4.3.1
Sea $X(i : p) = (X_1, .., X_{i-1}, [p, p], X_{i+1}, .., X_n),\ p \in X_i$, *el vector de intervalos obtenido a partir de* X, *cuya componente* i*-ésima es el intervalo punto* $[p, p]$.

Una vez fijada la componente i del intervalo X, (X_i), usaremos básicamente tres casos:

- cuando p es el extremo izquierdo de X_i denotaremos $X_i^l = X(i : \underline{x}_i)$,
- cuando p es el extremo derecho denotaremos $X_i^r = X(i : \overline{x}_i)$,
- y cuando p es el punto medio denotaremos $X_i^m = X(i : m(X_i))$.

De forma general podemos definir el límite inferior de $f(X)$ como $lbf(X) \in \mathbb{R}$ cumpliéndose que $lbf(X) \leq f(x),\ \forall x \in X$.

Definición 4.3.2
Sea $sp(X) = (sp(X_1), \ldots, sp(X_n))$, *la función soporte de* $f(X)$ *en los extremos del intervalo* X, *donde* $sp(X_i) = \{sp(X_i^l), sp(X_i^r)\} = \{lbf(X_i^l), lbf(X_i^r)\}, i = 1 \ldots n$.

Teorema 4.3.1
Sean X *y* S *dos vectores de intervalos cerrados que cumplen que* $X \subseteq S \subset \mathbb{I}^n$ *y* $f : S \to \mathbb{R}$ *una función continua y diferenciable. Supongamos que para un punto*

$c = (c_1, \ldots, c_n) \in X$, se determina un límite inferior $lbf(X(i : c_i))$, con $X(i : c_i) \in X$, de $f(X(i : c_i))$ y se obtiene una función de inclusión $F'(X)$ de $f'(X)$. Para un determinado límite superior actualizado $\overline{f^\sim}$ de la solución f^*, existe un conjunto $X^a_{go} \subseteq X$, donde están incluidos todos los mínimos globales de X, si existen.

Demostración: Dado que para un punto minimizador $x^* \in S^*$, se verifica que $f(x^*) \leq \overline{f^\sim}$. Del Lema 3.3.1 del capitulo anterior, un minimizador $x^* \in X \cap S^*$ tiene que cumplir que:

$$lbf(X(i : c_i)) + \min\{F'_i(X) \cdot (x^*_i - c_i)\} \leq f(x^*) \leq \overline{f^\sim}.$$

y por lo tanto, únicamente puede estar localizado en el conjunto:

$$X^a_{go} = \left\{x \in X : lbf(X(i : c_i)) + \min\{F'_i(X) \cdot (x_i - c_i)\} \leq \overline{f^\sim}\right\}.$$

□

Del Teorema 4.3.1 podemos deducir que si $\overline{f^\sim} < lbf(X(i : c_i))$, entonces $X(i : c_i) \notin S^*$ y se puede construir un intervalo $X^a_{go} = X \backslash V$, donde V viene dado por:

$$V = \left\{x \in X : x_i \in \left[c_i - \frac{lbf(X(i:c_i)) - \overline{f^\sim}}{\overline{F}'_i(X)}, c_i - \frac{lbf(X(i:c_i)) - \overline{f^\sim}}{\underline{F}'_i(X)}\right]\right\} \tag{4.9}$$

cuando $0 \in F'(X)$. Como ejemplo, mostramos V en la Figura 4.3, para el caso concreto de $c_1 = X^m_1$ y $0 \in F'(X)$. Si $0 \notin F'(X)$ entonces $x^* \notin X$.

Teorema 4.3.2
Sea $f : S \to \mathbb{R}$ una función continua y diferenciable, donde S es un vector de intervalos cerrado de $\mathbb{R}^n$, y sean X, Y, dos vectores de intervalos tal que $X \subseteq Y \subseteq S$. Si para algún $i \in 1, \ldots, n$:

1. *los límites inferiores $lbf(X^l_i)$ y $lbf(X^r_i)$ de $f(X^l_i)$ y $f(X^r_i)$, han sido evaluados;*
2. *un límite superior actualizado de f^*, $\overline{f^\sim}$ verifica que:*

$$\overline{f^\sim} \leq \min\{lbf(X^l_i), lbf(X^r_i)\};$$

3. *acotaciones $G_i = F'_i(Y)$ de $f'_i(Y)$ tales que $0 \in F'(Y)$, de forma que también son acotaciones de $f'_i(X)$, han sido obtenidos.*

Entonces, solamente el intervalo definido como:

$$X^b_{go} = \left\{x \in X : x_i \in \left[\underline{x}_i - \frac{lbf(X^l_i) - \overline{f^\sim}}{\underline{G}_i}, \overline{x}_i - \frac{lbf(X^r_i) - \overline{f^\sim}}{\overline{G}_i}\right]\right\} \tag{4.10}$$

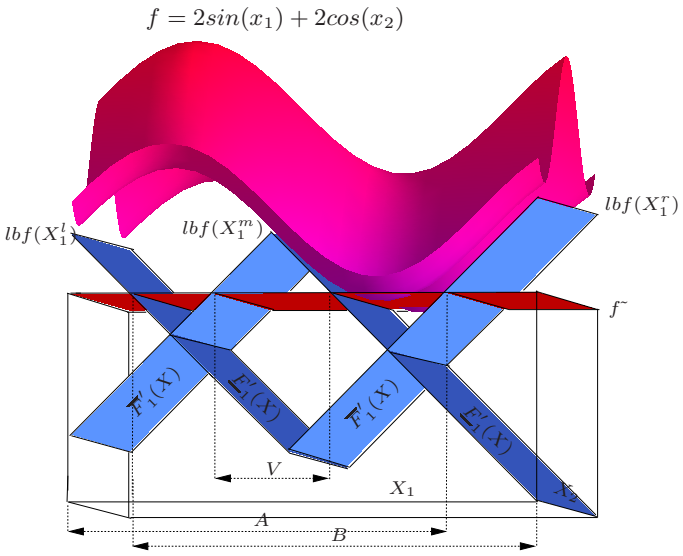


Figura 4.3: Funciones soporte $F_1'(X)$, $lbf(X_1^l)$, $lbf(X_1^m)$ y $lbf(X_1^r)$. F' determina la pendiente de los planos.

puede contener minimizadores globales y un nuevo límite inferior $z(X)$ *de* $f(X)$ *puede ser calculado como sigue:*

$$z(X) = \max_{j\in\{1,\ldots,n\}} \left\{ \frac{lbf(X_j^l)\cdot\overline{G}_j - lbf(X_j^r)\cdot\underline{G}_j}{w(G_j)} + w(X)\cdot\frac{\underline{G}_j\cdot\overline{G}_j}{w(G_j)} \right\}. \tag{4.11}$$

Demostración: Aplicando el Teorema 4.3.1 para $X(i:c_i) = X_i^l$ y para $X(i:c_i) = X_i^r$, obtenemos los subvectores de intervalos $A = X_{go}^a(X_i^l)$ y $B = X_{go}^a(X_i^r)$ de X y el vector de intervalos $X_{go}^b = A \cap B$ (ver la Figura 4.3).

La Ecuación (4.11) puede ser demostrada, considerando que $X \subseteq Y$, $F'(X) \subseteq F'(Y) = G$, y aplicando el teorema del valor medio como se hizo en la demostración del Teorema 3.3.2. □

Corolario 4.3.1
Si para un vector de intervalos X *se cumple la desigualdad* $z(X) > \overline{f^{\sim}}$*, entonces podemos concluir que* X *no contiene ningún mínimo global.*

Volviendo ahora sobre el problema definido en la Ecuación (1.6), podemos usar la información sobre los valores de $F(x)$, $F(X)$ y $F'(X)$ obtenidos durante la búsqueda

global. Así pues, usando $\underline{F}(X)$ junto con el valor de $z(X)$ de la Ecuación (4.11), y la forma centrada $F_c(X, m(X))$ (ver Definición 2.3.5), podemos construir una nueva función soporte $lbzf(X)$ de $f(X)$ de la siguiente manera:

$$lbzf(X) = \max\{\underline{F}(X), z(X), \underline{F_c}(X, m(X))\}. \tag{4.12}$$

La parte fundamental de este estudio, es que para cualquier vector de intervalos W obtenido a partir del vector de intervalos X de acuerdo con la Ecuación (4.10) ($W = X^b_{go}$), el valor actualizado de $\underline{f^\sim}$ es un límite inferior de $f(x)$ en W^l_i y W^r_i; por ejemplo $\underline{f^\sim} \leq f(W^l_i)$ y $\underline{f^\sim} \leq f(W^r_i)$, y por lo tanto $lbf(W^l_i) = lbf(W^r_i) = \underline{f^\sim}$ son límites que pueden ser obtenidos fácilmente.

Aplicando la forma centrada, es posible obtener un límite inferior de $f(X(i : c_i))$, con $c = (c_1, c_2, \ldots, c_n) \in X$, es decir:

$$lbf(X(i : c_i)) = \underline{F_c}(X(i : c_i), c). \tag{4.13}$$

Además, si previamente hemos evaluado $F(c)$ y $F'(X)$, el valor de $\underline{F_c}(X(i : c_i), c)$ puede ser calculado sin necesidad de evaluaciones adicionales de la función de inclusión.

A continuación se muestra un ejemplo bidimensional donde $c = m(X)$ y se conocen $F(c)$ y $F'(X)$ (ver Figura 4.4). Por lo tanto,

$$F_c(X, m(X)) = F(m(X)) + F'(X)(X - m(X)) =$$

$$F(m(X)) + F'_1(X)(X_1 - m(X_1) + F'_2(X_2 - m(X_2))$$

y como

$$m(X^m_i) = m(X)$$

y

$$F'(X^m_i) \subseteq F'(X)$$

entonces tenemos que:

$$F_c(X^m_1, m(X^m_1)) = F(m(X)) + F'_1(X)(m(X_1) - m(X_1)) + F'_2(X)(X_2 - m(X_2))$$

y

$$F_c(X^m_2, m(X^m_2)) = F(m(X)) + F'_1(X)(X_1 - m(X_1)) + F'_2(X)(m(X_2) - m(X_2))$$

Por lo que podemos obtener un límite inferior de $f(X^m_i)$ sin necesidad de realizar evaluaciones intervalares adicionales.

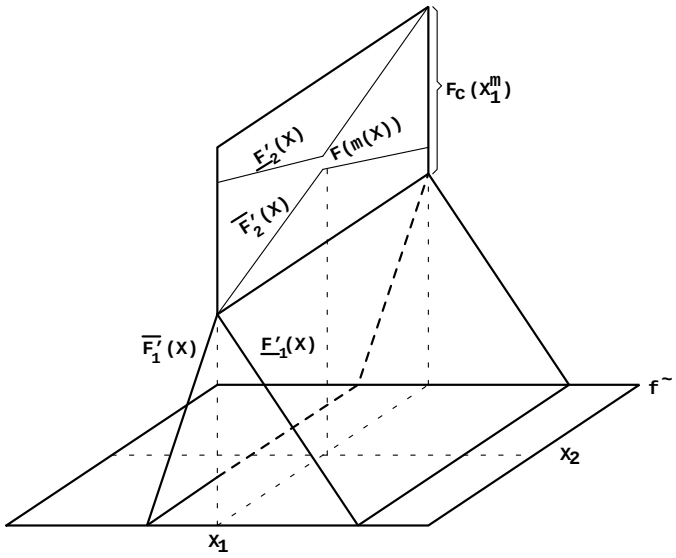


Figura 4.4: Reducción de evaluaciones intervalares mediante el uso de la forma centrada.

4.4. MIAG: multidimensional IAG

El algoritmo MIAG (multidimensional Interval analysis global optimization Algorithm using Gradient information [86]) es la extensión al caso multidimensional del Algoritmo IAG. El Algoritmo MIAG también puede verse como el resultado de incluir las aportaciones matemáticas mostradas en la sección anterior al Algoritmo MTIAM (Sección 4.2). El algoritmo MIAG esta detalladamente descrito en el Algoritmo 4.1.

El algoritmo MIAG obtiene un límite inferior de la función ($lbzf(X)$ Ecuación (4.12)) mediante el uso del límite inferior de la función obtenida por Aritmética de Intervalos ($\underline{F}(X)$), la aplicación de la forma centrada y el cálculo del valor de $z(X)$. El valor de $z(X)$ se obtiene a partir de los valores del intervalo que acota el gradiente de la función y de las cotas inferiores de la función en dos de los extremos del intervalo X. Los valores de la función $lbzf(X)$ en un intervalo X permiten mejorar las reglas de Selección y de Acotación, el *Test del Punto Medio* y el *Test de Corte*.

También se ha extendido al caso multidimensional el *Test Gradient* que ahora esta basado en el Teorema 4.3.2 y se describe en el Algoritmo 4.2. Siguiendo la notación presentada al principio de esta sección, para poder aplicar el *Test Gra-*

Algorithm 4.1: Algoritmo n-dimensional (**MIAG**).

Entrada: S, F, ϵ
Salida : Q,$f^{\sim}$

1 $sp(S) = \big(\{\underline{F}(S_1^l), \underline{F}(S_1^r)\}, \ldots, \{\underline{F}(S_n^l), \underline{F}(S_n^r)\}\big)$;
2 Eval $F(S)$, $f^{\sim} = F(m(S))$, $F'(S)$, and $lbzf(S)$;
3 L=$\{ListNode(S)\}$;
4 Q=$\{\emptyset\}$;
5 **while** *($L \neq \emptyset$)* **do** `/* lista de trabajo no vacía */`
6 $ListNode(X)$=PopHead (L);
7 $\underline{F_c}(X_k^m, m(X)) = \max_i\{\underline{F_c}(X_i^m, m(X))\}$;
8 $W^1 = X$; $W_k^1 = [\underline{x}_k, m(\overline{X}_k)]$;
9 $W^2 = X$; $W_k^2 = [m(X_k), \overline{x}_k]$;
10 $sp(W^1) = sp(X)$;
11 $sp(W_k^1) = \big\{sp(X_k^l), \underline{F_c}(X_k^m, m(X))\big\}$;
12 $sp(W^2) = sp(X)$;
13 $sp(W_k^2) = \big\{\underline{F_c}(X_k^m, m(X)), sp(X_k^r)\big\}$;
14 **for** *(i:=1,2)* **do** `/* Para cada subproblema generado */`
15 GradTest (W_k^i, $F_k'(X)$, $sp(W_k^i)$, $f^{\sim}$);
16 **if** $w(W^i) > 0$ **then**
17 Eval $F(W^i)$, $F'(W^i)$;
18 **if** MonotonocityTest *($F'(W^i)$)* **then**
19 Continuar;
20 **if** $\overline{F}(m(W^i)) < \overline{f^{\sim}}$ **then**
21 TestCorte ($\overline{f^{\sim}}$,L,Q);
22 **if** $lbzf(W^i) \leq \overline{f^{\sim}}$ **then**
23 **if** *($w(W^i) \leq \epsilon$)* **then** `/* Crit. de Parada */`
24 Almacenar $ListNode(W^i)$ en Q;
25 **else** Almacenar $ListNode(W^i)$ en L;

dient necesitamos calcular límites inferiores de la función objetivo en los lados del intervalo seleccionado X, es decir, la función soporte $sp(X)$. Esta evaluación se realiza directamente sobre el intervalo de búsqueda inicial (linea 1, Algoritmo 4.1). Para obtener una mayor eficiencia del *Test Gradient*, la coordenada k del intervalo seleccionado X, donde se realizará una bisección, es la que verifica la línea 7 del Algoritmo 4.1, generando los subintervalos W^1 y W^2 (líneas 8 y 9). Los valores de la función soporte para los intervalos W^1 y W^2 se establecen en las líneas 10 a 13 del Algoritmo 4.1. La elección de la coordenada k está motivada porque la misión del algoritmo *Test Gradient* consiste en encontrar los cortes de las pendientes con el plano $f^{\sim}$ para eliminar aquellos subintervalos del intervalo de entrada donde no puede existir un mínimo global de la función. Como las pendientes se apoyan en

Algorithm 4.2: Algoritmo TestGradient multidimensional.

```
1 if ( sp(X^l) > f~ ) then
2     Y = [x - sp(X^l) - f~ / F'(X), ∞) ;
3     sp(X^l) = f~ ;
4     X = Y ∩ X ;
5 if ( w(X) > 0 & sp(X^r) > f~ ) then
6     Y = (-∞, x - sp(X^r) - f~ / F'(X)] ;
7     sp(X^r) = f~ ;
8     X = Y ∩ X ;
```

los valores de la función $sp(W_k^i),\ i = 1, 2$, se elige la coordenada k que obtenga los valores más altos y que permite rechazar una mayor región de búsqueda.

El resultado del Algoritmo 4.2, es que frecuentemente el intervalo de entrada se reduce en uno o en ambos extremos de la dirección de la coordenada seleccionada (por la derecha y/o por la izquierda).

El algoritmo MIAG usa una lista de trabajo (L), donde se encuentran todos los intervalos que tienen que ser procesados y una lista final (Q) donde se van almacenando todos los intervalos cuyo tamaño es inferior a un umbral ϵ, para los que no se ha podido demostrar la inexistencia de un mínimo global de la función. Para cada elemento de estas listas se almacena toda la información relativa a los límites del intervalo, de la función y del gradiente de la función ($ListNode(X) = (X, lbzf(X), sp(X), F(m(X)), F'(X))$.

4.5. Eficiencia de MIAG frente a MTIAM

El algoritmo MIAG ha sido exhaustivamente evaluado y numéricamente comparado con el método estándar descrito en la Sección 4.2 (Algoritmo MTIAM). Se ha usado un conjunto de cuarenta funciones de prueba, que se listan en la Tabla 4.1, y que han sido tomadas de la literatura de Optimización Global [24, 30, 116, 133, 141]. Las referencias donde se describe cada una de las funciones se suministran en la columna Ref. La columna n especifica la dimensión de cada una de las funciones. La mitad de las funciones son bidimensionales, siete de ellas tridimensionales y el resto corresponden a $4, 6, 7, 9$ y 10 dimensiones. En el Apéndice A se suministra información detallada sobre todas las funciones de prueba que se han utilizado en este capítulo. Las columnas Esf_1 y Esf_2 de la Tabla 4.1 muestran los resultados de la evaluación de los algoritmos MTIAM y MIAG, respectivamente. En lugar de medir

tiempos de ejecución que pueden venir afectados por cuestiones relacionadas con la gestión de las listas de trabajo, con cuestiones relacionadas con fallos de memoria, o con el coste de evaluación de una función, hemos realizado la evaluación de ambos algoritmos en términos de complejidad computacional, es decir en términos del número de veces que se evalúa la función de prueba o su gradiente. Suponiendo que FE representa el número de veces que se evalúa la función en un intervalo ($F(X)$) mas el número de veces que se evalúa la función en un punto ($F(x)$) y que GE es el número de evaluaciones de la función gradiente ($F'(X)$), Esf_1 y Esf_2 se han calculado como $FE + nGE$ para los algoritmos MTIAM y MIAG, respectivamente. La columna $SpUp$ muestra los valores de $SpUp = Esf_1/Esf_2$, suministrando información sobre la aceleración relativa del nuevo método propuesto frente al tradicional. Se han ejecutado ambos algoritmos con un criterio de parada de $\epsilon = 10^{-8}$ y un tiempo límite de ejecución de una hora. Para la mayoría de las funciones ambos algoritmos terminaron la ejecución, pero para un conjunto de diez funciones de prueba MTIAM no pudo acabar la ejecución. Para este subconjunto de funciones se muestran los valores de ϵ para los que ambos algoritmos pudieron terminar la ejecución, no obstante MIAG es capaz de encontrar soluciones para estas funciones con una mayor precisión.

Como puede verse en la Tabla 4.1, los valores del $SpUp$ son menores que uno solamente para tres de las cuarenta funciones de prueba y que en promedio (ver la penúltima fila de la Tabla 4.1) MIAG es 1,93 veces menos costoso que MTIAM. Nótese que el rango de valores de $SpUp$ se encuentra en el intervalo $[0.6, 10.7]$. Como muestran los resultados numéricos de la Tabla 4.1, el algoritmo MIAG mejora computacionalmente al método MTIAM mediante el uso refinado de la información del gradiente. Esta mejora es consecuencia de la reducción del espacio de búsqueda, la cual se obtiene de un aprovechamiento inteligente y adecuado de los datos, que en cualquier caso (también para el algoritmo MTIAM) tienen que ser evaluados. Para aquellos problemas en los que el test de monotonía no tiene ningún efecto positivo, el uso de la información suministrada por el gradiente no puede incrementar la eficiencia del algoritmo, así pues las regiones no pueden reducirse, y esto puede afectar a la convergencia del algoritmo. Esto es lo que justifica la perdida de eficiencia en algunas (solo tres) de las funciones de prueba. Por otro lado, para funciones en las que el gradiente demuestra suministrar suficiente información (por ejemplo, la función Goldstein-Price) las mejoras son sorprendentemente buenas (SpUp=10,7).

4.6. Refinamiento de los algoritmos MTIAM y MIAG

Como todos los algoritmos, tanto MTIAM como MIAG son susceptibles de mejoras. En esta sección describimos los pequeños refinamientos introducidos en los algoritmos MTIAM y MIAG, que disminuyen el costo computacional, simplemente, haciendo un mejor uso de la información disponible en cada instante. En el algoritmo

Tabla 4.1: Comparación entre los algoritmos MTIAM y MIAG.

Name	Ref	n	Esf_1	Esf_2	$SpUp$	ϵ
Schwefel 3.1	[116]	3	874	994	0.9	10^{-8}
Price	[31]	2	5322	5951	0.9	10^{-8}
Levy 5	[116]	2	1587	1598	1.0	10^{-8}
Shekel 10	[116]	4	1365	1374	1.0	10^{-8}
Schwefel 3.7	[141]	2	1762	1772	1.0	10^{-8}
Levy 8	[116]	3	851	857	1.0	10^{-8}
Shekel 5	[116]	4	1339	1348	1.0	10^{-8}
Schwefel 2.1 (Beale)	[141]	2	5560	5523	1.0	10^{-8}
Shekel 7	[116]	4	1365	1350	1.0	10^{-8}
Levy 3	[116]	2	7116	6979	1.0	10^{-8}
Rastrigin	[133]	2	1564	1496	1.0	10^{-8}
Schwefel 2.5 (Booth)	[141]	2	488	466	1.0	10^{-8}
Henriksen-Madsen 3	[54]	2	12204	11575	1.1	10^{-8}
Henriksen-Madsen 4	[54]	3	63693	59870	1.1	10^{-8}
Treccani	[30]	2	2430	2227	1.1	10^{-8}
EX1	[24]	2	488	443	1.1	10^{-8}
Branin	[116]	2	4869	4367	1.1	10^{-8}
Chichinadze	[31]	2	653	576	1.1	10^{-8}
Griewank 2	[133]	2	1952	1642	1.2	10^{-8}
Schwefel 1.2	[141]	4	27975	22963	1.2	10^{-8}
Schwefel 3.2	[141]	3	3170	2484	1.3	10^{-8}
Rosenbrock 2	[30]	2	1279	887	1.4	10^{-8}
Ratz 4	[116]	2	7096	4772	1.5	10^{-8}
Hartman 6	[116]	6	20996	13020	1.6	10^{-8}
Three-Hump-Camel-Back	[30]	2	3990	2138	1.9	10^{-8}
Hartman 3	[116]	3	4463	2046	2.2	10^{-8}
Schwefel 2.18 (Matyas)	[141]	2	10944	4812	2.3	10^{-8}
Six-Hump-Camel-Back	[133]	2	6824	2638	2.6	10^{-8}
Simplified Rosenbrock	[30]	2	2386	831	2.9	10^{-8}
Goldstein-Price	[116]	2	320969	30128	10.7	10^{-8}
Schwefel 2.14 (Powell)	[116]	4	387176	595993	0.6	10^{-5}
Schwefel 3.1p	[116]	3	7854	4131	1.9	10^{-3}
Ratz 5	[116]	3	917495	331049	2.8	10^{-3}
Ratz 6	[116]	5	2162657	468513	4.6	10^{-3}
Griewank 10	[133]	10	3875828	3869704	1.0	10^{-2}
Schwefel 2.10 (Kowalik)	[141]	4	673544	496155	1.4	10^{-2}
Rosenbrock 10	[30]	10	2708380	2045727	1.3	10^{-2}
EX2	[24]	5	1016177	256975	4.0	10^{-2}
Ratz 7	[116]	7	245691	50229	4.9	10^{-2}
Ratz 8	[116]	9	388997	71367	5.5	10^{-2}
Av. Val.			322734.3	209674.3	1.93	
$\sum$			12909373	8386970	1.54	

MTIAM hemos incorporado los conceptos relacionados con las formas centradas (ver Definición 2.3.5). Esto simplemente implica la modificación de la regla de acotación. En el algoritmo MTIAM la regla de acotación se basaba únicamente en la evaluación de $F(X)$, obteniendo un límite inferior para $f(X)$ dado por $lbf(X) = \underline{F}(X)$. La incorporación de la forma centrada en la regla de acotación simplemente significa que el límite inferior de $f(X)$ se obtiene como $lbf(X) = \max\{\underline{F}(X), \underline{F_c}(X, m(X))\}$. El algoritmo MTIAM con esta nueva regla de acotación lo hemos denominado MIGO [85].

Por otro lado, para el algoritmo MIAG también hemos ideado un nuevo refinamiento que mejora su eficiencia computacional. Debe tenerse en cuenta que MIAG ya incluye la idea de la forma centrada en su propia regla de acotación. Las mejoras introducidas en el algoritmo MIAG están relacionadas con la regla de división. Para los dos algoritmos descritos hasta ahora (MTIAM y su versión refinada MIGO), la regla de división consistía en elegir la coordenada k que verifica que $w(X_k) = \max\{w(X_i),\ i = 1, \ldots, n\}$. En el algoritmo MIAG la coordenada k a dividir se obtenía como el k tal que $\underline{F_c}(X_k^m, m(X)) = \max_i\{\underline{F_c}(X_i^m, m(X))\}$. La nueva regla de selección de la coordenada de bisección consiste en realizar una estimación de las regiones que serian eliminadas cuando la bisección de un intervalo se hace en cada una de las n coordenadas. La coordenada k a dividir seria aquella que elimina una mayor región. El algoritmo MIAG modificado con esta nueva regla de selección de la coordenada de bisección se le ha denominado AMIGO (Advanced Multidimensional Interval analysis Global Optimization algorithm) [85], y se describe detalladamente en el Algoritmo 4.3.

El algoritmo AMIGO llama en la linea 8 a la rutina CoordinateToSplit (Algoritmo 4.4) como implementación de la regla de selección de la dirección de subdivisión. Esta rutina devuelve el valor de la coordenada k a dividir y un límite inferior de $f(X_k^m)$. La rutina CoordinateToSplit juega un papel importante dentro del algoritmo AMIGO. Esta rutina selecciona la coordenada k que permite que la regla de eliminación implementada como el algoritmo de GradientTest (Algoritmo 4.2) maximice el área a ser rechazada. CoordinateToSplit usa la rutina EvalRejectArea (Algoritmo 4.5) para calcular el valor del área que seria rechazada en caso de que el algoritmo dividiera la caja X en la dirección i. EvalRejectArea es evaluado para $i = 1, \ldots, n$ y se queda con el valor de k que maximiza EvalRejectArea(i), con $i = 1, \ldots, n$. EvalRejectArea evalúa la Ecuación (4.9) usando los valores de $f(X_m^i)$, $\overline{f^*}$ y $F_i'(X)$, con $c_i = m(X_i)$.

La rutina CoordinateToSplit analiza las siguiente posibilidades:

1. Aplicar el procedimiento EvalRejectArea, para evaluar las posibles áreas rechazadas usando $\underline{F_c}(X_i^m, m(X))$ como valor del límite inferior $lbf(X_i^m)$, para cada componente $i = 1, \ldots, n$. Esto permitirá evitar el cálculo de funciones de inclusión adicionales ($F(X_i^m)$ $i = 1, \ldots, n$). Si el resultado es un valor mayor que cero (área a rechazar), la rutina devuelve la coordenada k_1 que maximiza el volumen rechazado y el valor de $\underline{F_c}(X_i^m, m(X))$ como valor de $sp(X_{k_1}^m)$ (Algoritmo 4.4, línea 7)

2. Si el área rechazada con la opción anterior es nula, el algoritmo aplica EvalRejectArea usando $\underline{F}(X_i^m)$ en lugar de $\underline{F_c}(X_i^m, m(X))$. Esto viene motivado por el hecho de que la extensión natural de intervalos usualmente suministra mejores acotaciones que la forma centrada cuando los intervalos son grandes. En este caso, tienen que realizarse n evaluaciones adicionales de la función de inclu-

Algorithm 4.3: AMIGO.

Entrada: S, F, ϵ
Salida : Q,$f^{\sim}$

$sp(S) = \big(\{\underline{F}(S_1^l), \underline{F}(S_1^r)\}, \ldots, \{\underline{F}(S_n^l), \underline{F}(S_n^r)\}\big)$;
$NaiveF(S)$=$\{NaiveF(S_i^m) = True\}$, $i = 1, \ldots, n$;
Eval $F(S)$, $f^{\sim} = F(m(S))$, $F'(S)$, and $lbzf(S)$;
L=$\{ListNode(S) = (S, lbzf(S), sp(S), F(m(S)), F'(S), NaiveF(S))\}$;
Q=$\{\emptyset\}$;
while $L \neq \emptyset$ **do** `/* lista de trabajo no vacía */`
  $ListNode(X)$=PopHead (L);
  $\{k, sp(X_k^m)\}$=CoordinateToSplit $(X$,F,$NaiveF(X)$,$f^{\sim})$;
  $W^1 = X$; $W_k^1 = [\underline{x}_k, m(X_k)]$;
  $sp(W^1) = sp(X)$; $sp(W_k^1) = \{sp(X_k^l), sp(X_k^m)\}$;
  $NaiveF(W^1) = NaiveF(X)$;
  $W^2 = X$; $W_k^2 = [m(X_k), \overline{x}_k]$;
  $sp(W^2) = sp(X)$; $sp(W_k^2) = \{sp(X_k^m), sp(X_k^r)\}$;
  $NaiveF(W^2) = NaiveF(X)$;
  for *(i:=1,2)* **do** `/* Para cada subproblema generado */`
    GradTest $(W_k^i$,$F_k'(X)$,$sp(W_k^i)$,$f^{\sim})$;
    if $w(W^i) > 0$ **then**
      Eval $F(W^i)$, $F'(W^i)$;
      if MonotonocityTest *($F'(W^i)$)* **then**
        Continuar;
      if $\overline{F}(m(W^i)) < \overline{f^{\sim}}$ **then**
        TestCorte ($\overline{f^{\sim}}$,L,Q);
      if $lbzf(W^i) \leq \overline{f^{\sim}}$ **then**
        if *($w(W^i) \leq \epsilon$)* **then** `/* Crit. de Parada */`
          Almacenar $ListNode(W^i)$ en Q;
        else Almacenar $ListNode(W^i)$ en L;

Algorithm 4.4: CoordinateToSplit Algorithm.

$lbf(X_i^m) = \underline{F_c}(X_i^m, m(X)),\ i = 1 \dots n$;
for $i = 1 \dots n$ **do**
  $RejectArea(i) = EvalRejectArea(X, i, lbf(X_i^m), \overline{f^{\sim}}, F_i'(X))$;
$RejectArea(k_1) = \max_i\{RejectArea(i)\}$;
if $(RejectArea(k_1) > 0)$ **then**
  $sp(X_{k_1}^m) = lbf(X_{k_1}^m)$;
  Return k_1, $sp(X_{k_1}^m)$;
$lbf(X_{k_2}^m) = \max_i\{lbf(X_i^m)\}$;
if $(\ NaiveF(X_{k_2}^m) = True\ )$ **then**
  if $(\ w(F(X_{k_2}^m)) > w(F_c(X_{k_2}^m, m(X))))$ **then**
    $NaiveF(X_{k_2}^m) = False$;
  $RejectArea(k_2) = EvalRejectArea(X, k_2, \underline{F}(X_{k_2}^m), \overline{f^{\sim}}, F_i'(X))$;
if $(RejectArea(k_2) > 0)$ **then**
  $sp(X_{k_2}^m) = \underline{F}(X_{k_2}^m)$;
  Return k_2, $sp(X_{k_2}^m)$;
$w(F'_{k_3}(X) \cdot (X_{k_3} - m(X_{k_3}))) = \max_i\{w(F_i'(X) \cdot (X_i - m(X_i)))\}$;
$sp(X_{k_3}^m) = lbf(X_{k_3}^m)$;
Return k_3, $sp(X_{k_3}^m)$;

Algorithm 4.5: EvalRejectArea Algorithm.

$RejectA = 0$;
$Vol(X) = \prod_{i=1\dots n} w(X_i)$;
if $(\ lbf(X_i^m) > \overline{f^{\sim}}\ )$ **then**
  if $((\overline{f^{\sim}} - lbf(X_i^m))/\underline{F_i'}(X) < w(X_i)/2)$ **then**
    $RejectA = (\overline{f^{\sim}} - lbf(X_i^m))/\underline{F_i'}(X) \cdot Vol(X)/w(X_i)$
  else $RejectA = Vol(X)/2$;
  if $((lbf(X_i^m) - \overline{f^{\sim}})/\overline{F_i'}(X) < w(X_i)/2)$ **then**
    $RejectA = RejectA + (lbf(X_i^m) - \overline{f^{\sim}})/\overline{F_i'}(X) \cdot Vol(X)/w(X_i)$
  else $RejectA = RejectA + Vol(X)/2$;
Return $RejectA$

sión. Para evitar este incremento en el número de evaluaciones adicionales, que pueden ser muchas, se usa la siguiente heurística: solo se evalúa la componente $k_2 \in 1, \ldots, n$ que tiene el máximo valor de $\{\underline{F_c}(X_i^m, m(X))\},\ i = 1, \ldots, n$; es decir, $F(X_{k_2}^m)$. Basado en los estudios sobre el grado de convergencia de la forma centrada y de la extensión natural de intervalos [92, 111], $F(X_{k_2}^m)$ únicamente se evalúa si $w(F(Y_{k_2}^m)) < w(F_c(Y_{k_2}^m, m(Y)))$, $X \subset Y$. El algoritmo AMIGO tiene esto en cuenta usando el vector de flags $NaiveF(X)$ que se inicializa a True en el Algoritmo 4.3, linea 2. Las cajas heredan este vector de flags de la caja madre de la que proceden. Los valores de $NaiveF(X_{k_2}^m)$ se actualizan en el algoritmo CoordinateToSplit (Algoritmo 4.4). Si se obtiene una reducción del intervalo, CoordinateToSplit devuelve la coordenada k_2 y el valor de $F(X_{k_2}^m)$ como valor de $sp(X_{k_2}^m)$ (Algoritmo 4.4, línea 15)

3. Si las anteriores posibilidades no funcionan, ningún área será rechazada por la aplicación de la Ecuación (4.9) y la coordenada seleccionada como dirección de subdivisión (k_3) viene dada por la Regla C de Csendes y Ratzs descrita en la Sección 4.1.4 (Algoritmo 4.4, línea 16). La rutina CoordinateToSplit devuelve el valor de k_3 y $\underline{F_c}(X_{k_3}^m, m(X))$ como valor de $sp(X_{k_3}^m)$ (Algoritmo 4.4, línea 18).

4.7. Eficiencia de AMIGO frente a MIGO

Los algoritmos MIGO y AMIGO han sido numéricamente comparados tal como se ha hecho en la Sección 4.5 con los algoritmos MTIAM y MIAG. Se ha usado también un conjunto de cuarenta funciones de prueba en las que al menos 38 de ellas son las mismas que las usadas en la Sección 4.5. Puede verse de la Tabla 4.2 que el valor del SpUp es menor que uno solo para tres de las cuarenta funciones de prueba, y que en promedio (ver la penúltima fila de la Tabla 4.2) AMIGO es 1.84 veces más rápido que MIGO. De la última línea de la Tabla puede verse que el Speed-Ups total, considerando globalmente todas las funciones, es de 2.37, que es un valor muy superior al promedio de los speedup. Esto demuestra que AMIGO es mejor para problemas más complejos.

Los resultados numéricos demuestran que el algoritmo AMIGO supone una mejora computacional respecto del algoritmo MIGO y esto es debido a un aprovechamiento refinado de la información del gradiente. La mejora se obtiene como resultado de una reducción de la región de búsqueda, que se obtiene como consecuencia de un uso eficiente de toda la información disponible en cada instante. La eficacia del proceso de reducción depende, principalmente, de la calidad del límite superior del mínimo y de las sobreestimaciones en el cálculo de la derivada. En los casos en los que ambos valores son lo suficientemente exactos, la eficacia de la reducción de las regiones de búsqueda puede hacer que AMIGO sea entre 4 y 6 veces más rápido

Tabla 4.2: Comparación entre los algoritmos MIGO y AMIGO.

Name	Ref	n	Eff_1	Eff_2	$SpUp$	ϵ
Schwefel 3.1	[116]	3	874	1340	0.7	10^{-8}
Price	[31]	2	5322	6023	0.9	10^{-8}
Schwefel 3.7	[141]	2	1762	1907	0.9	10^{-8}
Shekel 10	[116]	4	1365	1424	1.0	10^{-8}
Shekel 7	[116]	4	1365	1401	1.0	10^{-8}
Schwefel 2.1 (Beale)	[141]	2	5560	5350	1.0	10^{-8}
Levy 8	[116]	3	851	788	1.1	10^{-8}
Levy 5	[116]	2	1587	1466	1.1	10^{-8}
Schwefel 2.18 (Matyas)	[141]	2	5144	4595	1.1	10^{-8}
Levy 3	[116]	2	7116	6336	1.1	10^{-8}
EX1	[24]	2	488	429	1.1	10^{-8}
Chichinadze	[31]	2	653	547	1.2	10^{-8}
Schwefel 1.2	[141]	4	27975	22341	1.3	10^{-8}
Three-Hump-Camel-Back	[30]	2	2482	1957	1.3	10^{-8}
Six-Hump-Camel-Back	[133]	2	3484	2747	1.3	10^{-8}
Branin	[116]	2	4869	3742	1.3	10^{-8}
Rosenbrock 2	[30]	3	1279	972	1.3	10^{-8}
Goldstein-Price	[116]	2	41839	30493	1.4	10^{-8}
Schwefel 2.5 (Booth)	[141]	2	1993	1376	1.4	10^{-8}
Ratz 4	[116]	2	6792	4391	1.5	10^{-8}
Hartman 6	[116]	6	20996	12998	1.6	10^{-8}
Schwefel 3.2	[141]	3	3170	1951	1.6	10^{-8}
Henriksen-Madsen 3	[54]	2	14961	8892	1.7	10^{-8}
Treccani	[30]	2	2430	1439	1.7	10^{-8}
Griewank 2	[133]	2	1952	1113	1.8	10^{-8}
Rastrigin	[133]	2	1564	867	1.8	10^{-8}
Hartman 3	[116]	3	4463	2020	2.2	10^{-8}
Henriksen-Madsen 4	[54]	3	63639	28726	2.2	10^{-8}
Simplified Rosenbrock	[30]	2	2386	780	3.1	10^{-8}
Schwefel 2.14 (Powell)	[116]	4	387176	139335	2.8	10^{-5}
Schwefel 3.1p	[116]	3	1090616	640275	1.7	10^{-4}
Ratz 5	[116]	3	917495	364215	2.5	10^{-3}
Ratz 6	[116]	5	2162657	502237	4.3	10^{-3}
Ratz 7	[116]	7	3932091	645129	6.1	10^{-3}
Schwefel 2.10 (Kowalik)	[141]	4	672219	520749	1.3	10^{-2}
Griewank 10	[133]	10	3875828	2436103	1.6	10^{-2}
Rosenbrock 10	[30]	10	2708380	1524310	1.8	10^{-2}
Neumaier 2	[99]	4	898506	449367	2.0	10^{-2}
EX2	[24]	5	1010335	261241	3.9	10^{-2}
Ratz 8	[116]	9	388997	75231	5.2	10^{-2}
Av. Val.			457066.5	192915.1	1.84	
$\sum$			18282661	7716603	2.37	

que MIGO, como por ejemplo para las funciones de Ratz.

4.8. Evaluación comparativa

Para cerrar este capitulo dedicado a los algoritmos de Optimización Global para el caso multidimensional, en esta sección se hace un resumen comparativo de los

cuatro algoritmos descritos en este capitulo. Se han recogido los resultados numéricos de las Tablas 4.1 y 4.2 que se resumen en las Tablas 4.3 y 4.4. Estas suministran información de un total de treinta y nueve funciones de prueba. La Tabla 4.3 resume los resultados correspondientes al valor de $FE + n \cdot GE$ obtenido para los cuatro algoritmos. El criterio de parada (ϵ) se ha escogido para que todos los algoritmos terminaran su ejecución en menos de una hora. No obstante, para algunas de las funciones el algoritmo AMIGO, en el tiempo establecido de una hora, puede alcanzar mayores precisiones de la solución, como por ejemplo para las funciones Schwefel 2.14 (Powell) y Ratz 7.

En la Tabla 4.3, los algoritmos están presentados en un orden que representa una complejidad creciente. Así, el algoritmo MTIAM, el más simple, esta representado en la columna cinco y es el que no incorpora ninguno de los dispositivos aceleradores diseñados en el trabajo de esta tesis. La columna sexta representa los datos correspondientes al algoritmo MIGO, que básicamente consiste en añadir a MTIAM el concepto de forma centrada como herramienta para obtener una *función de inclusión* más precisa.

En la columna séptima hemos representado los resultados numéricos del algoritmo MIAG que incluye las aportaciones teóricas descritas en este capitulo. Finalmente la última columna de la tabla contiene información sobre la evaluación de AMIGO que mejora a MIAG haciendo una estimación de la coordenada sobre la que se debe aplicar bisección para obtener un rendimiento máximo de las reglas de eliminación.

Los esfuerzos de los distintos algoritmos sobre las funciones de prueba se muestran en la Tabla 4.3. Estos resultados numéricos demuestran la eficiencia de nuestra propuesta, que para alguna función consigue reducir a la décima parte el número de evaluaciones de intervalos. En promedio, AMIGO es la mitad de costoso que MTIAM y MIGO, y aproximadamente 1.4 veces más rápido que MIAG. Esto queda más claramente reflejado en los datos representados en la Tabla 4.4, que por facilidad de lectura se han representado gráficamente en las Figuras 4.5, 4.6 y 4.7.

Tabla 4.3: Resultados experimentales de la ejecución de los cuatro algoritmos descritos en este capítulo (MTIAM, MIAG, MIGO y AMIGO).

Name	*Ref*	n	ϵ	MTIAM	MIGO	MIAG	AMIGO
Schwefel 3.1	[116]	3	10^{-8}	874	874	994	1340
Price	[31]	2	10^{-8}	5322	5322	5951	6023
Levy 5	[116]	2	10^{-8}	1587	1587	1598	1466
Shekel 10	[116]	4	10^{-8}	1365	1365	1374	1424
Schwefel 3.7	[141]	2	10^{-8}	1762	1762	1772	1907
Levy 8	[116]	3	10^{-8}	851	851	857	788
Shekel 5	[116]	4	10^{-8}	1339	1339	1348	1366
Schwefel 2.1 (Beale)	[141]	2	10^{-8}	5560	5560	5523	5350
Shekel 7	[116]	4	10^{-8}	1365	1365	1350	1401
Levy 3	[116]	2	10^{-8}	7116	7116	6979	6336
Rastrigin	[133]	2	10^{-8}	1564	1564	1496	867
Schwefel 2.5 (Booth)	[141]	2	10^{-8}	488	1993	466	1376
Henriksen-Madsen 3	[54]	2	10^{-8}	12204	14961	11575	8892
Henriksen-Madsen 4	[54]	3	10^{-8}	63693	63639	59870	28726
Treccani	[30]	2	10^{-8}	2430	2430	2227	1439
EX1	[24]	2	10^{-8}	488	488	443	429
Branin	[116]	2	10^{-8}	4869	4869	4367	3742
Chichinadze	[31]	2	10^{-8}	653	653	576	547
Griewank 2	[133]	2	10^{-8}	1952	1952	1642	1113
Schwefel 1.2	[141]	4	10^{-8}	27975	27975	22963	22341
Schwefel 3.2	[141]	3	10^{-8}	3170	3170	2484	1951
Rosenbrock 2	[30]	2	10^{-8}	1279	1279	887	972
Ratz 4	[116]	2	10^{-8}	7096	6792	4772	4391
Hartman 6	[116]	6	10^{-8}	20996	20996	13020	12998
Three-Hump-Camel-Back	[30]	2	10^{-8}	3990	2482	2138	1957
Hartman 3	[116]	3	10^{-8}	4463	4463	2046	2020
Schwefel 2.18 (Matyas)	[141]	2	10^{-8}	10944	5144	4812	4595
Six-Hump-Camel-Back	[133]	2	10^{-8}	6824	3484	2638	2747
Simplified Rosenbrock	[30]	2	10^{-8}	2386	2386	831	780
Goldstein-Price	[116]	2	10^{-8}	320969	41839	30128	30493
Schwefel 2.14 (Powell)	[116]	4	10^{-5}	387176	387176	595993	139335
Ratz 5	[116]	3	10^{-3}	917495	917495	331049	364215
Ratz 6	[116]	5	10^{-3}	2162657	2162657	468513	502237
Griewank 10	[133]	10	10^{-2}	3875828	3875828	3869704	2436103
Schwefel 2.10 (Kowalik)	[141]	4	10^{-2}	673544	672219	496155	520749
Rosenbrock 10	[30]	10	10^{-2}	2708380	2708380	2045727	1524310
EX2	[24]	5	10^{-2}	1016177	1010335	256975	261241
Ratz 8	[116]	9	10^{-2}	388997	388997	71367	75231
Neumaier 2	[99]	4	10^{-2}	898506	898506	460443	449367
Av. Val.				347547.0	340033.2	225462.9	164937.6
$\sum$				13554334	13261293	8793053	6432565

Tabla 4.4: Valores del Speedup de los cuatro algoritmos (MTIAM, MIAG, MIGO y AMIGO).

Name	$\frac{MTIAM}{MIGO}$	$\frac{MTIAM}{MIAG}$	$\frac{MTIAM}{AMIGO}$	$\frac{MIGO}{MIAG}$	$\frac{MIGO}{AMIGO}$	$\frac{MIAG}{AMIGO}$
Schwefel 3.1	1.0	0.9	0.7	0.9	0.7	0.7
Price	1.0	0.9	0.9	0.9	0.9	1.0
Levy 5	1.0	1.0	1.1	1.0	1.1	1.1
Shekel 10	1.0	1.0	1.0	1.0	1.0	1.0
Schwefel 3.7	1.0	1.0	0.9	1.0	0.9	0.9
Levy 8	1.0	1.0	1.1	1.0	1.1	1.1
Shekel 5	1.0	1.0	1.0	1.0	1.0	1.0
Schwefel 2.1 (Beale)	1.0	1.0	1.0	1.0	1.0	1.0
Shekel 7	1.0	1.0	1.0	1.0	1.0	1.0
Levy 3	1.0	1.0	1.1	1.0	1.1	1.1
Rastrigin	1.0	1.0	1.8	1.0	1.8	1.7
Schwefel 2.5 (Booth)	0.2	1.0	0.4	4.3	1.4	0.3
Henriksen-Madsen 3	0.8	1.1	1.4	1.3	1.7	1.3
Henriksen-Madsen 4	1.0	1.1	2.2	1.1	2.2	2.1
Treccani	1.0	1.1	1.7	1.1	1.7	1.5
EX1	1.0	1.1	1.1	1.1	1.1	1.0
Branin	1.0	1.1	1.3	1.1	1.3	1.2
Chichinadze	1.0	1.1	1.2	1.1	1.2	1.1
Griewank 2	1.0	1.2	1.8	1.2	1.8	1.5
Schwefel 1.2	1.0	1.2	1.3	1.2	1.3	1.0
Schwefel 3.2	1.0	1.3	1.6	1.3	1.6	1.3
Rosenbrock 2	1.0	1.4	1.3	1.4	1.3	0.9
Ratz 4	1.0	1.5	1.6	1.4	1.5	1.1
Hartman 6	1.0	1.6	1.6	1.6	1.6	1.0
Three-Hump-Camel-Back	1.6	1.9	2.0	1.2	1.3	1.1
Hartman 3	1.0	2.2	2.2	2.2	2.2	1.0
Schwefel 2.18 (Matyas)	2.1	2.3	2.4	1.1	1.1	1.0
Six-Hump-Camel-Back	2.0	2.6	2.5	1.3	1.3	1.0
Simplified Rosenbrock	1.0	2.9	3.1	2.9	3.1	1.1
Goldstein-Price	7.7	10.7	10.5	1.4	1.4	1.0
Schwefel 2.14 (Powell)	1.0	0.6	2.8	0.6	2.8	4.3
Ratz 5	1.0	2.8	2.5	2.8	2.5	0.9
Ratz 6	1.0	4.6	4.3	4.6	4.3	0.9
Griewank 10	1.0	1.0	1.6	1.0	1.6	1.6
Schwefel 2.10 (Kowalik)	1.0	1.4	1.3	1.4	1.3	1.0
Rosenbrock 10	1.0	1.3	1.8	1.3	1.8	1.3
EX2	1.0	4.0	3.9	3.9	3.9	1.0
Ratz 8	1.0	5.5	5.2	5.5	5.2	0.9
Neumaier 2	1.0	2.0	2.0	2.0	2.0	1.0
Av. Val.	1.2	1.9	2.0	1.6	1.7	1.2
$\sum$	1.0	1.5	2.1	1.5	2.1	1.4

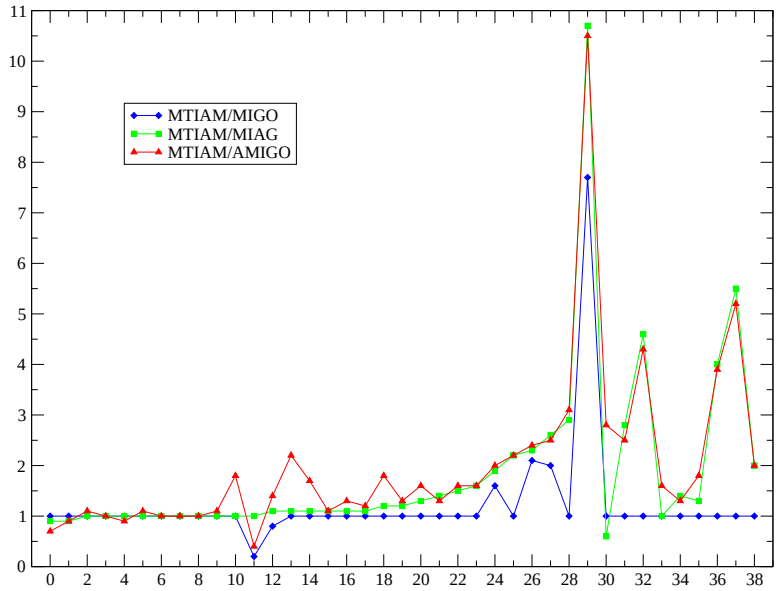


Figura 4.5: Representación gráfica de la aceleración de AMIGO, MIAG y MIGO respecto de MTIAM.

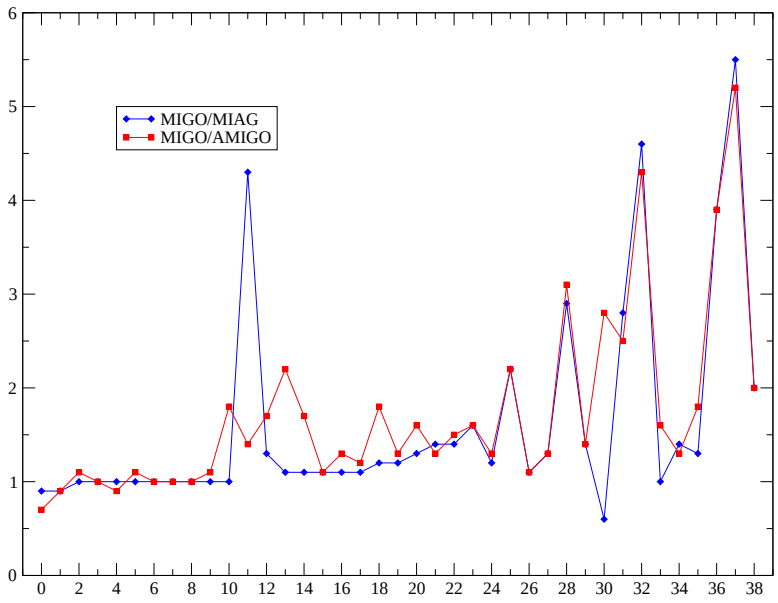


Figura 4.6: Representación gráfica de la aceleración de AMIGO y MIAG respecto de MIGO.

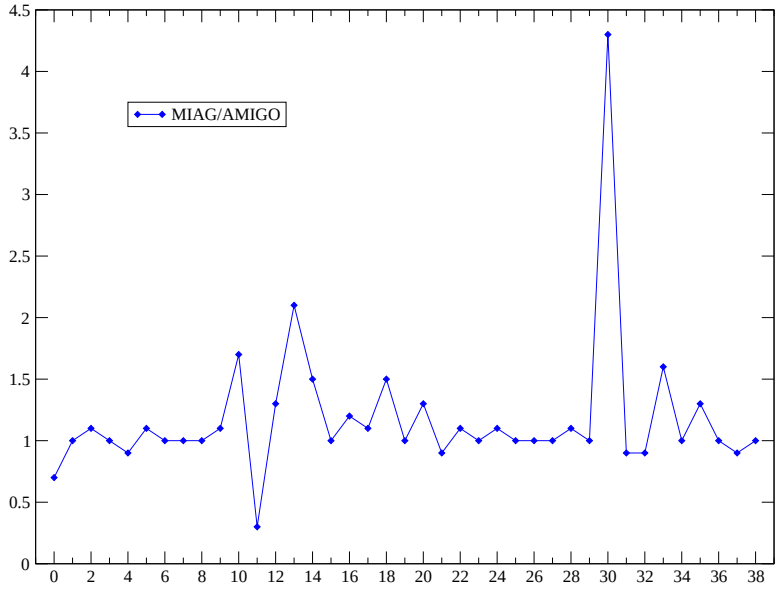


Figura 4.7: Representación gráfica de la aceleración de AMIGO respecto de MIAG.

Capítulo 5

Computación de altas prestaciones en Branch and Bound

En este capítulo se analizan y evalúan las aportaciones de esta tesis en relación con la computación de altas prestaciones. Como se ha demostrado en los capítulos previos, dentro del campo de la Optimización Global existen problemas cuya complejidad computacional desborda la potencia de un solo procesador. Muchos de los problemas de Optimización Global que encontramos en el mundo de la industria, la economía o, en general la ingeniería, son NP completos. Hablando de una forma simple, ello implica que cuando la precisión con la que se quiere obtener la solución, o cuando la dimensión del problema crece, el tiempo de procesamiento requerido crece exponencialmente. En el Capítulo4 se han evaluado los algoritmos de Optimización Global propuestos en esta tesis, usando un amplio conjunto de funciones de prueba y una arquitectura uniprocesador. Se ha comprobado que existe un subconjunto de las funciones para las que solo se han podido obtener soluciones con muy poca precisión. Este capítulo traslada las técnicas de Optimización Global basadas en Branch and Bound y Aritmética de Intervalos, desde el espacio secuencial hacia una perspectiva paralela.

5.1. Arquitecturas Paralelas

La computación paralela es un modelo eficiente para reducir los tiempos de cálculo de aplicaciones que tienen una elevada carga computacional. Los algoritmos paralelos ejecutados sobre una arquitectura paralela obtienen un mejor rendimiento

cuando:

- Se tiene un conocimiento profundo de la aplicación.
- El programador es capaz de explotar el paralelismo de la aplicación, especialmente el paralelismo de datos.
- Se tiene un buen conocimiento de la arquitectura sobre la que se trabaja.

Un computador paralelo está compuesto por un número de procesadores, p, que colaboran en la resolución de un problema. El usuario espera que el tiempo de ejecución obtenido en un computador paralelo con p procesadores sea $1/p$ veces el tiempo de respuesta usando un solo procesador. Esta aceleración o ganancia de velocidad (Speed-Up) no se alcanza en todos los casos, ya que la administración de los procesadores, así como la comunicación (o compartición) de datos entre los mismos, consumen tiempo.

En esta sección se describen las arquitecturas paralelas más usadas actualmente, es decir, la arquitectura MIMD (Multiple Instruction Multiple Data). Las arquitecturas MIMD pueden ser a su vez divididas en dos grandes grupos: arquitecturas de memoria compartida y arquitecturas de memoria distribuida.

5.1.1. Arquitecturas de memoria compartida

En esta arquitectura, los procesadores del computador paralelo comparten un único espacio de direcciones de almacenamiento que es accesible por todos los procesadores mediante una red de interconexión. Esta arquitectura puede verse como una extensión natural de las arquitecturas monoprocesador convencionales, que nos proporcionan una visión única del sistema. Como consecuencia, la principal ventaja de esta arquitectura es su sencillez en la programación: la comunicación entre procesadores se lleva a cabo leyendo o escribiendo en alguna dirección de memoria. Esto nos lleva a que el sistema debe implementar técnicas que nos permitan asegurar el acceso exclusivo de los procesadores a memoria, sobre todo a la hora de escribir, ya que podrían aparecer problemas de inconsistencia de los datos. Estos mecanismos de contención de datos pueden provocar un efecto de *cuello de botella*, sobre todo cuando muchos procesadores intentan acceder concurrentemente a la misma posición de memoria.

El principal problema que presentan estas arquitecturas frente a las arquitecturas de memoria distribuida es su escasa escalabilidad. Para incrementar la escalabilidad, se han usado varias alternativas:

- Incorporar caches en el sistema de memoria, de forma que cada procesador disponga de una cache local.

- Usar redes de interconexión de menor latencia y mayor ancho de banda que sustituyan al bus.
- Distribuir físicamente los módulos de memoria entre los procesadores.

Estas alternativas nos llevan a definir nuevas subarquitecturas dentro de estas arquitecturas.

Arquitecturas de procesamiento simétrico

La principal característica de las arquitecturas de procesamiento simétrico SMP, es que el tiempo de acceso a memoria es el mismo para cada procesador, independientemente de la posición física donde se encuentre, por esta circunstancia, estas arquitecturas también reciben el nombre de arquitecturas UMA (*Uniform Memory Access*).

Las arquitecturas que usan un bus como red de interconexión tienen una mayor relación coste-eficiencia para un número bajo de procesadores, por ejemplo, entre 8 y 16 procesadores. Mientras que sistemas que usan redes de interconexión más complejas, pueden escalar hasta un mayor número de procesadores. Algunos ejemplos de estas arquitecturas son: Los nuevos servidores biprocesador y cuatriprocesador que nos ofrecen las principales marcas comerciales, y los novedosos procesadores multicore de Intel y AMD ; un ejemplo de arquitectura con una red de interconexión más compleja que el bus y que llegan a escalar hasta un centenar de procesadores es la SUN Fire 15K, que soporta hasta 106 procesadores UltraSparc III a 1,05 Ghz.[88].

Arquitecturas de memoria compartida-distribuida

Las arquitecturas de memoria compartida-distribuida, se caracterizan por el hecho de que el tiempo de acceso a la memoria no es el mismo para todos los procesadores, sino que depende de la posición física donde se encuentre. Por esta característica, se denominan arquitecturas NUMA (*Non Uniform Memory Access*).

Las arquitecturas NUMA se componen de módulos que agrupan un conjunto de recursos de computación: Procesadores, memoria y slots de entrada/salida.

Varios de estos módulos se pueden unir a través de un dispositivo de comunicación que realiza la interconexión. De esta manera, un procesador puede acceder muy rápidamente a la memoria que está en el mismo módulo. Si por el contrario tiene que acceder a memoria situada en otro módulo será necesario atravesar el dispositivo de comunicación, con lo cual se requiere un mayor tiempo de acceso. Este es el motivo de la asimetría en el tiempo de acceso a memoria.

La principal ventaja con respecto a las arquitecturas UMA es que son más fácilmente escalables, llegando a varios cientos de procesadores en un único computador. Ejemplos de estas arquitecturas son: Standford DASH con 64 procesadores MIPS R3000/3010 a 33 Mhz; todas las series Origin de Silicon Graphics, con hasta 512 procesadores MPIS R12000 a 400 Mhz.

Entre las máquinas tipo NUMA, existe una característica muy importante derivada del hecho de que existen distintos procesadores con sus respectivas memorias cache. Esto propicia la posibilidad de que exista más de una copia de una misma variable en distintas memorias cache de distintos procesadores que estén trabajando en paralelo sobre datos compartidos. Cuando existen diversas copias hay que garantizar que si cualquiera de ellas es modificada por uno de los procesadores, las demás copias serán actualizadas convenientemente.

Los sistemas multiprocesador pueden incluir la lógica necesaria y los protocolos adecuados para garantizar la coherencia de las distintas copias del mismo dato que puedan existir simultáneamente en el sistema. Los sistemas NUMA que presentan esta característica son denominados cc-NUMA (caché-coherent Non Uniform Memory Access).

Una forma sencilla de implementar la coherencia de caché consiste en monitorizar el tráfico de las conexiones de acceso a memoria y establecer para cada línea de la caché una señal de estado que indica en qué situación se encuentra. Suele utilizarse un protocolo de tipo snoopy. Esto implica la necesidad de un cierto hardware específico que mantenga los estados de las líneas de caché según las modificaciones que sufran los datos almacenados en ellas y que se encargue de actualizar las copias de los datos que hayan sido modificados por otros procesadores. Unos mensajes internos son generados automáticamente para restituir las líneas de caché cuando los datos que contienen son modificados y enviados a los nodos del sistema.

Ejemplos de estas últimas arquitecturas son: La SICS Simple COMA [134], la Illinois I-ACOMA y la nueva serie ALTIX 3000 de Silicon Graphics que pueden escalar hasta varios centenares de procesadores Itanium II a 1.5 Ghz. [46]

5.1.2. Arquitecturas de memoria distribuida

Muchos de los computadores paralelos de hoy en día usan memoria distribuida. Cada procesador tiene su propio espacio de direcciones de memoria local, a la cual no pueden acceder directamente otros procesadores del computador. Las ventajas de este modelo frente al de memoria compartida son:

- Cada procesador puede usar todo el ancho de banda para el acceso a su memoria local.

- Es más escalable; el tamaño del sistema sólo está limitado por el tipo de red de interconexión utilizada para conectar los procesadores entre sí.
- No hay problemas de coherencia de cache.

La programación de este tipo de sistemas es más costosa, ya que el intercambio de datos entre procesadores se hace mediante el paso de mensajes a través de una red de interconexión. Debido a esta característica, estas arquitecturas también se denominan arquitecturas de paso de mensajes. En el siguiente apartado trataremos con mas detalle algunos aspectos de la programación de este tipo de arquitectura.

Ejemplos de plataformas de memoria distribuida son el Cray T3E, Mare Nostrum, IBM eServer Blue Gene y los cluster de estaciones de trabajo (COW).

Es fácil simular una arquitectura de paso de mensajes de p procesadores en una arquitectura de memoria compartida con idéntico número de procesadores. Esto se puede llevar a cabo particionando el espacio de direcciones en p conjuntos disjuntos y asignando cada conjunto a cada uno de los p procesadores de manera exclusiva. Un procesador puede entonces *enviar* y *recibir* mensajes escribiendo o leyendo de una partición de un procesador en otra partición. En este caso se usarán primitivas de sincronización que informan a los procesadores cuando han finalizado las operaciones de lectura y escritura de datos. Simular una arquitectura de memoria compartida con una arquitectura de paso de mensajes, supone acceder al espacio de memoria de un procesador remoto mediante el envío y recepción de un mensaje.

5.2. Modelos de programación para multicomputadores

Para obtener un alto rendimiento de la implementación de un determinado algoritmo sobre una arquitectura multicomputador, el modelo de programación usado debe ser elegido de forma que tenga en cuenta las características del hardware utilizado. A groso modo se pueden distinguir dos modelos de programación, que están directamente relacionados con los dos grandes modelos de arquitecturas paralelas:

- Modelo de programación por paso de mensajes: arquitecturas de memoria distribuida.
- Modelo de programación de variables compartidas: arquitecturas de memoria compartida.

Con el modelo de paso de mensajes suponemos que cada procesador en el sistema tiene su propio espacio de direcciones. Los mensajes llevan datos de uno a otro espacio de direcciones y además se pueden usar para sincronizar los procesos. Los

datos transferidos están duplicados en el sistema de memoria. Con el modelo de programación de variables compartidas, se supone que los procesadores del sistema comparten el mismo espacio de direcciones. Al compartir este espacio, no se necesita transferir datos explícitamente, la transferencia se realiza utilizando instrucciones de lectura y escritura en memoria.

5.2.1. Programación usando paso de mensajes

Para escribir un programa usando un modelo de paso de mensajes, se usan principalmente librerías que implementan el interfaz de comunicaciones que llevan a cabo el paso de mensajes, como MPI [126] o PVM [42]. Las funciones básicas de comunicación que ofrecen las librerías han de permitir al menos, comunicación entre dos procesos.

Generalmente, existe una función de envío donde se especifica el proceso destino y la información a enviar. En la función de recepción se especifica el proceso fuente de la información y la variable donde se almacena la información recibida. Nos podemos encontrar por lo general con transmisiones síncronas y asíncronas. Las primeras son bloqueantes; el proceso que ejecuta la instrucción de envío se bloquea hasta que el proceso correspondiente ejecute la instrucción de recepción; igualmente el proceso que ejecuta una instrucción de recepción se bloquea hasta que el proceso correspondiente no realice una instrucción de envío. En una transmisión asíncrona, la operación de envío, y la operación de recepción de información, no deja bloqueado al proceso que las ejecuta; para llevar a cabo estas operaciones es necesario un buffer donde se colocan los datos antes de llegar al destino o donde se almacenarán cuando lleguen al mismo.

5.2.2. Programación usando variables compartidas

Hay diversas herramientas para llevar a cabo la implementación de programas que usen el modelo de variables compartidas. El modelo más costoso, es el modelo de procesos concurrentes; en este modelo la comunicación entre procesos se lleva a cabo reservando y definiendo, en tiempo de ejecución, segmentos de datos que son compartidos por los distintos procesos. La escritura y lectura de datos de estos segmentos debe ser realizada de manera exclusiva, es decir, un único proceso puede acceder en un instante dado a los datos del segmento. Por otro lado tenemos los modelos de threads: los más comunes son la librería de funciones POSIX-Threads (pthread) [127] y la librería OpenMP [143]; en este caso, la comunicación entre threads se realiza accediendo a variables compartidas, por tanto, mediante instrucciones de lectura y escritura en memoria. Los threads de un mismo proceso creados por el sistema operativo pueden compartir variables globales. Esta es la principal

ventajas de usar un modelo de threads frente a un modelo de procesos para programar las arquitecturas de memoria compartida.

El interfaz de programación de aplicaciones OpenMP consiste en un conjunto de directivas del compilador, librerías de funciones y variables de entorno que ayudan a la implementación paralela de los algoritmos. Los elementos clave de OpenMP son:

- Los constructores para la creación de threads, que son usados para crear threads adicionales.
- Los constructores de balanceo de la carga, que son usados para especificar cuando asignar trabajo independiente a uno o a todos los threads.
- Los constructores para mantener datos compartidos; ya que OpenMP es usado para implementar algoritmos en arquitecturas de memoria compartida, la mayoría de las variables son compartidas por todos los threads, pero algunas veces son necesarias variables privadas y es necesario pasarlas entre las partes del código secuencial y las partes paralelas.
- Los constructores de estructuras de datos para realizar la sincronización de los threads.
- Las rutinas para que el usuario sepa en tiempo de ejecución ciertas características del sistema. Por ejemplo, para saber el número de threads en ejecución en un instante dado o saber cuantos procesadores hay disponibles en el sistema actualmente.
- Las variables de entorno, que son un método para alterar las características de ejecución de las aplicaciones OpenMP: son usadas por ejemplo para controlar las iteraciones de los bucles o el número de threads por defecto.

Las principales ventajas de usar OpenMP para implementar programas paralelos son:

- Sencillo de usar.
- La descomposición de datos se lleva a cabo de manera automática mediante directivas.
- Paralelismo incremental: Puede ser aplicado solo en una parte del programa a la vez, no es necesario realizar grandes cambios en el código secuencial.
- Se usa el mismo código tanto para la aplicación serie como para la paralela: Los constructores de OpenMP son tratados como comentarios cuando se compila en modo secuencial.

Los principales inconvenientes de OpenMP son:

- Baja eficiencia del paralelismo: Realmente solo son paralelizables los bucles, dejando fuera un alto porcentaje de código secuencial que puede ser potencialmente paralelizable.
- No se pueden crear threads de manera dinámica en tiempo de ejecución. Esto nos lleva a tener que usar estrategias de balanceo estático de la carga de los procesadores.

Estos dos últimos inconvenientes son los principales para habernos decidido por una librería de funciones POSIX-Threads para implementar los algoritmos presentados en esta tesis. Si queremos realizar una implementación paralela realmente eficiente de los algoritmos de Branch and Bound necesitamos paralelizar todas las regiones de nuestro código que lo permitan, no solo los bucles. Además, el tamaño del árbol de búsqueda de nuestros programas no es constante, es decir, en nuestro árbol de búsqueda el número de nodos puede aumentar o disminuir en diferentes instantes de la ejecución; luego necesitamos una herramienta que permita usar un mayor o menor número de threads durante la ejecución, dependiendo del número de nodos activos en el árbol de búsqueda. En el siguiente apartado se dan las claves necesarias para la programación usando el modelo de threads. En la Subsección 5.3.3 se especifica concretamente como se implementa el modelo de threads en el estándar POSIX.

5.3. Programación usando threads

Un *thread* de ejecución, en el contexto del sistema operativo del computador, es similar a un proceso: ambos representan una secuencia simple de instrucciones ejecutada en paralelo con otras secuencias. El termino *multithread* hace referencia a la capacidad del sistema operativo para mantener varios *threads* de ejecución dentro del mismo proceso. Un ejemplo de sistema operativo que soporta un solo proceso de usuario y un solo thread de ejecución es el MS-DOS. Otros sistemas operativos, como las primeras versiones de UNIX, soportan múltiples procesos de usuario, pero solo un thread de ejecución por proceso. Un ejemplo de sistema con un proceso y múltiples threads de ejecución es un entorno de ejecución Java. También es interesante considerar el uso de múltiples procesos, cada uno de ellos formado por múltiples threads de ejecución; esta es la solución de que nos aportan las nuevas versiones de UNIX , Linux y Solaris.

En un entorno multithread, un proceso se define como la unidad de asignación de recursos. A cada proceso se les asocian los siguientes elementos:

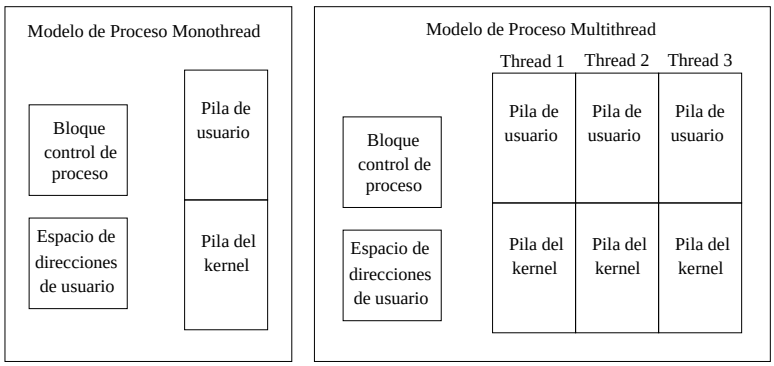


Figura 5.1: Modelo de proceso monothread y multithread.

- Un espacio de direcciones virtuales, que contienen la imagen del proceso.
- Acceso protegido a los procesadores.
- Archivos y recursos de E/S.

En un proceso, puede haber uno o más threads, cada uno con:

- El estado de ejecución del thread.
- El contexto del procesador, que se salva cuando no se está ejecutando; una forma de ver el thread es como un contador de programa independiente operando dentro de un proceso.
- Una pila de ejecución.
- Almacenamiento estático para las variables locales.
- Acceso a la memoria y a los recursos del proceso compartidos por todos los otros threads del mismo proceso.

La Figura 5.1 muestra las diferencias entre procesos y threads desde el punto de vista de la gestión de procesos. En un modelo de proceso monothread, donde no hay un concepto diferenciado entre thread y proceso, la representación de un proceso incluye su bloque de control de proceso y un espacio de direcciones de usuario, así como las pilas de usuario y *kernel* para gestionar la secuencia de llamadas-retornos durante la ejecución del proceso. Mientras se está ejecutando el proceso, los registros del procesador están controlados por ese proceso, y cuando el proceso no se está ejecutando, se salva el contenido de esos registros. En un entorno multithread,

continúa existiendo un solo bloque de control de proceso y un espacio de direcciones de usuario asociadas al proceso, pero ahora hay pilas separadas para cada thread, así como distintos bloques de control para cada thread, que contienen los valores de los registros, prioridad y otras informaciones relativas al estado de los threads.

Así pues, todos los threads de un proceso comparten el estado de los recursos del proceso, residen dentro del mismo espacio de direcciones y tienen acceso a los mismos datos. Cuando un thread modifica un dato de memoria, los otros threads usan el resultado cuando acceden al dato. Si un thread abre un archivo con permisos de lectura, el resto de los threads del mismo proceso también pueden acceder para leer de él.

Las ventajas de los threads frente al modelo de procesos procede de los siguientes aspectos parciales:

- Se tarda mucho menos tiempo en crear un thread en un proceso existente que en crear un nuevo proceso.
- Se tarda mucho menos tiempo en terminar un thread que un proceso.
- Se tarda mucho menos tiempo en cambiar la ejecución entre threads de un mismo proceso que entre procesos.
- Los threads aumentan la eficiencia de la comunicación entre programas en ejecución. En la mayoría de los sistemas operativos, la comunicación entre procesos independientes requiere la intervención del *kernel* para ofrecer protección y para proporcionar los mecanismos necesarios para la comunicación, Sin embargo, puesto que los threads de un mismo proceso comparten memoria y archivos, pueden comunicarse entre si sin invocar al *kernel*.

Por lo tanto si hay una aplicación o una función que pueda implementarse como un conjunto de unidades de ejecución relacionadas, es más eficiente hacerlo con una colección de threads que con una colección de procesos separados.

5.3.1. Estado y sincronización de los threads

Como en los procesos, los principales estados de un thread son: Running, ready y stopped. No tiene sentido asociar estados de suspensión a los threads porque esos estados pertenecen al concepto de proceso. En concreto, si un proceso está expulsado de la memoria principal, todos los threads del proceso deben también estar expulsados, porque todos comparten el espacio de direcciones del proceso.

Hay cuatro operaciones básicas relacionadas con el cambio de estado de los threads:

- Creación: Normalmente, cuando se crea un nuevo proceso, también se crea un thread para ese proceso. Posteriormente, un thread de un proceso puede crear otros threads dentro del mismo proceso, proporcionando un puntero de instrucción y los argumentos del nuevo thread. El nuevo thread tendrá su propio contexto y su propio espacio de pila, y pasara a la cola de ready.
- Bloqueo: Cuando un thread necesita esperar a un evento, éste pasa al estado de stopped (salvando sus registros de usuario, contador de programa y punteros de pila). Ahora, el procesador podrá pasar a ejecutar otro thread ready.
- Desbloqueo: Cuando se produce el evento por el que un thread pasó al estado de stopped, el thread pasa a la cola de ready.
- Terminación: Cuando un thread finaliza, se libera su contexto y sus pilas.

En cuanto a la sincronización, decir que todos los threads de un proceso comparten el mismo espacio de direcciones y otros recursos, como los archivos abiertos. Cualquier modificación de un recurso desde un thread afecta al entorno del resto de los threads del mismo proceso. Por lo tanto, es necesario sincronizar la actividad de los distintos threads para que no interfieran unos con otros o corrompan las estructuras de datos. Por ejemplo, si dos threads intentan añadir cada uno un elemento a una lista doblemente enlazada, se podría perder un elemento de la lista o la lista podría acabar en un estado inconsistente.

5.3.2. Threads ULT, KLT y combinados

Existen dos grandes categorías para la implementación de threads: Los threads a nivel de usuario (ULT (User Level Threads)) y los threads a nivel de kernel (KLT (Kernel Level Threads)). A estos últimos también se les denomina threads soportados por el kernel o procesos ligeros (LWP (Light Weight Process)). También existe una aproximación combinada que aporta los beneficios de las dos implementaciones.

Threads a nivel de usuario

En una aplicación ULT pura, todo el trabajo de gestión de los threads lo realiza la aplicación y el kernel no es consciente de la existencia de threads. La Figura 5.2 muestra una solución ULT pura. Es posible programar cualquier aplicación como multithread mediante una librería de threads, que es un paquete de funciones para gestionar ULT. La librería de threads contiene un código para crear o destruir threads, para intercambiar mensajes y datos entre threads, para planificar la ejecución de los threads y para salvar y restaurar el contexto de los threads.

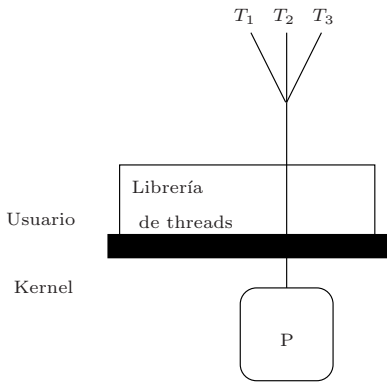


Figura 5.2: Modelo de threads a nivel de usuario puro. T1, T2 y T3 representan los threads a nivel de usuario.

Por defecto, una aplicación comienza su ejecución con un thread único. Esta aplicación y su thread pertenecen a un único proceso, gestionado por el kernel. En cualquier instante de la ejecución de la aplicación, ésta puede crear un nuevo thread que se ejecuta dentro del mismo proceso. La creación se lleva a cabo ejecutando la correspondiente función de la librería de threads. El flujo de control pasa a esa función mediante una llamada a procedimiento. La librería de threads, crea una estructura de datos para el nuevo thread y cede el control a uno de los threads del proceso que están en el estado ready, elegido mediante algún algoritmo de planificación. Cuando el control pasa a la librería, se salva el contexto del thread actual que se restaura cuando el control pasa de la librería al thread.

Todas las operaciones descritas anteriormente, se llevan a cabo en el espacio de usuario dentro de un mismo proceso. El kernel no tiene conocimiento de ellas. El kernel continua planificando el proceso como una unidad y asignándole un único estado.

Son varias las ventajas de usar ULT frente a KLT :

- El intercambio de threads no necesita de los privilegios del modo kernel. El proceso no debe cambiar a modo kernel para gestionar los threads; con ello se evitan las sobrecargas de los cambios de modo (modo kernel y modo usuario).
- Se puede realizar una planificación específica. Para una aplicación puede ser mejor la planificación mediante turno rotatorio mientras que para otra puede ser mejor la planificación por propiedades. Se puede realizar una planificación a medida de la aplicación sin afectar a la planificación subyacente del sistema operativo.

- Los ULT se pueden ejecutar en cualquier sistema operativo. Para dar soporte a los ULT no es necesario realizar cambios en el kernel subyacente. La librería de threads es un conjunto de funciones compartidas por todas las aplicaciones.

Existen dos desventajas en el uso de ULT frente a KLT :

- En un sistema operativo, la mayoría de las llamadas al sistema son bloqueantes. Así pues, cuando un ULT ejecuta una llamada al sistema no sólo bloquea ese thread, sino todos los threads del proceso.
- En una estrategia ULT pura, una aplicación multithread no puede aprovechar las ventajas de los multiprocesadores. El kernel asigna un proceso a un sólo procesador cada vez. Por lo tanto, sólo puede ejecutar un thread de cada proceso en cada instante. De hecho, se realiza un multithreading a nivel de aplicación dentro de un proceso.

Existen formas de evitar estos problemas. Por ejemplo, ambos se pueden superar escribiendo una aplicación en forma de múltiples procesos en lugar de múltiples threads. Pero esta solución elimina la principal ventaja de los threads: Cada cambio de proceso es más costoso que un cambio de thread, lo que produce una sobrecarga. Otra forma de superar el problema de bloqueo de los threads es mediante una técnica denominada *jacketing*; esta consiste en convertir una llamada bloqueante del sistema en otra no bloqueante.

Threads a nivel de kernel

En una aplicación KLT pura, todo el trabajo de gestión de los threads lo realiza el kernel. En el entorno de la aplicación no hay código para la gestión de los threads, únicamente un interfaz de programación de aplicación (API).

La Figura 5.3 representa una solución KLT pura. Se puede programar cualquier aplicación como multithread. Todos los threads de la misma aplicación pertenecen a un único proceso. El kernel mantiene la información de contexto del proceso como un todo y la de cada thread dentro del proceso. El kernel realiza la planificación en función de los threads. Este método resuelve los dos principales inconvenientes de la solución ULT. En primer lugar, el kernel puede planificar simultáneamente múltiples threads del mismo proceso en múltiples procesadores. En segundo lugar, si se bloquea uno de los threads de un proceso, el kernel puede planificar otro thread del mismo proceso. Otra ventaja de la solución KLT es que las propias funciones del kernel pueden ser multithread.

La principal desventaja de la solución KLT comparada con la solución ULT es

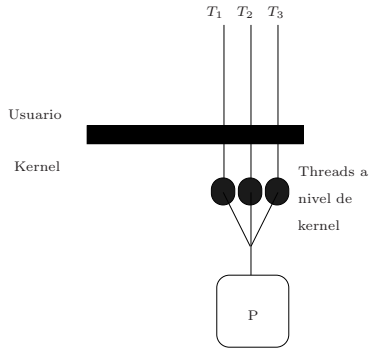


Figura 5.3: Modelo de threads a nivel de kernel puro. T1, T2 y T3 representan los threads a nivel de usuario y están correlacionados con threads del kernel.

que el paso de control de un thread a otro, dentro del mismo proceso, necesita un cambio de modo del kernel, con el consiguiente aumento de la latencia.

Aproximaciones combinadas

Algunos sistema operativos ofrecen un mecanismo que combina ULT y KLT (ver Figura 5.4). En un sistema combinado, la creación de threads, así como la mayor parte de la planificación y sincronización de los threads de una aplicación se realiza por completo en el espacio del usuario. Los múltiples ULTs de una sola aplicación se asocian con varios (el mismo o menor número) KLTs. El programador puede ajustar el número de KLTs para cada aplicación y máquina para obtener el mejor rendimiento.

En un método combinado, los múltiples threads de una aplicación se pueden ejecutar en paralelo en múltiples procesadores y las llamadas bloqueantes al sistema no necesitan parar todo el proceso. Si está correctamente diseñado, este enfoque puede combinar las ventajas de las soluciones ULT y KLT puras.

5.3.3. Threads POSIX

Los threads POSIX implementan un soporte de threads multinivel diseñado para ofrecer una buena flexibilidad en la explotación de los recursos del procesador. Los threads POSIX hacen uso de cuatro conceptos independientes relativos a los threads:

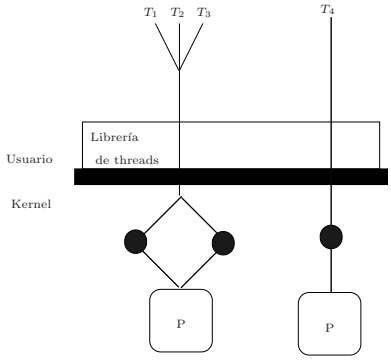


Figura 5.4: Modelo de threads combinado. Los threads de usuario T1, T2 y T3 están mapeados sobre dos threads del kernel. El thread T4 está mapeado sobre un solo thread del kernel.

- Proceso: Es el proceso de UNIX convencional que incluye el espacio de direcciones de usuario, la pila y el bloque de control de proceso.
- Threads a nivel de usuario: Implementados en el espacio de direcciones de un proceso por medio de una biblioteca de threads (pthreads), estos son invisibles para el sistema operativo. Los threads a nivel de usuario (ULT) son la interfaz para el paralelismo de aplicaciones.
- Procesos ligeros: Un proceso ligero (LWP) puede verse como una correspondencia entre ULT y threads del kernel. Cada LWP soporta uno o más ULTs y los hace corresponder con un thread del kernel. El kernel planifica los LWPs independientemente y pueden ejecutarse en paralelo sobre multiprocesadores.
- Threads del kernel: Son las entidades básicas de planificación y expedición en cada uno de los procesadores del sistema.

La Figura 5.5 muestra la relación entre estas cuatro entidades; hay un thread del kernel por cada LWP. Un LWP es visible para la aplicación dentro del proceso. De este modo, las estructuras de datos LWP existen dentro de los respectivos espacios de direcciones de los procesos. Al mismo tiempo, cada LWP está mapeado en un único thread del kernel y la estructura de datos de este thread del kernel se mantiene dentro del espacio de direcciones del kernel. En este ejemplo, el Proceso 1 está formado por un thread único ULT unido a un sólo LWP. Así pues, hay un sólo thread de ejecución, lo que se corresponde con un proceso UNIX clásico. El Proceso 2, se corresponde con una estrategia ULT pura; un único LWP soporta todos los ULT del proceso, por lo tanto sólo un ULT puede ejecutarse en cada instante.

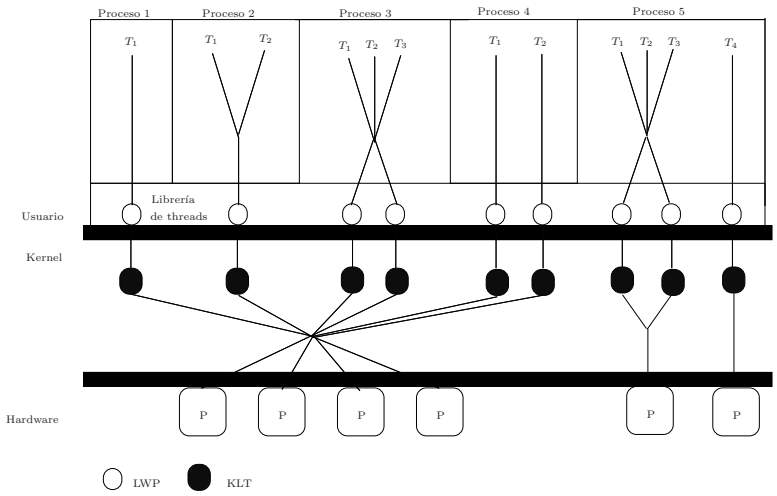


Figura 5.5: Ejemplo de una arquitectura multithread implementada con threads POSIX.

Esta estructura es útil en aplicaciones que pueden programarse por métodos concurrentes pero que no necesitan de la ejecución de múltiples threads en paralelo. El Proceso 3 muestra múltiples threads de ejecución multiplexados sobre un número menor número de LWP. El Proceso 4, tiene cada uno de sus threads unidos con un LWP; esta estructura hace que el paralelismo a nivel de kernel sea visible para la aplicación. El Proceso 5, muestra tanto una correspondencia de múltiples ULTs sobre múltiples LWPs como la unión de un ULT con un LWP; además, un LWP está ligado a un procesador concreto. En la Figura 5.5 no se muestra la presencia de threads del kernel que no estén asociados con un LWP. El kernel, crea, ejecuta y destruye esos threads para realizar funciones específicas del sistema. El uso de threads en vez de procesos para implementar las funciones del sistema reduce la sobrecarga de intercambio dentro del kernel.

5.4. Medidas de rendimiento de algoritmos paralelos

El uso de un computador paralelo para la resolución de un problema numérico sólo tiene sentido, si el tiempo total de resolución del problema en el computador paralelo se reduce significativamente, en comparación con un computador secuencial. Sería deseable que el tiempo secuencial del algoritmo se reduzca alrededor de un factor p, si se usan p procesadores para resolverlo en paralelo. Una medida de la aceleración producida por el algoritmo paralelo, frente al secuencial, se denota por

la ganancia de velocidad o Speed-Up, que se define como:

$$S(p) = \frac{t_{sec}}{t_{par}(p)},$$

donde p es el número de procesadores, t_{sec} es el tiempo total de resolución del algoritmo en un computador secuencial y $t_{par}(p)$ es el tiempo total requerido para resolverlo en un computador paralelo con p procesadores. La relación entre la ganancia en velocidad y el número de procesadores, se conoce como la eficiencia:

$$E(p) = \frac{S(p)}{p}.$$

Generalmente se cumple que $1 \leq S(p) \leq p$. En algunos casos, se consigue una ganancia de velocidad superlineal con $S(p) > p$. Para el problema de Branch and Bound que tratamos en esta tesis, esto puede deberse a que el algoritmo serie examina más subproblemas que en el algoritmo paralelo, lo que nos hace pensar que el algoritmo serie puede ser mejorado. Otra de las causas por las que es posible obtener un Speed-Up superlineal se debe a que en la mayoría de los computadores paralelos cada procesador tiene una pequeña memoria (cache) rápida y una cantidad mayor de memoria más lenta. Cuando un problema se ejecuta sobre un número grande de procesadores, muchos de sus datos se encuentran en la memoria cache, por lo que se reduce el número de fallos de cache, decreciendo así el tiempo necesario de acceso a los datos.

Por otro lado, el objetivo de un programa paralelo es realizar una buena división del problema, para balancear la carga entre los procesadores. Normalmente las técnicas de balanceo de la carga han sido propuestas para algoritmos paralelos basados en descomposición de datos que se ejecutan sobre computadores de memoria distribuida. Esta descomposición de datos puede realizarse de dos formas:

- Estáticamente. Si se conoce la carga computacional del problema antes de resolverlo.
- Dinámicamente. Si la carga computacional se crea y/o se destruye durante la ejecución del algoritmo.

La caracterización exacta de un algoritmo paralelo eficiente no es fácil. Muchos factores pueden contribuir a limitar la ganancia en velocidad. Un parámetro importante es el tamaño del problema de entrada, ya que si no hay bastante carga computacional para el número de procesadores disponibles, la ganancia de velocidad será baja. Otro factor importante, determinado por la cantidad de código secuencial en los algoritmos paralelos, se describe en la *Ley de Amdahl* [2].

5.5. Anomalías en algoritmos paralelos de Branch and Bound

La naturaleza heurística de los algoritmos de Branch and Bound aparece cuando en la selección del siguiente subproblema a tratar hay que decidir cuál tiene mayor probabilidad de ser un *camino crítico mínimo*. Varios subproblemas pueden tener la misma prioridad, por ejemplo, el mismo límite inferior, si se considera una regla de selección de *Primero el Mejor*. Debido a la naturaleza de estos problemas, es difícil, si no imposible, predecir qué dirección de la búsqueda se debe tomar para reducir, tanto como sea posible, el trabajo a realizar. A mediados de los 80, varios autores se dieron cuenta de la existencia de anomalías en la ganancia de velocidad debidas a la *Regla de Selección* utilizada, cuando existían varios problemas con el mismo límite inferior, igual a f^* [71, 72] (Notas históricas pueden verse en [108]). De hecho, Fox et al. en los 70 ya habían observado que incluso en el caso de un solo procesador, cuando varios nodos tienen el mismo límite inferior, algunas estrategias *Primero el Mejor* no son óptimas [40].

Se define $I(p)$ como el número de iteraciones llevadas a cabo con p procesadores en un problema de Branch and Bound. Se define la ganancia en velocidad con respecto al número de iteraciones realizado $S_I(p) = I(1)/I(p)$ [23].

Generalizando, cuando se usa un modelo de comunicación síncrono, la ganancia en velocidad $S_I(p)$ está determinada solamente por la dificultad del problema ya que usando un modelo síncrono la ejecución está completamente definida. En la práctica, cuando se usan varios procesadores, normalmente son inevitables situaciones del tipo $1 < S_I(p) < p$. Las anomalías de la ganancia en velocidad son de dos tipos: *Anomalías Aceleratorias*, donde $S_I(p) > p$ y *Anomalías Detrimentales*, donde $S_I(p) < p$ [71, 72, 78]. Es obvio que se desean preservar las anomalías aceleratorias y evitar las detrimentales. Entre los subproblemas con el mismo límite inferior, pueden existir unos que estén más cerca de la solución que otros. Por lo tanto, si el algoritmo paralelo explora los mejores subproblemas primero se obtendrán anomalías aceleratorias y viceversa.

Estas anomalías han sido teórica y extensivamente estudiadas para algoritmos paralelos de Branch and Bound síncronos [7, 40, 71, 72, 73, 77, 78, 82]. El caso asíncrono se ha tratado en [20, 21, 27].

5.5.1. Condiciones para prevenir anomalías detrimentales

Estas condiciones aparecen en algoritmos que usan una estrategia de selección *Primero el Mejor*. En [71, 72] se demostró, que para algoritmos de Branch and Bound paralelos síncronos, el número de iteraciones realizadas por el algoritmo secuencial no se incrementaba en la versión paralela cuando la función de acotación (Definición 2.1.1) $li(v) \neq f^*$ y v no es un nodo solución [78]. Además, la función

de prioridad heurística h, que establece la ordenación de los subproblemas en la estructura de datos D para su procesamiento, debe cumplir que:

1. $h(v) \neq h(v')$ si $v \neq v',\ v, v' \in V$,
2. $h(v) \leq h(v')$, si v' es descendiente de v,

es decir, que la función de prioridad heurística h sea inyectiva y además sea consistente con la función de acotación li ($h(v_1) \leq h(v_2) \Rightarrow li(v_1) \leq li(v_2),\ \forall v_1, v_2 \in V$).

Si existen relaciones de dominancia, éstas deben ser consistentes con la función de prioridad heurística h (Definición 2.1.6). Bajo estas condiciones los nodos divididos por el algoritmo secuencial, también llamados *nodos primarios*, no pueden ser eliminados por un test de dominancia [27]. El uso de una regla de prioridad heurística no ambigua es una condición suficiente para prevenir anomalías detrimentales, ya que garantiza que al menos un nodo primario es descompuesto en cada iteración de la ejecución del algoritmo síncrono, y que una vez que todos los nodos primarios son descompuestos o eliminados, la ejecución síncrona termina. La demostración para el caso síncrono puede encontrarse en [78].

En [82] se definen reglas del tipo *FIFO*, *LIFO* y *Consistentes* para ordenar los nodos con el mismo valor de la función de prioridad h, demostrando que la regla $FIFO$ es práctica y teóricamente menos propensa a cualquier tipo de anomalías de los algoritmos paralelos síncronos, usando la regla de selección *Primero el Mejor*.

5.5.2. Condiciones para preservar anomalías aceleratorias

Las anomalías aceleratorias son una consecuencia del hecho de que los algoritmos de Branch and Bound paralelos pueden ser más eficientes en sus selecciones heurísticas, ya que descomponen simultáneamente varios subproblemas en cada iteración y por lo tanto el árbol de búsqueda generado es menor que el del algoritmo secuencial. Esto sólo puede ocurrir si se eliminan algunos de los *nodos primarios*. En [77, 78] se muestra que las anomalías de tipo aceleratorio ocurren en los siguientes casos:

- Cuando se usa una *Regla de Selección en anchura* o *en profundidad*.
- Para la *Regla de Selección Primero el Mejor*, cuando varios nodos tienen el mismo valor de la función de prioridad heurística, es decir, cuando la función de prioridad heurística h no es completamente consistente con la función de acotación li. La función h es completamente consistente con li, si $f(v_1) < f(v_2) \Rightarrow li(v_1) < li(v_2),\ \forall v_1, v_2 \in V$.

Añadir más procesadores puede dar lugar a encontrar más rápidamente un mejor límite superior de la solución ($\overline{f^*}$) y por lo tanto evitar la evaluación de subproblemas que podrían evaluarse sin la utilización de estos nuevos procesadores. Se asume implícitamente que si el problema en cuestión es suficientemente grande, el número de subproblemas activos es suficiente para mantener ocupados a los procesadores en todo momento [16]. Resultados sobre anomalías mostraron que árboles de búsqueda densos con *caminos críticos mínimos* cortos producían mejores rendimientos, que árboles poco densos con *caminos críticos mínimos* largos [71]. Análisis teóricos y experimentales han demostrado que las anomalías son raras cuando se buscan las soluciones óptimas usando una *Regla de Selección Primero el Mejor* y el problema es suficientemente grande. Por ejemplo, para el problema de la *Mochila* [71, 22], el problema del *Viajante de Comercio* [105, 108, 109], el problema de *Cobertura de Vértices* [139], para los problemas: de la *Mochila*, *Programación Entera* y *Cobertura de Vértices* [140] y para problemas de *Asignación Cuadrática* [79, 102, 118].

5.6. Trabajos previos en algoritmos Branch and Bound paralelos

En [12] se hace una revisión bibliográfica de varias implementaciones paralelas de algoritmos de Optimización Global Intervalar. Todas ellas tienen en común que están implementadas sobre arquitecturas de memoria distribuida.

- La implementación aportada por Henriksen et. al [53, 54] propone una implementación maestro-esclavo; el procesador maestro envía problemas a los procesadores esclavos y este recoge los resultados. La desventaja principal de esta solución es que si el número de procesadores esclavos aumenta, las comunicaciones con el procesador maestro son un cuello de botella; otro problema es que la longitud máxima de la estructura de datos que almacena los problemas sin resolver está limitada por la cantidad de memoria que tiene el procesador maestro.
- La implementación de Erikson et al. aporta una estrategia de balanceo descentralizado. En cada procesador se ejecutan dos procesos: el proceso organizador y el proceso esclavo. El proceso organizador es el encargado de seleccionar el siguiente problema a resolver y de proporcionárselo al proceso esclavo. Implementaron diversas estrategias para comunicar los procesos organizadores de los distintos procesadores, con el fin de encontrar una manera óptima de balancear la carga.
- Moore et al. [91] hacen una implementación parecida a la anterior; cuando un procesador se queda sin problemas a resolver, se elige otro procesador aleatoriamente para hacerle una petición de problemas, si este contesta negativa-

mente, entonces se envía la misma petición al procesador que tiene un índice de procesador siguiente. Un procesador que recibe una petición de problemas, devuelve la mitad de sus problemas sin resolver, pero nunca un número mayor que un umbral previamente establecido, para evitar mensajes muy largos.

- Berner [5] intenta aprovechar las ventajas de los modelos de balanceo centralizado y balanceo distribuido; para ello usa el concepto de mediador centralizado, ya visto en [7]. Existe un procesador que no realiza trabajo sobre los problemas sin resolver, lo que hace, es esperar mensajes del resto de procesadores para enviarles problemas. El procesador que actúa como mediador centralizado, establece un umbral que cambia de manera dinámica dependiendo del tamaño de la estructura de datos que mantiene los problemas sin resolver y de las peticiones recibidas. Los procesadores con un número de problemas sin resolver mayor que el umbral de mediador centralizado, envían problemas a éste. Ahora la estructura para almacenar los problemas sin resolver está repartida entre los demás procesadores. El principal inconveniente es que el procesador que hace de mediador no resuelve ningún problema.

En [142] presenta una implementación sobre una arquitectura de memoria distribuida. Todos los procesadores ejecutan dos procesos, uno encargado de balancear los problemas entre los procesadores y otro proceso encargado de resolver los problemas. En este caso, la implementación es similar a la presentada por Erikson et al., pero en este caso, los procesadores solo se comunican con los cuatro vecinos mas cercanos; es decir, con los dos procesadores a los que está conectado y con los dos siguientes. En este caso el hardware presentaba una topología en anillo. Este modelo no presenta cuellos de botella, pero cuando el número de procesadores es muy grande, la eficiencia baja debido a que las comunicaciones son muy lentas.

En [8, 9, 12, 87] se hace una implementación de un algoritmo de Optimización Global basado en Aritmética de Intervalos, sobre arquitecturas de memoria distribuida. La principal aportación en el campo de la computación paralela, se hace sobre el criterio a seguir para el reparto eficiente de la carga computacional entre los procesadores. Se presenta un nuevo estimador de la carga computacional para realizar un balanceo dinámico más eficiente. Normalmente, se usa como estimador de la carga la cantidad de problemas pendientes de resolver y la calidad de los mismos, dada por el $\underline{F}(X^i)$. En [12], se propone como estimador de la carga de trabajo, el parámetro $p\overline{f}^*$ para realizar el balanceo. También se establece un modelo híbrido de recorrido del árbol de búsqueda (ver Sección 2.1.1).

En [61] se propone un nuevo modelo de balanceo descentralizado; presenta un modelo maestro-esclavo, pero el procesador maestro no es único, sino que puede ser cualquier procesador. Introduce el concepto de leader: el procesador que hace las veces de maestro es el procesador *leader*. La característica que hace que un procesador sea leader es que es el último que encontró la mejor solución. Con esto consigue

un balanceador dinámico descentralizado, pero en algunas ocasiones pueden aparecer problemas y actuar como un balanceador centralizado. Por ejemplo, si no se actualiza la mejor solución, es decir, esta se encuentra al principio, o simplemente se proporciona al problema, entonces no hay actualización posible, con lo cual el procesador leader será siempre el mismo.

Con la aparición de las arquitecturas SMP y la incorporación de éstas a los cluster de estaciones de trabajo (COW), en [3], se presenta una implementación híbrida. En este modelo, se realiza una paralelización a dos niveles: una internodos, se usa PVM; y otra intranodo, se usa OpenMP. Presentan una aproximación maestro-esclavo, que ellos denominan coordinador-trabajador. El proceso coordinador es el encargado de iniciar los procesos trabajadores, que son los encargados de realizar la computación. El proceso coordinador también es el encargado de distribuir la información entre los procesos trabajadores.

La paralelización internodo consiste en las siguiente fases:

- Un proceso coordinador produce p instancias de procesos trabajadores, descomponiendo el espacio de búsqueda del problema asignado en p conjuntos disjuntos.
- Cada proceso trabajador, explora su propio espacio de búsqueda de la solución con una estrategia *depth-first* y actualiza su mejor solución local y la envía al proceso coordinador.
- Cuando el proceso coordinador recibe una mejor solución de algún proceso trabajador, este lo envía al resto de procesos trabajadores.

El balanceo de la carga entre los nodos se realiza de la siguiente manera:

- Cuando un proceso trabajador completa su subespacio de búsqueda, envía un mensaje al proceso coordinador y espera a recibir una nuevo problema a resolver.
- El proceso coordinador mantiene una estructura de datos que informa sobre los procesos trabajadores que están ociosos. Si esta no está vacía, entonces sondea a los procesos trabajadores pidiendo problemas sin resolver, si alguno le contesta, entonces le envía el problema recibido al procesador que estaba en la lista de procesos ociosos.

La paralelización intranodo la realizan varios threads OpenMP, dependiendo del número de procesadores que tenga el nodo. Cada thread realiza las operaciones sobre una estructura que almacena los problemas sin resolver asignados por el proceso

coordinador. Esta estructura es compartida por los distintos threads del mismo nodo. También se comparte la mejor solución encontrada hasta el momento. Los distintos threads tienen que mantener exclusión mutua cuando: (i) Cogen un nuevo problema de la estructura, (ii) introducen un nuevo problema en la estructura, y (iii) actualizan la mejor solución encontrada. El acceso a estas tres secciones criticas penalizan el rendimiento del algoritmo.

En esta implementación, aunque se aprovechan las ventajas de las arquitecturas SMP, volvemos a tener los problemas de los modelos maestros-esclavo. Existe el proceso coordinador a través del cual pasan todos los mensajes de balanceo de la carga, además de los mensajes de actualización de la mejor solución. En un intento por aumentar el rendimiento del procesamiento intranodo, los autores dividieron la estructura de datos que almacena los problemas abiertos en varias subestructuras, una por cada thread, con esto no es necesario acceder en exclusión mutua a la estructura que mantiene los problemas sin resolver, ya que cada thread accede a la suya. Un thread solo accede a la estructura del thread vecino en caso de que se quede sin problemas.

En [32] se hace un estudio exhaustivo sobre las librerías paralelas de resolución de problemas de Branch and Bound. Todas usan el modelo de programación maestro-esclavo y están implementadas sobre maquinas de memoria distribuida. En [33] se aporta una solución implementada sobre un arquitectura de memoria compartida. Esta implementación se ha llevado a cabo con OpenMP. El algoritmo presentado, usa una estructura de datos global donde se almacenan los problemas abiertos (problemas sin resolver). El número de threads que usa es proporcionado al programa previamente. Entonces, los subproblemas son sacados de la estructura y asignados a cada thread. Cada thread trabaja con su propio subproblema. Cuando se encuentra una mejor solución, solo un thread puede cambiar la variable que la almacena. Cuando un thread termina de procesar su subproblema, los nuevos subproblemas generados, si los hay, son almacenados en la estructura global que los alberga. El eficiencia obtenida es bastante baja. Hay que hacer notar que estas librerías no intentan explotar el paralelismo de las arquitecturas, se trata de herramientas que proporcionan mecanismos para implementar los programas paralelos de manera sencilla.

5.7. Algoritmo paralelo

La primera cuestión que nos planteamos a la hora de implementar nuestro algoritmo en una arquitectura de memoria compartida, es acerca de que variables son las que necesitamos compartir con el resto de threads, y cuales son usadas de forma local por los threads. En esta tesis se presentan dos alternativas de implementación de un algoritmo de Optimización Global Intervalar sobre arquitecturas de memoria

compartida. Las dos alternativas, se originan en función del modo de acceso (global o local) a las estructuras de datos que maneja el programa: el árbol de trabajo (W) y el árbol de resultados (R).

Tal como se ha presentado en secciones anteriores, todos los threads deben conocer la mejor actualización de ($f^{\sim}$), por lo tanto en todas las versiones presentadas la variable ($f^{\sim}$) será global, es decir, accesible por todos los threads. La principal idea de las implementaciones presentadas es la de mantener un thread que realice un trabajo útil por procesador. La localidad o globalidad de las estructuras de datos va a influenciar el modo en el que se generan estos threads.

A continuación se detallan las dos implementaciones correspondientes con las dos alternativas en función del modo de acceso a las variables compartidas.

Global-PAMIGO

Partiendo de sus especificaciones generales:

- El árbol de trabajo y el árbol final son estructuras globales, por lo que los threads acceden a estas estructuras en exclusión mutua.
- El número de threads creados es fijo e igual al número de procesadores asignados al problema.
- Cada thread procesa una caja y devuelve los resultados al árbol de trabajo o final.
- No se necesita un balanceador dinámico.

Global-PAMIGO crea un thread principal que realiza las funciones presentadas en el Algoritmo 5.1. En la linea 3, se realiza una fase de inicialización descrita en el Algoritmo 5.2, que consiste en evaluar las funciones soporte iniciales, evaluar el mejor límite superior de la función, generar el primer nodo e insertarlo en el árbol de trabajo e inicializar el árbol de resultados.

Una vez realizada la fase de inicialización, las líneas 4 y 5 del thread principal inicia tantos threads PAMIGO-GThread como procesadores estén disponibles ($p = MaxThreads$). La variable $MaxThreads$ almacena el número de procesadores del que podemos hacer uso. La asignación de cada PAMIGO-GThread a cada procesador la realiza el sistema operativo. En este caso esta operación es relativamente simple. Una vez iniciados los threads PAMIGO-GThread, el thread principal pasa al estado de *Stopped*, esperando que todos los threads creados anteriormente terminen su trabajo (líneas 6 y 7).

Algorithm 5.1: Thread principal del algoritmo Global-PAMIGO.

```
1 .
2 .
3 Inicializacion() ;                          /* Inicializacion de W */
4 for (i = 0; i < MaxThreads; i++) do
5 |  Crear PAMIGO-GThread(i);
6 for (i = 0; i < MaxThreads; i++) do
7 |  Esperar PAMIGO-GThread(i);
8 .
9 .
```

Algorithm 5.2: Algoritmo de Inicialización.

Entrada: S, F
Salida : W

1 $sp(S) = \big(\{\underline{F}(S_1^l), \underline{F}(S_1^r)\}, \ldots, \{\underline{F}(S_n^l), \underline{F}(S_n^r)\}\big)$;
2 Eval $F(S)$, $f^{\sim} = F(m(S))$, $F'(S)$, and $lbzf(S)$;
3 W=$\{TreeNode(S)\}$;
4 R=$\{\emptyset\}$;

En toda implementación paralela hay que determinar de la condición de parada del algoritmo paralelo. Los PAMIGO-GThread pueden estar en los siguientes estados:

Trabajando : cuando está procesando cajas y/o existen cajas en el árbol de trabajo.

Espera : no se está procesando ninguna caja y no existen cajas en el árbol de trabajo.

Nótese que el que haya PAMIGO-GThreads en el estado de *Espera* no implica que algoritmo paralelo haya acabado, ya que pueden existir otros PAMIGO-GThreads que están trabajando y que realimenten con nuevas cajas el árbol del trabajo. En la implementación PAMIGO-Global existe un vector de estados global, donde cada PAMIGO-GThread establece el estado en el que se encuentra. La condición general de parada será cierta cuando un PAMIGO-GThread pasa al estado de *Espera* y comprueba que todos los demás PAMIGO-GThreads están también en el estado de *Espera*.

El Algoritmo 5.3 muestra el pseudo-código de los PAMIGO-GThreads. En primer lugar, el algoritmo recibe como entrada un identificador de thread, denotado

Algorithm 5.3: PAMIGO-GThread.

```
   Entrada: idthread
 1 while (Parada == False) do      /* Mientras no se cumpla la parada */
 2     if (W == ∅) then              /* El árbol no tiene elementos */
 3         if (Resto de Threads en Espera) then
 4             Parada = True ;              /* Activamos la parada */
 5             Desbloquea Threads en Espera ;     /* Desb. threads */
 6         else
 7             Bloquea Thread(idthread) ;     /* Bloqueo este thread */
 8         Continue ;                /* Volvemos al inicio del bucle */
 9     X=ExtraerCaja(W) ;              /* Coge una caja de W */
10     ProcesaCaja(X);                /* Procesa el intervalo */
11     if (Genero dos cajas y existe un Thread_en_Espera) then
12         Desbloquea un Thread_en_Espera ;     /* Desb. thread */
```

como *idthread*. Este identificador nos servirá para poder bloquear nuestro thread en espera de trabajo o para que sea desbloqueado por otro thread cuando haya trabajo que realizar. El algoritmo empieza en la linea 1 con un bucle que se ejecuta mientras no se cumpla el criterio de parada establecido en la línea 1. El criterio de parada se basa en el valor de la variable global *Parada*. Si la variable *Parada* toma el valor $True$, quiere decir que todos los threads están en el estado de *Espera* y podemos salir del bucle. Las instrucciones que aparecen entre las líneas 2 y 8 chequean la posibilidad de establecer la variable *Parada* a *True* o de pasar a un estado de *Espera*. Si el árbol de trabajo no tiene elementos (línea 2) se comprueba si los demás PAMIGO-GThreads están en un estado de *Espera* (línea 3):

- En caso afirmativo, se establece la variable *Parada* al valor $True$ (linea 4), y además se desbloquean al resto de PAMIGO-GThreads para que puedan acabar su ejecución (linea 5).
- En caso negativo, existen threads PAMIGO-GThread en el estado *Trabajando*. La linea 7, bloquea este thread PAMIGO-GThread en espera de ser desbloqueado por otro thread PAMIGO-GThread o de que se alcance el final de su ejecución.

Si el árbol de trabajo tiene al menos un nodo, es condición suficiente para que este PAMIGO-GThread pueda seguir trabajando, extrayendo un nodo del árbol de trabajo en la linea 9 y procesándolo en la linea 10. La línea 10 engloba todas las operaciones descritas en el algoritmo 4.3. Una vez procesada una caja, si se han generado dos nuevas cajas que deben ser procesadas y existe algún PAMIGO-

GThread en el estado de *Espera*, este será desbloqueado para que pueda procesar una de las nuevas cajas.

El algoritmo Global-PAMIGO intenta que todos los procesadores estén trabajando sobre las cajas más prometedoras y por lo tanto haciendo un trabajo útil. Su principal inconveniente puede ser la contención en el acceso a las estructuras de datos globales cuando el número de procesadores es grande.

Local-PAMIGO

Partiendo de sus especificaciones generales:

- Cada thread tiene un árbol de trabajo y un árbol final que son locales, de forma que se evita la exclusión mutua en el acceso a estas estructuras de datos.
- El número total de threads creados es variable y depende del problema.
- Se intenta que el número de threads activos en cada momento sea igual al número de procesadores asignados.
- Cada thread procesa la región de búsqueda que le ha sido asignada, la cual genera una estructura de datos dinámica. Cuando un thread se queda sin trabajo muere.
- Es necesario el uso de un balanceador dinámico.

En esta implementación se nos presentan los siguientes problemas:

1. Teniendo en cuenta que los threads de Local-PAMIGO (PAMIGO-LThreads) mueren cuando no tienen más trabajo que realizar, se necesita un algoritmo de balanceo dinámico de la carga, ya que pueden existir otros PAMIGO-LThreads en ejecución con mucha carga computacional pendiente y procesadores que no tengan threads asignados.
2. Un PAMIGO-LThread muere cuando su árbol de trabajo está vacío, pero puede tener cajas en su árbol final. Las cajas finales generadas por los distintos PAMIGO-LThreads hay que recogerlas en una estructura de datos final.
3. No sabemos a que procesador es asignado un thread por el sistema operativo, lo que dificulta la obtención de estadísticas de uso por procesador físico.

Como solución al Problema 1, Local-PAMIGO usa un balanceo dinámico basado en una variable global (NTh) que indica el número de PAMIGO-LThreads activos. Un PAMIGO-LThread, con al menos dos cajas en su árbol de trabajo, que observe que el número de PAMIGO-LThreads activo es menor que el número de procesadores asignado al problema ($NTh < MaxThreads$), genera un nuevo PAMIGO-LThread. La carga computacional asignada al nuevo thread se obtiene dividiendo el árbol de trabajo y generando de este modo dos nuevos subárboles. Uno de los subárboles será procesado por el PAMIGO-LThread que lo generó y el otro subárbol será procesado por el nuevo PAMIGO-LThread. El PAMIGO-LThread padre debe incrementar la variable NTh en exclusión mutua para hacer saber a los demás PAMIGO-LThreads activos que se ha generado un nuevo thread. Nótese que el PAMIGO-LThread que va a realizar el balanceo de la carga es aquel que primero observe que se cumple la condición ($NTh < MaxThreads$). De los experimentos realizados con este mecanismo de balanceo de la carga, se deduce que el árbol de threads creados varía de una ejecución a otra.

Una solución al Problema 2 puede ser que un thread que termina su ejecución, es decir, su árbol de trabajo se queda vacío, devolverá al thread que lo generó el árbol de resultados que generó (puede estar vació), terminando de este modo su ejecución. Un thread no enviará su árbol final al thread que lo generó mientras no reciba los árboles finales de los threads que el generó. De esta forma también se resuelve el problema de detectar la terminación del algoritmo paralelo. Cuando el thread raíz del árbol de threads generados reciba los árboles finales de todos los threads que generó, la ejecución del algoritmo paralelo termina. En este modelo, un thread puede estar en los siguientes estados:

Activo : el thread esta ejecutando el algoritmo de Ramificación y Acotación Intervalar sobre su árbol de trabajo local y almacenando o no cajas finales en su árbol local final.

Balanceando : el thread ha detectado que la condición $NTh < MaxThreads$ es cierta y está realizando el mecanismo de balanceo de la carga mostrado anteriormente.

Espera : el thread espera recibir los arboles finales de los threads que generó.

Finalizando : Una vez terminado el estado de espera, el thread manda su árbol final a su thread padre. Nótese que el thread raíz del árbol de threads no puede estar en este estado al no tener ningún thread padre.

De esta forma, el número de threads en estado activo no puede ser superior a $MaxThreads$ que es igual al número de procesadores asignados a este problema, pero puede haber muchos más threads en el estado de *Espera* o *Finalizando*.

Algorithm 5.4: Thread principal del algoritmo Local-PAMIGO.

```
1  .
2  .
3  for (i = 0; i < Maxthreads; i++) do      /* Inicializamos los slots */
4  |  PV[i].thread = -1 ;                   /* Al principio todos están libres */
5  Inicializacion() ;                              /* Inicializacion de W */
6  PV[0].thread = idthread ;   /* Asignamos el primer thread al slot[0] */
7  Crear PAMIGO-LThread(W, idtread);
8  wait(Parada) ;                              /* Semáforo parada threads */
9  .
10 .
```

La principal desventaja de este modelo de generación de threads es que los árboles finales deben recorrer el camino en el árbol de threads hasta el thread raíz, lo que puede ser computacionalmente costoso. Para resolver este problema y el Problema 3, nos definimos un vector de procesadores virtuales $PV[MaxThreads]$. Cada slot del vector PV tiene información de que thread activo está asociado a ese slot, y el árbol final local a ese slot que es generado por todos los threads que han sido asignados en ese slot. Un thread que termina su ejecución deja el slot al que estaba asignado como libre y muere, pero el slot mantiene el árbol de cajas final asociado a ese slot y las correspondientes estadísticas de uso. De esta forma, el árbol de trabajo es local a cada thread pero el árbol final está repartido por los slots de PV. El algoritmo Local-PAMIGO solo permite que exista un thread por slot por lo que no se necesita exclusión mutua para el acceso a los datos almacenados en cada slot. El método de balanceo dinámico se mantiene en esta versión por lo que el número máximo de threads activos no puede superar $MaxThreads$, y solo existe uno por slot.

Ahora cada thread sólo puede estar en los estados *Activo* o *Balanceando* por lo que no existe un árbol de threads y debe establecerse un criterio de detección de parada del algoritmo paralelo diferente, pero similar al del algoritmo PAMIGO-Global. El thread que termina su trabajo y detecta que $NTh = 1$, establece $NTh = 0$ y la variable global *Parada* a *True* para que el thread principal del programa termine. El thread principal del programa Local-PAMIGO se muestra en el Algoritmo 5.4.

El Algoritmo 5.4 inicializa del árbol de trabajo W en la linea 5. La principal diferencia con el Algoritmo 5.1 es que ahora se crea un único thread PAMIGO-LThread. El resto de threads PAMIGO-LThread, hasta alcanzar el número de procesadores ($MaxThreads$), son creados por los propios threads PAMIGO-LThread, tal como se ha explicado en el método de balanceo dinámico de la carga. Cada slot del vector PV, o procesador virtual, almacena una gran cantidad de información, sin embargo. En los pseudo-códigos que representan al algoritmo PAMIGO-Local sólo se referen-

Algorithm 5.5: Thread PAMIGO-LThread.

```
   Entrada: W,idthread
1  for (i = 0; i < MaxThreads; i++) do      /* Localizamos nuestro slot */
2      if (PV[i].thread == idthread) then  Me = i; break;
3  while (W ≠ ∅) do                  /* Mientras no se cumpla la parada */
4      X=ExtraerCaja(W) ;                        /* Coge una caja de W */
5      ProcesaCaja(X);                          /* Procesa el intervalo */
6      Inserta cajas finales en PV[Me].R ;
7      if (NTh < MaxThreads)&&(nElem(W) >= 2) then
8          NTh++;       /* Incrementamos el numero de threads activos */
9          Divide(W,W1,W2); W = W1;        /* Div. el árbol de trabajo */
10         for (i = 0; i < MaxThreads; i++) do   /* Localizamos el slot */
11             if (slot[i] == −1) then New = i; break;
12         PV[New].Thread = idthread_New;    /* Asignamos thread a slot */
13         Crear PAMIGO-LThread(W2,idthread_New)
       ;
14 PV[Me].Thread = −1;                              /* Liberamos un slot */
15 NTh = NTh − 1;       /* Decrementamos el numero de threads activos */
16 if (NTh == 0) then
       signal(Parada);                  /* Activamos el thread principal */
```

cian el identificador de thread asignado $PV[i].thread$ (ver línea 6 del Algoritmo 5.4) y el subárbol final asociado a ese procesador virtual $PV[i].R$ (línea 6 del Algoritmo 5.5).

El Algoritmo 5.5 muestra el pseudo-código del thread PAMIGO-LThread. El thread PAMIGO-LThread primero determina a qué procesador virtual ha sido asignado (líneas 1 y 2), cuyo índice se almacena en la variable *Me*. Mientras su árbol de trabajo local W no esté vacío (línea 3) se procesan las cajas del árbol de trabajo (línea 5). Las cajas finales se almacenan en el árbol final asociado al procesador virtual al que ha sido asignado este thread (línea 6). En cada iteración, el thread PAMIGO-LThread chequea si se necesita realizar un balanceo dinámico de la carga (línea 7), es decir, si el número de threads en estado activo es menor que el número de procesadores asignados y existen al menos dos cajas en el árbol de trabajo. Si se cumple la condición para realizar un mecanismo de balanceo, se genera un nuevo thread, se le asigna a un procesador libre y con una carga igual a la mitad del árbol de trabajo del thread actual (líneas 8 a 13). Cuando PAMIGO-LThread termina su trabajo, deja el procesador virtual libre (línea 14) y decrementa el número de threads PAMIGO-LThread en estado activo (línea 15). Además, si es el el último PAMIGO-LThread, antes de morir, informa al thread principal del algoritmo Local-PAMIGO (Algoritmo 5.4) que el algoritmo paralelo ha terminado (línea 16).

En el modelo PAMIGO-LThreads las únicas variables globales son sólo $f^{\tilde{}}$ y Nth, reduciéndose de esta forma la contención en el acceso a memoria compartida.

Nótese, que cuando los threads PAMIGO-LThreads terminan, el Algoritmo 5.4 debe recolectar las cajas finales y la información estadística almacenada en los procesadores virtuales. Cuando el número de procesadores es grande, esta parte final puede tener cierto peso en el rendimiento del algoritmo.

5.8. Evaluación

El entorno hardware de ejecución de los algoritmos paralelos descritos en la Sección 5.7, se ha llevado a cabo sobre una plataforma Altix 330 de Silicon Graphics con 16 procesadores Intel Itanium II y 64 GB de memoria RAM. Los procesadores se encuentran divididos en dos conjuntos:

- Conjunto *boot*: Destinado a tareas de administración, compilación y ejecución de programas y utilidades, con un solo procesador.
- Conjunto *green*: Destinado a la evaluación de programas paralelos. Con 15 procesadores.

En cuanto a las características del software utilizado para la implementación de los algoritmos son las siguientes:

- El Sistema Operativo que corre esta arquitectura es GNU/Linux con la versión del kernel 2.6.5-7.244-sn2.
- El compilador de C es el gnu C++ versión 3.3.3.
- La librería de aritmética de intervalos es la C-XSC version 2.1.1, que ha sido modificado para ser *thread safe*.
- La librería de pthreads que tiene nuestro sistema es la NPTL version 2.3.5.

Para la evaluación de la eficiencia de los algoritmos propuestos en esta tesis, en el Capítulo 4 se ha hecho uso de un conjunto de funciones de prueba que son estándares en el ámbito de la Optimización Global. Este conjunto de funciones de pruebas está originalmente pensado para evaluar y comparar la eficiencia de cada una de las reglas y aceleradores que definen un algoritmo específico frente a propuesta previas o clásicas. Sin embargo, desde un punto de vista computacional, la mayoría de ellas no requieren computadores de altas prestaciones para su solución,

Tabla 5.1: Equivalencia de funciones multidimensional, precisión para la que se ha calculado el mínimo, dimensiones y referencia bibliográfica.

No.	Función	Épsilon	Dim	Ref
0	EX2	10^{-9}	5	[116]
1	Griewank 10	10^{-4}	10	[133]
2	Schwefel 2.10 (Kowalik)	10^{-5}	4	[141]
3	Neumaier 2	10^{-10}	4	[99]
4	Ratz 5	10^{-5}	3	[116]
5	Ratz 6	10^{-4}	5	[116]
6	Ratz 7	10^{-4}	7	[116]
7	Ratz 8	10^{-3}	9	[116]
8	Rosenbrock 10	10^{-14}	10	[30]
9	Schwefel 2.14 (Powell)	10^{-8}	4	[116]
10	Schwefel 2.7	10^{-2}	3	[116]
11	Schwefel 3.1p	10^{-4}	3	[116]

especialmente para los algoritmos desarrollados en esta tesis que consiguen reducir suficientemente el número de evaluaciones de la función objetivo y de sus derivadas. Un subconjunto de 30 funciones de las Tablas 4.1, 4.2, 4.3 y 4.4 consumen menos de 10 segundos, resolviéndolas con una precisión de $\epsilon = 10^{-14}$.

Los modelos paralelos de computación propuestos en este capítulo se han evaluado sobre un conjunto de doce funciones de prueba, once de ellas son funciones más costosas utilizadas en la evaluación de resultados mostrados en el Capítulo 4 (Tablas 4.1, 4.2, 4.3, 4.4); la función adicional está descrita en el Apéndice A y se denomina: Schwefel 2.7 (Powell) [116].

La evaluación de los modelos de paralelismo se ha realizado desde varias perspectivas. Hay que considerar que existen diferentes factores que pueden influir en la eficiencia de las implementaciones paralelas que se proponen aquí; en primer lugar, tal como se describe en la Sección 5.5, los factores relacionados con las anomalías detrimentales o aceleratorias propias de la alteración de la secuencia de evaluación y creación de subproblemas que, dependiendo del número de procesadores sobre los que se ejecute el algoritmo paralelo, pueden dar lugar a formas muy diferentes del árbol de subproblemas. Por otro lado, la eficiencia de las implementaciones se puede ver afectada por factores tales como contenciones en el acceso a memoria, desbalanceo de la carga entre procesadores y/o arboles de subproblemas desequilibrados, en los que no es posible generar suficientes subproblemas para alimentar un número determinado de procesadores.

La evaluación llevada a cabo en esta sección ofrece ejemplos de funciones que se ven afectadas por uno o varios de estos factores. Para poder justificar claramente

Tabla 5.2: Resultados del esfuerzo $Esf(p)$ y desbalanceo para cinco ejecuciones usando ocho procesadores.

Modelo	Evaluaciones					
EX2	1	2	3	4	5	Desb
Global	5525064	5512155	5524307	5525691	5525637	0.001
Local	5621732	5567741	5581428	5557925	5657451	0.007
Ratz 7	1	2	3	4	5	Desb
Global	5009724	5009481	5009612	5011499	5010110	0.0001
Local	5015474	5015391	5013923	5016132	5013759	0.0002

los resultados obtenidos hemos diseñado un conjunto de experimentos que nos permiten valorar la eficiencia del algoritmo paralelo. Los algoritmos se han evaluado usando un número de procesadores $NP = 1, 2, 4, 8, 13$ y 15. Debido a que en la implementación con *pthreads* el sistema operativo toma decisiones acerca de donde ejecutar cada thread, cada ejecución del mismo algoritmo, bajo las mismas condiciones iniciales, puede presentar ciertas variaciones en lo referente a sus resultados computacionales (no respecto a los resultados numéricos del problema). Para ofrecer una idea de este efecto hemos ejecutado cinco veces los algoritmos paralelos usando 8 procesadores. Los resultados de esta primera prueba se muestran en la Tabla 5.2, donde se suministran valores del esfuerzo computacional de los dos modelos computacionales. El esfuerzo computacional se mide en términos del número de evaluaciones de la función objetivo y de sus derivadas, que es también un indicador del número de subproblemas que se evalúan. En la Tabla 5.2 se muestran los valores de $Esf(p) = FE + n \cdot GE$ para dos funciones de prueba (EX2 y Ratz 7) en cinco ejecuciones y el desbalanceo medido como la relación entre la desviación estándar y el valor medio de las cinco ejecuciones, para los dos modelos de paralelismo (Global y Local). De estos datos podemos deducir que realmente no existen grandes variaciones al repetir las ejecuciones. Estos resultados indican que no es necesario llevar a cabo ejecuciones múltiples bajo las mismas condiciones para poder valorar el comportamiento de los modelos de paralelismo.

La evaluación computacional de los algoritmos paralelos se ha llevado a cabo midiendo varios parámetros. En primer lugar, se hace una valoración de la eficiencia de las implementaciones paralelas basada en el Speed-Up. En la Figura 5.6 se muestran los valores del Speed-Up para los dos modelos de paralelismo y para el conjunto de las doce funciones de prueba seleccionadas. Las funciones se identifican por un número del 0 al 11. La Tabla 5.1 muestra la relación entre el nombre de la función, el valor numérico asignado y la precisión para la que hemos calculado el mínimo.

Como puede observarse en la Figura 5.6 existen cuatro funciones (8, 9, 10 y 11) que presentan un comportamiento no lineal, es decir, que el Speed-Up no crece al aumentar el número de procesadores sino que incluso en alguna de ellas llega a

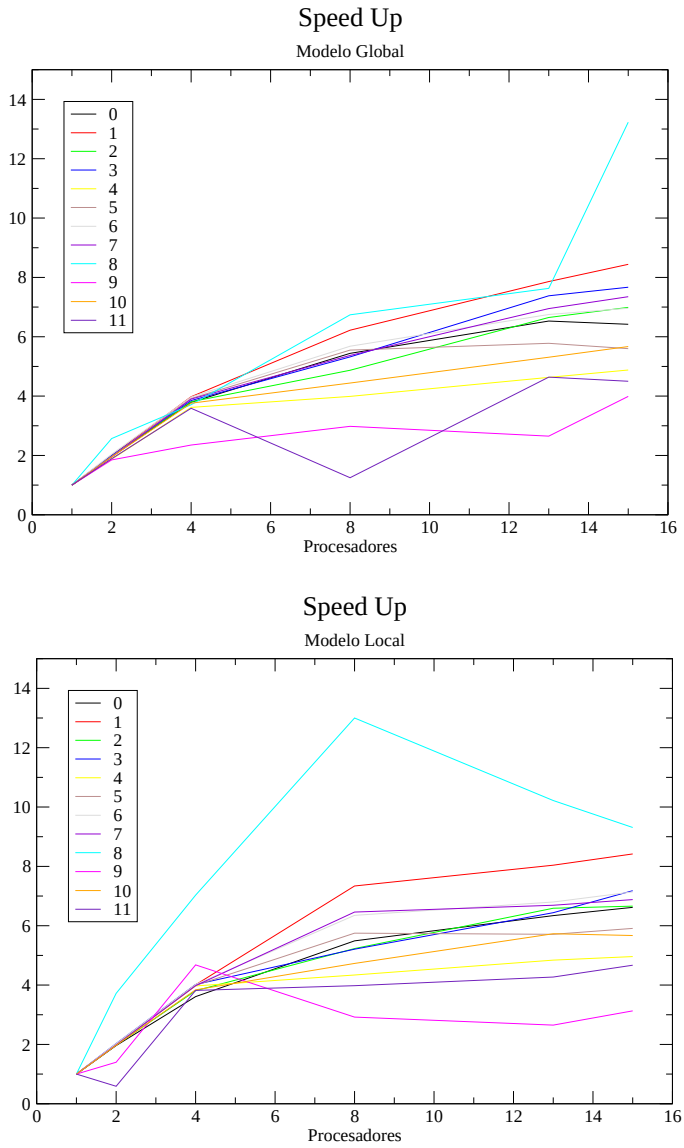


Figura 5.6: Valores del Speed-Up para el modelo Global y Local.

decrementarse. Por ejemplo para la función 11 en el modelo global se obtiene un Speed-Up menor que 2 cuando se usan 8 procesadores y algo similar le ocurre a la función 9 en el modelo Local. Nótese también que la función 8, en el modelo Local, presenta un Speed-Up superlineal con 8 procesadores. No obstante, si se ignora este

subconjunto de cuatro funciones, hay que comentar que claramente se observa un caída de los valores del Speed-Up cuando se usan 8 o mas procesadores. Esta pérdida de eficiencia de los dos modelos de paralelismo puede ser debida a múltiples causas, que son las se analizan y evalúan a continuación.

En primer lugar, dado que los algoritmos de Branch and Bound pueden presentar anomalías detrimentales y/o aceleratorias debidas a la modificación de la regla de selección, que claramente se ve afectada por el paralelismo, se hace una valoración del esfuerzo computacional ($Esf(p)$) en función del número de procesadores, para todas las funciones de prueba. Este análisis permite valorar, las posibles anomalías por el efecto del paralelismo y, simultáneamente, el reparto de la carga computacional entre los threads del modelo Global y entre los procesadores virtuales del modelo Local. Nótese que de esta forma se ignoran los posibles efectos debidos por ejemplo a la contención de memoria.

En las Tablas C.4 y C.5 del Apéndice C, se muestran los valores numéricos del $Esf(p) = FE + n \cdot GE$ para los dos modelos de paralelismo, Global y Local, respectivamente. De los datos de estas tablas cabe también destacar que el modelo Global requiere un número menor de evaluaciones que el modelo Local, y aunque usando un solo procesador esta diferencia es despreciable, cuando se usan más procesadores se obtienen diferencias que vienen a indicar que el modelo global sigue una secuencia de evaluación de subproblemas mas eficiente que el modelo Local. No obstante, estas diferencias no superan nunca el 7 %. En la Figura 5.7 se han representado gráficamente los datos de dichas tablas.

En la Figura 5.8 se han representado los valores de las anomalías propias de cada uno de los modelos de paralelismo (ver Tablas C.6 y C.7 del Apéndice C para los valores numéricos). Las anomalías se han medido como la relación entre el esfuerzo computacional total del algoritmo sobre un procesador y sobre p procesadores $A(p) = \frac{Esf(1)}{Esf(p)}$. Se puede observar que solamente cuatro de las funciones de prueba presentan anomalías, en algunos casos detrimentales y otros aceleratorias. También queda claramente reflejado que el modelo de paralelismo que se ve menos afectado por las anomalías es el modelo Global, cuyo rango de variación esta en el intervalo (0.28-1.71), mientras que para el modelo Local el rango de valores de $A(p)$ es (0.3-2.36). Nótese que si se eliminan las cuatro funciones (8, 9, 10 y 11) con fuertes anomalías, el resto de funciones no se ven prácticamente afectadas por el efecto del paralelismo. El rango de valores de las anomalías para las funciones 0 a la 7 es de (0.99-1.02) para el modelo global y de (0.93-1.00) para el modelo Local. De estos datos se deduce que la perdida de eficiencia de los modelos Global y Local no es una consecuencia de las anomalías.

Dadas las características dinámicas e irregulares de los datos, podría pensarse que la perdida de eficiencia de los modelos paralelos puede ser debida a algún problema relacionado con el desbalanceo de la carga que, de alguna manera, podría dejar ociosos a algunos procesadores mientras otros estuvieran sobrecargados.

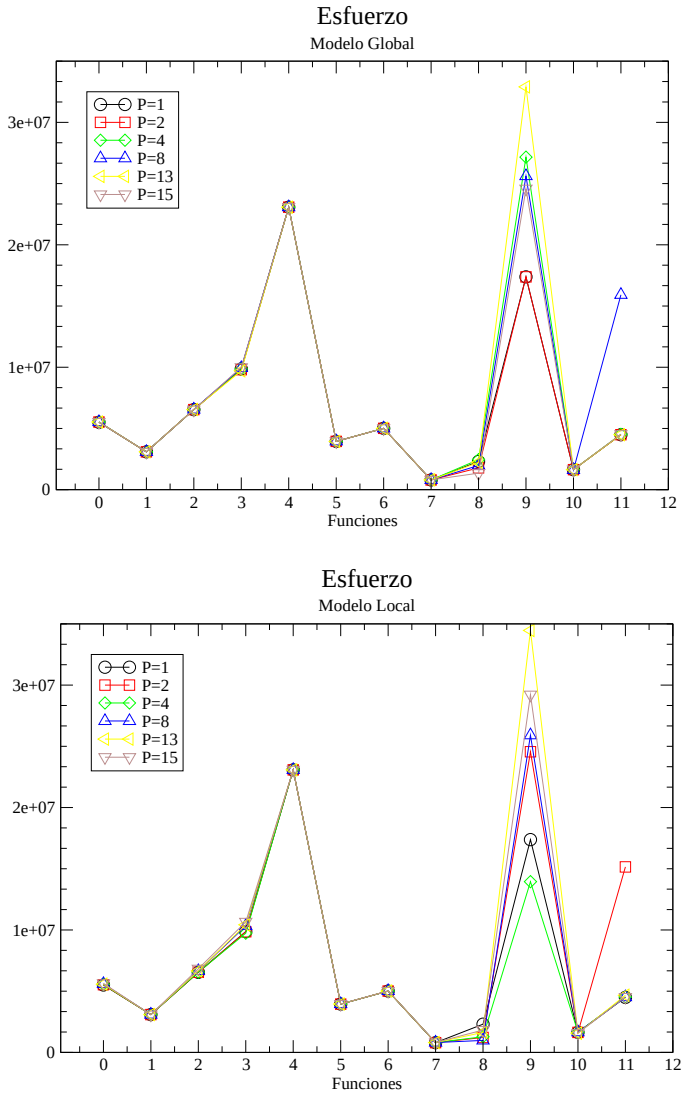


Figura 5.7: Medidas del esfuerzo computacional $Esf(p) = FE + n \cdot GE$ para un conjunto de 12 funciones de prueba, en función del número de procesadores, para los modelos Global y Local.

Para valorar este posible efecto, se ha llevado a cabo una evaluación del desbalanceo de la carga. El desbalanceo se ha medido en base al esfuerzo computacional de cada uno de los threads en el modelo Global y en base al esfuerzo asociado a cada procesador virtual para el modelo Local. La medida es la relación entre la desvia-

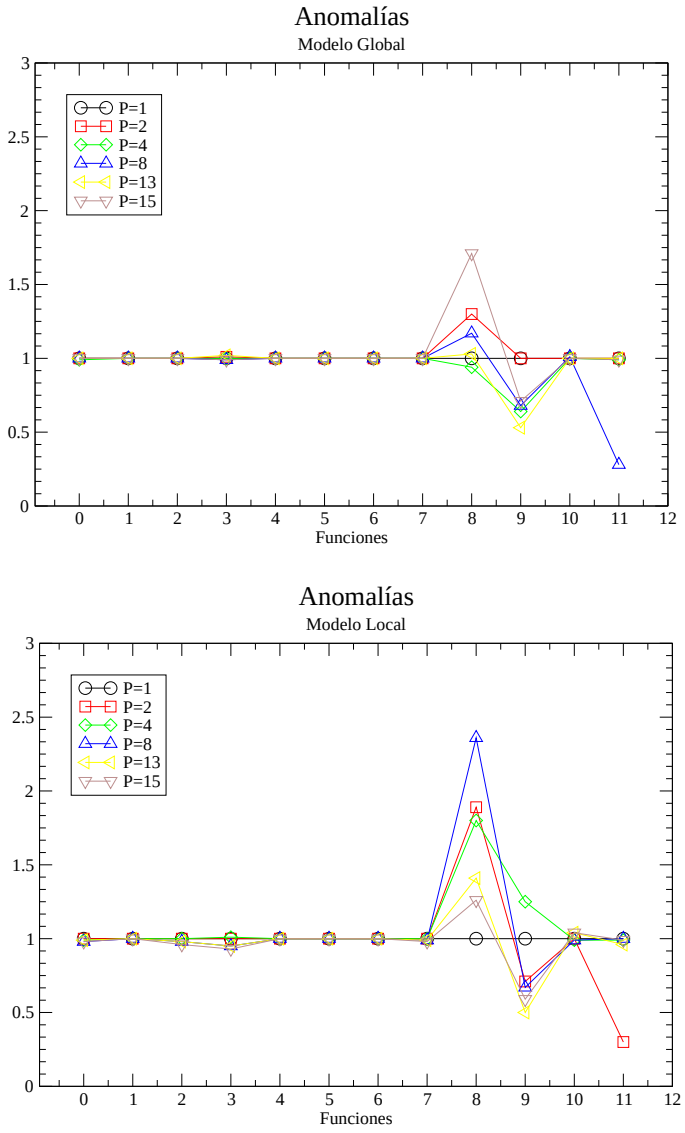


Figura 5.8: Anomalías debidas al paralelismo para los modelos Global y Local.

ción estándar y el valor medio del esfuerzo. Una representación gráfica de los valores del desbalanceo se muestra en la Figura 5.9 (ver también las Tablas C.8 y C.9 del Apéndice C, que muestran los mismos resultados en forma de valores numéricos).

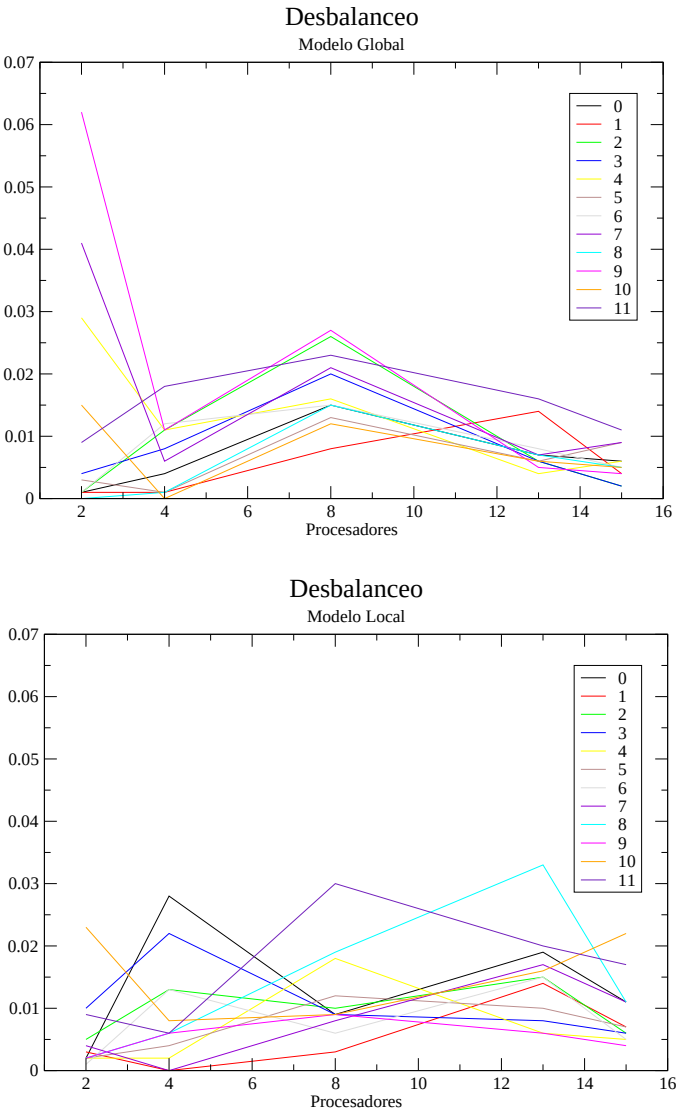


Figura 5.9: Valores del desbalanceo de la carga para el modelo local.

En el modelo Global el máximo valor del desbalanceo es de 0.062 y se obtiene para dos procesadores, en el resto de de configuraciones no supera el valor de 0.030. En el modelo Local, donde se ha aplicado un balanceador dinámico, el desbalanceo en general esta por debajo de 0.033. De estos resultados se puede concluir que ambos modelos de computación paralela son bastante robustos en cuanto al equilibrio de

la carga computacional se refiere.

De todas las posibles medidas que conduzcan a encontrar las causas de la perdida de eficiencia de los modelos paralelos propuestos en esta tesis, solo queda analizar los tiempos de ejecución. Considerando que en los modelos de paralelismo basados en threads, para arquitecturas de memoria compartida-distribuida, puede existir un overhead debido a la creación de los threads o a la contención de memoria, se han llevado a cabo medidas del número de threads creados en cada ejecución y de los tiempos de usuario, sistema y tiempo real.

En relación con el número de threads creados (para el modelo Local), cabe comentar que el número threads creados en una ejecución, como es lógico, aumenta con el número de procesadores, pero en ningún caso hemos encontrado ejecuciones que generen más de 160 threads. Esto induce a pensar que la creación de threads no es la causa de la perdida de rendimiento del sistema.

Finalmente, para terminar este análisis se suministran los valores de tiempo de usuario, sistema (acumulados por procesador) y tiempo real. En las Tablas C.10 y C.11, del Apéndice C se muestran los valores numéricos obtenidos de estos parámetros para todas las funciones de prueba, en función del número de procesadores, para los modelos Global y Local. Estos mismos datos pueden verse en forma gráfica en las Figuras 5.10, 5.11 y 5.12. En relación a los datos del tiempo real, puesto que existe una correlación directa con el análisis que se ha hecho para el Speed-Up, solo comentaremos los aspectos relacionados con la comparación entre los dos modelos. En este sentido cabe destacar que el modelo Global da lugar a tiempos de ejecución ligeramente superiores a los obtenidos por el modelo Local, y estas diferencias pueden llegar hasta el 16 %, en algunos casos. Es significativo destacar que para los tiempos de usuario ocurre algo similar a los datos de tiempo real, sin embargo para los tiempos del sistema existe una gran diferencia entre ambos modelos, verificándose que el tiempo consumido por el sistema en el modelo Global es siempre superior al del modelo Local y estas diferencias llegan a alcanzar valores del 170 %. Una comparativa de ambos modelos se ha representado en la Figura 5.13, donde se ha representado el porcentaje de la diferencia de los tiempos de los modelos Global y Local, relativa al valor medio de ambos modelos; es decir $100 \cdot (T_{Global} - T_{Local})/(T_{Global} + T_{Local}/2)$

Para terminar este análisis hay que resaltar que los tiempos consumidos por el sistema tanto en el modelo Global como en el Local crecen enormemente cuando el número de procesadores es superior a cuatro, que es cuando se obtiene una perdida de eficiencia en los algoritmos. Este incremento en el tiempo del sistema es más acusado en el modelo Global que en el Local.

De todo este análisis únicamente se puede concluir que la perdida de eficiencia en ambos modelos es debida principalmente a problemas de contención de memoria. Si se tiene en cuenta que cada vez que se evalúa un subproblema (un elemento del árbol de trabajo) es necesario acceder a datos compartidos, como por ejemplo a $f^{\sim}$,

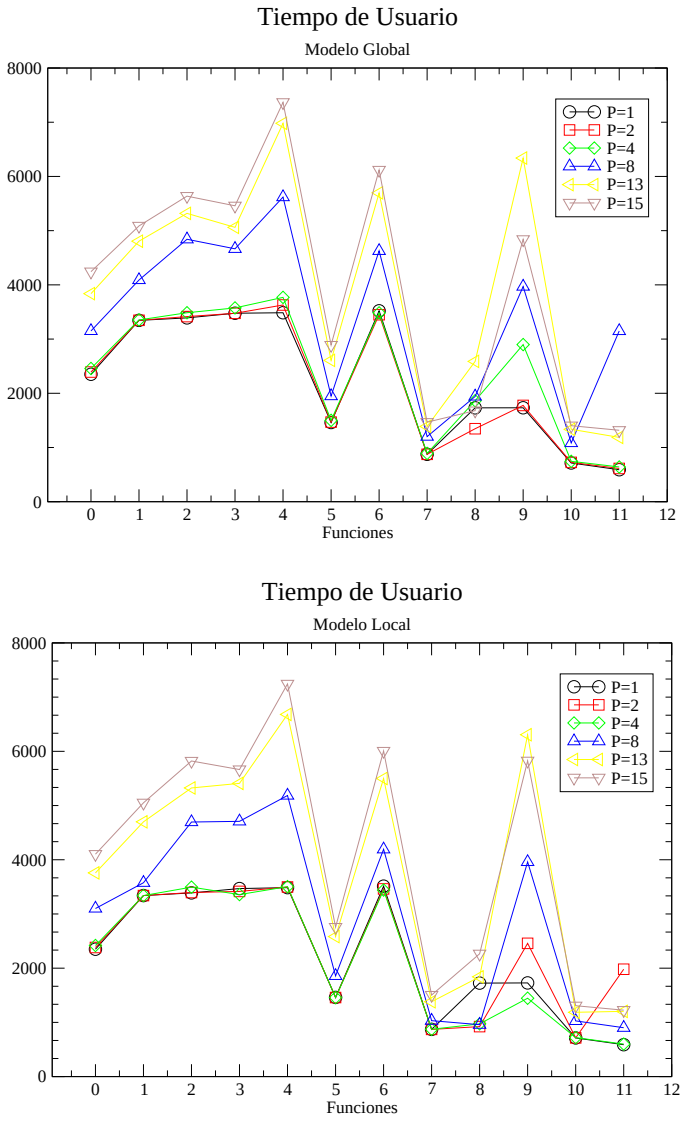


Figura 5.10: Tiempo de usuario en el modelo Global y Local.

para ambos modelos, y además al árbol de trabajo y al árbol final de resultados en el modelo Global, puede entenderse que para funciones en las que su evaluación (intervalos de la función y de sus derivadas) es muy poco costosa, el acceso simultáneo a memoria, por parte de un número de procesadores suficientemente grande, produzca problemas de contención.

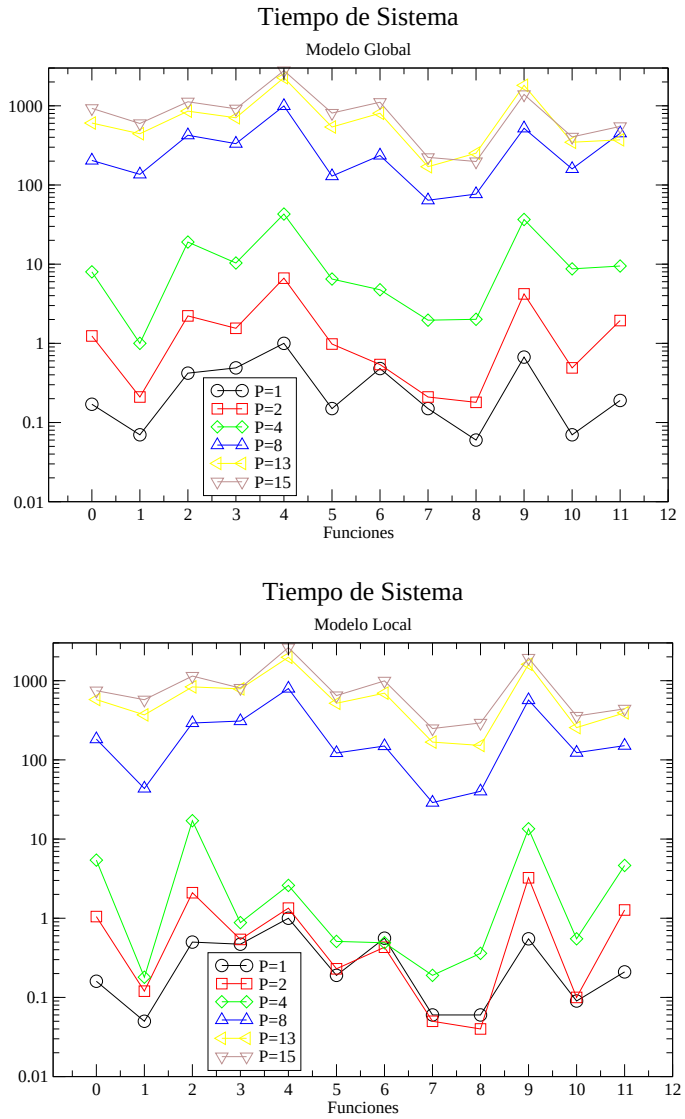


Figura 5.11: Tiempo de sistema en el modelo Global y Local.

Para corroborar esta hipótesis se ha repetido la evaluación de ambos modelos usando las mismas funciones de prueba pero en esta ocasión se ha endurecido el coste computacional de evaluación de cada subproblema. Esto se ha llevado a cabo incorporando un bucle de longitud S en el que se realiza una operación de suma. En primer lugar, para determinar la influencia de este bucle se ha experimentado

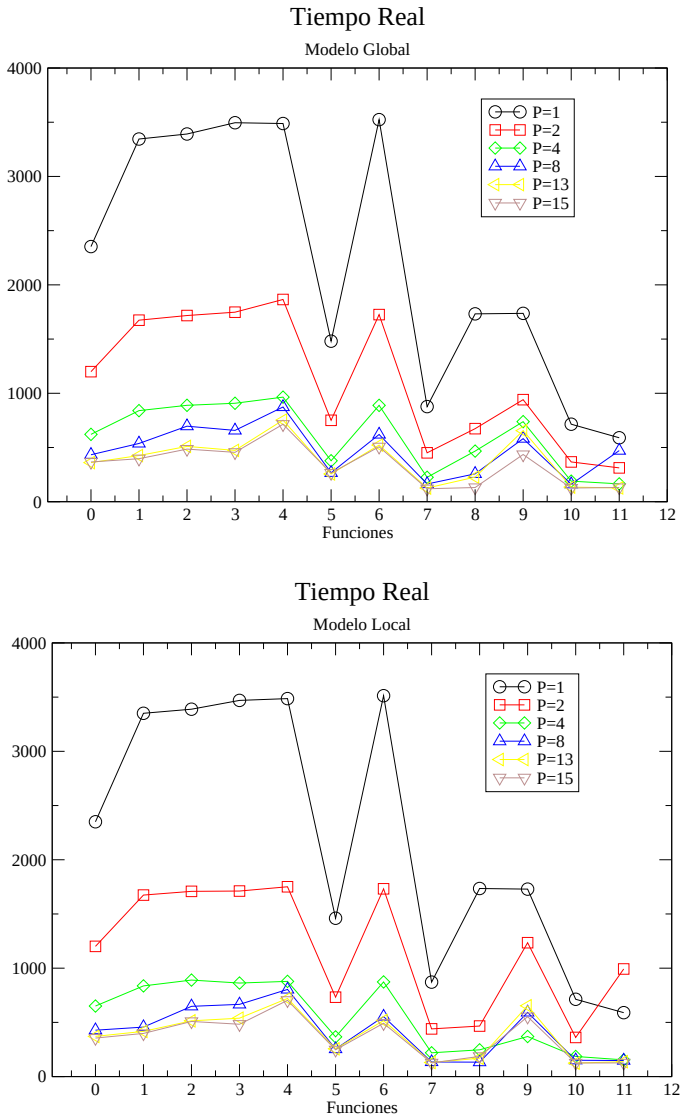


Figura 5.12: Tiempo real para el modelo Global y Local.

con distintos valores de S, usando una única función de prueba, en concreto la función EX2 (número 0). En las Figuras 5.14 y 5.15 se han representado los valores del tiempo del sistema y del Speed-Up en función del número de procesadores y del valor de S, para los dos modelos. En la Figura 5.14 se observa que el tiempo consumido por el sistema disminuye significativamente al aumentar el valor de S.

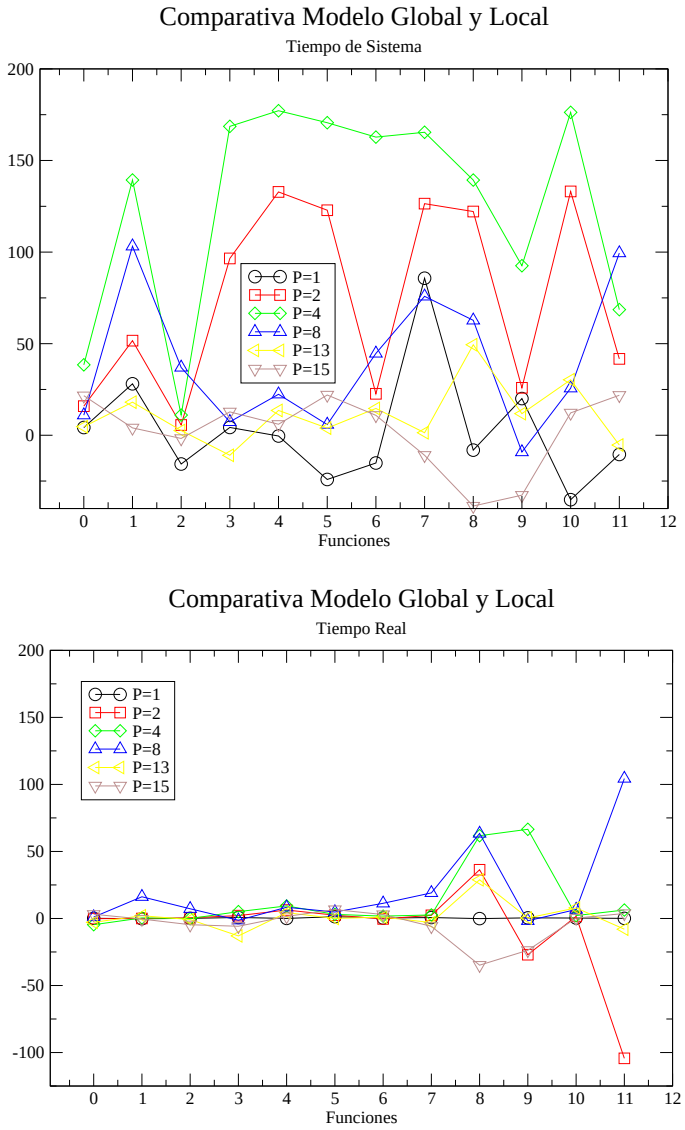


Figura 5.13: Comparativa de los modelos Global y Local en cuanto al tiempo de sistema y tiempo real.

Obsérvese que para $S = 10^6$ casi puede considerarse que el tiempo de sistema se incrementa casi linealmente al incrementar el número de procesadores. Lo mismo se puede decir en relación a los datos de la Figura 5.15, a medida que incrementamos el valor de S, la ganancia en velocidad es mayor. Podemos deducir que al endurecer

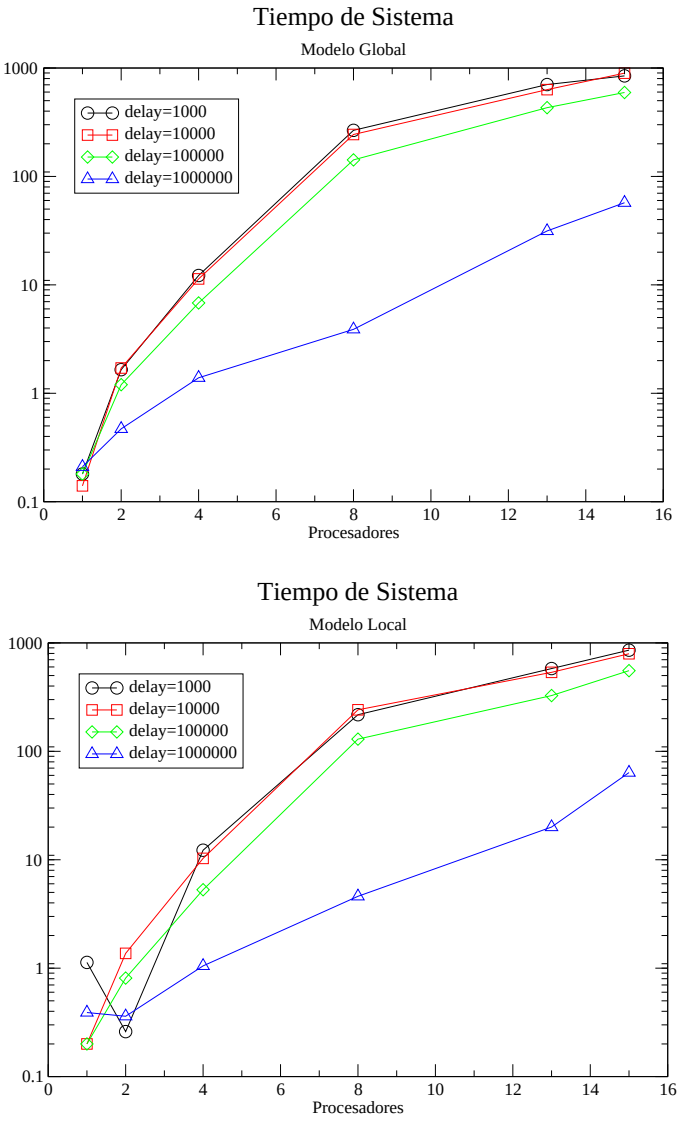


Figura 5.14: Tiempo de sistema en el modelo Global y Local para varios valores de S.

el problema insertando simplemente una suma de 10^6 enteros, damos tiempo al sistema para que el acceso concurrente a memoria no sea un factor crítico.

Para comprobar esta última afirmación, hemos repetido todos los experimentos anteriores, pero esta vez endureciendo las funciones con un bucle que realiza una

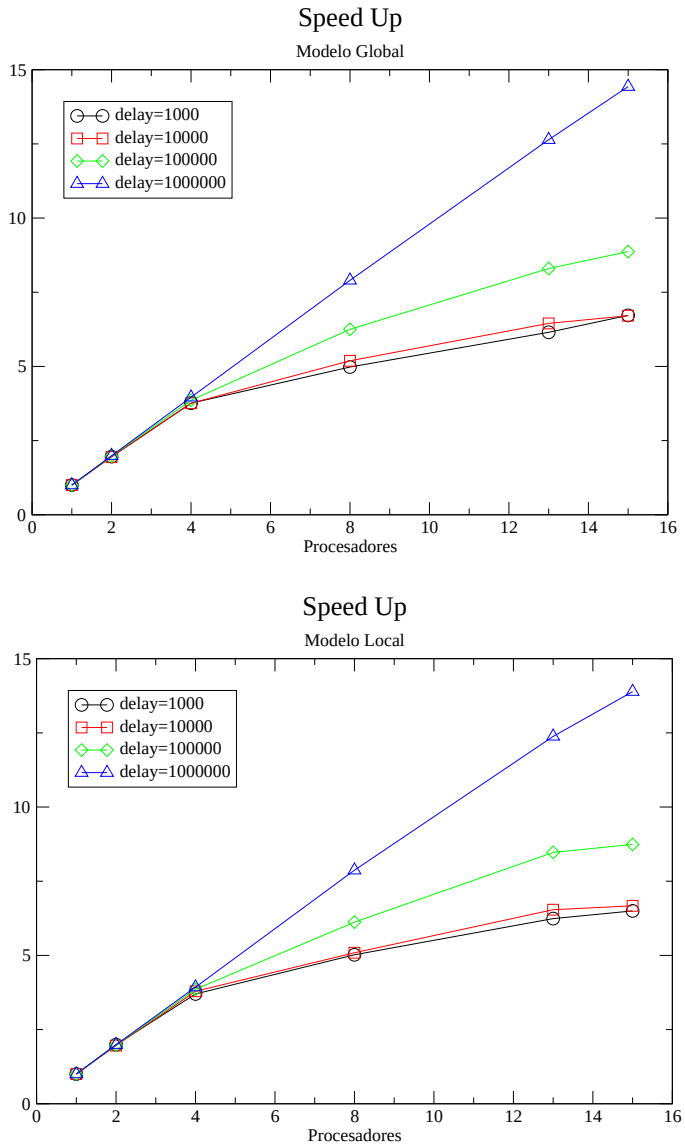


Figura 5.15: Valores del Speed-Up en el modelo Global y Local para varios valores de S.

suma de enteros $S = 10^6$ veces. En primer lugar, en la Figura 5.16 se muestran los valores obtenidos para el Speed-Up. Puede observarse que, en general, se obtiene un comportamiento lineal para casi todas las funciones; incluso algunas funciones

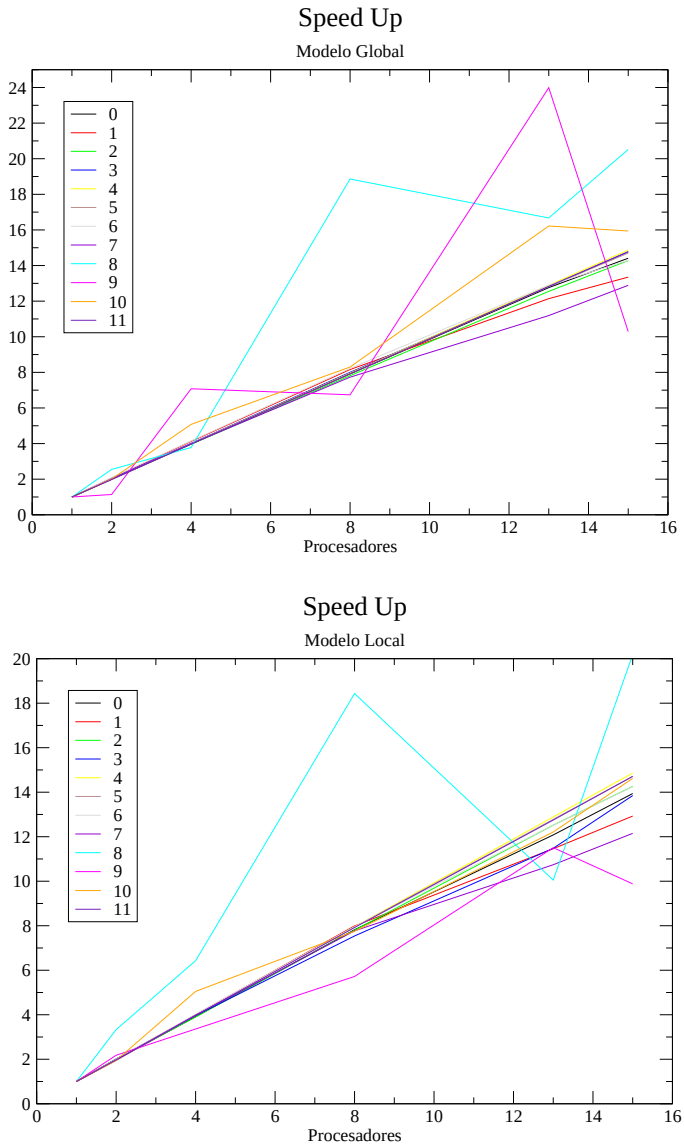


Figura 5.16: Valores del Speed Up para el modelo Global y Local. $S = 10^6$.

que en la evaluación previa presentaban fuertes anomalías, ahora se comportan de manera casi lineal (funciones 10 y 11). Esto puede ser debido a que los problemas de contención de memoria se han reducido y por lo tanto la secuencia de evaluación de subproblemas no se modifica al incrementar el número de procesadores.

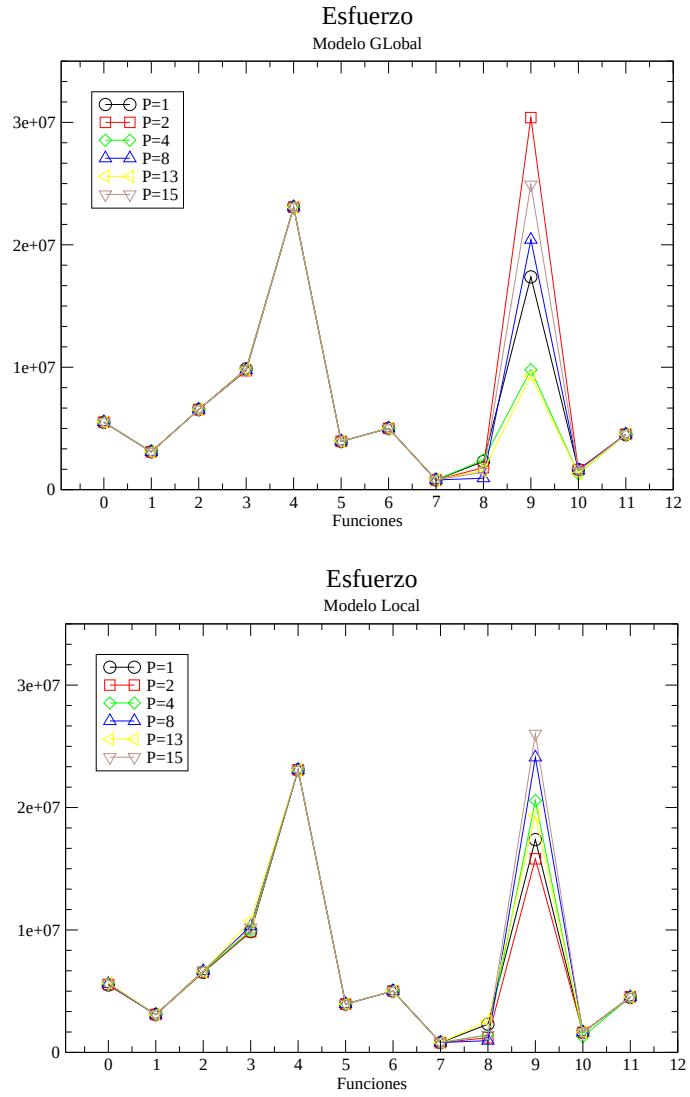


Figura 5.17: Medidas del esfuerzo computacional $Esf(p) = FE + n \cdot GE$ para un conjunto de 12 funciones de prueba, en función del número de procesadores, para los modelos Global y Local. $S = 10^6$.

En cuanto al esfuerzo, la Figura 5.17 muestra el resumen de los datos numéricos de las Tablas C.15 y C.16 del Apéndice C. Los resultados coinciden plenamente con los obtenidos cuando no endurecíamos la función; sólo el conjunto de funciones que presentaban anomalías (Funciones 8, 9, 10 y 11) se han visto ligeramente

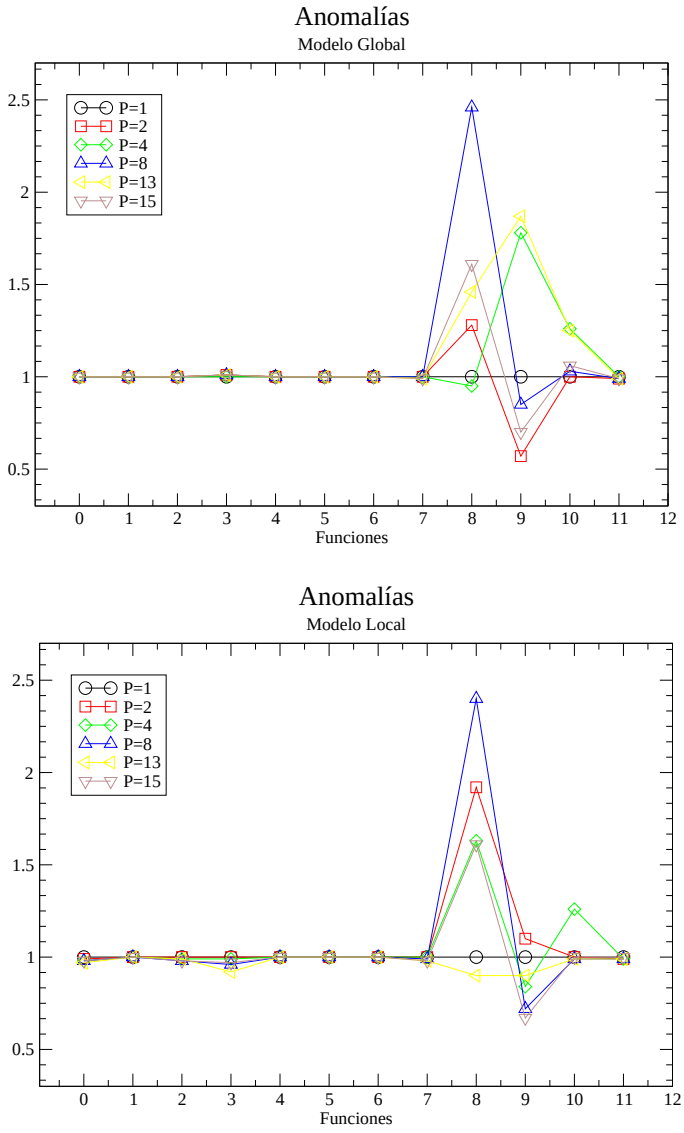


Figura 5.18: Anomalías debidas al paralelismo para los modelos Global y Local. $S = 10^6$.

beneficiadas. Ahora, para este subconjunto de funciones, el esfuerzo computacional requerido cuando se usan varios procesadores en menor que cuando no se incluía el bucle de retardo.

En cuanto a las anomalías, los datos numéricos de las Tablas C.17 y C.18 nos han servido para construir la Figura 5.18 que muestra, que tal como ocurría anteriormente, solamente el conjunto de funciones anómalas se ven afectadas, siguiendo un comportamiento similar al de la evaluación sin bucle de retardo. En el modelo global, las funciones anómalas, presentan una variación que va desde 0.57 para la Función 7 evaluada con 2 procesadores y 2.46 de la Función 8 evaluada con 8 procesadores. El resto de funciones tienen una variación comprendida entre 0.99 y 1.01; anomalías prácticamente inexistentes. Con el modelo Local ocurre algo similar; las funciones anómalas tienen una variación comprendida entre 0.67 para la Función 9 evaluada con 15 procesadores y 2.40 para la Función 8 evaluada con 8 procesadores. El resto de funciones presentan unas anomalías que se encuentran entre 0.92 y 1.00. De esto podemos deducir que el modelo global es mas robusto frente a las anomalías.

En relación con los resultados de la evaluación del desbalanceo puede decirse que se observan ciertas diferencias con respecto a los experimentos en los que no se ha incluido retardo. En las tablas C.19 y C.20 se suministran los valores del desbalanceo de la carga para $S = 10^6$, los cuales han sido representados gráficamente en la Figura 5.19. Aunque los valores de desbalanceo obtenido previamente eran bastante buenos, cuando hemos utilizado un valor de $S = 10^6$ los resultados han mejorado. Concretamente, para el modelo Global, a excepción de la Función 8 que tiene un desbalanceo máximo de 0.015 para 13 procesadores, el resto de funciones no sobrepasan el valor 0.009. En el modelo Local, sólo la Función 10 que presentan unos picos de 0.046 y 0.048 para 4 y 13 procesadores, respectivamente, el resto funciones no supera el 0.021.

En cuanto al análisis de los resultados del tiempo de usuario, del sistema (valores acumulados por procesador) y tiempo real, cuyos valores numéricos se muestran en las Tablas C.21 y C.22 , y que han sido representados gráficamente en las Figuras 5.20, 5.21 y 5.22, se puede decir que el tiempo de usuario no se ve afectado por el incremento del número de procesadores (recordar que es acumulado por procesador). En el modelo Global, solamente la Función 9, que presenta anomalías, ve modificado de manera ostentosa el tiempo de usuario cuando varía el número de procesadores. Para esta función el mayor tiempo de usuario se obtiene cuando se utilizan 2 procesadores, mientras que el menor tiempo de usuario se obtiene para 8 y 13 procesadores. Esto es consecuencia de que solamente presenta anomalías aceleratorias cuando se resuelve con ese número de procesadores, mientras que para el resto de los casos solo presenta anomalías detrimentales. En el modelo Local ocurre algo similar con la Función 9, pero en este caso los menores tiempos de usuario se obtienen para 1 y 2 procesadores.

En relación a los resultados del tiempo del sistema, ahora no son tan elevados como ocurría antes de introducir el retardo. Para el modelo Global, el valor mas alto del tiempo del sistema es de 195.47 segundos para la Función 1 resuelta con 15

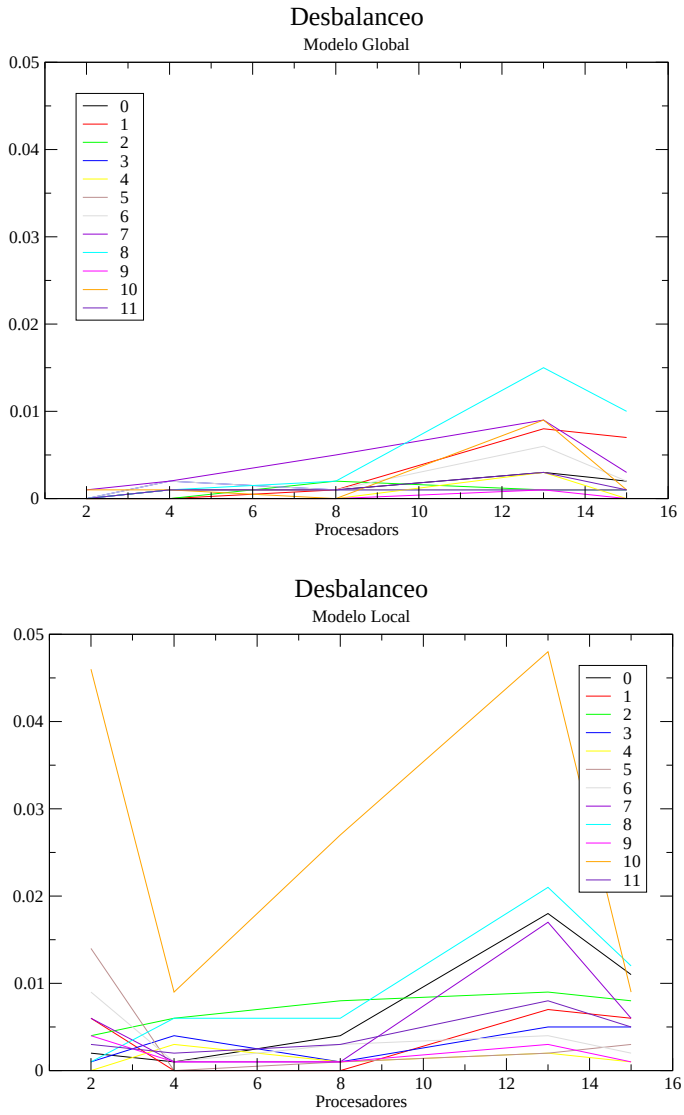


Figura 5.19: Valores del desbalanceo de la carga para el modelo local. $S = 10^6$.

procesadores. Esta misma función presenta para el caso de no introducir retardo, un tiempo de sistema de 598.32 segundos. En el primer caso, representa un 11.76 % y en el segundo caso, representa un 2.3 % sobre el tiempo de usuario. Algo similar ocurre en el modelo Local, el valor mas alto de tiempo de sistema lo obtenemos para la Función 1 resuelta para 15 procesadores, en este caso los porcentajes del tiempo

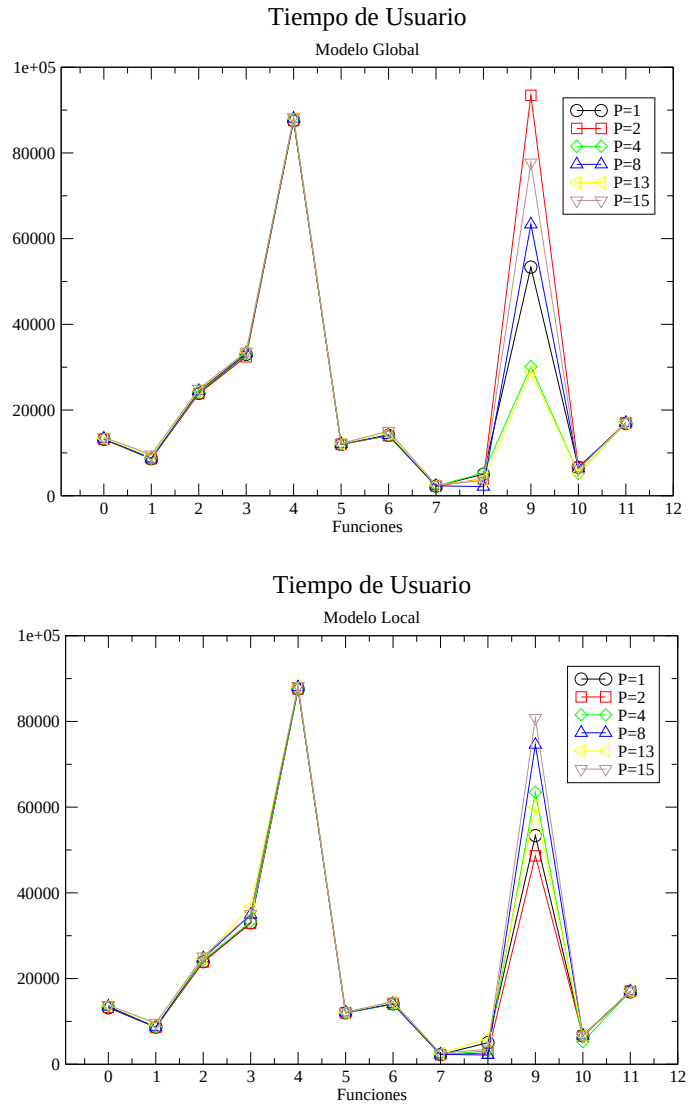


Figura 5.20: Tiempo de usuario en el modelo Global y Local. $S = 10^6$.

de sistema respecto al tiempo de usuario son de 11.37 % y 1.95 %, respectivamente. Se puede deducir de estos resultados que, mientras que para el caso de no introducir retardo, los tiempos de sistema que se obtenían para ejecuciones de mas de 4 procesadores ya empezaban a ser importantes, en el caso de introducir un retardo de $S = 10^6$, los tiempos no representan un porcentaje de tiempo muy elevado ni

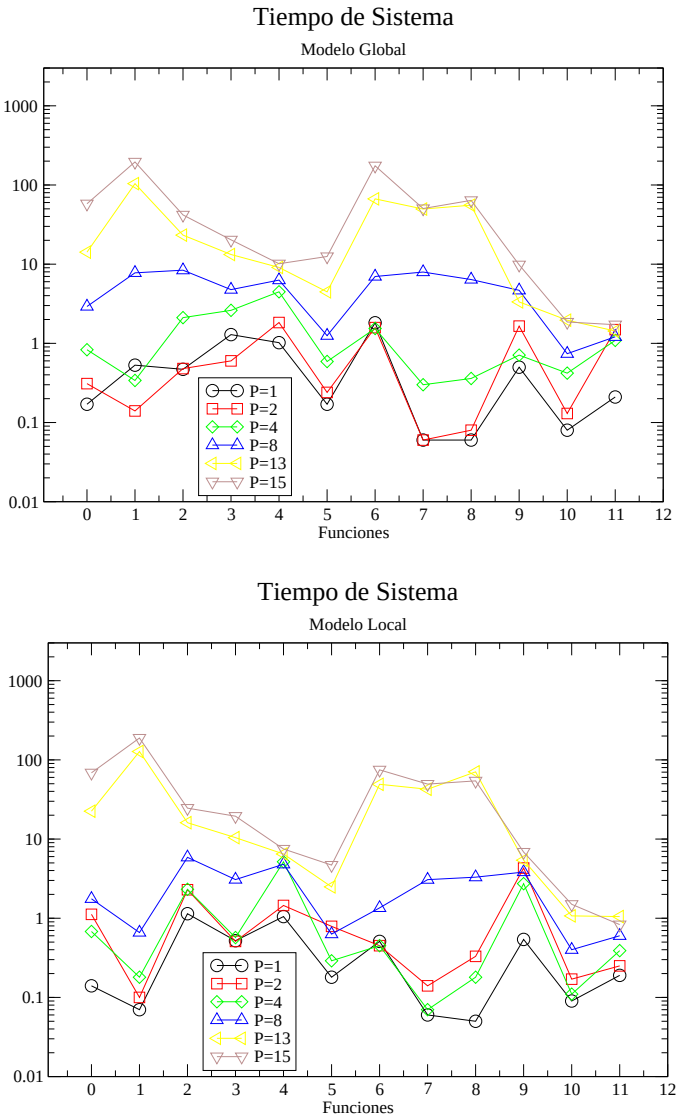


Figura 5.21: Tiempo de sistema en el modelo Global y Local. $S = 10^6$.

siquiera para el caso de usar 15 procesadores.

La Figura 5.22 representa los valores obtenidos para el tiempo real de ejecución medido en segundos para las 12 funciones. Podemos ver que el valor del tiempo real de ejecución para todas las funciones disminuye a medida que se incrementa el

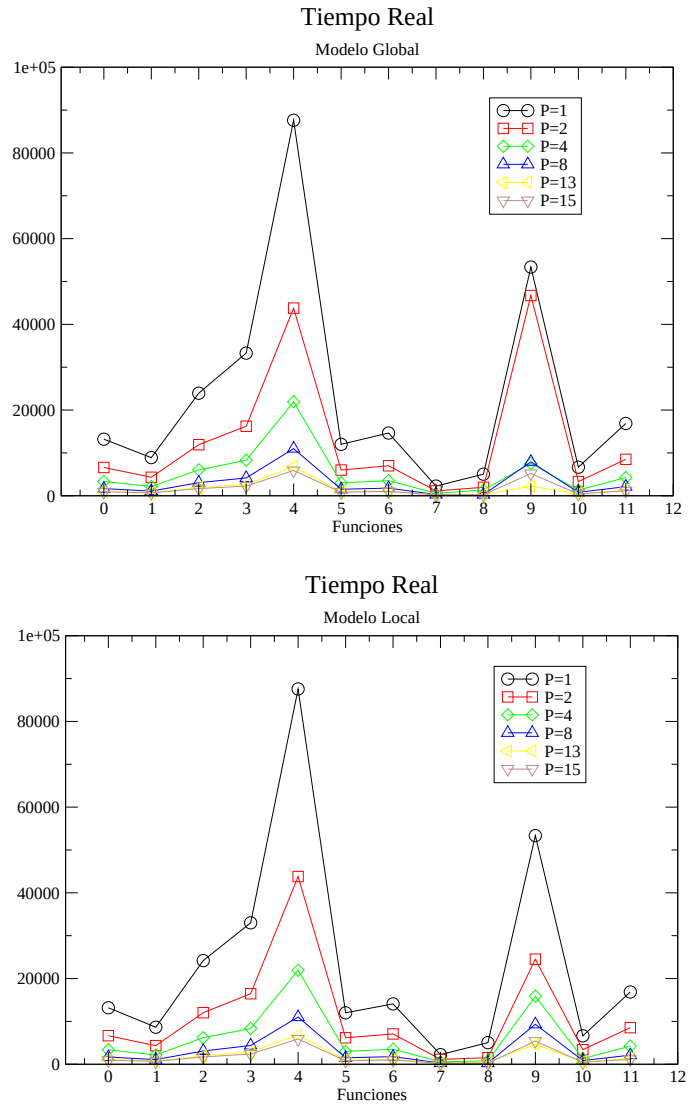


Figura 5.22: Tiempo real para el modelo Global y Local. $S = 10^6$.

número de procesadores. Solamente la Función 9, tanto en el modelo Global como en el Local, presenta ciertas anomalías en cuanto a los tiempos de usuario cuando se incrementa el número de procesadores.

Finalmente, en la Figura 5.23 se ha representado una comparación e los tiempos real y de sistema. Como puede observarse en la gráfica correspondiente a los valores

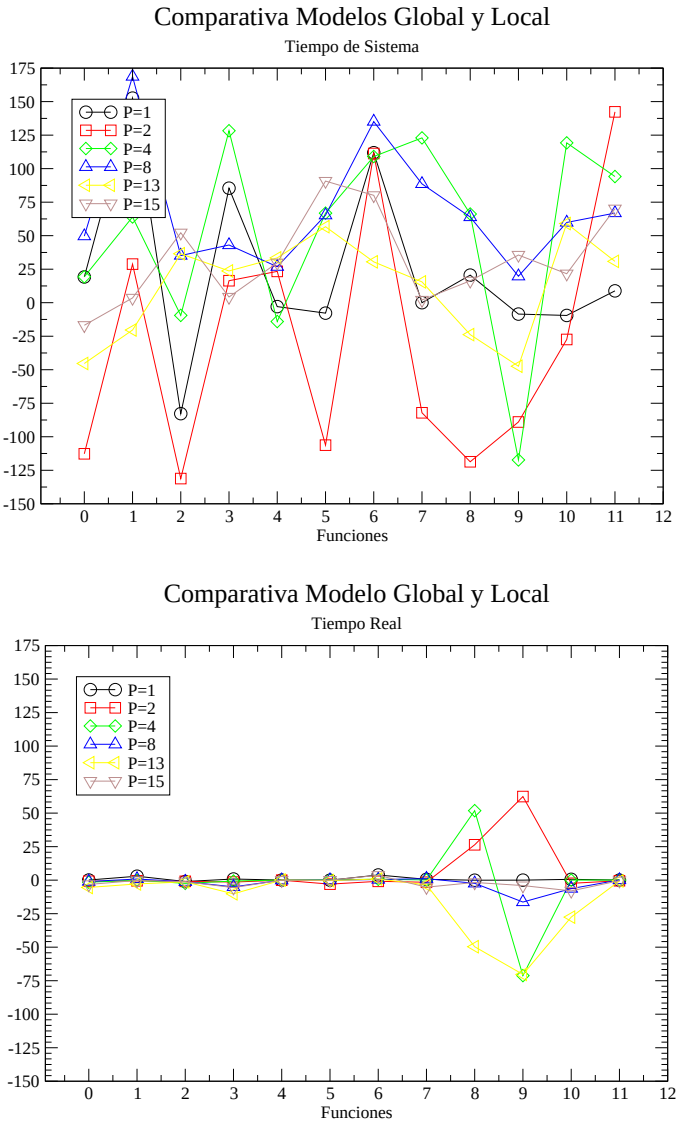


Figura 5.23: Comparativa de los modelos Global y Local en cuanto al tiempo de sistema y tiempo real. $S = 10^6$.

del tiempo real, excepto para las funciones anómalas, no existen grandes diferencias entre los dos modelos. Los únicos casos que merece la pena comentar son: la Función 3 evaluada con 13 procesadores presenta un tiempo real un 10.20 % inferior para el modelo Global y la Funcion 6 evaluada con 1 procesador, para la que el modelo

Local reduce el tiempo real en un 3.85 % respecto al modelo Global.

En relación a los tiempos de sistema, se observa que el modelo Global presenta mayores valores de tiempo de sistema que el modelo Local. El tiempo de sistema del modelo Global es hasta un 168.71 % mayor que en el modelo Local y menor hasta un 131,21 %, pero en muy pocos casos.

De los resultados presentados en esta sección, se puede concluir que los modelos de paralelismo propuestos en esta tesis (Global y Local) obtienen valores del Speed Up casi lineales cuando las funciones a optimizar son suficientemente costosas. No obstante, de la evaluación exhaustiva que se ha llevado a cabo, descrita en este capítulo, se puede hacer los siguientes comentarios generales: El modelo Global da lugar a valores del tiempo de ejecución (tiempo real) y tiempo del sistema mayores que el modelo Local. El esquema de balanceo de la carga propuesto en el modelo Local produce resultados tan satisfactorios como el modelo Global que implícitamente balancea la carga. El modelo Global, en general, tiende a presentar menos anomalías que el modelo Local.

Conclusiones y trabajo futuro

Esta tesis ha abordado la resolución de problemas de optimización global mediante técnicas de Ramificación-Acotación y Aritmética de Intervalos. Las motivaciones de esta tesis son dos:

- Reducir el tiempo de ejecución de los algoritmos actuales.
- Poder resolver en un tiempo razonable problemas más complejos:
  - Por un lado, porque se requiere mayor precisión en los resultados.
  - Por otro, para resolver problemas que, por su complejidad, no era posible resolver en el tiempo establecido.

Para alcanzar estos objetivos se han aportado nuevos estudios que permiten mejorar los algoritmos secuenciales que hacen uso de los valores de la función objetivo y de su derivada. Más concretamente:

- Se ha incorporado una nueva regla de eliminación de la región de búsqueda.
- Esta regla de eliminación también modifica la regla de división tradicionalmente utilizada.
- Se ha mejorado la *función de inclusión*, lo que permite obtener una mejor cota inferior de la función en el intervalo procesado.
- Con esta mejor acotación, se ha mejorado la regla se selección del siguiente intervalo a procesar.

Los experimentos realizados sobre un conjunto extenso de funciones unidimensionales de prueba muestran, que en media, las mejoras introducidas en el algoritmo reducen a la mitad el esfuerzo computacional.

También se han resuelto los problemas de extender los estudios realizados sobre funciones uni-dimensionales al caso multidimensional. Los resultados obtenidos de forma experimental sobre un conjunto de funciones de prueba extenso son prometedores ya que las reducciones obtenidas en el esfuerzo computacional son similares a las obtenidas en el caso unidimensional.

Se ha abordado la paralelización de estos algoritmos para mejorar los objetivos alcanzados. Se ha optado por la realización de una solución sobre memoria compartida por varios motivos:

- Disponibilidad de un hardware de altas prestaciones como es el Altix 330 con 16 procesadores Itanium2 y 64 GB de RAM.
- Es el modelo de programación que deberá extenderse en un futuro próximo si se quiere explotar el paralelismo de los nuevos procesadores multicore, ya que son la tendencia actual para mejorar las prestaciones de los computadores de uso personal.

Para ello se han llevado a cabo los siguientes trabajos:

- Se ha realizado una programación basada en threads.
- Se han modificado las librerías de Aritmética de Intervalos para que sean *thread safe*.
- Se han propuesto dos versiones del algoritmo paralelo basadas en la localidad de las estructuras de datos:
  - Una versión Global, donde las estructuras de datos son compartidas por todos los threads.
  - Una versión Local, donde las estructuras de datos son locales a cada thread. Para esta versión se ha tenido que implementar un balanceador dinámico de la carga que intenta que el número de threads activo sea igual al número de procesadores.

Los resultados obtenidos en ambas versiones paralelas hacen prever una ganancia de velocidad cercana a la lineal para computadores que hacen uso de dos o cuatro procesadores (Dual o Quad-Core).

Cuando el número de procesadores es mayor de cuatro los resultados no son tan esperanzadores, ya que el tiempo requerido por el sistema para la gestión y

mantenimiento de los threads y de la coherencia de los datos es relevante si lo comparamos con el tiempo de usuario. Esto es debido a que, aunque este tipo de problemas de prueba son computacionalmente muy costosos porque hay que evaluar muchos intervalos, el coste computacional por intervalo es muy bajo.

Se han realizado nuevamente todos los experimentos, aumentando artificialmente la carga computacional requerida para evaluar un intervalo. En las pruebas realizadas hasta quince procesadores, se ha demostrando que para problemas de optimización global suficientemente costosos, ambas versiones del algoritmo paralelo alcanzan una ganancia de velocidad próxima a la lineal.

El trabajo realizado en esta tesis ha permitido que se aborden nuevos problemas tanto en el caso secuencial como en el paralelo. Sin ser exhaustivos, nuestro interés a corto plazo se centrará en los siguientes aspectos:

- Extender las nuevas mejoras de los algoritmos secuenciales a problemas multiobjetivo con restricciones.
- Estudiar algoritmos que adapten el número de procesadores asignados a la complejidad del problema en tiempo real.
- Mejorar el algoritmo de balanceo dinámico usado en esta tesis de forma que exista una prioridad en cuanto al thread que realizará el balanceo de la carga.
- Hacer uso de estructuras de datos que permitan un mayor nivel de concurrencia.

Bibliografía

[1] G. Alefeld and J. Herzberger. *Introduction to Interval Computations*. Academic Press, New York, 1983.

[2] G. Amdahl. Validity of the single-processor approach to achieving large-scale computing capabilities. In *AFIPS Conf.*, volume 30, page 483. AFIPS Press, 1967.

[3] R. Aversa, B. Di Martino, N. Mazzocca, and S. Venticinque. A hierarchical distributed-shared memory parallel branch & bound application with PVM and OpenMP for multiprocessor clusters. *Parallel Computing*, 31(10-12):1034–1047, 2005.

[4] S. Berner. New results in verified global optimization. *Computing*, 57(4):323–343, 1996.

[5] S. Berner. Parallel methods for verified global optimization, practice and theory. *Journal of Global Optimization*, 9(1):1–22, 1996.

[6] I.M. Bomze, T. Csendes, R. Horst, and P.M. Pardalos, editors. *Developments in Global Optimization*, volume 18 of *Noncovex optimization and its applications*. Kluwer Academic Publishers, 1997.

[7] F.W. Burton, M.M. Huntbach, G.P. McKeown, and V.J. Rayward-Smith. Parallelisms in Branch-and-Bound algorithms. Technical Report CSA/3/19983, University of East Anglia, Norwich, 1983.

[8] L. G. Casado and I. García. New load balancing criterion for parallel interval global optimization algorithms. In *Proceedings of the 16th IASTED International Conference on Applied Informatics*, pages 321–323, Garmisch-Partenkirchen, Germany, 1998.

[9] L. G. Casado and I. García. Work load balance approaches for branch and bound algorithms on distributed systems. In *Proceedings of the 7th Euromicro Workshop on Parallel and Distributed Processing*, pages 155–162. IEEE Press, 1999.

[10] L. G. Casado, I. García, and T. Csendes. A new multisection technique in interval methods for global optimization. *Computing*, 65(3):263–269, 2000.

[11] L. G. Casado, I. García, and T. Csendes. A heuristic rejection criterion in interval global optimization algorithms. *BIT*, 41(4):683–692, 2001.

[12] L.G. Casado. *Optimización Global Basada en Aritmética de Intervalos y Ramificación y Acotación: Paralelización*. PhD thesis, Universidad de Málaga, Málaga, 1999.

[13] L.G. Casado, I. García, T. Csendes, and V.G. Ruíz. Heuristic rejection in interval global optimization. *Journal of Optimization Theory and Applications (JOTA)*, 118(1):27–43, 2003.

[14] L.G. Casado, I. García, J.A. Martínez, and Ya.D. Sergeyev. New interval analysis support functions using gradient information in a global minimization algorithm. *Journal of Global Optimization*, 25:345–362, 2003.

[15] L.G. Casado, J.A. Martínez, and I. García. Experimenting with a new selection criterion in a fast interval search algorithm. *Journal of Global Optimization*, 19(3):247–264, 2001.

[16] J. Clausen. Do inherently sequential branch-and-bound exist? *Parallel Processing Letters*, 4(1-2):3–13, 1994.

[17] A.R. Conn, N. Gould, and Ph.L. Toint. *LANCELOT: A Fortran Package for Large-Scale Nonlinear Optimization*, volume 17 of *Springer Series in Computational Mathematics*. Springer-Verlag, 1992.

[18] T. Cormen, C. Leiseron, and R. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

[19] H. Cornelius and L. Lohner. Computing the range of values of real functions with accuracy higher than second order. *Computing*, 33(3):331–347, 1984.

[20] R. Corrêa and A. Ferreira. A distributed implementation of asynchronous parallel branch-and-bound. In A. Ferreira and J. Rolim, editors, *Solving Irregular Problems in Parallel: State of the Art*. Kluwer Academic Publisher, Boston (USA), 1995.

[21] R. Corrêa and A. Ferreira. Modeling parallel branch-and-bound for asynchronous implementations. *DIMACS series in Discrete Mathematics and Theoretical Computer Science*, 22:45–56, 1995.

[22] R. Corrêa and A. Ferreira. On the effectiveness of synchronous branch-and-bound algorithms. *Parallel Processing Letters*, 5(3):375–386, 1995.

[23] R. Corrêa and A. Ferreira. Parallel best-first branch and bound in discrete optimization: a framework. In A. Ferreira and P.M. Pardalos, editors, *IRREGULAR '95, Solving Combinatorial Optimization Problems in Parallel – Methods and Techniques*, pages 145–170. Springer LNCS 1054, 1996.

[24] T. Csendes and D. Ratz. Subdivision direction selection in interval methods for global optimization. *SIAM Journal on Numerical Analysis*, 34:922–938, 1997.

[25] P. Daponte, D. Grimaldi, A. Molinaro, and Ya. D. Sergeyev. An algorithm for finding the zero crossing of time signals with Lipschitzeans derivatives. *Measurements*, 16:37–49, 1995.

[26] P. Daponte, D. Grimaldi, A. Molinaro, and Ya. D. Sergeyev. Fast detection of the first zero-crossing in a measurement signal set. *Measurements*, 19(1):29–39, 1996.

[27] A. de Bruin, G.A.P. Kindervater, and H.W.J.M Trienekens. Asynchronous parallel branch and bound and anomalies. Technical Report EUR-CS-95-05, Erasmus University, Department of Computer Science, 1995.

[28] A. de Bruin, G.A.P. Kindervater, and H.W.J.M Trienekens. Towards an abstract parallel branch and bound machine. In A. Ferreira and P.M. Pardalos, editors, *IRREGULAR '95, Solving Combinatorial Optimization Problems in Parallel – Methods and Techniques*, pages 145–170. Springer LNCS 1054, 1996.

[29] C.G. Diderich and M. Gengler. Solving traveling salesman problems using a parallel synchronized branch and bound algorithm. In *International Conference and Exhibition on High-Performance Computing and Networking (HPCN Europe '96)*, pages 633–638, Brussels, Belgium, 1996. Springer-Verlag.

[30] L.C.W. Dixon and G.P. Szego. *Towards Global Optimization, 1.* North-Holland, Amsterdam, 1975.

[31] L.C.W. Dixon and G.P. Szego. *Towards Global Optimization, 2.* North-Holland, Amsterdam, 1978.

[32] I. Dorta. *Esqueletos Paralelos para la Técnica de Ramificación y Acotación.* PhD thesis, Universidad de La Laguna, La Laguna, Tenerife, 2004.

[33] I. Dorta, C. León, C. Rodríguez, and A. Rojas. MPI and OpenMPI implementations of branch-and-bound skeletons. In Gerhard R. Joubert, Wolfgang E. Nagel, F. J. Peters, and W. V. Walter, editors, *PARCO*, volume 13 of *Advances in Parallel Computing*, pages 185–192. Elsevier, 2003.

[34] K. Du. *Cluster Problem in Global Optimization using Interval Arithmetic*. PhD thesis, University of Southeastern Louisiana, 1994.

[35] P. S. Dwyer. Computation with approximate numbers. In P. S. Dwyer, editor, *Linear Computations*, pages 11–35, New York, 1951. Wiley & Sons Inc.

[36] W. L. Eastman. *Linear programming with pattern constraints :–a thesis*. PhD thesis, Harvard University, 1958.

[37] J.S. Ely. *Prospects for Using Variable Precision Interval Software in C++ for Solving some Contemporary Scientific Problems*. PhD thesis, Ohio State University, 1990.

[38] C. A. Floudas and P. M. Pardalos. *A Collection of Test Problems for Constrained Global Optimization Algorithms*, volume 455 of *LNCS*. Springer-Verlag, 1990.

[39] C.A. Floudas and P.M. Pardalos. *Recent Advances in Global Optimization*. Princeton series in Computer Science. Princeton, 1992.

[40] B. Fox, J. Lenstra, A. Rinnoy Kan, and L. Schrage. Branching from the largest upper bound: Folklore and facts. *European Journal of Operational Research*, 2(3):191–194, 1978.

[41] F.C. García López. *Programación en paralelo y técnicas algorítmicas*. PhD thesis, Universidad de la Laguna, 1995.

[42] R.P. Geist. *PVM: Parallel Virtual Machine, A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press, 1994.

[43] M. Gengler and G. Coray. A parallel best-first B&B algorithm and its axiomatization. *Journal of Parallel Algorithms and Applications*, 2:61–80, 1994.

[44] P.C. Gilmore. Optimal and suboptimal algorithms for the quadratic assignment problem. *J. Soc. Indust. Apl. Math.*, 10:305–313, 1962.

[45] S. Gnesi, U. Montanari, and A. Martinelli. Dynamic programming as graph searching: An algebraic approach. *Journal of the ACM*, 28(4):737–751, 1981.

[46] Silicon Graphics. *SGI Altix 3000 System User's Guide*. Silicon Graphics, 2006.

[47] A. Griewank and G. Corlis. Automatic differentiation of algorithms: Theory, implementation and applications. In *Workshop on Automatic Differentiation*, Breckenridge, Philadelphia, 1991. SIAM.

[48] I.E. Grossmann. *Global Optimization in Engineering Design*. Kluwer Academic Publishers, 1996.

[49] R. Hammer, M. Hocks, U. Kulisch, and D. Ratz. *Numerical Toolbox for Verified Computing I.* Springer-Verlag, New York, 1993.

[50] E. Hansen. *Global Optimization Using Interval Analysis*, volume 165 of *Pure and Applied Mathematics.* Marcel Dekker, Inc, New York, NY, 1992.

[51] P. Hansen, B. Jaumard, and S-H. Lu. Global optimization of univariate Lipschitz functions: I. Survey and properties. *Math. Program*, 55:251–272, 1992.

[52] P. Hansen, B. Jaumard, and S-H. Lu. Global optimization of univariate Lipschitz functions: II. New algorithms and computational comparison. *Math. Program*, 55:273–292, 1992.

[53] T. Henriksen and K. Madsen. Parallel algorithms for Global Optimization. *Interval Computations*, 3(5):88–95, 1992.

[54] T. Henriksen and K. Madsen. Use of a depth-first strategy in parallel Global Optimization. Technical Report 92-10, Institute for Numerical Analysis, Technical University of Denmark, 1992.

[55] J. Hormigo, J. Villalva, and E.L. Zapata. A hardware approximation to interval arithmetic for sine and cosine functions. In *SCAN'98, Volume of extended Abstracts*, Budapest, Hungary, 1998. IMACS/GAMM International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics.

[56] R. Horst and P.M. Pardalos, editors. *Handbook of Global Optimization*, volume 2 of *Noncovex Optimization and its Applications.* Kluwer Academic Publishers, Dordrecht, Holland, 1995.

[57] T. Ibaraki. Theoretical comparisons of search strategies in branch and bound algorithms. *Int. J. Comput. Inform. Sci.*, 5:315–344, 1976.

[58] T. Ibaraki. On the computational efficiency of branch and bound algorithms. *J. Oper. Res. Soc.*, 20:16–35, 1977.

[59] T. Ibaraki. The power of dominance relations in branch and bound algorithms. *J. Assoc. Comput. Match.*, 24:264–279, 1977.

[60] T. Ibaraki. *Enumerative approaches to combinatorial optimization*, volume I and II. J.C. Baltzer AG, Basel-Switzerland, 1987.

[61] S. Ibraev. *A New Parallel Method for Verified Global Optimization.* PhD thesis, University of Wuppertal, Wuppertal, 2001.

[62] K. Ichida and Y. Fujii. An interval arithmetic method for global optimization. *Computing*, 23:85–97, 1979.

[63] W.M. Kahan. A more complete interval analysis. In *Lecture Notes for a Summer Course at the University of Michigan*, 1968.

[64] R. B. Kearfott. *Rigorous Global Search: Continuous Problems*. Kluwer Academic Publishers, Dordrecht, Holland, 1996.

[65] R. B. Kearfott. Empirical evaluation of innovations in interval branch and bound algorithms for nonlinear systems. *SIAM Journal on Scientific Computing*, 18(2):574–594, 1997.

[66] O. Knüppel. Bias-basic interval arithmetic subroutines. Technical Report 93.3, University of Hamburg, 1993.

[67] R. Krawczyk and K. Nickel. The centered form in interval arithmetics: Quadratic convergence and inclusion isotonicity. *Computing*, 28(2):117–137, 1982.

[68] U.W. Kullish and W.L. Miranker. The arithmetic of the digital computer: A new approach. *SIAM Rev.*, 1(28):1–40, 1986.

[69] V. Kumar and L. Kanal. The CPD: a unifying formulation for heuristic search, dynamic programming and branch and bound. In *National Conference on Artificial Intelligence*, 1983.

[70] V. Kumar and L. Kanal. A general branch and bound formulation for understanding and synthesizing and/or tree search procedures. *Artificial Intelligence*, 21:179–198, 1983.

[71] T. Lai and S. Shani. Anomalies in parallel branch-and-bound algorithms. *Communications of the ACM*, 27:594–602, 1984.

[72] T. Lai and A. Sprague. Performance of parallel branch-and-bound algorithms. *IEEE Transactions on Computers*, C-34(10):962–964, 1985.

[73] T. Lai and A. Sprague. A note on anomalies on parallel branch-and-bound algorithms with one-to-one bounding functions. *Information Processing Letters*, 23:119–122, 1986.

[74] A.H. Lang and A.G. Doig. An automatic method for solving discrete programming problems. *Econometrica*, 28:497–520, 1960.

[75] E.L. Lawer and D.E. Wood. Branch and bound methods: A survey. *Operation Research*, 14:699–719, 1966.

[76] A.P. Leclerc. *Efficient and Reliable Global Optimization*. PhD thesis, Ohio State University, 1992.

[77] G. Li and B. Wah. Computational efficiency on parallel approximate branch-and-bound algorithms. Technical Report TR-EE 84-6, Purdue University, West Lafayette, 1984.

[78] G. Li and B. Wah. Coping with anomalies in parallel branch-and-bound algorithms. *IEEE Transactions on Computers*, C-35(6):568–573, 1986.

[79] Y. Li and P. Pardalos. Parallel algorithms for the quadratic assignment problem. In P. Pardalos, editor, *Advances in Optimization and Parallel Computing*, pages 177–189. North-Holland, 1992.

[80] J.D.C. Little, K.G. Murty, D.W. Sweeney, and C. Karel. An algorithm for the traveling salesman problem. *Operation Research*, 11:972–989, 1963.

[81] R. Lüling, B. Monien, A. Reinefeld, and S. Tschöke. Mapping tree-structured combinatorial optimization problems onto parallel computers. In A. Ferreira and P.M. Pardalos, editors, *IRREGULAR '95, Solving Combinatorial Optimization Problems in Parallel – Methods and Techniques*. Springer LNCS 1054, 1996.

[82] B. Mans and C. Roucairol. Performances of parallel branch and bound algorithms with best-first search. *Discrete Applied Mathematics*, 66(1):57–76, 1996.

[83] S. Markov. Some applications on extended interval arithmetic to interval iterations. *Computing (Suppl.)*, 2:69–84, 1980.

[84] S. Markov. On interval arithmetic and its applications. In *Proceedings of the 5th Symposium on computer Arithmetic*, U. Michigan, MI, USA, 1981. IEEE.

[85] J. A. Martínez, L. G. Casado, I. García, and B. Tóth. AMIGO: Advanced Multidimensional Interval analysis Global Optimization algorithm. In C.A. Floudas and P.M. Pardalos, editors, *Nonconvex Optimization and Applications Series. Frontiers in Global Optimization*, volume 74, pages 313–326. Kluwer Academic Publishers, 2004.

[86] J.A. Martínez, L.G. Casado, I. García, Ya.D. Sergeyev, and B. Tóth. On an efficient use of gradient information for accelerating interval global optimization algorithms. *Numerical Algorithms*, 37:61–69, 2004.

[87] J.A. Martínez, L.G. Casado, J.A. Álvarez, and I. García. Interval parallel global optimization with Charm++. *Lecture Notes in Computer Science*, 3732:161–168, 2006.

[88] Sun Microsystems. *Sun fire E15k server. http://www.sun.com/server/highend/whitepapers/SUN_SF15K_DS_0103_V3.PDF*. Sun Microsystems, 2003.

[89] L. G. Mitten. Branch and bound methods: general formulation and properties. *Operation Research*, 18:24–34, 1970.

[90] R. Mladineo. Convergence rates of a global optimization algorithm. *Math. Programming*, 54:223–232, 1992.

[91] R. Moore, E. Hansen, and A. Leclerc. Rigorous methods for global optimization. In Christodoulos A. Floudas and Panos M. Pardalos, editors, *Recent advances in global optimization*, Princeton series in computer science, pages 321–342, Princeton, NJ, USA, 1992. Princeton University Press.

[92] R.E. Moore. *Interval analysis*. Prentice-Hall, New Jersey, (USA), 1966.

[93] R.E. Moore. A test for existence of solutions to linear systems. *SIAM Journal of Numerical Analysis*, 14:611–615, 1977.

[94] R.E. Moore. Methods and applications of interval analysis. In *SIAM*, Philadelphia, 1979.

[95] R.E. Moore and S.T. Jones. Safe starting regions for iterative methods. *SIAM Journal of Numerical Analysis*, 14:1051–1065, 1977.

[96] T. Morin and R. Marsten. Branch-and-bound strategies for dynamic programming. *Operations Research*, 24(4):611–627, 1976.

[97] B.A. Murtagh and P.A. Saunders. Minos 5.1 user's guide. Technical Report SOL 83-20R, Dept. Operations Res., Stanford University, 1987.

[98] D.S. Nau, V. Kumar, and L.N. Kanal. General branch and bound and its relation to A* and AO*. *Artificial Intelligence*, 23:29–58, 1984.

[99] A. Neumaier. *Interval Methods for Systems of Equations*. Cambridge University Press, Cambridge, 1990.

[100] A. Neumaier. Second-order sufficient optimality conditions for local and global nonlinear programming. *Journal of Global Optimization*, pages 141–151, 1996.

[101] M. Novoa. *Advances in the Interval Solution of Algebraic Systems*. Univ. of Southwestern Louisiana, 1993.

[102] P.M. Pardalos and J. Crouse. A parallel algorithm for quadratic assignment problem. In *Proceedings Supercomputing*, pages 351–360. ACM Press, 1989.

[103] P.M. Pardalos and J.B. Rosen. *Constrained global optimization: algorithms and applications*. Springer-Verlag New York, Inc., New York, NY, USA, 1987.

[104] D.A. Patterson and J.L. Hennessy. *Organización y diseño de computadores. La interfaz hardware/software*. McGrawHill, 1999.

[105] J. Pekny and D. Miller. A parallel branch and bound algorithm for solving large asymmetric traveling salesman problems. *Mathematical Programming*, 55:17–33, 1992.

[106] S. A. Pijavskii. An algorithm for finding the absolute extremum of a function. *USSR Maths. Math. Physics*, 12:57–67, 1972.

[107] J. D. Pintér. *Global Optimization in Action*, volume 6 of *Noncovex optimization and its applications*. Kluwer Academic Publishers, Dordrecht, The Netherland, 1996.

[108] E. Pruul, G. Nemhauser, and R. Rushmeier. Branch-and-bound and parallel computation: A historical note. *Op. Res. Letters*, 7(2):65–69, 1988.

[109] M.J. Quinn. Analysis and implementation of branch and bound algorithms on a hypercube multicomputer. *IEEE Transactions on Computers*, 39(3):384–387, 1990.

[110] H. Ratschek and J. Rokne. *Computer methods for the range of functions*. Ellis Horwood Ltd.(John Willey & Sons), 1984.

[111] H. Ratschek and J. Rokne. *New Computer Methods for Global Optimization*. Ellis Horwood Ltd., Chichester, England, 1988.

[112] D. Ratz. *Automatische Ergebnisverifikation bei globalen Optimierungsproblemen*. PhD thesis, University of Karlsruhe, 1992.

[113] D. Ratz. On branching rules in second-order branch-and-bound methods for global optimization. *Scientific Computing and Validated Numerics*, pages 221–227, 1995.

[114] D. Ratz. On extended interval arithmetic and inclusion isotonicity. Preprint, U. Karlsruhe, Germany, 1996.

[115] D. Ratz. *Automatic Slope Computation and its Application in Nonsmooth Global Optimization*. Shaker Verlag, Aachen, Germany, 1998.

[116] D. Ratz and T. Csendes. On the selection of subdivision directions in interval branch and bound methods for global optimization. *Journal of Global Optimization*, 7:183–207, 1995.

[117] M.J. Rossman, R.J. Twery, and F.D. Stone. A solution to the traveling salesman problem by combinatorial programming. Chicago, 1958.

[118] C. Roucairol. A parallel branch and bound for the quadratic assignment problem. *Discrete Applied Mathematics*, 18:211–225, 1987.

[119] S.M. Rump. Algorithm for verified inclusions – theory and practice. In R.E. Moore, editor, *Reliability in Computing*, pages 109–126. Academic Press, 1988.

[120] F. Schoen. Stochastic techniques for global optimization: A survey of recent advances. *Journal of Global Optimization*, 11(6):207–228, 1991.

[121] Ya. D. Sergeyev. A one-dimensional deterministic global minimization algorithm. *Comput. Maths. Math. Phys*, 35(5):705–717, 1995.

[122] Ya. D. Sergeyev. Global one-dimensional optimization using smooth auxiliary functions. *Mathematical Programming*, 81(1):127–146, 1998.

[123] S.P. Shary. Algebraic approach to interval linear static identification, tolerance and control problems. *Reliable Computing*, 2(1):3–34, 1996.

[124] Y. Shimano, K. Harada, and R. Hirabayashi. Control schemes in a generalized utility for parallel branch-and-bound algorithms. In *11th International Parallel Symposium (IPPS'97)*, pages 621–627, 1997.

[125] S. Skelboe. Computation of rational interval functions. *BIT*, 14:87–95, 1974.

[126] M. Snir, S.W. Otto, S. Huss-Lederman, D.W. Walker, and J. Dongarra. *MPI The Complete Reference*. MIT Press, 1996.

[127] W. Stalling. *Sistemas Operativos*. Prentice-Hall, 2001.

[128] O. Stauning. *Automatic Validation of Numerical Solutions*. PhD thesis, Dept. of Math. Modelling, Technical University Of Denmark DTU, 1997.

[129] D. Stevenson. IEEE standard for binary floating point arithmetic (IEEE/ANSI 754-1985). Technical report, Floating-Point Working Group, Microprocessor Standards Subcommittee. IEEE, 1985.

[130] R. G. Strongin. *Numerical Methods on Multiextremal Problems*. Nauka, Moscow, 1978.

[131] R. G. Strongin and Ya. D. Sergeyev. *Global optimization with non-convex constraints: Sequential and parallel algorithms*. Kluwer Academic Publishers, Dordrecht, The Netherland, 2000.

[132] T. Sunaga. Theory of interval algebra and its application to numerical analysis. *RAAG memoirs*, 3:29–46, 1958.

[133] A. Törn and A. Žilinskas. *Global Optimization*, volume 3350. Springer-Verlag, Berlin, Germany, 1989.

[134] J. Torrellas and D. Padua. The Illinois aggresive COMA multiprocessors. In *IEEE Symposium on the Frontiers of Massively Parralel Computing*, pages 106–117, 1996.

[135] H. Trienekens. *Parallel Branch and Bound Algorithms*. PhD thesis, Erasmus University, Rotterdam, 1990.

[136] S. Tschöke, R. Lüling, and B. Monien. Solving the traveling salesman problem with a distributed branch-and-bound algorithm on a 1024 processor network. In *9th International Parallel Processing Symposium (IPPS '95)*, pages 182–189, Santa Barbara, 1995.

[137] B. Tóth and T. Csendes. Empirical investigation of the convergence speed of inclusion functions. *Reliable Computing*, 11(4):253–273, 2005.

[138] B. Tóth, J. Fernández, and T. Csendes. Empirical convergence speed of inclusion functions for facility location problems. *Journal of Computational and Applied Mathematics*, 199(2):384–389, 2007. to appear, DOI: 10.1016/j.cam.2005.07.037.

[139] B. Wah and Y. Eva. MANIP - A multicomputer architecture for solving combinatorial extremum-search problems. *IEEE Transactions on Computers*, C-33(5):377–390, 1984.

[140] B. Wah and C. Yu. Stochastic modeling of branch-and-bound algorithms with best-first search. *IEEE Transactions of Software Engineering*, SE-11(0):922–934, 1985.

[141] G. W. Walster, E. R. Hansen, and S. Sengupta. Test results for a global optimization algorithm. In P. T. Boggs, R. H. Byrd, and R. B. Schnabel, editors, *Numerical Optimization 1984*, pages 272–287, Philadelphia, 1985. SIAM.

[142] A. Wiethoff. *Verifizierte Globale Optimierung auf Parallelrechnern*. PhD thesis, Universität Karlsruhe, Karlsruhe, 1998.

[143] B. Wilkinson and M. Allen. *Parallel Programing. Techniques and Applications Using Networked Workstations and Parallel Computer*. Prentice-Hall, 2005.

Apéndice A

Funciones test

Branin (BR) [133]:

Ecuación de la función:

$$f_{BR}(x) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)cos(x_1) + 10$$

Dominio (S): $[-5, 10] \times [0, 15]$

Anchura del rango real: $w(f(S)) \simeq 300$

Existen 3 mínimos globales con:

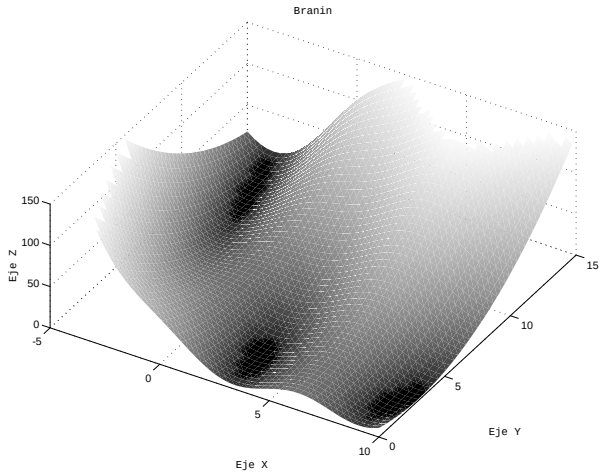
$$f^* = 0.397887$$

$$X^* = \begin{cases} (\text{-}3.14159,\ 12.275)^T \\ (3.14159,\ 2.275)^T \\ (9.42478,\ 2.475)^T \end{cases}$$

Branin 2 (BR2) [30]:

Ecuación de la función:

$$f_{BR2}(x) = \left(1 - 2x_2 + \frac{1}{20}\sin(4\pi x_2) - x_1\right)^2 + \left(x_2 - \frac{1}{2}\sin(2\pi x_1)\right)^2$$



Dominio (S): $-10 \leq x_i \leq 10,\ i = 1, 2$

Anchura del rango real: $w(f(S)) \simeq 1100$

Existen 5 mínimos globales con:

$$f^* = 0$$

$$X^* = \begin{cases} (1,\, 0)^T \\ (0.148696,\, 0.402086)^T \\ (0.402537,\, 0.287408)^T \\ (1.59746,\, –0.287408)^T \\ (1.85130,\, –0.402086)^T \end{cases}$$

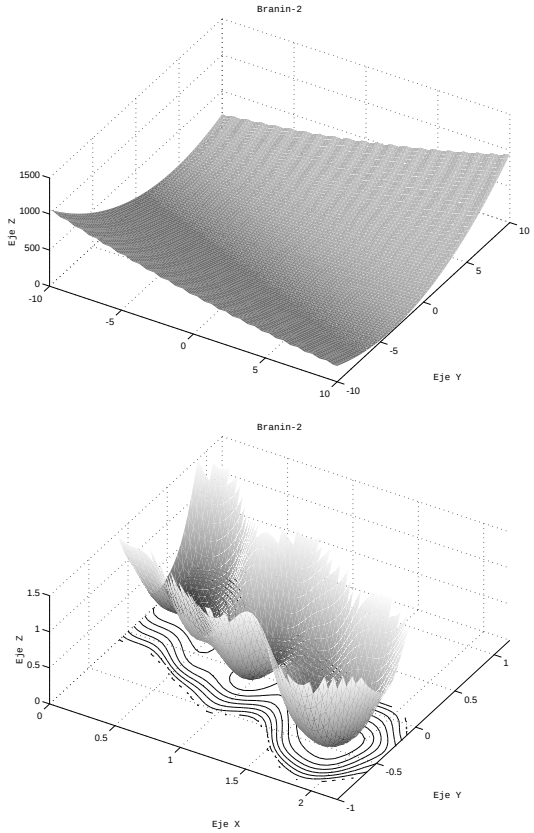
Chichinadze [31]:

Ecuación de la función:

$$f_{CHI}(x) = x_1^2 - 12x_1 + 11 + 10\cos(\frac{\pi}{2}x_1) + 8\sin(5\pi x_1) - \frac{1}{\sqrt{5}}\exp\left(-\frac{\left(x_2 - \frac{1}{2}\right)^2}{2}\right)$$

Dominio (S): $-30.0 \leq x_i \leq 30.0,\ i = 1, 2$

Anchura del rango real: $w(f(S)) \simeq 1300$



Mínimo global: $f^* = -43.315862,\ x^* = (5.90133, 0.5)^T$

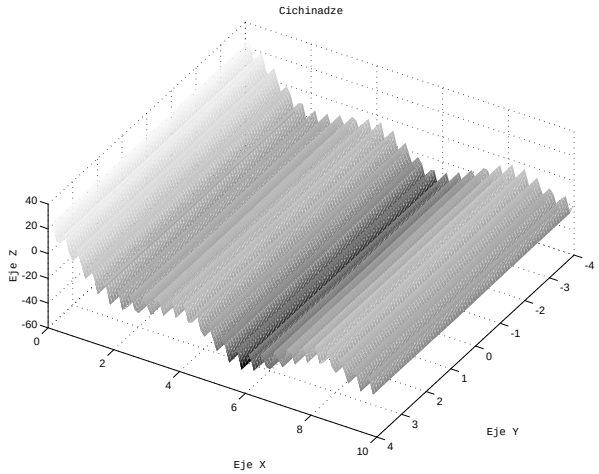
Función de inclusión:

$$F_{BL}(X) = X_1(X_1 - 12) + 11 + 10\cos(\frac{\pi}{2}X_1) + 8\sin(5\pi X_1) - \frac{1}{\sqrt{5}}\exp\left(-\frac{\left(X_2 - \frac{1}{2}\right)^2}{2}\right)$$

EX1 [24]:

Ecuación de la función:

$$f_{EX1}(x) = 0.1((1 - x_1)^2 + \sin(10x_1)) + (11 - x_2)^2 + \sin(10x_2)$$



Dominio (S): $[0.0, 2.0] \times [10.0, 12.0]$

Anchura del rango real: $w(f(S)) \simeq 4$

Mínimo global: $f^* = -1.076182,\ x^* = (1.0976052, 11.14966027)^T$

EX2 [24]:

Ecuación de la función:

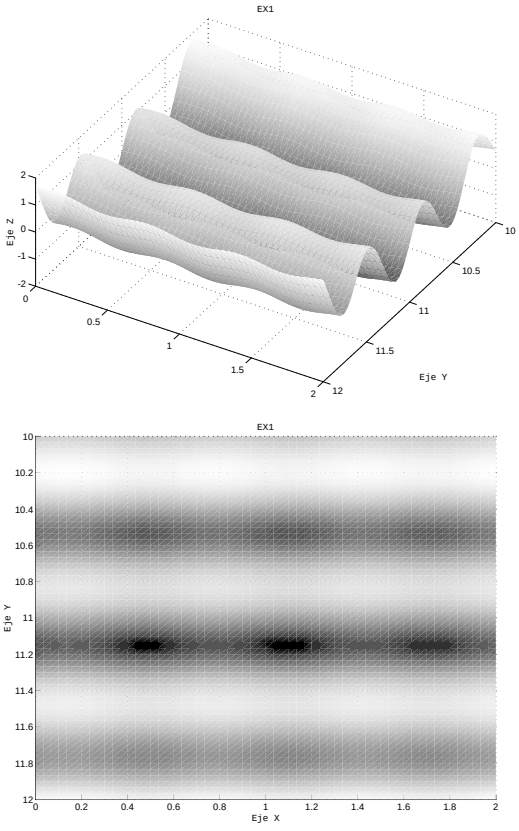
$$f_{EX2}(x) = \sum_{i=1}^{6} \left| f_i - \left(x_1 + \frac{x_2}{\omega_i^{x_3}} + i \left(\omega_i x_4 - \frac{x_5}{\omega_i^{x_3}} \right) \right) \right|^2$$

Dominio (S): $[0.0, 1.0]^2 \times [1.1, 1.3] \times [0.0, 1.0]^2$

Coeficientes:

$$f_i = (5.0 - 5.0i,\ 3.0 - 2.0i,\ 1.5 - 0.5i,\ 1.2 - 0.2i,\ 1.1 - 0.1i)$$
$$w_i = \frac{\pi i}{20},\ i = 1, \ldots, 6$$

Mínimo global: Según los autores $f^* \approx 0.212$ Con nuestros algoritmos se han



obtenido los siguientes valores:

$$\begin{aligned} f^* &= 0.212459838694341 \\ x^* &= (0.606295463259812, \\ &\quad 0.556762695328871, \\ &\quad 1.131807708718409, \\ &\quad 0.750201387128982, \\ &\quad 0.621900754986200)^T \end{aligned}$$

Función de inclusión:

$$F_{EX2}(X) = \sum_{i=1}^{6} \left(yre_i - \left(X_1 + \frac{X_2}{\omega_i^{X_3}} \right) \right)^2 + \left(yim_i - \left(\omega_i X_4 - \frac{X_5}{\omega_i^{X_3}} \right) \right)^2$$

$$
\begin{aligned}
yre_i &= (5.0,\ 3.0,\ 2.0,\ 1.5,\ 1.2,\ 1.1) \\
yim_i &= (-5.0,\ -2.0,\ -1.0,\ -0.5,\ -0.2,\ -0.1) \\
w_i &= (0.05\pi,\ 0.1\pi,\ 0.15\pi,\ 0.2\pi,\ 0.25\pi,\ 0.3\pi)
\end{aligned}
$$

Observación:

El termino $|\cdot|^2$ para números complejos puede calcularse simplemente como la suma de los cuadrados de los términos real e imaginario.

Goldstein-Price [133]:

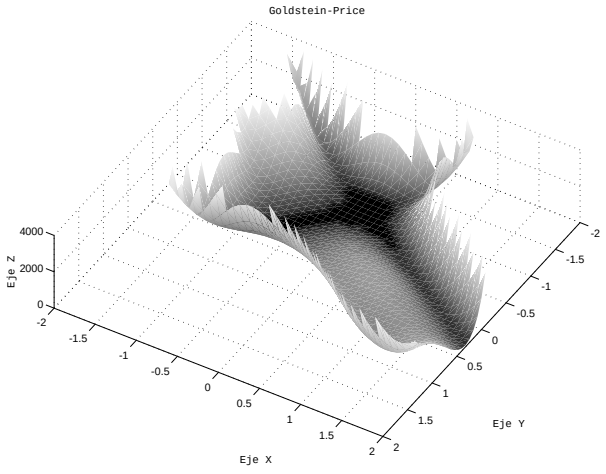
Ecuación de la función:

$$
\begin{aligned}
f_{GP}(x) =& [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \times \\
& [30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]
\end{aligned}
$$

Dominio (S): $-2 \leq x_i \leq 2,\ i = 1, 2$

Anchura del rango real: $w(f(S)) \simeq 10^6$

Mínimo global: $f^* = 3,\ x^* = (0, -1)^T$



Función de inclusión:

$$\begin{aligned} F_{GP}(X) =& (1 + (Y_1 + 1)^2(19 - 14Y_1 + 3Y_1^2)) \times \\ & (30 + Y_2^2(18 - 16Y_2 + 3Y_2^2)) \\ & \text{con } Y_1 = X_1 + X_2, \quad Y_2 = 2X_1 - 3X_2 \end{aligned}$$

Griewank 2 [133]:

Ecuación de la función:

$$f_{G2}(x) = \frac{x_1^2 + x_2^2}{200} - \cos(x_1) \cdot \cos(\frac{x_2}{\sqrt{2}}) + 1$$

Dominio (S): $-100 \leq x_i \leq 100,\ i = 1, 2$

Anchura del rango real: $w(f(S)) \simeq 100$

Mínimo global: $f^* = 0,\ x^* = (0, 0)^T$

Observación: Existen aproximadamente 500 mínimos locales en la región de búsqueda.

Griewank 10 [133]:

Ecuación de la función:

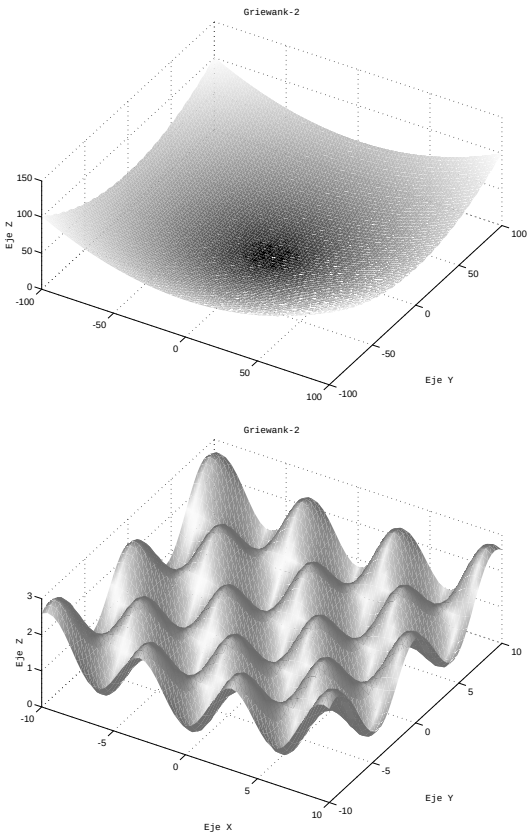
$$f_{G10}(x) = \sum_{i=1}^{10} \frac{x_i^2}{4000} - \prod_{i=1}^{10} \cos(\frac{x_i}{\sqrt{2}}) + 1$$

Dominio (S): $-600 \leq x_i \leq 600,\ i = 1, \ldots, 10$

Anchura del rango real: $w(f(S)) \simeq 900$

Mínimo global: $f^* = 0,\ x^* = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0)^T$

Observación: Existen varios miles de mínimos locales en la región de búsqueda.



Hartman 3 (H3) [133]:

Ecuación de la función:

$$f_{H3}(x) = -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{3} \alpha_{ij}(x_j - p_{ij})^2\right]$$

Coeficientes:

i	α_i	c_i	p_i
1	(3.0,10.0,30.0)	1.0	(0.36890,0.11700,0.26730)
2	(0.1,10.0,35.0)	1.2	(0.46990,0.43870,0.74700)
3	(3.0,10.0,30.0)	3.0	(0.10910,0.87320,0.55470)
4	(0.1,10.0,35.0)	3.2	(0.03815,0.57430,0.88280)

Dominio (S): $0 \leq x_i \leq 1,\ i = 1, .., 3$

Anchura del rango real: $w(f(S)) \simeq 3.9$

Mínimo global:

$$f^* = -3.86278214$$
$$x^* = (0.1146143, 0.55564988, 0.85254695)^T$$

Hartman 6 (H6) [133]:

Ecuación de la función:

$$f_{H6}(x) = -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{6} \alpha_{ij}(x_j - p_{ij})^2\right]$$

Coeficientes:

i	α_i	c_i	p_i
1	(10.00, 3.00,17.00, 3.50, 1.70, 8.00)	1.0	(0.1312,0.1696,0.5569,0.0124,0.8283,0.5886)
2	(0.05,10.00,17.00, 0.10, 0.80,14.00)	1.2	(0.2329,0.4135,0.8307,0.3736,0.1004,0.9991)
3	(3.00, 3.50, 1.70,10.00,17.00, 8.00)	3.0	(0.2348,0.1451,0.3522,0.2883,0.3047,0.6650)
4	(17.00, 8.00, 0.05,10.00, 0.10,14.00)	3.2	(0.4047,0.8828,0.8732,0.5743,0.1091,0.0381)

Dominio (S): $0 \leq x_i \leq 1,\ i = 1, .., 6$

Anchura del rango real: $w(f(S)) \simeq 3.3$

Mínimo global:

$f^* = -3.32236801$

$x^* = (0.2016895, 0.1500106, 0.4768739, 0.2753324, 0.31165161, 0.65730053)^T$

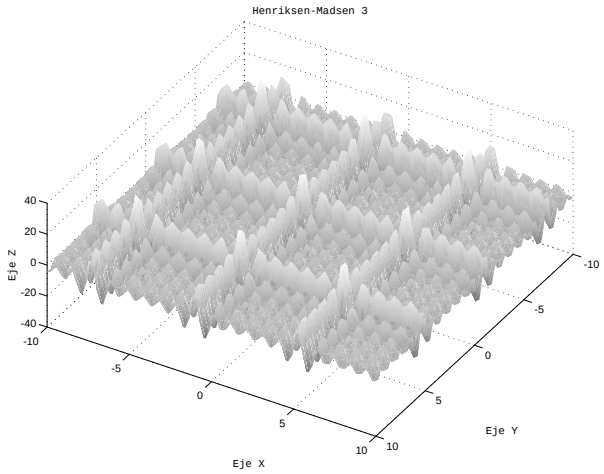
Henriksen-Madsen No. 3 (HM3) [54]:

Ecuación de la función:

$$f_{HM3}(x) = -\sum_{i=1}^{2}\sum_{j=1}^{5} j \sin((j+1)x_i + j)$$

Dominio (S): $-10 \leq x_i \leq 10,\ i = 1, 2$

Anchura del rango real: $w(f(S)) \simeq 50$



Existen 9 mínimos globales con:

$$f^* = -24.06249888$$

$$X^* = \begin{cases} (\ -6.77457614, -0.491390836)^T \\ (\ -0.491390836, 5.79179447)^T \\ (\ 5.79179447, -0.491390836)^T \\ (\ -0.491390836, -0.491390836)^T \\ (\ -0.491390836, -6.77457614)^T \\ (\ -6.77457614, -6.77457614)^T \\ (\ 5.79179447, -6.77457614)^T \\ (\ -6.77457614, 5.79179447)^T \\ (\ 5.79179447, 5.79179447)^T \end{cases}$$

Henriksen-Madsen No. 4 (HM4) [54]:

Ecuación de la función:

$$f_{HM4}(x) = -\sum_{i=1}^{3}\sum_{j=1}^{5} j \sin((j+1)x_i + j)$$

Dominio (S): $-10 \leq x_i \leq 10,\ i = 1, ..3$

Anchura del rango real: $w(f(S)) \simeq 80$

Mínimo global: $f^* = -36.09374832,\ x^* = (0.4913908, 0.4913908, 0.4913908)^T$

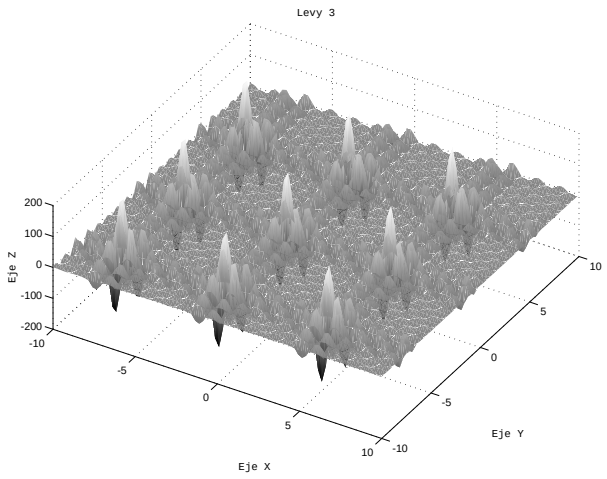
Levy No. 3 (L3) [141]:

Ecuación de la función:

$$f_{L3}(x) = \sum_{i=1}^{5} i\cos((i+1)x_1 + i) \sum_{j=1}^{5} j\cos((j+1)x_2 + j)$$

Dominio (S): $-10 \leq x_i \leq 10,\ i = 1, 2$

Anchura del rango real: $w(f(S)) \simeq 380$



Existen 9 mínimos globales con:

$$f^* = -176.54179313$$

$$X^* = \begin{cases} (4.97647760, 4.85805687)^T \\ (4.97647760, -1.42512842)^T \\ (4.97647760, -7.70831373)^T \\ (-1.30670770, 4.85805687)^T \\ (-1.30670770, -1.42512842)^T \\ (-1.30670770, -7.70831373)^T \\ (-7.58989301, 4.85805687)^T \\ (-7.58989301, -1.42512842)^T \\ (-7.58989301, -7.70831373)^T \end{cases}$$

Levy No. 5 (L5) [141]:

Ecuación de la función:

$$f_{L5}(x) = \sum_{i=1}^{5} i\cos((i+1)x_1 + i) \sum_{j=1}^{5} j\cos((j+1)x_2 + j) + (x_1 + 1.42513)^2 + (x_2) + 0.80032)^2$$

Dominio (S): $-10 \leq x_i \leq 10,\ i = 1, 2$

Anchura del rango real: $w(f(S)) \simeq 460$

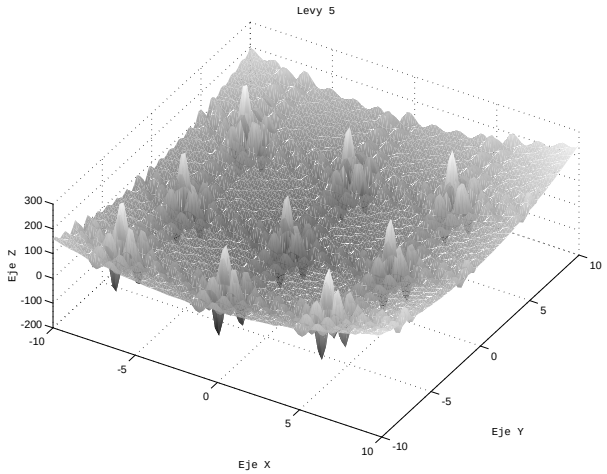
Mínimo global: $f^* = -176.137578,\ x^* = (-1.306853, -1.424845)^T$

Levy No. 8 (L8) [141]:

Ecuación de la función:

$$f_{L8}(x) = \sin^2(\pi y_1) + \sum_{i=1}^{2}(y_i - 1)^2(1 + 10\sin^2(\pi y_{i+1})) + (y_3 - 1)^2$$
$$\text{con } y_i = 1 + (x_i - 1)/4$$

Dominio (S): $-10 \leq x_i \leq 10,\ i = 1, .., 3$



Anchura del rango real: $w(f(S)) \simeq 160$

Mínimo global: $f^* = 0,\ x^* = (1,1,1)^T$

Neumaier 2 [99]:

Ecuación de la función:

$$\begin{aligned} f_{n2}(x) =& (8 - (x_1 + x_2 + x_3 + x_4))^2 + \\ & (18 - ((x_1)^2 + (x_2)^2 + (x_3)^2 + (x_4)^2))^2 + \\ & (44 - ((x_1)^3 + (x_2)^3 + (x_3)^3 + (x_4)^3))^2 + \\ & (144 - ((x_1)^4 + (x_2)^4 + (x_3)^4 + (x_4)^4))^2 + \end{aligned}$$

Dominio (S): $0 \leq x_i \leq 4,\ i = 1,..,4$

Mínimo global: $f^* = 0$

Price (P) [31]:

Ecuación de la función:

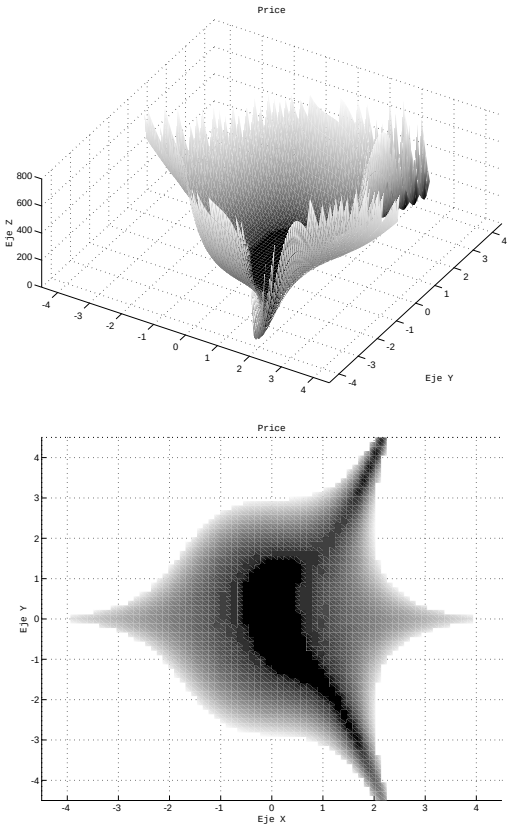
$$f_P(x) = (2x_1^3 x_2 - x_2^3)^2 + (6x_1 - x_2^2 + x_2)^2$$

Dominio (S): $-10 \leq x_i \leq 10,\ i = 1, 2$

Anchura del rango real: $w(f(S)) \simeq 4.4 \cdot 10^8$

Existen 3 mínimos globales con:

$$f^* = 0$$
$$X^* = \begin{cases} (0,\ 0)^T \\ (2,\ 4)^T \\ (1.464352,\ –2.506012)^T \end{cases}$$



Función de inclusión:

$$F_P(X) = \left(X_2\left(2X_1^3 - X_2^2\right)\right)^2 + \left(X_2\left(1 - X_2\right) + 6X_1\right)^2$$

return sqr(x[1])+sqr(x[2])-cos(18.0*x[1])-cos(18.0*x[2]);

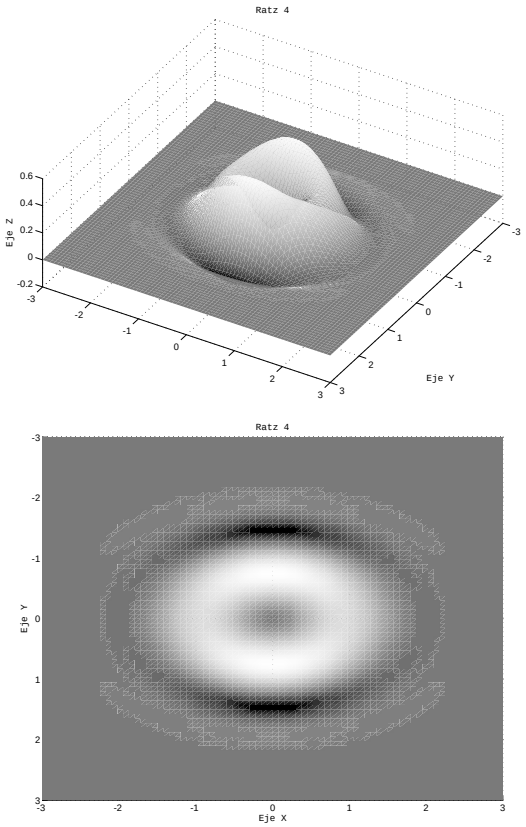
Rastrigins [133]:

$$f_{RT}(x) = (x_1)^2 + (x_2)^2 - \cos(18x_1) - \cos(18x_2)$$

Dominio (S): $-1 \leq x_i \leq 1,\ i = 1, 2$

Mínimo global: $f^* = -2,\ x^* = (0, 0)^T$

Ratz No. 4 (R4) [116]:



Ecuación de la función:

$$f_{R4}(x) = \sin(x_1^2 + 2x_2^2)\exp(-x_1^2 - x_2^2)$$

Dominio (S): $-3 \leq x_i \leq 3,\ i = 1, 2$

Anchura del rango real: $w(f(S)) \simeq 0.6$

Existen 2 mínimos globales con:

$$f^* = -0.10689134$$
$$X^* = \begin{cases} (0.0, \text{-}1.4575221047)^T \\ (0.0, 1.4575221047)^T \end{cases}$$

Ratz No. 5 (R5) [116]:

Ecuación de la función:

$$f_{R5}(x) = \left(\sin^2 \left(\pi \frac{x_1 + 3}{4} \right) + \sum_{i=1}^{2} \left(\frac{x_i - 1}{4} \right)^2 \left(1 + 10 \sin^2 \left(\pi \frac{x_{i+1} + 3}{4} \right) \right) \right)^2$$

Dominio (S): $[-10, 10]^3$

Mínimo global:

$$f^* = 0.0$$
$$x^* = (1, 1, [-10, 10])^T$$

Ratz No. 6 (R6) [116]:

Ecuación de la función:

$$f_{R6}(x) = \left(\sin^2 \left(\pi \frac{x_1 + 3}{4} \right) + \sum_{i=1}^{4} \left(\frac{x_i - 1}{4} \right)^2 \left(1 + 10 \sin^2 \left(\pi \frac{x_{i+1} + 3}{4} \right) \right) \right)^2$$

Dominio (S): $[-10, 10]^5$

Mínimo global:

$$f^* = 0.0$$
$$x^* = (1, 1, 1, 1, [-10, 10])^T$$

Ratz No. 7 (R7) [116]:

Ecuación de la función:

$$f_{R7}(x) = \left(\sin^2\left(\pi\frac{x_1+3}{4}\right) + \sum_{i=1}^{6}\left(\frac{x_i-1}{4}\right)^2\left(1+10\sin^2\left(\pi\frac{x_{i+1}+3}{4}\right)\right)\right)^2$$

Dominio (S): $[-10,10]^7$

Mínimo global:

$$f^* = 0.0$$
$$x^* = (1,1,1,1,1,1,[-10,10])^T$$

Ratz No. 8 (R8) [116]:

Ecuación de la función:

$$f_{R8}(x) = \left(\sin^2\left(\pi\frac{x_1+3}{4}\right) + \sum_{i=1}^{8}\left(\frac{x_i-1}{4}\right)^2\left(1+10\sin^2\left(\pi\frac{x_{i+1}+3}{4}\right)\right)\right)^2$$

Dominio (S): $[-10,10]^9$

Mínimo global:

$$f^* = 0.0$$
$$x^* = (1,1,1,1,1,1,1,1,[-10,10])^T$$

Rosenbrock 2 [30]:

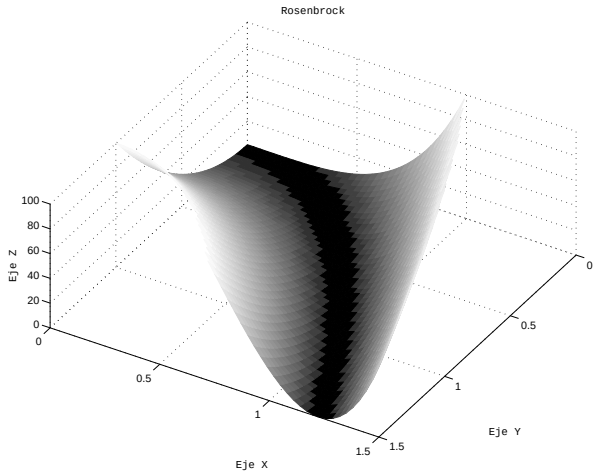
Ecuación de la función:

$$f_{RB}(x) = 100(x_1^2 - x_2)^2 + (x_1-1)^2$$

Dominio (S): $-2 \leq x_i \leq 8,\ i = 1,2$

Anchura del rango real: $w(f(S)) \simeq 4 \cdot 10^5$

Mínimo global: $f^* = 0,\ x^* = (1,1)^T$



Rosenbrock 10 [30]:

$$f_{RB10}(x) = \sum_{i=2}^{10} 100(x_i^2 - x_{i-1})^2 + (x_{1-1} - 1)^2$$

Dominio (S): $-2 \leq x_i \leq 2,\ i = 1, .., 2$

Mínimo global: $f^* = 0,\ x^* = (1, 1, 1, 1, 1, 1, 1, 1, 1, 1)^T$

Schwefel No. 1.2 [141]:

Ecuación de la función:

$$f_{S1.2}(x) = (x_1^2) + (x_1 + x_2)^2 + (x_1 + x_2 + x_3)^2 + (x_1 + x_2 + x_3 + x_4)^2$$

Dominio (S): $-5 \leq x_i \leq 10,\ i = 1, .., 4$

Mínimo global: $f^* = 0,\ x^* = (0, 0)^T$

Schwwfel No. 2.1 (Beale) [141]:

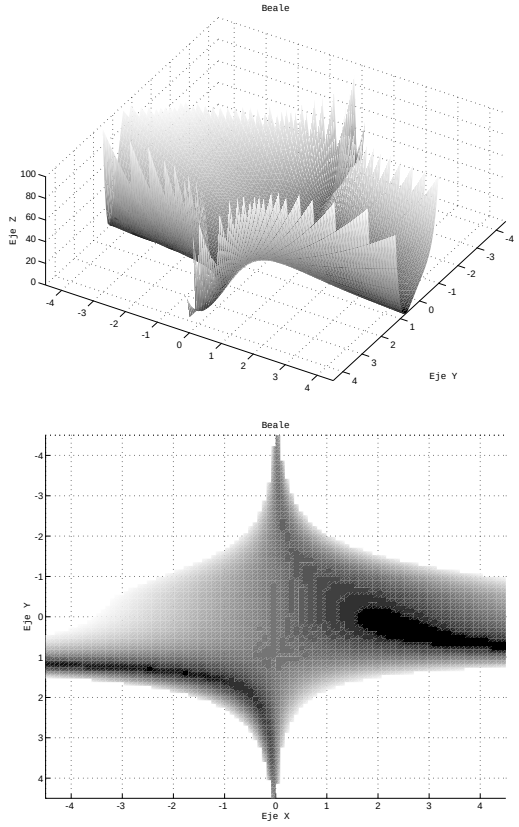
Ecuación de la función:

$$f_{S2.1}(x) = (1.5 - x_1 + x_1x_2)^2 + (2.25 - x_1 + x_1x_2^2)^2 + (2.625 - x_1 + x_1x_2^3)^2$$

Dominio (S): $-4.5 \leq x_i \leq 4.5,\ i = 1, 2$

Anchura del rango real: $w(f(S)) \simeq 1.8 \cdot 10^5$

Mínimo global: $f^* = 0,\ x^* = (3, 0.5)^T$



Función de inclusión:

$$F_{S2.1}(X) = (1.5 + X_1(X_2 - 1))^2 + \left(2.25 + X_1(X_2^2 - 1)\right)^2 + \left(2.625 + X_1(X_2^3 - 1)\right)^2$$

Schwefel No. 2.5 (Booth) [141]:

Ecuación de la función:

$$f_{S2.5}(x) = (x_1 + 2x_1 - 7)^2 + (2x_1 + x_2 - 5)^2$$

Dominio (S): $-10 \leq x_i \leq 10,\ i = 1, 2$

Mínimo global: $f^* = 0,\ x^* = (1, 3)^T$

Schwefel No. 2.7 (Box 3D) [116]:

Ecuación de la función:

$$f_{S2.7}(x) = \sum_{i=1}^{10} \left(\exp\left(\frac{-kx_1}{10}\right) - \exp\left(\frac{-kx_2}{10}\right) - \left(\exp\left(\frac{-k}{10}\right) - \exp\left(-k\right)\right) x_3 \right)^2$$

Dominio (S): $-10 \leq x_i \leq 30,\ i = 1, 2, 3$

Schwefel No. 2.10 (Kowalik) [141]:

Ecuación de la función:

$$f_{S2.10}(x) = \sum_{i=1}^{11} \left(a_i - x_1 \frac{b_i^2 + b_i x_2}{b_i^2 + b_i x_3 + x_4} \right)^2$$

Coeficientes:

i	a_i	$1/b_i$
1	0.1957	0.25
2	0.1947	0.50
3	0.1735	1.00
4	0.1600	2.00
5	0.0844	4.00
6	0.0627	6.00
7	0.0456	8.00
8	0.0342	10.00
9	0.0323	12.00
10	0.0235	14.00
11	0.0246	16.00

Dominio (S): $0 \leq x_i \leq 0.42,\ i = 1, .., 4$

Anchura del rango real: $w(f(S)) \simeq 36$

Mínimo global:

$f^* = 3.074859878 \cdot 10^{-4}$

$x^* = (0.192833452982, 0.19083623878, 0.12311729627, 0.13576598998)^T$

Schwefel No. 2.14 (Powell) [133]:

Ecuación de la función:

$$f_{S2.14}(x) = (x_1 + 10x_2)^2 + 5\,(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10\,(x_1 - x_4)^4$$

Dominio (S): $-4 \leq x_i \leq 5,\ i = 1, .., 4$

Mínimo global: $f^* = 0,\ x^* = (0, 0)^T$

Schwefel No. 2.18 (Matyas) [141]:

Ecuación de la función:

$$f_{S2.18}(x) = 0.26(x_1^2 + x_2^2) - 0.48x_1x_2$$

Dominio (S): $-10 \leq x_i \leq 10,\ i = 1, 2$

Anchura del rango real: $w(f(S)) \simeq 100$

Mínimo global: $f^* = 0,\ x^* = (0, 0)^T$

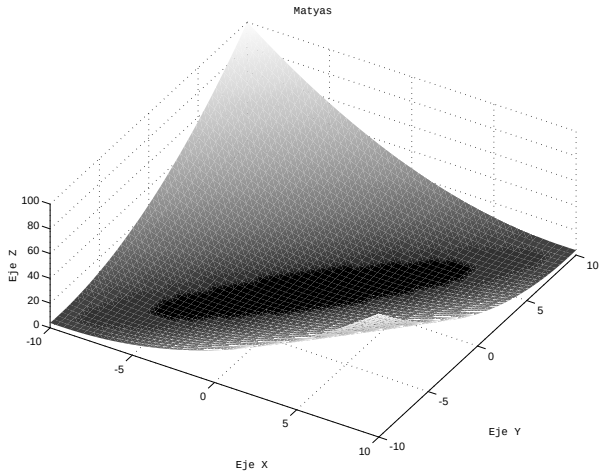
Función de inclusión:

$$F_{S2.18}(X) = 0.26(X_1 - X_2)^2 + 0.04X_1X_2$$

Schwefel No. 3.1 [141]:

Ecuación de la función:

$$f_{S3.1}(x) = \sum_{i=1}^{3} \left[\left(x_1 - x_i^2\right)^2 + (x_i - 1)^2 \right]$$



Dominio (S): $-10 \leq x_i \leq 10,\ i = 1, .., 3$

Mínimo global: $f^* = 0,\ x^* = (1, 1, 1)^T$

Schwefel No. 3.1p [141]:

Ecuación de la función:

$$f_{S3.1p}(x) = \sum_{i=1}^{3} \left[\left(x_1 \pi - x_i^2 \pi \right)^2 + \left(x_i \pi - \pi \right)^2 \right]$$

Dominio (S): $-10 \leq x_i \leq 10,\ i = 1, .., 3$

Mínimo global: $f^* = 0,\ x^* = (1, 1, 1)^T$

Schwefel No. 3.2 [141]:

Ecuación de la función:

$$f_{S3.2}(x) = \sum_{i=2}^{3} \left[\left(x_1 - x_i^2 \right)^2 + \left(x_i - 1 \right)^2 \right]$$

Dominio (S): $-10 \leq x_i \leq 10,\ i = 1, .., 3$

Mínimo global: $f^* = 0,\ x^* = (1,1,1)^T$

Schwefel No. 3.7 [141]:

Ecuación de la función:

$$f_{S3.7}(x) = \sum_{i=1}^{5}(x_i)^{10}$$

Dominio (S): $-0.5 \leq x_i \leq 0.4,\ i = 1,..5$

Mínimo global: $f^* = 0,\ x^* = (0,0,0,0,0)^T$

Shekel 5 (S5) [133]:

Ecuación de la función:

$$f_{S5}(x) = -\sum_{i=1}^{5}\frac{1}{(x-a_i)(x-a_i)^T + c_i}$$

Coeficientes:

i	a_i	c_i
1	(4.0,4.0,4.0,4.0)	0.1
2	(1.0,1.0,1.0,1.0)	0.2
3	(8.0,8.0,8.0,8.0)	0.2
4	(6.0,6.0,6.0,6.0)	0.4
5	(3.0,7.0,3.0,7.0)	0.4

Dominio (S): $0 \leq x_i \leq 10,\ i = 1,..,4$

Anchura del rango real: $w(f(S)) \simeq 10$

Mínimo global:

$$f^* = -10.15319967$$
$$x^* = (4.0000371, 4.0001332, 4.0000371, 4.0001332)^T$$

Observación: Los 5 mínimos locales con $f \approxeq -1/c_i$ están aproximadamente en a_i.

Shekel 7 (S7) [133]:

Ecuación de la función:

$$f_{S7}(x) = -\sum_{i=1}^{7} \frac{1}{(x-a_i)(x-a_i)^T + c_i}$$

Coeficientes:

i	a_i	c_i
1	(4.0,4.0,4.0,4.0)	0.1
2	(1.0,1.0,1.0,1.0)	0.2
3	(8.0,8.0,8.0,8.0)	0.2
4	(6.0,6.0,6.0,6.0)	0.4
5	(3.0,7.0,3.0,7.0)	0.4
6	(2.0,9.0,2.0,9.0)	0.6
7	(5.0,5.0,3.0,3.0)	0.3

Dominio (S): $0 \leq x_i \leq 10,\ i = 1,..,4$

Anchura del rango real: $w(f(S)) \simeq 10$

Mínimo global:

$$f^* = -10.40294056$$
$$x^* = (4.0005729, 4.0006893, 3.999489, 3.9996061)^T$$

Observación: Los 7 mínimos locales con $f \cong -1/c_i$ están aproximadamente en a_i.

Shekel 10 (S10) [133]:

Ecuación de la función:

$$f_{S7}(x) = -\sum_{i=1}^{10} \frac{1}{(x-a_i)(x-a_i)^T + c_i}$$

Coeficientes:

i	a_i	c_i
1	(4.0,4.0,4.0,4.0)	0.1
2	(1.0,1.0,1.0,1.0)	0.2
3	(8.0,8.0,8.0,8.0)	0.2
4	(6.0,6.0,6.0,6.0)	0.4
5	(3.0,7.0,3.0,7.0)	0.4
6	(2.0,9.0,2.0,9.0)	0.6
7	(5.0,5.0,3.0,3.0)	0.3
8	(8.0,1.0,8.0,1.0)	0.7
9	(6.0,2.0,6.0,2.0)	0.5
10	(7.0,3.6,7.0,3.6)	0.5

Dominio (S): $0 \leq x_i \leq 10,\ i = 1, .., 4$

Anchura del rango real: $w(f(S)) \simeq 10$

Mínimo global:

$$f^* = -10.53640981$$
$$x^* = (4.000746, 4.00059, 3.999663, 3.999509)^T$$

Observación: Los 10 mínimos locales con $f \approxeq -1/c_i$ están aproximadamente en a_i.

Simplified Rosenbrock (SRB) [30]:

Ecuación de la función:

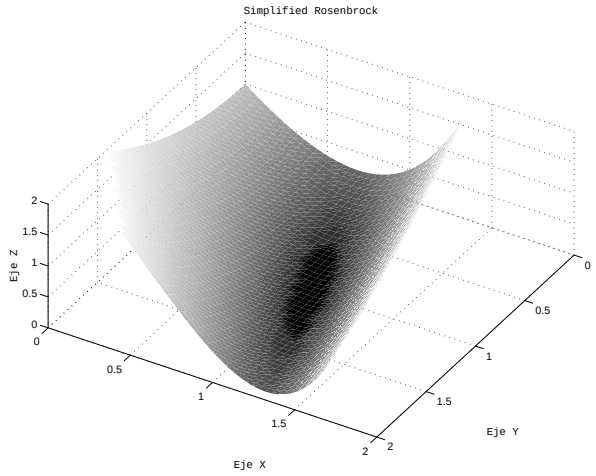
$$f_{RB}(x) = \frac{1}{2}(x_1^2 - x_2)^2 + (x_1 - 1)^2$$

Dominio (S): $-2 \leq x_i \leq 8,\ i = 1, 2$

Anchura del rango real: $w(f(S)) \simeq 2200$

Mínimo global: $f^* = 0,\ x^* = (1, 1)^T$

Observación: Se ha reducido la pendiente del valle de la función RB.



Six Hump Camel Back (C) [133]:

Ecuación de la función:

$$f_C(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$$

Dominio (S): $-5 \leq x_i \leq 5,\ i = 1, 2$

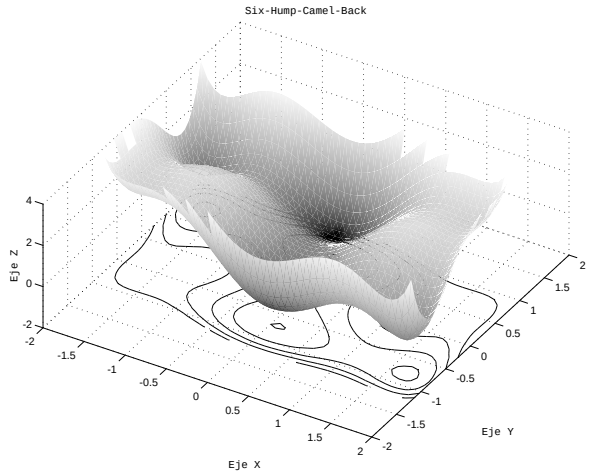
Anchura del rango real: $w(f(S)) \simeq 6400$

Existen dos mínimos globales con:

$$f^* = -1.03162845$$
$$X^* = \begin{cases} (\ 0.08984201, -0.71265640)^T \\ (-0.08984201,\ 0.71265640)^T \end{cases}$$

Función de inclusión:

$$F_C(X) = (4Y_1 - 2.1Y_1^2 + \frac{1}{3}Y_1^3) + X_1X_2 + 4(Y_2(Y_2 - 1))$$
$$\text{con } Y_1 = X_1^2,\ Y_2 = X_2^2$$



Three Hump Camel Back (C3) [30]:

Ecuación de la función:

$$f_{C3}(x) = 2x_1^2 - 1.05x_1^4 + \frac{1}{6}x_1^6 + x_1x_2 + x_2^2$$

Dominio (S): $-5 \leq x_i \leq 5,\ i = 1, 2$

Anchura del rango real: $w(f(S)) \simeq 2000$

Mínimo global: $f^* = 0,\ x^* = (0, 0)^T$

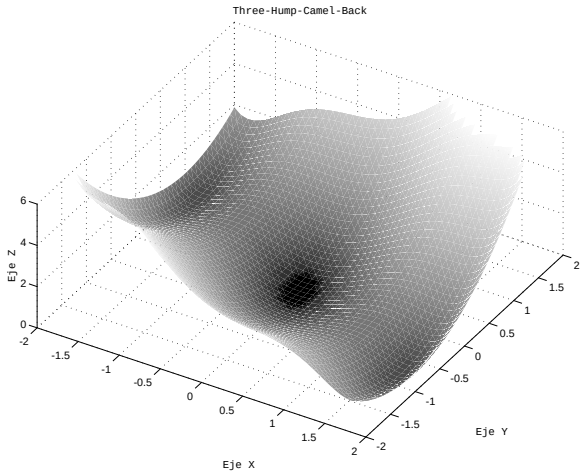
Treccani (TR) [30]:

Ecuación de la función:

$$f_{TR}(x) = x_1^4 + 4x_1^3 + 4x_1^2 + x_2^2$$

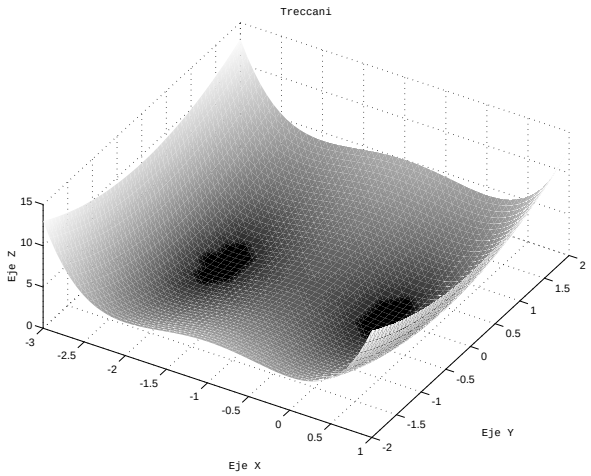
Dominio (S): $-5 \leq x_i \leq 5,\ i = 1, 2$

Anchura del rango real: $w(f(S)) \simeq 1500$



Existen 2 mínimos globales con:

$$f^* = 0$$
$$X^* = \begin{cases} (0,\ 0)^T \\ (\text{-}2,\ 0)^T \end{cases}$$



Función de inclusión:

$$F_{TR}(X) = X_1^2 \left(X_1 + 2\right)^2 + X_2^2$$

Observación: (-1,0) y (-2,0) son dos puntos de "silla de montar".

Apéndice B

Direcciones de interés en Internet

B.1. Aritmética de Intervalos

1. **Errores de la Aritmética Computacional:**
   http://www.ima.umn.edu/ arnold/455.f97/notes.html
2. **Interval Computations:**
   http://www.cs.utep.edu/interval-comp/
3. **Interval Methods:**
   http://www.mat.univie.ac.at/∼neum/interval.html

B.2. Arquitecturas y Algoritmos Paralelos

1. **Parallel Architectures and Algorithms :**
   http://www.ace.ual.es/∼leo/parallel.html
2. **Network of workstations :**
   http://now.cs.berkeley.edu/

B.3. Diferenciación Automática

1. **Automatic Differentiation:**
   http://www.mat.univie.ac.at/∼neum/glopt/related.html#autodiff

2. **Cray Assembler for MPP:**
   http://www.ciemat.es:8080/library/t3e/2510_2.2/8319

B.4. IEEE-754

1. **IEEE-754 status:**
   http://www.cs.berkeley.edu/∼wkahan/ieee754status/754story.html

B.5. Optimización Global

1. **A Survey of GO Methods:**
   http://www.cs.sandia.gov/opt/survey/
2. **Global (and Local) Optimization:**
   http://www.mat.univie.ac.at/∼neum/glopt.html
3. **Decision Tree for Optimization Software:**
   http://plato.la.asu.edu/guide.html
4. **List of Useful Optimization Software:**
   http://realtime.snu.ac.kr/bwkim/free-OR-codes.html
5. **Continuous GO Software: A Brief Review**
   http://plato.la.asu.edu/gom.html
6. **The Optimization Technology Center:**
   http://www.ece.northwestern.edu/OTC/
7. **Global Optimization Page:**
   http://www.bu.edu/pcms/simon/index.html
8. **Mathematical Optimization:**
   http://www.phy.ornl.gov/csep/CSEP/MO/MO.html
9. **Michael Trick's Operations Research Page:**
   http://mat.gsia.cmu.edu/index.html
10. **Links to Global Optimization:**
    http://www.ace.ual.es/∼leo/optimizacion.html

B.6. Problemas de Optimización Combinatoria

1. **TSPBIB Home Page :**
   http://www.ing.unlp.edu.ar/cetad/mos/TSPBIB_home.html

Apéndice C

Tablas de resultados

Tabla C.1: Equivalencia de funciones multidimensional, precisión para la que se ha calculado el mínimo, dimensiones y refencia bibliográfica.

No.	Función	Epsilon	Dim	Ref
0	EX2	10^{-9}	5	[116]
1	Griewank 10	10^{-4}	10	[133]
2	Schwefel 2.10 (Kowalik)	10^{-5}	4	[141]
3	Neumaier 2	10^{-10}	4	[99]
4	Ratz 5	10^{-5}	3	[116]
5	Ratz 6	10^{-4}	5	[116]
6	Ratz 7	10^{-4}	7	[116]
7	Ratz 8	10^{-3}	9	[116]
8	Rosenbrock 10	10^{-14}	10	[30]
9	Schwefel 2.14 (Powell)	10^{-8}	4	[116]
10	Schwefel 2.7	10^{-2}	3	[116]
11	Schwefel 3.1p	10^{-4}	3	[116]

Tabla C.2: Resultados del speed-up en función del número de procesadores para el modelo Global.

No.	Número de procesadores					
	1	2	4	8	13	15
0	1.00	1.96	3.79	5.44	6.53	6.42
1	1.00	2.00	3.98	6.22	7.86	8.44
2	1.00	1.98	3.81	4.87	6.65	6.99
3	1.00	2.00	3.85	5.32	7.38	7.67
4	1.00	1.87	3.62	3.99	4.63	4.88
5	1.00	1.97	3.92	5.55	5.78	5.60
6	1.00	2.04	3.97	5.68	6.75	6.96
7	1.00	1.95	3.90	5.37	6.95	7.35
8	1.00	2.57	3.71	6.74	7.63	13.23
9	1.00	1.85	2.35	2.98	2.65	3.99
10	1.00	1.95	3.76	4.44	5.31	5.67
11	1.00	1.89	3.59	1.25	4.64	4.50

Tabla C.3: Resultados del speed-up en función del número de procesadores para el modelo Local.

No.	Número de procesadores					
	1	2	4	8	13	15
0	1.00	1.96	3.61	5.49	6.34	6.62
1	1.00	2.00	4.01	7.34	8.04	8.42
2	1.00	1.98	3.81	5.23	6.59	6.66
3	1.00	2.03	4.02	5.21	6.44	7.18
4	1.00	1.99	3.97	4.34	4.84	4.96
5	1.00	1.99	3.99	5.75	5.71	5.91
6	1.00	2.03	4.02	6.35	6.80	7.14
7	1.00	1.98	3.98	6.46	6.69	6.88
8	1.00	3.72	7.03	13.00	10.22	9.31
9	1.00	1.40	4.68	2.92	2.65	3.13
10	1.00	1.97	3.84	4.73	5.73	5.67
11	1.00	0.59	3.82	3.98	4.27	4.67

Tabla C.4: Resultados del esfuerzo $Esf(p)$ en función del número de procesadores para el modelo Global.

No.	Número de procesadores					
	1	2	4	8	13	15
0	5507272	5525018	5537128	5524426	5525981	5524132
1	3088391	3088391	3088391	3088391	3088391	3088391
2	6541965	6541961	6541881	6548759	6548536	6534961
3	9865499	9802108	9861616	9939129	9707866	9992403
4	23074801	23074801	23074801	23074939	23074877	23074837
5	3946189	3946071	3946239	3946819	3946166	3946427
6	5009697	5009576	5009596	5009992	5010582	5010473
7	795541	795680	796024	796562	798941	797973
8	2303866	1778666	2455569	1972934	2229650	1348024
9	17379909	17379849	27161122	25612988	32900328	24580115
10	1635608	1635643	1635791	1615364	1637018	1636645
11	4491251	4490741	4520144	15917387	4492066	4529372

Tabla C.5: Resultados del esfuerzo $Esf(p)$ en función del número de procesadores para el modelo Local.

No.	Número de procesadores					
	1	2	4	8	13	15
0	5507289	5529997	5552189	5591695	5573798	5636114
1	3088423	3088423	3088423	3088423	3088423	3088423
2	6541979	6543736	6556626	6648497	6657242	6830707
3	9865513	9857232	9736743	10336141	10423568	10660830
4	23074812	23075483	23075054	23076023	23075283	23075652
5	3946206	3946239	3946838	3948613	3949394	3949855
6	5009720	5010585	5012181	5017153	5016932	5017819
7	795570	796566	798160	805203	806866	813467
8	2303898	1219522	1277500	977863	1633093	1825221
9	17379923	24554719	13943163	25905497	34467362	29215001
10	1635619	1636281	1649760	1650840	1570024	1573316
11	4491262	15151655	4524233	4513681	4669909	4520552

Tabla C.6: Resultados de las anomalias debidas al paralelismo en funcion del numero de procesadores para el modelo Global.

No.	Número de procesadores					
	1	2	4	8	13	15
0	1.00	1.00	0.99	1.00	1.00	1.00
1	1.00	1.00	1.00	1.00	1.00	1.00
2	1.00	1.00	1.00	1.00	1.00	1.00
3	1.00	1.01	1.00	0.99	1.02	0.99
4	1.00	1.00	1.00	1.00	1.00	1.00
5	1.00	1.00	1.00	1.00	1.00	1.00
6	1.00	1.00	1.00	1.00	1.00	1.00
7	1.00	1.00	1.00	1.00	1.00	1.00
8	1.00	1.30	0.94	1.17	1.03	1.71
9	1.00	1.00	0.64	0.68	0.53	0.71
10	1.00	1.00	1.00	1.01	1.00	1.00
11	1.00	1.00	0.99	0.28	1.00	0.99

Tabla C.7: Resultados de las anomalias debidas al paralelismo en funcion del numero de procesadores para el modelo Local.

No.	Número de procesadores					
	1	2	4	8	13	15
0	1.00	1.00	0.99	0.98	0.99	0.98
1	1.00	1.00	1.00	1.00	1.00	1.00
2	1.00	1.00	1.00	0.98	0.98	0.96
3	1.00	1.00	1.01	0.95	0.95	0.93
4	1.00	1.00	1.00	1.00	1.00	1.00
5	1.00	1.00	1.00	1.00	1.00	1.00
6	1.00	1.00	1.00	1.00	1.00	1.00
7	1.00	1.00	1.00	0.99	0.99	0.98
8	1.00	1.89	1.80	2.36	1.41	1.26
9	1.00	0.71	1.25	0.67	0.50	0.59
10	1.00	1.00	0.99	0.99	1.04	1.04
11	1.00	0.30	0.99	1.00	0.96	0.99

Tabla C.8: Resultados de desbalaceo en funcion del numero de procesadores para el modelo Global.

No.	Número de procesadores					
	1	2	4	8	13	15
0	0.000	0.001	0.004	0.015	0.007	0.006
1	0.000	0.001	0.001	0.008	0.014	0.004
2	0.000	0.001	0.011	0.026	0.006	0.002
3	0.000	0.004	0.008	0.020	0.006	0.002
4	0.000	0.029	0.011	0.016	0.004	0.006
5	0.000	0.003	0.001	0.013	0.006	0.009
6	0.000	0.001	0.012	0.015	0.008	0.005
7	0.000	0.041	0.006	0.021	0.007	0.009
8	0.000	0.000	0.001	0.015	0.007	0.005
9	0.000	0.062	0.011	0.027	0.005	0.004
10	0.000	0.015	0.000	0.012	0.006	0.005
11	0.000	0.009	0.018	0.023	0.016	0.011

Tabla C.9: Resultados de desbalanceo en funcion del numero de procesadores para el modelo Local.

No.	Número de procesadores					
	1	2	4	8	13	15
0	0.000	0.002	0.028	0.009	0.019	0.019
1	0.000	0.003	0.000	0.003	0.014	0.014
2	0.000	0.005	0.013	0.010	0.015	0.015
3	0.000	0.010	0.022	0.009	0.008	0.008
4	0.000	0.002	0.002	0.018	0.006	0.006
5	0.000	0.002	0.004	0.012	0.010	0.010
6	0.000	0.001	0.013	0.006	0.015	0.015
7	0.000	0.004	0.000	0.008	0.017	0.017
8	0.000	0.002	0.006	0.019	0.033	0.033
9	0.000	0.002	0.006	0.009	0.006	0.006
10	0.000	0.023	0.008	0.009	0.016	0.016
11	0.000	0.009	0.006	0.030	0.020	0.020

Tabla C.10: Tiempo de usuario, del sistema (acumulados por procesador) y real (segundos) en función del número de procesadores, para el modelo Global.

Función	ϵ	Número de procesadores					
		1	2	4	8	13	15
Tiempo de usuario							
0	10^{-9}	2351	2392	2458	3150	3150	4243
1	10^{-4}	3345	3346	3353	4089	4089	5087
2	10^{-5}	3391	3415	3485	4841	4841	5637
3	10^{-10}	3477	3475	3575	4665	4665	5458
4	10^{-5}	3486	3629	3769	5620	5620	7367
5	10^{-4}	1460	1469	1500	1949	1949	2890
6	10^{-4}	3521	3449	3483	4627	4627	6120
7	10^{-3}	874	875	891	1201	1201	1467
8	10^{-14}	1732	1347	1862	1942	1942	1681
9	10^{-8}	1732	1776	2898	3969	3969	4839
10	10^{-2}	714	726	746	1083	1083	1402
11	10^{-4}	590	614	641	3150	3150	1318
Tiempo del sistema							
0	10^{-9}	0.17	1.24	8.00	204.04	605.76	929.22
1	10^{-4}	0.07	0.21	1.00	136.24	443.66	598.32
2	10^{-5}	0.42	2.22	19.01	425.03	856.22	1124.23
3	10^{-10}	0.49	1.55	10.34	332.63	706.48	916.56
4	10^{-5}	1.00	6.65	42.96	996.38	2260.78	2781.76
5	10^{-4}	0.15	0.98	6.49	129.54	539.32	809.36
6	10^{-4}	0.48	0.54	4.75	235.78	805.66	1109.08
7	10^{-3}	0.15	0.21	1.96	64.15	170.04	223.17
8	10^{-14}	0.06	0.18	2.01	76.77	252.04	197.79
9	10^{-8}	0.67	4.21	36.68	519.87	1814.73	1389.91
10	10^{-2}	0.07	0.49	8.72	159.62	346.77	404.60
11	10^{-4}	0.19	1.94	9.48	450.31	371.92	549.07
Tiempo real							
0	10^{-9}	2353	1199	621	433	360	367
1	10^{-4}	3345	1674	840	537	425	396
2	10^{-5}	3392	1717	889	697	510	485
3	10^{-10}	3495	1748	908	657	473	456
4	10^{-5}	3488	1865	964	874	753	714
5	10^{-4}	1480	751	378	267	256	264
6	10^{-4}	3524	1726	888	620	522	507
7	10^{-3}	877	451	225	163	126	119
8	10^{-14}	1733	674	467	257	227	131
9	10^{-8}	1737	941	738	582	655	435
10	10^{-2}	715	367	190	161	135	126
11	10^{-4}	590	312	165	471	127	131

Tabla C.11: Tiempo de usuario, del sistema (acumulados por procesador) y real (segundos) en función del número de procesadores, para el modelo Local.

Función	ϵ	Número de procesadores					
		1	2	4	8	13	15
Tiempo de usuario							
0	10^{-9}	2350	2382	2419	3101	3758	4103
1	10^{-4}	3339	3340	3339	3576	4700	5050
2	10^{-5}	3388	3395	3495	4696	5324	5822
3	10^{-10}	3469	3416	3357	4709	5409	5664
4	10^{-5}	3485	3497	3504	5185	6682	7244
5	10^{-4}	1460	1458	1457	1858	2583	2758
6	10^{-4}	3513	3462	3441	4193	5499	6006
7	10^{-3}	871	871	873	1032	1381	1504
8	10^{-14}	1724	922	972	958	1840	2265
9	10^{-8}	1728	2458	1445	3960	6307	5826
10	10^{-2}	713	714	719	1027	1186	1307
11	10^{-4}	589	1980	600	902	1204	1225
Tiempo del sistema							
0	10^{-9}	0.16	1.05	5.42	182.80	578.17	748.07
1	10^{-4}	0.05	0.12	0.18	43.51	370.08	574.66
2	10^{-5}	0.50	2.10	17.03	292.25	834.48	1143.98
3	10^{-10}	0.47	0.54	0.88	309.14	788.17	806.55
4	10^{-5}	1.00	1.34	2.60	795.86	1974.01	2619.69
5	10^{-4}	0.19	0.23	0.51	122.17	518.58	648.99
6	10^{-4}	0.56	0.43	0.49	149.67	695.73	994.68
7	10^{-3}	0.06	0.05	0.19	28.87	167.73	248.91
8	10^{-14}	0.06	0.04	0.36	40.10	152.14	292.50
9	10^{-8}	0.55	3.24	13.48	569.69	1612.24	1933.35
10	10^{-2}	0.09	0.10	0.55	123.29	255.59	358.09
11	10^{-4}	0.21	1.27	4.64	151.20	392.29	440.95
Tiempo real							
0	10^{-9}	2350	1201	651	428	371	355
1	10^{-4}	3351	1674	837	457	417	398
2	10^{-5}	3389	1708	890	648	514	509
3	10^{-10}	3470	1711	863	666	539	483
4	10^{-5}	3487	1751	878	804	721	704
5	10^{-4}	1461	732	366	254	256	247
6	10^{-4}	3513	1732	873	554	517	492
7	10^{-3}	871	440	219	135	130	127
8	10^{-14}	1735	467	247	133	170	186
9	10^{-8}	1729	1235	370	592	653	553
10	10^{-2}	713	361	186	151	124	126
11	10^{-4}	589	993	154	148	138	126

Tabla C.12: Comparacion entre el modelo Global y Local de tiempos del usuario, del sistema (acumulados por procesador) y real (segundos) en función del número de procesadores.

Función	ϵ	Número de procesadores					
		1	2	4	8	13	15
				Tiempo de usuario			
0	10^{-9}	0.07	0.43	1.59	1.57	2.12	3.36
1	10^{-4}	0.17	0.18	0.42	13.4	2.23	0.73
2	10^{-5}	0.07	0.57	-0.28	3.04	-0.1	-3.24
3	10^{-10}	0.24	1.71	6.28	-0.93	-6.66	-3.7
4	10^{-5}	0.02	3.7	7.29	8.05	4.38	1.69
5	10^{-4}	0	0.74	2.96	4.79	0.92	4.68
6	10^{-4}	0.24	-0.36	1.22	9.84	3.52	1.87
7	10^{-3}	0.35	0.38	2.05	15.09	0.56	-2.5
8	10^{-14}	0.44	37.43	62.78	67.89	33.8	-29.63
9	10^{-8}	0.23	-32.23	66.9	0.23	0.51	-18.5
10	10^{-2}	0.16	1.62	3.63	5.3	11.88	7.04
11	10^{-4}	0.09	-105.3	6.62	110.98	-1.46	7.29
				Tiempo del sistema			
0	10^{-9}	4.23	15.87	38.43	10.98	4.66	21.6
1	10^{-4}	28.1	51.63	139.35	103.18	18.09	4.03
2	10^{-5}	-15.68	5.61	11.01	37.02	2.57	-1.74
3	10^{-10}	4.26	96.54	168.63	7.32	-10.93	12.77
4	10^{-5}	-0.39	132.84	177.16	22.38	13.54	6
5	10^{-4}	-24.14	122.84	170.62	5.85	3.92	21.99
6	10^{-4}	-15.13	22.63	162.76	44.68	14.64	10.88
7	10^{-3}	85.72	126.44	165.39	75.84	1.37	-10.91
8	10^{-14}	-8.13	122.13	139.31	62.75	49.43	-38.64
9	10^{-8}	20.05	25.9	92.54	-9.15	11.82	-32.71
10	10^{-2}	-35.15	133.11	176.28	25.68	30.28	12.2
11	10^{-4}	-10.47	41.7	68.57	99.45	-5.33	21.84
				Tiempo real			
0	10^{-9}	21.6	-0.14	-4.67	1.07	-2.85	3.12
1	10^{-4}	4.03	-0.01	0.42	16.21	2.01	-0.42
2	10^{-5}	-1.74	0.5	-0.08	7.22	-0.79	-4.68
3	10^{-10}	12.77	2.13	5.1	-1.33	-13	-5.81
4	10^{-5}	6	6.32	9.35	8.32	4.34	1.5
5	10^{-4}	21.99	2.46	3.03	4.82	0.04	6.64
6	10^{-4}	10.88	-0.35	1.74	11.3	1.02	2.9
7	10^{-3}	-10.91	2.37	2.62	19.08	-3.09	-5.87
8	10^{-14}	-38.64	36.33	61.73	63.4	28.85	-34.84
9	10^{-8}	-32.71	-27.04	66.53	-1.72	0.34	-23.76
10	10^{-2}	12.2	1.58	2.18	6.66	7.86	0.27
11	10^{-4}	21.84	-104.36	6.29	104.38	-8.01	3.71

Tabla C.13: Resultados del speed-up en función del número de procesadores para el modelo Global. Se ha introducido un retardo $S = 10^6$.

No.	Número de procesadores					
	1	2	4	8	13	15
0	1.00	2.00	3.97	7.91	12.76	14.41
1	1.00	2.06	4.12	8.16	12.14	13.35
2	1.00	2.01	3.96	7.81	12.55	14.28
3	1.00	2.05	4.01	8.00	12.82	14.77
4	1.00	2.00	3.99	7.96	12.89	14.87
5	1.00	2.00	3.98	7.93	12.82	14.70
6	1.00	2.09	4.15	8.22	12.90	14.29
7	1.00	2.01	3.99	7.73	11.19	12.89
8	1.00	2.55	3.78	18.86	16.67	20.52
9	1.00	1.14	7.08	6.74	23.99	10.29
10	1.00	2.01	5.08	8.30	16.22	15.94
11	1.00	1.98	3.97	7.92	12.83	14.78

Tabla C.14: Resultados del speed-up en función del número de procesadores para el modelo Local. Se ha introducido un retardo $S = 10^6$.

No.	Número de procesadores					
	1	2	4	8	13	15
0	1.00	1.98	3.91	7.82	12.08	13.94
1	1.00	1.99	4.00	7.99	11.46	12.93
2	1.00	2.01	3.90	7.80	12.52	14.26
3	1.00	2.01	3.96	7.54	11.48	13.85
4	1.00	2.00	3.99	7.95	12.89	14.86
5	1.00	1.94	4.00	7.94	12.78	14.71
6	1.00	1.99	4.02	7.95	12.52	14.25
7	1.00	1.97	3.98	7.77	10.74	12.15
8	1.00	3.33	6.43	18.43	10.05	20.18
9	1.00	2.18	3.36	5.72	11.51	9.88
10	1.00	1.95	5.05	7.75	12.21	14.62
11	1.00	1.97	3.97	7.92	12.75	14.71

Tabla C.15: Resultados del esfuerzo $Esf(p)$ en función del número de procesadores para el modelo Global. Se ha introducido un retardo $S = 10^6$.

No.	Número de procesadores					
	1	2	4	8	13	15
0	5507272	5515497	5523666	5525955	5526021	5525724
1	3088391	3088391	3088391	3088391	3088391	3088391
2	6541965	6542689	6541977	6548323	6548675	6548689
3	9865499	9722006	9849103	9778041	9812949	9785642
4	23074801	23075058	23075009	23075224	23075527	23075079
5	3946189	3946197	3946595	3948120	3949152	3950242
6	5009697	5009520	5009721	5009895	5014141	5013271
7	795541	795634	795937	796721	799583	799668
8	2303866	1798993	2437022	936788	1579522	1431833
9	17379909	30388935	9788058	20417450	9301385	24913582
10	1635608	1636622	1296407	1581650	1310295	1536206
11	4491251	4529866	4513106	4516561	4515469	4521717

Tabla C.16: Resultados del esfuerzo $Esf(p)$ en función del número de procesadores para el modelo Local. Se ha introducido un retardo $S = 10^6$.

No.	Número de procesadores					
	1	2	4	8	13	15
0	5507289	5551087	5608025	5592651	5695185	5592567
1	3088423	3088423	3088423	3088423	3088423	3088423
2	6541979	6544370	6588007	6650961	6604367	6664994
3	9865513	9854115	9955882	10307506	10769148	10201632
4	23074812	23075057	23075089	23075433	23075401	23075371
5	3946206	3946485	3946918	3948244	3949871	3949344
6	5009720	5010461	5011917	5013667	5018114	5018582
7	795570	796500	799051	805291	809185	812004
8	2303898	1198657	1410752	959691	2555152	1433145
9	17379923	15808790	20574033	24099668	19368315	26019172
10	1635619	1636258	1296768	1650822	1644220	1635461
11	4491262	4548454	4516589	4513984	4517739	4513818

Tabla C.17: Resultados de las anomalias debidas al paralelismo en funcion del numero de procesadores para el modelo Global. Se ha introducido un retardo $S = 10^6$.

No.	Número de procesadores					
	1	2	4	8	13	15
0	1.00	1.00	1.00	1.00	1.00	1.00
1	1.00	1.00	1.00	1.00	1.00	1.00
2	1.00	1.00	1.00	1.00	1.00	1.00
3	1.00	1.01	1.00	1.01	1.01	1.01
4	1.00	1.00	1.00	1.00	1.00	1.00
5	1.00	1.00	1.00	1.00	1.00	1.00
6	1.00	1.00	1.00	1.00	1.00	1.00
7	1.00	1.00	1.00	1.00	0.99	0.99
8	1.00	1.28	0.95	2.46	1.46	1.61
9	1.00	0.57	1.78	0.85	1.87	0.70
10	1.00	1.00	1.26	1.03	1.25	1.06
11	1.00	0.99	1.00	0.99	0.99	0.99

Tabla C.18: Resultados de las anomalias debidas al paralelismo en funcion del numero de procesadores para el modelo Local. Se ha introducido un retardo $S = 10^6$.

No.	Número de procesadores					
	1	2	4	8	13	15
0	1.00	0.99	0.98	0.98	0.97	0.98
1	1.00	1.00	1.00	1.00	1.00	1.00
2	1.00	1.00	0.99	0.98	0.99	0.98
3	1.00	1.00	0.99	0.96	0.92	0.97
4	1.00	1.00	1.00	1.00	1.00	1.00
5	1.00	1.00	1.00	1.00	1.00	1.00
6	1.00	1.00	1.00	1.00	1.00	1.00
7	1.00	1.00	1.00	0.99	0.98	0.98
8	1.00	1.92	1.63	2.40	0.90	1.61
9	1.00	1.10	0.84	0.72	0.90	0.67
10	1.00	1.00	1.26	0.99	0.99	1.00
11	1.00	0.99	0.99	0.99	0.99	1.00

Tabla C.19: Resultados de desbalaceo en funcion del numero de procesadores para el modelo Global. Se ha introducido un retardo $S = 10^6$.

No.	Número de procesadores					
	1	2	4	8	13	15
0	0.000	0.000	0.001	0.001	0.003	0.002
1	0.000	0.000	0.000	0.001	0.008	0.007
2	0.000	0.000	0.000	0.002	0.001	0.001
3	0.000	0.000	0.002	0.001	0.001	0.001
4	0.000	0.000	0.000	0.000	0.003	0.000
5	0.000	0.000	0.001	0.001	0.001	0.001
6	0.000	0.000	0.002	0.001	0.006	0.002
7	0.000	0.001	0.002	0.005	0.009	0.003
8	0.000	0.000	0.001	0.002	0.015	0.010
9	0.000	0.000	0.000	0.000	0.001	0.000
10	0.000	0.001	0.001	0.000	0.009	0.001
11	0.000	0.000	0.001	0.001	0.003	0.001

Tabla C.20: Resultados de desbalanceo en funcion del numero de procesadores para el modelo Local. Se ha introducido un retardo $S = 10^6$.

No.	Número de procesadores					
	1	2	4	8	13	15
0	0.000	0.002	0.001	0.004	0.018	0.011
1	0.000	0.006	0.000	0.000	0.007	0.006
2	0.000	0.004	0.006	0.008	0.009	0.008
3	0.000	0.001	0.004	0.001	0.005	0.005
4	0.000	0.000	0.003	0.001	0.002	0.001
5	0.000	0.014	0.000	0.001	0.002	0.003
6	0.000	0.009	0.001	0.003	0.004	0.002
7	0.000	0.006	0.001	0.001	0.017	0.006
8	0.000	0.001	0.006	0.006	0.021	0.012
9	0.000	0.004	0.001	0.001	0.003	0.001
10	0.000	0.046	0.009	0.027	0.048	0.009
11	0.000	0.003	0.002	0.003	0.008	0.005

Tabla C.21: Tiempo de usuario, del sistema (acumulados por procesador) y real (segundos) en función del número de procesadores, para el modelo Global. Se ha introducido un retardo $S = 10^6$.

Función	ϵ	Número de procesadores					
		1	2	4	8	13	15
Tiempo de usuario							
0	10^{-9}	13176	13207	13284	13331	13396	13626
1	10^{-4}	8638	8639	8650	8716	9304	9635
2	10^{-5}	23905	23834	24170	24458	24673	24926
3	10^{-10}	32987	32449	33100	33236	33597	33631
4	10^{-5}	87548	87576	87773	88016	88193	88309
5	10^{-4}	11994	11988	12037	12092	12138	12203
6	10^{-4}	14093	14009	14081	14210	14581	15096
7	10^{-3}	2246	2243	2252	2319	2526	2544
8	10^{-14}	5034	3951	5325	2122	3807	3539
9	10^{-8}	53356	93450	30138	63331	28888	77730
10	10^{-2}	6625	6629	5249	6424	5321	6259
11	10^{-4}	16860	17009	16981	17037	17051	17097
Tiempo del sistema							
0	10^{-9}	0.17	0.31	0.83	2.92	14.17	57.92
1	10^{-4}	0.53	0.14	0.34	7.77	103.99	195.47
2	10^{-5}	0.47	0.48	2.11	8.39	23.26	41.90
3	10^{-10}	1.29	0.60	2.61	4.77	13.24	20.28
4	10^{-5}	1.02	1.83	4.48	6.29	9.08	10.12
5	10^{-4}	0.17	0.24	0.59	1.25	4.47	12.49
6	10^{-4}	1.82	1.58	1.54	6.98	66.81	174.85
7	10^{-3}	0.06	0.06	0.30	7.95	49.73	50.45
8	10^{-14}	0.06	0.08	0.36	6.40	55.55	63.90
9	10^{-8}	0.50	1.65	0.71	4.67	3.34	9.82
10	10^{-2}	0.08	0.13	0.42	0.74	1.96	1.86
11	10^{-4}	0.21	1.49	1.09	1.21	1.43	1.72
Tiempo real							
0	10^{-9}	13198	6605	3322	1668	1034	916
1	10^{-4}	8906	4320	2163	1092	734	667
2	10^{-5}	23935	11919	6047	3064	1907	1676
3	10^{-10}	33279	16227	8289	4161	2596	2253
4	10^{-5}	87628	43790	21947	11006	6799	5893
5	10^{-4}	11998	5995	3011	1512	936	816
6	10^{-4}	14639	7006	3526	1780	1135	1025
7	10^{-3}	2261	1122	566	292	202	175
8	10^{-14}	5038	1976	1332	267	302	246
9	10^{-8}	53365	46729	7536	7919	2224	5186
10	10^{-2}	6670	3319	1313	804	411	419
11	10^{-4}	16877	8507	4250	2131	1315	1142

Tabla C.22: Tiempo de usuario, del sistema (acumulados por procesador) y real (segundos) en función del número de procesadores, para el modelo Local. Se ha introducido un retardo $S = 10^6$.

Función	ϵ	Número de procesadores					
		1	2	4	8	13	15
Tiempo de usuario							
0	10^{-9}	13181	13286	13472	13455	13787	13776
1	10^{-4}	8647	8647	8647	8651	9379	9629
2	10^{-5}	23927	23871	24283	24731	24766	25148
3	10^{-10}	32998	32912	33266	34918	36825	35060
4	10^{-5}	87566	87576	87573	87995	88126	88251
5	10^{-4}	12001	11998	12003	12091	12129	12154
6	10^{-4}	14079	14024	14004	14097	14447	14618
7	10^{-3}	2247	2248	2256	2300	2530	2576
8	10^{-14}	5040	2648	3120	2164	6050	3496
9	10^{-8}	53360	48633	63469	74566	60042	80785
10	10^{-2}	6627	6629	5242	6709	6688	6658
11	10^{-4}	16864	17079	16985	17010	17039	17027
Tiempo del sistema							
0	10^{-9}	0.14	1.12	0.68	1.76	22.49	68.58
1	10^{-4}	0.07	0.10	0.18	0.66	127.56	188.69
2	10^{-5}	1.14	2.29	2.32	5.88	16.08	24.56
3	10^{-10}	0.52	0.51	0.57	3.08	10.43	19.44
4	10^{-5}	1.05	1.45	5.16	4.80	6.48	7.49
5	10^{-4}	0.18	0.79	0.29	0.63	2.49	4.69
6	10^{-4}	0.51	0.45	0.45	1.35	49.16	74.86
7	10^{-3}	0.06	0.14	0.07	3.07	42.58	49.45
8	10^{-14}	0.05	0.33	0.18	3.30	70.58	54.26
9	10^{-8}	0.54	4.30	2.73	3.83	5.41	6.85
10	10^{-2}	0.09	0.17	0.11	0.40	1.07	1.50
11	10^{-4}	0.19	0.25	0.39	0.60	1.05	0.83
Tiempo real							
0	10^{-9}	13183	6660	3373	1685	1092	945
1	10^{-4}	8647	4341	2164	1083	755	669
2	10^{-5}	24182	12022	6195	3099	1931	1695
3	10^{-10}	33002	16459	8343	4375	2875	2383
4	10^{-5}	87571	43798	21934	11012	6795	5892
5	10^{-4}	12002	6175	3001	1512	939	816
6	10^{-4}	14086	7069	3502	1771	1125	988
7	10^{-3}	2247	1140	565	289	209	185
8	10^{-14}	5041	1515	784	273	502	250
9	10^{-8}	53359	24517	15873	9325	4637	5400
10	10^{-2}	6628	3398	1312	856	543	453
11	10^{-4}	16866	8544	4248	2128	1323	1146

Tabla C.23: Comparacion entre el modelo Global y Local de tiempos del usuario, del sistema (acumulados por procesador) y real (segundos) en función del número de procesadores. Se ha introducido un retardo $S = 10^6$.

Función	ϵ	Número de procesadores					
		1	2	4	8	13	15
Tiempo de usuario							
0	10^{-9}	-0.04	-0.60	-1.41	-0.92	-2.87	-1.09
1	10^{-4}	-0.10	-0.09	0.03	0.74	-0.80	0.06
2	10^{-5}	-0.09	-0.16	-0.47	-1.11	-0.38	-0.89
3	10^{-10}	-0.03	-1.42	-0.50	-4.94	-9.17	-4.16
4	10^{-5}	-0.02	0.00	0.23	0.02	0.08	0.07
5	10^{-4}	-0.06	-0.08	0.28	0.01	0.08	0.40
6	10^{-4}	0.10	-0.11	0.55	0.80	0.92	3.22
7	10^{-3}	-0.05	-0.22	-0.18	0.81	-0.19	-1.27
8	10^{-14}	-0.12	39.50	52.23	-1.98	-45.52	1.20
9	10^{-8}	-0.01	63.08	-71.21	-16.29	-70.06	-3.86
10	10^{-2}	-0.02	0.01	0.14	-4.35	-22.78	-6.16
11	10^{-4}	-0.02	-0.41	-0.02	0.16	0.07	0.41
Tiempo del sistema							
0	10^{-9}	19.19	-112.71	19.32	49.67	-45.36	-16.85
1	10^{-4}	152.60	28.80	63.93	168.71	-20.36	3.53
2	10^{-5}	-82.77	-131.21	-9.44	35.18	36.49	52.18
3	10^{-10}	85.44	16.39	128.31	43.04	23.68	4.23
4	10^{-5}	-2.93	23.36	-14.07	26.86	33.45	29.77
5	10^{-4}	-7.78	-106.26	66.82	65.29	56.70	90.74
6	10^{-4}	111.91	111.28	109.07	135.14	30.45	80.08
7	10^{-3}	0.00	-82.13	122.96	88.62	15.50	2.00
8	10^{-14}	20.56	-118.66	66.19	63.93	-23.82	16.31
9	10^{-8}	-8.48	-89.00	-117.30	19.75	-47.38	35.59
10	10^{-2}	-9.52	-27.45	119.26	59.84	59.06	21.60
11	10^{-4}	8.82	142.38	94.13	66.89	30.82	70.10
Tiempo real							
0	10^{-9}	0.11	-0.83	-1.53	-1.03	-5.42	-3.17
1	10^{-4}	2.95	-0.48	-0.03	0.86	-2.80	-0.23
2	10^{-5}	-1.03	-0.86	-2.42	-1.14	-1.26	-1.18
3	10^{-10}	0.83	-1.42	-0.65	-5.01	-10.20	-5.57
4	10^{-5}	0.07	-0.02	0.06	-0.05	0.06	0.00
5	10^{-4}	-0.03	-2.96	0.33	-0.02	-0.32	0.01
6	10^{-4}	3.85	-0.89	0.68	0.48	0.85	3.63
7	10^{-3}	0.63	-1.56	0.17	1.10	-3.45	-5.23
8	10^{-14}	-0.05	26.39	51.84	-2.35	-49.65	-1.71
9	10^{-8}	0.01	62.35	-71.23	-16.32	-70.33	-4.04
10	10^{-2}	0.63	-2.36	0.05	-6.29	-27.59	-8.01
11	10^{-4}	0.07	-0.44	0.04	0.13	-0.59	-0.38