UNIVERSIDAD DE MÁLAGA
Departamento de Arquitectura de Computadores

OPTIMIZACIÓN GLOBAL BASADA EN ARITMÉTICA DE INTERVALOS Y RAMIFICACIÓN Y ACOTACIÓN: PARALELIZACIÓN

Leocadio González Casado

Septiembre de 1999

Dr. **Inmaculada García Fernández**
Catedrática de Arquitectura de Computadores
de la Universidad de Almería.

CERTIFICA:

Que la memoria titulada, **OPTIMIZACIÓN GLOBAL BASADA EN ARITMÉTICA DE INTERVALOS Y RAMIFICACIÓN Y ACOTACIÓN: PARALELIZACIÓN**, ha sido realizada por **D. Leocadio González Casado** bajo mi dirección en el Departamento de Arquitectura de Computadores de la Universidad de Málaga y concluye la Tesis que presenta para optar al grado de Doctor en Ingeniería Informática.

Málaga 1 de Septiembre de 1999

Dr. **Inmaculada García Fernández**
Director de la Tesis

Dr. **Emilio López Zapata**
Director del Departamento de
Arquitectura de Computadores

Dr. **Julio Villalba Moreno**
Titular de Escuela Universitaria.
Tutor de la Tesis

A Raquel y Alberto

"La mitad está hecha cuando tienen buen principio las cosas"
Fernando de Rojas (1475-1541)

"La formulación de un problema, es más importante que su solución"
Albert Einstein (1879-1955)

"Nunca vayas por el camino trazado, porque conduce hacia donde otros han ido ya"
Alexander Graham Bell (1847-1922)

"La inspiración existe, pero tiene que encontrarte trabajando"
Pablo Ruiz Picasso (1881-1973)

"El pensador sabe considerar las cosas más sencillas de lo que son"
Friedrich Nietzsche (1844-1900)

"Todo lo ajustable tarde o temprano requiere un ajuste"
Conclusión de Kagel

"Bien acierta quien sospecha que siempre yerra"
Francisco de Quevedo (1580-1645)

"No se sale adelante celebrando éxitos, sino superando fracasos"
Jaime Pericles

Agradecimientos

Para no olvidarme de nadie, quiero agradecer a todos aquellos que directa o indirectamente me ayudaron, no sólo a realizar esta tesis, sino también a tener la oportunidad de poder realizarla.

A mi mujer y a mi hijo, agradezco la comprensión y la paciencia que han tenido durante la realización de esta tesis y espero poder dedicarles un poco más de mi vida a partir de ahora.

Quiero agradecer a mi familia, el apoyo constante que siempre me han dado; por enseñarme que con esfuerzo y trabajo se pueden conseguir la mayoría de las cosas. A mis padres, Leocadio y Presentación, y hermanos, Inma y Sergio, por la suerte que he tenido de tenerlos, sin olvidarme de mi abuela María, ni de Mari Pepa y Marcelino, ni del resto de mi familia.

Agradezco también la ayuda y colaboración prestada por la que considero mi familia científica: muy especialmente a Inmaculada García Fernández por su inestimable y constante ayuda en todo, a la que nunca he podido encontrar una inconsistencia en sus razonamientos. Agradezco su colaboración y el valioso tiempo que me han dedicado Tibor Csendes y Yaroslav D. Sergeyev, cuyas ideas y consejos han permitido mejorar considerablemente esta tesis.

A José Antonio Martínez García y Vicente González Ruiz por permitirme colaborar en sus proyectos, como la construcción de CAESARG. A Pilar Martínez Ortigosa por acercarme al mundo de la Optimización Global. Igualmente agradezco a todos los miembros del Departamento de Arquitectura de Computadores y Electrónica de la Universidad de Almería, el haberme acogido entre ellos y hacer del trabajo un sitio agradable y solidaridario.

A Peter Gabor Szabo, por su sentido crítico y perfeccionista que siempre ha mostrado en los trabajos en los que he colaborado con él. A Janos Balog por hacer mi estancia en Hungría más llevadera.

A D. Emilio López Zapata y D. Julio Villalva Moreno, del Departamento de Arquitectura de Computadores de la Universidad de Málaga, por hacer posible la lectura de esta tesis.

Gracias a las instituciones que me han dado el soporte económico para el desarrollo de esta tesis: Junta de Andalucía, Universidad de Almería y programa TEMPUS.

Prólogo

Muchos de los problemas prácticos que aparecen en diversas áreas pueden plantearse como problemas de Optimización Global. Consideremos, por ejemplo, el diseño de un sistema que exige ciertas condiciones críticas de funcionamiento. Este sistema debe incluir características que pueden ser modificadas por el diseñador, dentro de ciertos límites. El rango de valores que puedan tomar estos parámetros especifica el espacio de definición de las variables de entrada para el problema de optimización. Cuando el rendimiento del sistema se expresa como una función matemática de las variables a optimizar, esta función posee frecuentemente más de un mínimo (o máximo) local. Los algoritmos que distinguen entre estos mínimos locales y localizan el mejor, se conocen como aquellos que resuelven el problema de Optimización Global.

La Optimización Global está relacionada con la caracterización y computación del mínimo o máximo global de funciones no lineales con o sin ligaduras. Tales problemas aparecen en un amplio rango de aplicaciones del mundo real que se pueden modelar matemáticamente. Entre tales aplicaciones, se encuentran: finanzas, estadística, control en plantas de producción industrial, diseño de circuitos, problemas de redes y transporte, diseño y control en ingeniería química, biología molecular, y otras muchas [52]. Estos problemas pertenecen a la clase de los problemas NP-Duros [131], por lo que son difíciles de resolver.

La Optimización Global es sustancialmente más complicada que la Optimización Local, en la cual se busca un mínimo local en las proximidades del punto de comienzo de la búsqueda. Una manera de obtener los mínimos globales consiste en determinar todos los mínimos locales y de ellos quedarse con los menores, pero esta aproximación es impracticable, debido a que muchos problemas se caracterizan por tener un gran número de mínimos locales. Además, incluso la determinación de un mínimo local no es siempre fácil. Por lo tanto, en vez de una información local hay que usar una información global para evitar la búsqueda en soluciones locales.

Los métodos basados en el análisis de intervalos son apropiados para obtener una información global con un coste bajo. Los intervalos se representan en el computador mediante dos números reales que deben ser internamente representables mediante la Aritmética Computacional. Un intervalo es un subconjunto infinito de los números reales cerrado y acotado, y no sólo las cantidades finitas representadas en los límites del intervalo, por lo tanto tienen una cantidad infinita de información, lo que implica que tienen información global. En la sección 1.2, se definirá la Aritmética de Intervalos que fue desarrollada inicialmente con el propósito de tener en cuenta los errores de redondeo cometidos por la Aritmética Computacional [124], permitiendo así probar en el computador sentencias matemáticamente correctas. Por ejemplo, usando una sola evaluación de la Aritmética de Intervalos, se puede obtener información fiable sobre la inclusión del rango de la función objetivo en un intervalo (su coste es, aproximadamente, dos veces el de la Aritmética Real), con lo que se puede determinar si una función tiene un cero en un intervalo determinado.

Para la obtención de soluciones rigurosas, aparecieron una nueva clase de algoritmos de Optimización Global, distintos de los tradicionales, basados en la Aritmética de Intervalos. El primer algoritmo de este tipo es el llamado Moore-Shelboe, ya que Skelboe [167] desarrolló el primer algoritmo de Optimización Global, basado en las ideas de Aritmética de Intervalos de Moore [123] y en el modelo de Ramificación y Acotación, siendo éste mejorado posteriormente por Moore [126]. Las mejoras introducidas en el algoritmo de Moore-Skelboe pueden encontrarse en [88]. Tanto el algoritmo básico, como los dispositivos aceleradores más ampliamente utilizados, serán descritos en este trabajo, en el que se presentan nuevas versiones secuenciales y paralelas.

Los computadores paralelos son los que ofrecen una mayor potencia de cálculo. Se usan principalmente para resolver problemas cuyo tiempo de resolución sobre una máquina secuencial es muy grande. La ventaja de los computadores paralelos es el uso de varios procesadores para la resolución de un problema. Para obtener buenos resultados sobre un computador paralelo, normalmente hay que adaptar los programas secuenciales a la nueva arquitectura. Los algoritmos de Optimización Global basados en Aritmética de Intervalos y Ramificación y Acotación, debido a que determinan con la precisión deseada todos los mínimos globales, son llamados también algoritmos de Optimización Global precisos. Debido a su precisión, estos algoritmos tienen una complejidad exponencial con el número de dimensiones del problema a resolver. Ésto, junto con el hecho de que los algoritmos de Ramificación y Acotación son altamente paralelizables, los hacen buenos candidatos para su resolución en paralelo.

Esta tesis se ha organizado de la siguiente forma: en el Capítulo 1 se introducen las bases matemáticas y científicas de los algoritmos de Optimización Global basados en intervalos y la nomenclatura usada. También se describen los distintos tipos de problemas de optimización y los distintos algoritmos propuestos para su resolución. En esta tesis nos centraremos en los algoritmos de Optimización Global basados en intervalos. Para ello se realizará una breve descripción de la Aritmética Computacional y de Intervalos, así como de la Aritmética de Intervalos Extendida. También se describen los distintos tests que permiten determinar la inexistencia de mínimos globales. Varios de estos tests, como el test de monotonía, requieren que el problema a resolver sea diferenciable. En esta tesis las mejoras propuestas para los algoritmos de Optimización son independientes de la diferenciabibidad o no, de los problemas a resolver. Además, para los estudios numéricos es importante probar las nuevas mejoras con la ausencia de otros dispositivos aceleradores. No obstante, en la Sección 1.3 se describen los dispositivos aceleradores más ampliamente utilizados en los algoritmos de Optimización Global basada en intervalos. Por el mismo motivo, se introduce la Diferenciación Automática que permite un cálculo eficiente de la función de inclusión del gradiente y de la matriz Hessiana, sin necesidad de expresar explícita y analíticamente, las derivadas de la función objetivo. Finalmente, se detallarán las reglas generales de: *División*, *Selección*, *Acotación*, *Eliminación* y *Terminación* que rigen los algoritmos de Ramificación y Acotación, las posibles opciones para las mismas y sus ventajas e inconvenientes. Estas reglas serán la base de los estudios posteriores.

En el capítulo 2 se estudia una clase de problemas de Optimización que consisten en

la determinación de la primera raíz de una función uni-dimensional y la primera raíz de un conjunto de funciones uni-dimensionales. La resolución de este tipo de problemas tiene un amplio rango de aplicaciones. En este capítulo se presentan nuevos algoritmos que resuelven eficientemente estos problemas.

En el capítulo 3 se describe un algoritmo simple de Optimización Global precisa y se presentarán las modificaciones propuestas para las reglas básicas de Ramificación y Acotación de este algoritmo, concretamente, la *Regla de División*, la *Regla de Eliminación* y la *Regla de Selección*, las cuales permitirán acelerar considerablemente el algoritmo inicial.

En el Capitulo 4 se hace un resumen de las bases de la computación paralela. Después de una clasificación de las arquitecturas paralelas y de la definición de los conceptos necesarios para la realización de medidas de rendimiento de los algoritmos paralelos, se introducen los criterios de balanceo de la carga, necesarios para los algoritmos de Ramificación y Acotación, que serán la base de los algoritmos paralelos de Optimización Global basados en intervalos. Después, se estudiará la paralelización de los algoritmos de Ramificación y Acotación, junto con las posibles elecciones a tener en cuenta en su implementación y las anomalías que se pueden presentar. Se describirán brevemente las implementaciones paralelas de los algoritmos de Optimización Global basados en intervalos que pueden encontrarse en la literatura y finalmente se propone un nuevo algoritmo paralelo basado en las mejoras que se introdujeron en el algoritmo secuencial, presentando resultados de las pruebas numéricas realizadas, así como de las medidas de rendimiento.

En el Apéndice A, se detallan los problemas de prueba utilizados en esta tesis. Se hace una descripción analítica de los problemas, se especifican sus dominios y la localización de sus mínimos globales.

Índice General

1 Introducción **1**
1.1 Optimización matemática . . . 1
1.1.1 Problema general . . . 1
1.1.2 Clasificación de los problemas de Optimización . . . 3
1.1.3 Clasificación de los algoritmos de Optimización . . . 5
1.2 Aritmética de Intervalos . . . 7
1.2.1 Ejemplo de errores de redondeo. . . . 8
1.2.2 Aritmética de Intervalos reales. . . . 8
1.2.3 Extensión de la Aritmética de Intervalos . . . 12
1.2.4 Extensiones de funciones estándar . . . 13
1.2.5 Propiedades de las extensiones de intervalos . . . 14
1.2.6 Aritmética de Intervalos Computacional . . . 17
1.3 Inexistencia de mínimos globales . . . 18
1.3.1 Test del punto medio . . . 18
1.3.2 Test de monotonía . . . 18
1.3.3 Test de concavidad . . . 18
1.3.4 El método de Newton unidimensional clásico . . . 19
1.3.5 Método de Newton de intervalos uni-dimensional . . . 21
1.3.6 El método de Newton multi-dimensional clásico . . . 22
1.3.7 El método de Newton de intervalos multi-dimensional . . . 22
1.4 Diferenciación Automática . . . 23
1.5 Algoritmos de Ramificación y Acotación secuenciales . . . 25
1.5.1 Reglas básicas de los algoritmos de Ramificación y Acotación . . . 26
1.6 Resumen . . . 31

2 Raíces en funciones unidimensionales **33**
2.1 Introducción . . . 33
2.2 Algoritmo para encontrar la mínima raíz en una función unidimensional . . . 35
2.2.1 Regla de División Adaptativa . . . 41
2.2.2 Reordenación de las evaluaciones de la función de inclusión . . . 43
2.2.3 Resultados . . . 45

2.3 Algoritmo para encontrar la mínima raíz de un conjunto de funciones unidimensionales . . . 49
2.3.1 Resultados . . . 52
2.4 Resumen . . . 55

3 Algoritmos secuenciales de Optimización Global basados en Aritmética de Intervalos y Ramificación y Acotación 59
3.1 Algoritmo básico de Optimización Global basado en Intervalos . . . 59
3.2 El parámetro $p\overline{f^*}(X)$ . . . 66
3.3 División múltiple en algoritmos de Optimización Global basados en intervalos . . . 70
3.3.1 Resultados numéricos . . . 76
3.3.2 Evaluación de los resultados numéricos . . . 77
3.4 Criterio de Eliminación Heurístico en Algoritmos de Optimización Global basados en Intervalos . . . 84
3.4.1 El parámetro $p\overline{f^*}(X)$ como criterio de eliminación . . . 84
3.4.2 Resultados numéricos . . . 92
3.5 Nuevos criterios de selección basados en $p\overline{f^*}(X)$ . . . 96
3.5.1 Mejor Primero basado en $p\overline{f^*}(X)$ . . . 97
3.5.2 Modelo de selección híbrido basado en $p\overline{f^*}(X)$ . . . 99
3.6 Algoritmo de Optimización Global basado en Intervalos con Restricciones Computacionales . . . 103
3.6.1 Resultados numéricos . . . 106
3.7 Resumen . . . 109

4 Algoritmos paralelos de Optimización Global basados en Aritmética de Intervalos y Ramificación y Acotación 111
4.1 Algunas nociones sobre paralelismo . . . 111
4.1.1 Clasificación de las Arquitecturas Paralelas . . . 112
4.1.2 Medidas de rendimiento de algoritmos paralelos . . . 120
4.1.3 Paralelización de algoritmos serie . . . 121
4.1.4 Balanceo de la carga . . . 123
4.2 Algoritmos de Ramificación y Acotación paralelos . . . 126
4.2.1 Anomalías de los algoritmos de Ramificación y Acotación paralelos . 131
4.3 Paralelizaciones existentes de algoritmos de Optimización Global basados en intervalos . . . 133
4.3.1 Henriksen y Madsen . . . 133
4.3.2 Eriksson y Lindström . . . 134
4.3.3 Moore, Hansen y Leclerc . . . 136
4.3.4 Berner . . . 136

4.4 Nuevo criterio de balanceo de la carga en algoritmos paralelos de Optimización Global basados en intervalos 138
4.4.1 Modelo estático 141
4.4.2 Modelo semi-estático 141
4.4.3 Modelo dinámico 144
4.4.4 Resultados numéricos 145
4.5 Resumen 151

Conclusiones y principales aportaciones **153**

A Funciones test. **157**

B Direcciones de interés en Internet **177**
B.1 Aritmética de Intervalos 177
B.2 Arquitecturas y Algoritmos Paralelos 177
B.3 Diferenciación Automática 177
B.4 IEEE-754 177
B.5 Optimización Global 178
B.6 Problemas de corte con cero 178
B.7 Problemas de Optimización Combinatoria 178

Bibliografía **179**

Índice de Figuras

1.1 Ejemplo de un problema con múltiples mínimos locales. 3
1.2 *Función de inclusión* de la función $f(x) = x + sin(5x)$ en el intervalo $X = [1.1, 2.1]$, obtenida con las rutinas BIAS [90]. 12
1.3 Descripción gráfica del método clásico de Newton. 19
1.4 Ejemplo de uso de la *Regla de Selección Híbrida*. 29

2.1 Resultados gráficos del algoritmo *MRF* para los problemas 39 y 33, descritos en la Tabla 2.1. 40
2.2 Heurística usada en la *Regla de División Adaptativa*. 42
2.3 Resultados gráficos del los algoritmos *RF* (arriba-izquierda) y *MRF* (arriba-derecha), *MRFda* (debajo-izquierda) y *MRFro* (debajo-derecha), para el problema 14 (ver Tabla 2.1). 44
2.4 Representaciones gráficas de los resultados mostrados en la Tabla 2.3. . . . 54
2.5 Algoritmos *MRFroEx* (columna izquierda) y *MRFroEx-II* (columna derecha), para las funciones 8, 27 y 16 de la Tabla 2.1 y un valor de $\epsilon = 0.34$. . 56

3.1 Ejemplo de ejecución del Algoritmo 3.1.1 sobre una función unidimensional. 64
3.2 En la columna de la izquierda se muestran las cajas no eliminadas por el *Test del Punto Medio*. Las cajas más claras tienen un mayor valor de $p\overline{f^*}(X)$. En la columna de la derecha se representan los valores de $p\overline{f^*}(X)$ respecto a la distancia del centro de la caja X con el mínimo global más cercano. . . 67
3.3 En la columna de la izquierda se muestran zooms de las cajas no eliminadas por el *Test del Punto Medio*. Las cajas más claras tienen un mayor valor de $p\overline{f^*}(X)$. En la columna de la derecha se representan los valores de $p\overline{f^*}(X)$ respecto a la distancia del centro de la caja X con el mínimo global más cercano. 68
3.4 Ejecución del Algoritmo 3.1.1 para las funciones GP, C y L3. En las tres primeras filas se ha usado ϵ=10^{-1}, 10^{-2} y 10^{-3}, respectivamente. En la cuarta fila sólo se han dibujado las cajas de la tercera fila con $p\overline{f^*}(X) > 0.4$. 71
3.5 Métodos de división sobre un problema bidimensional. El método de división adaptativa se basa en (a), (b) y (d). 72

3.6 Posibles divisiones realizadas en un ejemplo bidimensional usando bisección de una sola coordenada. 73
3.7 Zooms sobre un punto minimizador global de las funciones Goldstein-Price, Six-Hump-Camel-Back y Levy3. 75
3.8 Valores de $p\overline{f^*}(X)$ durante la ejecución del Algoritmo 3.1.1, con una *Regla de División* 1DA y $\overline{f^*} = f^*$, para el problema Six-Hump-Camel-Back. . . . 86
3.9 Valores de $p\overline{f^*}(X)$ de las descendientes (hijas y nietas) de algunas cajas en diferentes fases de ejecución del Algoritmo 3.1.1 para el problemas Six-Hump-Camel-Back con $\epsilon = 10^{-4}$. 87
3.10 Valores de $p\overline{f^*}(X)$ durante la ejecución del Algoritmo 3.4.1-3.4.2, con una *Regla de División* 1DA, $\overline{f^*} = f^*$ y $Dc = 0.04$, para el problema Six-Hump-Camel-Back. 91
3.11 Valores de $p\overline{f^*}(X)$ durante la ejecución del Algoritmo 3.1.1, realizando bisección en todas las coordenadas (1DA), sin conocer f^* y seleccionando la caja X con mayor valor de $p\overline{f^*}(X)$ primero. 98
3.12 Valores de $p\overline{f^*}(X)$ durante la ejecución del Algoritmo 3.1.1, realizando bisección en todas las coordenadas (1DA), sin conocer f^* y usando un método de selección híbrido, basado en *Primero el Mejor* $w(X) \cdot p\overline{f^*}(X)$, seguido de una búsqueda en profundidad basada en $p\overline{f^*}(X)$. En la gráfica inferior se muestran las cien primeras iteraciones de la gráfica superior. 100
3.13 Valores de $p\overline{f^*}(X)$ durante la ejecución del Algoritmo 3.1.1, realizando una bisección de todas las coordenadas (1DA), sin conocer f^* y usando un método de selección híbrido basado en *Primero el Mejor* $w(X) \cdot p\overline{f^*}(X)$, seguido de una búsqueda en profundidad basada en $p\overline{f^*}(X)$ y un criterio de eliminación heurístico con $Dc = 0.04$. 102
3.14 Valores de $p\overline{f^*}(X)$ durante la ejecución del Algoritmo 3.6.1, realizando bisección en todas las coordenadas (1DA), sin conocer f^* y $N_m = 20$. 106
3.15 Ejecución de cinco iteraciones del Algoritmo 3.6.1 con $N_m = 40$ y una *Regla de Selección* basada en $p\overline{f^*}(X)$ (izquierda) y en el límite inferior de $F(X)$ (derecha). 108

4.1 Bus de datos (M = memoria; P = Procesador). 114
4.2 Topologías de redes estándar en computadores paralelos. 117
4.3 (a) Crossbar switch. (b) Red Butterfly. 119
4.4 Esquema del método *maestro-esclavo*. 134
4.5 Estructura de comunicación con un *mediador centralizado*. 137
4.6 Primera etapa del modelo semi-estático sobre la función GP, para 4 *esclavos*, 1, 4 y 16 cajas iniciales por *esclavo* y $\epsilon = 10^{-2}$. 143
4.7 Resultados numéricos de la ganancia de velocidad y del desbalanceo de la carga. 150

Índice de Tablas

1.1 Clasificación de los problemas de optimización. 4

1.2 Cualidades deseadas en los algoritmos de optimización. 5

2.1 Descripción de las funciones de prueba. Las funciones marcadas con $\triangleright$ no son diferenciables en algunos puntos de $S = [0.2, 7.0]$. 46

2.2 Resultados numéricos para $\epsilon = w(S) \cdot 10^{-15}$. 48

2.3 Resultados numéricos para los algoritmos MRFEx (columna A), MRFdaEx (columna B), MRFroEx (columna C) y MRFroEx-II (columna D). La precisión de los resultados es $\epsilon = w(S) \cdot 10^{-15}$, en todos los casos. La fila i contiene los resultados numéricos del conjunto de i funciones cuyo número se especifica en la columna N, desde la fila 1 a la i. 53

3.1 Resultados numéricos de las seis variantes del Algoritmo 3.1.1 para $\epsilon = 10^{-1}$. 78

3.2 Resultados numéricos de las seis variantes del Algoritmo 3.1.1 para $\epsilon = 10^{-2}$. 79

3.3 Resultados numéricos de las seis variantes del Algoritmo 3.1.1 para $\epsilon = 10^{-3}$. 80

3.4 Resumen de los resultados numéricos para los distintos tipos de división. Los valores medios de las distintas ejecuciones se muestran en las columnas (Med.). También se muestran los valores relativos comparados con los obtenidos con el tipo de división 1D. 81

3.5 Resumen de los resultados numéricos para los distintos tipos de división, conociendo el mínimo global de la función ($\overline{f^*} = f^*$). Los valores medios de las distintas ejecuciones se muestran en las columnas (Med.). También se muestran los valores relativos de los demás métodos de división, comparados con los obtenidos con el tipo de división 1D. 83

3.6 Resultados numéricos para el Algoritmo 3.1.1 con un método de división 1DA y f^* conocido. 93

3.7 Resultados numéricos para el Algoritmo 3.4.1-3.4.2 con un método de división 1DA, f^* conocido y $Dc = 0.04$. 94

3.8 Resumen de los resultados numéricos. Se muestran los valores medios (abreviados como Med.1) de los datos de la Tabla 3.6. También se muestran los valores relativos (abreviados como Rel.Med.2 = Med.2 / Med.1) de los valores medios de los datos en la Tabla 3.7 comparados con los de la Tabla 3.6. 95
3.9 Resultados numéricos para el problema EX2 con el algoritmo básico de Optimización Global (Algoritmo 3.1.1, página 63), sin conocer f^*, usando como *Regla de División* una bisección simple (1D) y como *Regla de Terminación* $w(F(X)) \leq \epsilon$. 95
3.10 Resultados numéricos para el problema EX2 con el nuevo criterio heurístico de eliminación (Algoritmo 3.4.1-3.4.2), sin conocer f^*, usando como *Regla de División* una bisección simple (1D), como *Regla de Terminación* $w(F(X)) \leq \epsilon$ y $Dc = 0.004$. 95
3.11 Resultados numéricos para el Algoritmo 3.6.1 realizando una bisección de las dos dimensiones más anchas de la caja seleccionada a la vez, sin conocer f^* y $\epsilon = 10^{-8}$. 107

4.1 Resultados numéricos para el método estático $\epsilon = 10^{-3}$. 147
4.2 Resultados numéricos para el método semi-estático. $\epsilon = 10^{-3}$ 148
4.3 Resultados numéricos para el método dinámico. $\epsilon = 10^{-3}$ 149

Capítulo 1

Introducción

1.1 Optimización matemática

La Optimización Matemática es el nombre formal de la rama de la ciencia computacional, que busca una respuesta a la pregunta: ¿Qué es mejor?, para problemas donde la calidad de la respuesta puede expresarse mediante un valor. Estos problemas aparecen en áreas científicas como Matemáticas, Física, Química y Biología, Ingeniería, Arquitectura, Economía y Administración, existiendo también un amplio rango de técnicas disponibles para resolverlos.

En esta sección se realizará una breve, y por lo tanto no exhaustiva, clasificación de los diferentes problemas de Optimización y de los métodos que existen para resolverlos.

1.1.1 Problema general

El objetivo en un problema de optimización es encontrar la mejor solución, a menudo expresada como un vector de reales, de forma que se minimice (o maximice) el valor de una función objetivo, posiblemente sujeta a una serie de restricciones sobre los posibles candidatos. La descripción general del problema es:

$$\begin{aligned} minimizar \quad & f(x) \\ sujeto\ a: \quad & g(x) \leq 0, \\ & h(x) = 0, \\ & \underline{x} \leq x \leq \overline{x}, \\ & x \in \mathbb{R}^n. \end{aligned} \tag{1.1}$$

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 (1.1), como:

$$S = \{x \in \mathbb{R}^n : g(x) \leq 0,\ h(x) = 0,\ \underline{x} \leq x \leq \overline{x}\} \tag{1.2}$$

En (1.2), $\underline{x}$ y $\overline{x}$ son límites (finitos) implícitos. Cuando sólo se da el tipo de restricciones por límites implícitos en la región de búsqueda, el problema de optimización se suele incluir en el grupo de problemas denominados sin restricciones.

Una solución global del problema (1.1) viene dada por $x^* \in S$ de forma que $f^* = f(x^*) \leq f(x),\ \forall x \in S$. Una solución local del problema (1.1), 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 (1.1) hasta que no se demuestra que es una solución global. Sólo se considera el problema de minimización, ya que el de maximización puede reescribirse como uno de minimización ($\max_{x \in S}\{f(x)\} = \min_{x \in S}\{-f(x)\}$).

Si todas la funciones definidas anteriormente son continuas, el conjunto de soluciones óptimas para el problema (1.1) es no vacío. Para resolver el problema (1.1), la mayoría 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, si se da la propiedad de convexitud, de otra forma, dicha convergencia no está asegurada.

Aunque muchos tipos de problemas pertenecen a la clase anterior, 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 [144, 138, 177, 51, 50, 67, 9].

El campo emergente de la Optimización Global se ocupa de problemas de programación matemática, con la (posible) presencia de múltiples mínimos locales. Este número de mínimos locales es normalmente desconocido y puede ser bastante grande. 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. Por lo tanto la Optimización Global puede llegar a ser extremadamente importante. Dichas estructuras de mínimos dan lugar a representaciones parecidas a paisajes montañosos. La mayoría de las aproximaciones clásicas no pueden aplicarse directamente para resolver estos problemas. Como ejemplo, la Figura 1.1 muestra una, relativamente simple, composición de funciones trigonométricas con ar-

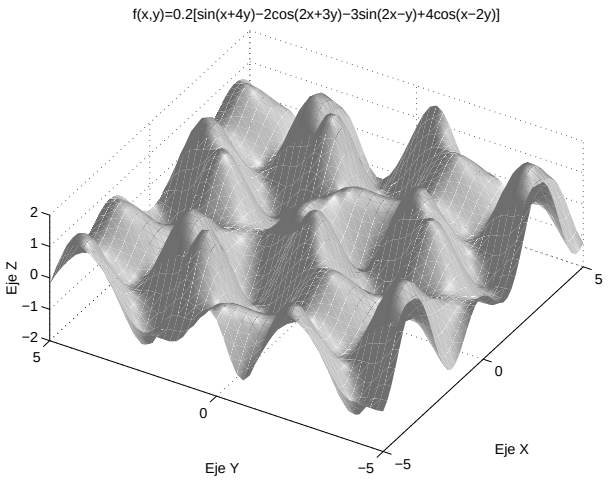


Figura 1.1: Ejemplo de un problema con múltiples mínimos locales.

gumentos polinomiales, sobre un espacio de búsqueda bidimensional. Naturalmente, ante tales circunstancias, es esencial usar una estrategia de búsqueda global apropiada.

Los primeros y esporádicos trabajos sobre Optimización Global surgieron a finales de los cincuenta. La evolución desde entonces ha sido muy grande, de forma que 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.

1.1.2 Clasificación de los problemas de Optimización

Una clasificación general, dependiente del número de variables, de su tipo y del tipo de funciones involucradas en (1.1) se describe en la Tabla 1.1 [B.5.8].

Por ejemplo, cuando las variables de los problemas de Optimización son variables enteras, estos problemas se llaman también problemas de *Programación Entera* (IP, Integer Programming). Si las funciones del problema son lineales se llaman problemas de *Programación Lineal* (LP, Linear Programming). Si por el contrario, dichas funciones no son lineales, nos encontramos ante problemas de *Programación No Lineal* (NLP Non Linear Programming), etc.

Una enumeración más concreta puede encontrarse en [145] y en [77], donde además existen numerosas referencias de los siguientes problemas de optimización: Programación Bilineal o Biconvexa, Optimización Combinatoria, Minimización Cóncava, Optimización Global Continua, Diferencial Convexa (DC), Programación Fraccional, Problemas Lineales y No Lineales Complementarios, Optimización Lipschitz, Problemas MiniMax, Optimización Multinivel, Programación Multiobjetivo, Programación Multiplicativa, Problemas de Redes de Trabajos, Programación Paramétrica No Convexa, Optimización Cuadrática, Programación Inversa Convexa, Optimización Global Separable y otros modelos proba-

Tabla 1.1: Clasificación de los problemas de optimización.

Característica	**Propiedad**	**Clasificación**
Número de variables	Una	Unidimensional
	Varias	Multidimensional
Tipo de variables	Reales continuas	Continua
	Enteras	Entera o discreta
	Reales continuas y enteras	Entera mezclada
	Permutaciones de enteros	Combinatoria
Tipo de Funciones	Funciones lineales	Lineal
	Funciones cuadráticas	Cuadrática
	Otras funciones no lineales	No Lineal.
Formulación del problema	Sujeto a restricciones	Con restricciones
	No sujeto a restricciones	Sin restricciones

bilísticos no convexos.

En cuanto a la complejidad de la resolución de estos problemas, Pardalos y Schnitger [139] mostraron que demostrar que un posible punto candidato es un óptimo local, para problemas de Programación Cuadrática, es un problema NP-Duro. Pardalos y Vavasis [140], demostraron también que la Programación Cuadrática con un autovalor negativo es un problema NP-Duro. Resolver el problema no convexo más simple de Programación No Lineal, es NP-Duro. Ésto indica, que el tiempo de resolución de cualquier algoritmo, en el peor caso, se incrementa exponencialmente con el tamaño del problema (teoría acerca de NP-Completitud puede encontrarse en [59]). Consecuentemente, problemas de dimensión cien, cincuenta o incluso diez, pueden considerarse como "grandes", dependiendo del problema de Optimización Global investigado (recuérdese la Figura 1.1). Algunos ejemplos en el ámbito de la Optimización Global con restricciones son [48]: Quesada y Grossman [148] resolvieron un problema de 20 variables con una función objetivo fraccional y restricciones convexas y un problema de 10 variables con una función objetivo cuadrática y restricciones no convexas y cuadráticas. Ryoo y Sabhinidis [162] resolvieron un problema de 12 variables con una función objetivo no convexa y restricciones cuadráticas. Philips y Rosen [143] han resuelto problemas de Programación Cuadrática, de 50 variables no lineales y 400 variables lineales, respectivamente. Horst y Tuy [78] han resuelto problemas de minimización cóncava de hasta 50 variables. Visweswaran y Floudas [182] han resuelto problemas de Programación Cuadrática Indefinida de 20 variables, problemas cuadráticos NLP con 39 variables y problemas cóncavos de Programación Cuadrática Indefinida de 25 variables, generados aleatoriamente.

Tabla 1.2: Cualidades deseadas en los algoritmos de optimización.

Corrección	No producir resultados incorrectos.
Completitud	Encontrar todas las posibles soluciones.
Finitud	Garantizar la convergencia.
Certeza	Probar la existencia o inexistencia de soluciones.

1.1.3 Clasificación de los algoritmos de Optimización

Existen muchas clases de algoritmos de Optimización Global que poseen fuertes propiedades teóricas de convergencia y al menos, en principio, se pueden implementar y aplicar directamente. Todos estos algoritmos rigurosos de Optimización Global, como se comentó anteriormente, tienen una demanda computacional que se incrementa en un orden no polinomial con respecto al tamaño del problema, incluso en las instancias más simples. Cuando se requiere una eficiencia mayor, los algoritmos de Optimización Global pueden hacer uso de fases de optimización local tradicionales, las cuales convergen rápidamente cuando están cerca de una solución o de un óptimo local, estando fuera del ámbito de estos métodos locales el encontrar todas las soluciones, o probar la existencia o ausencia de las mismas. Los métodos tradicionales pueden fallar, no obteniendo una convergencia en problemas difíciles de resolver. Sin embargo, la convergencia global debe garantizarse por medio de los componentes del algoritmo con ámbito global, los cuales, teóricamente, deberían usarse de una forma exhaustiva. Estas necesidades indican la dificultad de desarrollar algoritmos eficientes y robustos de Optimización Global. En la Tabla 1.1.3 se resumen las cualidades que debería tener un algoritmo de Optimización Global. Sólo unas pocas de las implementaciones existentes garantizan todas las cualidades mostradas. Normalmente, en las implementaciones se combinan ideas de diferentes aproximaciones. Una clasificación debida a Pinter [145] es la siguiente:

Partición adaptativa y estrategias de búsqueda: Algoritmos de Ramificación y Acotación, aproximaciones Bayesianas y métodos basados en Aritmética de Intervalos [153, 121, 133, 187, 71, 77, 78, 144, 88].

Algoritmos estocásticos adaptativos de búsqueda: Búsquedas aleatorias, enfriamiento simulado, algoritmos genéticos y algoritmos evolutivos [181, 187, 77, 118, 144].

Estrategias enumerativas: Por ejemplo, para resolver problemas de optimización combinatoria, o problemas de optimización con cierta estructura, como la que tienen los problemas de optimización cóncavos. [77, 78].

Búsqueda global mediante métodos locales: Aplicando una búsqueda de rejilla o búsquedas aleatorias para la fase global, y algoritmos de búsqueda local [77, 144].

Estrategias heurísticas: Deflación, tunelado, métodos de rellenado de función, aproximaciones infraestimadas de globales convexos, búsquedas tabú, etc [77, 144].

Métodos Homotópicos y otras aproximaciones relacionadas: Incluyendo métodos de punto fijo, algoritmos de pivote, etc.[77].

Métodos pasivos: Búsqueda uniforme en rejilla, búsqueda aleatoria pura [77, 144].

Métodos de aproximación sucesiva: También llamados de relajación, como los de corte de planos y otros tipos de corte, construcciones de minorantes y ciertas estrategias de descomposición y optimización anidadas [77, 144].

Métodos de trayectoria: Métodos basados en modelos de ecuaciones diferenciales, estrategias de caminos de búsqueda dirigidos, etc. [77].

Como puede verse, no existe un método universal de resolución de problemas de Optimización, eligiéndose uno u otro dependiendo de las características del problema, de los requerimientos de calidad en los resultados y del tiempo de respuesta del algoritmo.

Algunas direcciones de Internet, donde se pueden encontrar revisiones de los métodos de Optimización Global y del software que se encuentra actualmente disponible para este propósito, se muestran en [B.5.1-B.5.9].

La clasificación de los algoritmos de optimización mostrada anteriormente suele generalizarse haciendo una distinción entre algoritmos estocásticos y determinísticos.

Todos los métodos estocásticos usan algún factor aleatorio en sus algoritmos y la demostración de su convergencia depende de argumentaciones estadísticas [177, 163]. 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. Se basan en obtener valores 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 se encuentre la solución en un número finito de pasos, ni que el mínimo encontrado sea el global. Ejemplos de aplicaciones de métodos estocásticos con aportaciones del autor de esta tesis son: determinar los parámetros óptimos para la reconstrucción de imágenes a partir de proyecciones, donde se usó *Control Random Search* (CRS) en su versión paralela (PCRS) [56, 57] y la resolución del problema de geometría combinatoria, basado en encontrar el mejor empaquetado de n círculos de igual tamaño, en un cuadrado de tamaño unidad, de forma que el tamaño de los círculos sea el mayor posible y sin solapamientos,. Para este problema se aplicó una mezcla de los métodos de Aceptación por Umbral y de Búsqueda Estocástica de Agente Simple [175].

Al contrario que los métodos estocásticos, los métodos determinísticos no toman decisiones aleatorias en sus algoritmos y sus demostraciones de convergencia no dependen de la probabilidad. Algunos ofrecen un tiempo de terminación finito para problemas específicos

y otros convergen cuando el número de iteraciones se aproxima a infinito. La clase de los algoritmos determinísticos, donde se incluyen los métodos de Ramificación y Acotación, de cobertura, basados en intervalos, de tunelado y enumerativos, puede dividirse a su vez en dos categorías:

- 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 fiable para los problemas de optimización Global. Por otro lado, los métodos basados en cálculos de límites pueden producir soluciones globales rigurosas, si se implementan adecuadamente, teniendo en cuenta los errores de redondeo producidos por la Aritmética Computacional.

El método determinístico basado en el cálculo de los límites de la función objetivo que se estudiará en esta tesis está basado en el uso de la Aritmética de Intervalos y algoritmos de Ramificación y Acotación. El principal inconveniente de estos métodos es que se necesita conocer la descripción analítica del problema a resolver. En las siguientes secciones se desarrollarán los conceptos necesarios para el desarrollo de este tipo de algoritmos.

1.2 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 números, representados por 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. 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 [45, 174]. La Aritmética de Intervalos se atribuye a Moore con la aparición de su libro *Interval Analysis* en 1966 [125]. 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. A partir del trabajo de Moore han aparecido miles de artículos y decenas de libros dedicados a este tema [71].

1.2.1 Ejemplo de errores de redondeo.

Se pueden encontrar en [B.1.1] 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. Para enfatizar más sobre el problema, se mostrará a continuación un ejemplo debido a Rump [161]. La siguiente ecuación:

$$f(x,y) = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + \frac{x}{2y} \tag{1.3}$$

fue evaluada para $x = 77617$ e $y = 33096$. El programa para su cálculo fue escrito en Fortran. Éste 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 [102] se habla también de este ejemplo, mostrando el resultado exacto, que fue obtenido mediante la Aritmética de Intervalos implementada en el software VPI [47], 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 [115, 116, 157].

1.2.2 Aritmética de Intervalos reales.

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 reales. Se puede encontrar una amplia bibliografía sobre el Análisis de intervalos complejos en [71] y una descripción del mismo en [88].

Antes de describir con más detalle los conceptos relacionados con la Aritmética de Intervalos, se introducirán las notaciones que se utilizarán en este trabajo para la Aritmética Real y la Aritmética de Intervalos.

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.
$\mathbb{R}^{n\times m}$	El conjunto de las Matrices $n \times m$ de reales.

Los elementos 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:

$$x \cdot y = xy = \sum_{i=1}^{n} x_i \cdot y_i \tag{1.4}$$

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:

$$\nabla f(x) = f'(x) = \begin{pmatrix} \frac{\delta f}{\delta x_1}(x) \\ \frac{\delta f}{\delta x_2}(x) \\ \vdots \\ \frac{\delta f}{\delta x_n}(x) \end{pmatrix} \tag{1.5}$$

como el gradiente de la función f y

$$\nabla^2 f(x) = f''(x) = \begin{pmatrix} \frac{\delta^2 f}{\delta x_1^2}(x) & \frac{\delta^2 f}{\delta x_1 \delta x_2}(x) & \cdots & \frac{\delta^2 f}{\delta x_1 \delta x_n}(x) \\ \frac{\delta^2 f}{\delta x_2 \delta x_1}(x) & \frac{\delta^2 f}{\delta x_2^2}(x) & \cdots & \frac{\delta^2 f}{\delta x_2 \delta x_n}(x) \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\delta^2 f}{\delta x_n \delta x_1}(x) & \frac{\delta^2 f}{\delta x_n \delta x_2}(x) & \cdots & \frac{\delta^2 f}{\delta x_n^2}(x) \end{pmatrix} \tag{1.6}$$

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

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

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

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.
$\mathbb{I}^{n\times m}$	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:

$w(X)$	$= \overline{x} - \underline{x}$	Ancho de X.
$w(Y)$	$= \max_{1\leq i\leq n} w(Y_i)$	Ancho 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 punto real.

Extensión natural a intervalos

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

$$X \circ Y = [\min_{\substack{x\in X \\ y\in Y}} x \circ y, \max_{\substack{x\in X \\ y\in Y}} x \circ y] \tag{1.7}$$

Análogamente, para una función unaria ϕ sobre un intervalo:

$$\phi(X) = [\min_{x \in X} \phi(x), \max_{x \in X} \phi(x)] \tag{1.8}$$

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

$$X + Y = [\underline{x} + \underline{y}, \overline{x} + \overline{y}] \tag{1.9}$$

$$X - Y = [\underline{x} - \overline{y}, \overline{x} - \underline{y}] \tag{1.10}$$

$$X \cdot Y = [a, b] \tag{1.11}$$

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

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

$$\frac{1}{Y} = [\frac{1}{\overline{y}}, \frac{1}{\underline{y}}], \ 0 \notin Y \tag{1.12}$$

Definición 1.2.1 *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 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:

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

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 1.2.1 *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:*

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

Además, el rango real exacto de la función en el intervalo $[0,1]$ es $f([0,1]) = [-1/4, 0]$, por lo que la Aritmética de Intervalos puede obtener sobreestimaciones del 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.*

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 1.2.2 *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:*

$$f(X) \subseteq F(X), \ \forall X \subseteq D. \tag{1.13}$$

Teorema 1.2.1 *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 [88].*

Por lo tanto la Aritmética de Intervalos es una herramienta que nos permite obtener *funciones de inclusión* [125, 152]. Por ejemplo, en la Figura 1.2 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 [90].

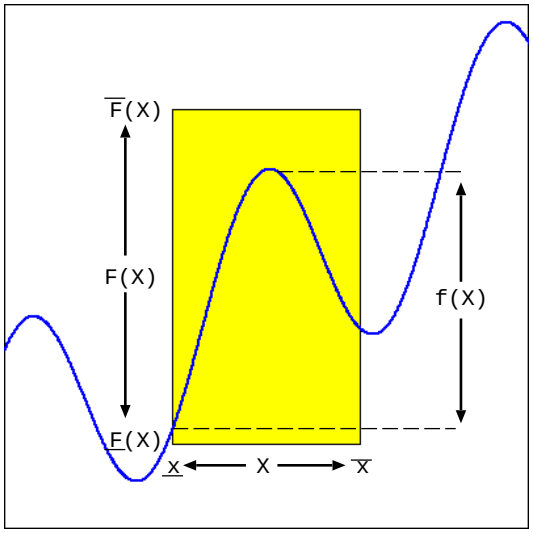


Figura 1.2: *Función de inclusión* de la función $f(x) = x + sin(5x)$ en el intervalo $X = [1.1, 2.1]$, obtenida con las rutinas BIAS [90].

1.2.3 Extensión de la Aritmética de Intervalos

En las operaciones básicas definidas anteriormente (1.9-1.12), 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 [85] y las mejoras de las mismas debidas a correcciones de inconsistencias [135, 156]. 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:

$$\frac{[\underline{x},\overline{x}]}{[\underline{y},\overline{y}]} = \begin{cases} X[1/\overline{y}, 1/\underline{y}] & \text{si } 0 \notin Y, \\ [-\infty,\infty] & \text{si } 0 \in X \text{ y } 0 \in Y, \\ [\overline{x}/\underline{y},\infty] & \text{si } \overline{x} < 0 \text{ e } \underline{y} < \overline{y} = 0, \\ [-\infty,\overline{x}/\overline{y}] \cup [\overline{x}/\underline{y},\infty] & \text{si } \overline{x} < 0 \text{ e } \underline{y} < 0 < \overline{y}, \\ [-\infty,\overline{x},\overline{y}] & \text{si } \overline{x} < 0 \text{ y } 0 = \underline{y} < \overline{y}, \\ [-\infty,\underline{x}/\underline{y}] & \text{si } 0 < \underline{x} \text{ e } \underline{y} < \overline{y} = 0, \\ [-\infty,\underline{x}/\underline{y}] \cup [\underline{x}/\overline{y},\infty] & \text{si } 0 < \underline{x} \text{ e } \underline{y} < 0 < \overline{y}, \\ [\underline{x}/\overline{y}],\infty] & \text{si } \underline{x} < 0 \text{ y } 0 = \underline{y} < \overline{y}, \\ 0 & \text{si } 0 \notin X \text{ y } 0 = Y. \end{cases} \tag{1.14}$$

Por ejemplo, de acuerdo con la fórmula (1.14), [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 [165]. Otro ejemplo es el debido a Markov [115, 116], 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*.

1.2.4 Extensiones de funciones estándar

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

$$\sqrt{X} = [\sqrt{\underline{x}}, \sqrt{\overline{x}}],\ X \geq 0 \tag{1.15}$$

$$ln(X) = [ln(\underline{x}), ln(\overline{x})],\ X \geq 0 \tag{1.16}$$

$$e^X = [e^{\underline{x}}, e^{\overline{x}}] \tag{1.17}$$

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 [76]. 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 1.2.2 *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 $w_j(X)$, de funciones estándar $\{w_j\}_{j=1}^p$, entremezcladas con las cuatro operaciones básicas sobre intervalos. Entonces esta extensión natural de intervalos, F, de f es una función de inclusión de f [88] .*

1.2.5 Propiedades de las extensiones de intervalos

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

Definición 1.2.3 *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 1.2.3 *La extensiones naturales de intervalos, tal como se definieron en el Teorema 1.2.2, son funciones de inclusión monótonas [88].*

La principal consecuencia del Teorema 1.2.3 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 1.2.4 *Supóngase que $f : \mathbb{R}^n \to \mathbb{R}$ se escribe como una secuencia de funciones estándar $\{w_j\}_{j=1}^p$, 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 w_i, entonces $F(X) = f(X)$, es decir, el intervalo de inclusión resultante es igual al rango exacto de la función real [88].*

Ejemplo 1.2.2 *Para $f(x) = x^2 - 2$, $F[-2,2] = [-2,2]^2 - 2 = [0,4] - [2,2] = [-2,2]$ y para $f(x_1, x_2) = x_1 x_2$, $F([-1,1],[-1,1])^T = [-1,1][-1,1] = [-1,1]$.*

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 1.2.4 *Sea $F : \mathbb{I}^n \to \mathbb{I}^m$ una extensión a 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 1.2.5 *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 [88, 2, 123, 152].*

Extensiones de segundo orden se pueden obtener mediante extensiones de intervalos de series de Taylor, también llamadas *expresiones centradas*, y limitando el rango de las derivadas.

Definición 1.2.5 *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 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 [88, 133].

Teorema 1.2.6 *Supóngase que los componentes de ∇F son extensiones a intervalos de ∇f de orden al menos uno. Entonces F_c es una extensión a intervalos de f de orden dos [91, 152].*

Desafortunadamente, aunque sea deseable en algoritmos de Optimización Global, parece difícil obtener extensiones de intervalos de orden mayor de dos para funciones arbitrarias [28, 43].

La expresión del valor medio es un caso especial de las *expresiones centradas* para funciones de inclusión basadas en intervalos. Otras *expresiones centradas*, así como su discusión matemática, aparecen en [152]. Dichas expresiones deben ser usadas para obtener límites más precisos en casos específicos.

Definición 1.2.6 *Krawczyk y Neumaier [133] introdujeron las expresiones basadas en la pendiente (Slope Forms), mostrando que si para un intervalo $S \in \mathbb{I}$ tal que, $\forall x \in X$ se tiene que*

$$f(x) = f(c) + s \cdot (x - c) \text{ para algún } s \in S, \tag{1.18}$$

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 a intervalos de la pendiente y no sólo por la extensión a intervalos de la derivada. Si se usa la extensión a intervalos de la pendiente, entonces se expande f respecto a un punto arbitrario pero fijo $c \in X$.

Definición 1.2.7 *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 por:

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

donde f no necesita ser diferenciable.

Notas:

1. Es fácil ver que $S = s_f(c,X)$ satisface (1.18) 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 a intervalos de la pendiente.
3. Si se asume que f es continuamente diferenciable se tiene que [133]:
$$\nabla f(X) = \bigcup_{c \in X} s_f(c,X)$$

Ejemplo 1.2.3 *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 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 forma del valor medio generalizada, usando el mismo valor de c [133].

En [153] se muestra que no es siempre bueno usar la extensión de intervalos 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 a intervalos tiende linealmente a infinito. Un problema todavía abierto en el Análisis de Intervalos es saber como debe ser la anchura de un intervalo para que la extensión del valor medio dé un resultado al menos tan bueno como el obtenido por la fórmula original [71].

Estudios más extensos sobre la Aritmética de Intervalos se encuentran en [2, 71, 123, 133]

1.2.6 Aritmética de Intervalos Computacional

En el Teorema 1.2.4 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. 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 [92]. 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.

El estándar IEEE 754-1985 [173], sobre aritmética de punto flotante, establece cuatro modos de redondeo: al *más cercano*, *superior*, *inferior* y *al cero*. Para obtener un intervalo resultado riguroso, el redondeo *superior* e *inferior* se usan cuando se computen los límites superiores e inferiores, 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 1.2.3, página 14) 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 $w(X) \to 0 \not\Rightarrow w(F(X)) \to 0$ [153].

La página principal de Aritmética de Intervalos en Internet se encuentra en la dirección

[B.1.2]. Otra página de interés es [B.1.3].

1.3 Inexistencia de mínimos globales

Los Teoremas 1.2.1 (página 12) y 1.2.3 (página 14), pueden usarse para determinar si existe un mínimo global en una caja determinada, utilizando los siguientes tests:

1.3.1 Test del punto medio

Supóngase que $\overline{f^*}$ 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 [84]. El 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 1.3.1 *Considérese la función $f(x) = x(x-1)$. Inspeccionado 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 haber 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 $f(x)$.

1.3.2 Test de monotonía

El gradiente de la función objetivo es cero en cualquier máximo, mínimo o punto de inflexión de la función. Si $0 \notin F_i'(X)$, 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 bordes de X [153].

Ejemplo 1.3.2 *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.*

1.3.3 Test de concavidad

Este test separa los mínimos de los máximos. 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 [68, 134].

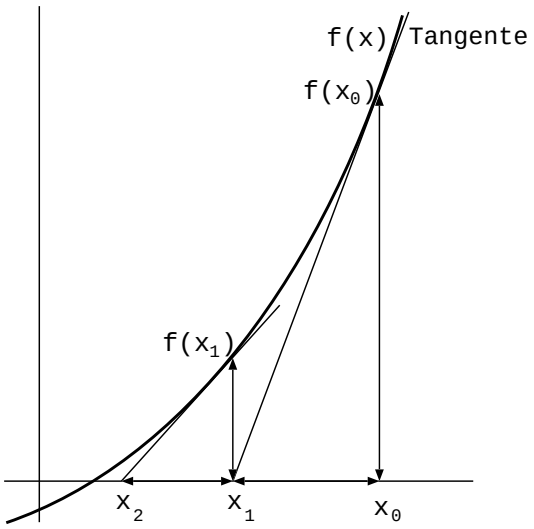


Figura 1.3: Descripción gráfica del método clásico de Newton.

1.3.4 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 1.3 puede observarse que:

$$f'(x_0) = \frac{f(x_0)}{x_0 - x_1} \tag{1.19}$$

por lo tanto,

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \tag{1.20}$$

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

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)},\ i = 0, \ldots, n \tag{1.21}$$

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. Como se pretende que $f(x) = 0$:

$$0 = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)(x - x_0)^2}{2!} + \cdots + \frac{f^{(n+1)}(\xi)(x - x_0)}{(n+1)!} \tag{1.22}$$

Usando sólo los dos primeros términos de la ecuación (1.22) se obtiene que:

$$0 \simeq f(x_0) + f'(x - 0)(x - x_0), \;\; x \approx x_0 - \frac{f(x_0)}{f'(x_0)} \tag{1.23}$$

Sustituyendo x por x_1, se obtiene la ecuación (1.20). 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:

$$\|x_{n+1} - x^*\| \leq c\|x_n - x^*\|^2 \tag{1.24}$$

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

$$\nabla f(x_i) \simeq \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}} \tag{1.25}$$

Obteniéndose la siguiente secuencia de puntos que converge a una raíz de f, siempre que se elija un buen punto inicial x_0:

$$x_{i+1} = x_i - f(x_i)\frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})} \tag{1.26}$$

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 que la función no tiene raíces. El siguiente ejemplo muestra una función para la cual el *método de Newton clásico* diverge.

Ejemplo 1.3.3 *Considérese la función $f(x) = x^{\frac{1}{3}}$. El método de Newton clásico establece que:*

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = -2x_n \tag{1.27}$$

Por lo que la secuencia diverge para cualquier punto inicial distinto de $x_0 = 0$.

1.3.5 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 [125] y carece de los errores debidos a las posibles divergencias del método clásico.

Teorema 1.3.1 *(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*

$$f(b) - f(a) = f'(\xi)(b - a) \tag{1.28}$$

La conclusión, desde el punto de vista geométrico, del Teorema 1.3.1 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 1.3.1, suponiendo que $0 = f(b)$:

$$0 = f(b) = f(a) + f'(\xi)(b - a) \tag{1.29}$$

entonces,

$$b = a - \frac{f(a)}{f'(\xi)} \Rightarrow b \in N(X) = a - \frac{f(a)}{F'(X)},\ si\ a, b \in X, \tag{1.30}$$

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

$$X_{n+1} = N(X_n) \cap X_n,\ \text{hasta que } X_{n+1} = X_n \text{ ó } X_{n+1} = \{\emptyset\} \tag{1.31}$$

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 [125, 153, 71]:

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, $w(X_{n+1}) \leq c \cdot w(X_n)^2$, siempre que los errores de redondeo computacionales sean menores que la precisión final requerida.

Ejemplo 1.3.4 *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.

1.3.6 El método de Newton multi-dimensional clásico

El método de Newton multi-dimensional es una generalización del uni-dimensional. Usando sólo los dos primeros términos de la versión multi-dimensional del Teorema de Taylor, se puede decir que:

$$0 = f(x) \approx f(x_0) + f'(x_0)(x - x_0),\ x \approx x_0 - f'(x_0)^{-1} f(x_0) \tag{1.32}$$

Reemplazando el vector x por el nuevo vector x_1 de la secuencia:

$$f'(x_0)(x_1 - x_0) = -f(x_0) \tag{1.33}$$

$$J_f(x_0)A = B_0, \tag{1.34}$$

donde $J_f(x_0)$ es el Jacobiano de f en x_0. Resolviendo el sistema lineal de la ecuación (1.34), se obtiene el nuevo vector $x_1 = A + x_0$.

1.3.7 El método de Newton de intervalos multi-dimensional

Exceptuando la opción de usar expresiones basadas en la pendiente (Definición 1.2.6, página 15), existe esencialmente un solo método de Newton de intervalos unidimensional. Por el contrario, el método de Newton de intervalos multi-dimensional varía dependiendo del método usado para resolver el sistema lineal de ecuaciones, precondicionamientos, etc.

La ecuación (1.30) puede generalizarse al caso multi-dimensional, estableciendo el vector $x_n = m(X_n)$ y $F'(X)$ como la extensión a intervalos del Jacobiano de f. Así :

$$N(X_{n+1}) = x_n - [F'(X_n)]^{-1} f(x_n),\ x_n \in X_n \tag{1.35}$$

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

$$F'(X_n)(N(X_{n+1}) - x_n) = -f(x_n) \tag{1.36}$$

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 (1.36), 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 [127, 128] y el método de Gauss-Seidel extendido a intervalos [89], aunque también se puede usar la extensión del método de eliminación de Gauss a intervalos [133]. Los libros [71, 133, 88] y [153] son buenas referencias para encontrar estudios de los diferentes métodos de Newton de intervalos multi-dimensionales, los cuales se escapan del ámbito de esta tesis.

1.4 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 (Sección 1.3.2, página 18) o el método de Newton (Sección 1.3.4, página 19). 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 [26], tienen programas hechos a medida para interpretar formatos de entrada estándar, como el formato MPS [130], SIF [26] o el propuesto por Neumaier [133], 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 [66] 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:

$$(u, u', u'') + (v, v', v'') = (u + v, u' + v', u'' + v'') \tag{1.37}$$

$$(u, u', u'') - (v, v', v'') = (u - v, u' - v', u'' - v'') \tag{1.38}$$

$$(u, u', u'') \cdot (v, v', v'') = (uv, uv' + u'v, uv'' + u'(v')^T + v'(u')^T + u''v) \tag{1.39}$$

$$\begin{aligned}(u, u', u'') \,/\, (v, v', v'') = {} & u/v, (u'v - uv')/v^2, \\ & (u'' - w'(v')^T - v'(w')^T - wv'')/v), \\ & v \neq 0 \ (0 \notin v)\end{aligned} \tag{1.40}$$

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

$$s((u, u', u'')) = (s(u), s'(u)u', s'(u)u'' + s''(u)u'(u')^T) \tag{1.41}$$

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 1.4.1 *Supóngase que se quiere conocer el valor de la siguiente función y de su derivada, en el intervalo* $X = [.99, 1.01]$*:*

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

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 $[2.920, 3.082]$ y el de $f'(X)$ es $[7.821, 8.183]$.

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 [172]. Enlaces a otros paquetes e información sobre diferenciación automática pueden encontrarse en [B.3.1].

1.5 Algoritmos de Ramificación y Acotación secuenciales

Los primeros algoritmos de Ramificación y Acotación 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 [46, 159, 100, 63]. 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 Ramificación y Acotación (*Branch and Bound*) a este método, fueron Little, Murty, Sweeny y Karel [108] (1963), en su artículo innovador sobre el problema del viajante de comercio. Lawer y Wood [101] estudiaron los algoritmos de Ramificación y Acotación, obteniendo una descripción independiente del problema, siendo así el primer artículo que presenta un modelo general de Ramificación y Acotación.

Los métodos de Ramificación y Acotación 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 Ramificación y Acotación [64, 95, 129, 132].

Una descripción de los modelos de algoritmos de Ramificación y Acotación que aparecen en los artículos [101, 120, 80, 81, 82, 96, 132], hecha por Bruin, Kindervater y Trienekens puede encontrarse en [39]. 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 Ramificación y Acotación está motivada para establecer un esquema común para todas las aplicaciones, mediante la descripción de unas reglas u operadores generales [80, 120]. Para especificar estas reglas nos basaremos en los formalismos utilizados por Corrêa y Ferreira [32], centrándonos en problemas donde se intenta encontrar el valor mínimo de una función objetivo.

1.5.1 Reglas básicas de los algoritmos de Ramificación y Acotación

Según Mitten e Ibaraki [120, 80] los algoritmos de Ramificación y Acotación pueden ser caracterizados por cuatro reglas:

- *Regla de Ramificación o División*: Establece cómo se descomponen los nodos del árbol de búsqueda.
- *Regla de Acotación*: Calcula un *límite inferior* de la solución óptima de un nodo.
- *Regla de Selección*: Define qué nodo será el siguiente a dividir.
- *Regla de Eliminación*: Establece cómo reconocer y eliminar nodos donde no se encuentra la mejor 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 del problema 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 [40, 108, 159, 180] [B.7.1], donde hay que encontrar el camino mínimo para visitar todas las ciudades deseadas.

Llamaremos a estas reglas las *reglas básicas* de los algoritmos de Ramificación y Acotación.

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 en cuanto a su tamaño.

Los algoritmos de Ramificación y Acotación 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 [27] 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 Ramificación y Acotación, 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 [80, 82, 120, 179].

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 1.5.1 *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 1.5.2 *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 [97]. 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*) [98]. Las hojas de T son los nodos solución. Los subproblemas críticos son aquellos $v \in V$ tal que $li(v) < f^*$ [149]. Los nodos críticos de un árbol-BB, T, forman el subárbol crítico T^* de T [98]. Es fácil ver que todos los subproblemas críticos deben seleccionarse durante la ejecución de un algoritmo de Ramificación y Acotación. 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 eficacia y al espacio de memoria requerido [58]. A continuación se describen las reglas de selección heurísticas más comúnmente utilizadas:

Búsqueda en Profundidad (Depth-First): La función de selección δ selecciona, 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 [111]. 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 la 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 ramificación.

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 $\overline{f^*}$.

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 1.5.5, página 30).

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:

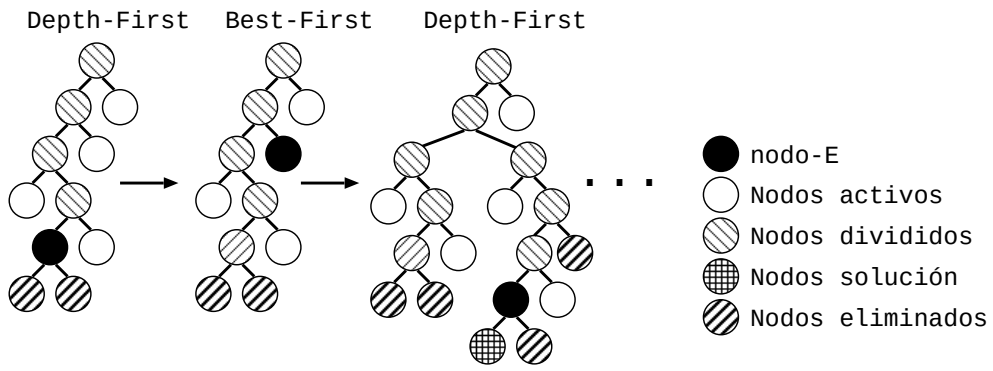


Figura 1.4: Ejemplo de uso de la *Regla de Selección Híbrida*.

- 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 [62] 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 se muestra en la Figura 1.4. 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 [166].

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 [83].

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

Definición 1.5.3 *La Regla de División de un nodo-E, v, en un algoritmo de Ramificación y Acotación 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 en el que se aplique el algoritmo de Ramificación y Acotación.

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 1.5.4 *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 1.5.1 *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 [31, 80].*

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

Definición 1.5.5 *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 [82], 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 1.5.6 *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 instancia, será aplicada en mayor o menor medida, 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.

1.6 Resumen

En este capítulo se han presentado de forma general los tipos de problemas de Optimización Global y los métodos propuestos para resolverlos. En esta tesis nos centraremos en problemas de Optimización Global con la única restricción de tener límites implícitos en la región de búsqueda y en los algoritmos que hacen uso de técnicas de Ramificación y Acotación, donde la *Regla de Acotación* está determinada por la *función de inclusión* que se obtiene mediante la Aritmética de Intervalos.

Capítulo 2

Raíces en funciones unidimensionales

En este capítulo se presentan varios algoritmos que resuelven eficientemente el problema de encontrar la mínima raíz en funciones unidimensionales. El primer problema consiste en determinar la mínima raíz de una función unidimensional sobre un determinado intervalo. Se supone que se tiene definida la función analíticamente y que puede ser no diferenciable con un gran número de máximos y mínimos. El segundo problema es una generalización del primero y trata de encontrar la mínima raíz en un conjunto de funciones unidimensionales. Ambos problemas se plantean como problemas de Optimización Global y serán tratados mediante algoritmos de Ramificación y Acotación con Aritmética de Intervalos. En la Sección 2.1 se hace una descripción de estos problemas y de los métodos generalmente utilizados para su resolución. En la Sección 2.2 se presentan nuevos algoritmos para resolver el problema de encontrar la mínima raíz de una función unidimensional y en la Sección 2.3 se generalizan los algoritmos anteriores para encontrar la mínima raíz de un conjunto de funciones unidimensionales.

2.1 Introducción

El problema de encontrar la mínima raíz en funciones unidimensionales con limitaciones en la región de búsqueda, puede encontrarse en muchas aplicaciones de diversas áreas. Un ejemplo típico de estas aplicaciones aparece dentro del campo de la computación gráfica y de la visualización, donde una de las operaciones fundamentales en el trazado de rayos (*ray tracing*), consiste en el cálculo de la intersección entre los rayos y la superficie de los objetos [86, 119]. Otros ejemplos de estas aplicaciones son: técnicas de procesado de señal, por ejemplo, en la identificación no paramétrica de una señal [188], en el procesamiento de imágenes [24], en las transformadas *wavelet* [113], en el cálculo de los autovalores de una matriz [5], en el diseño de herramientas software de simulación eléctrica y electrónica para el análisis del dominio temporal en redes con conmutadores [6], etc.

En este capítulo se consideran dos problemas que aparecen en tales aplicaciones. El primero consiste en establecer un algoritmo general, simple, riguroso y rápido para encontrar la mínima raíz de una función unidimensional definida analíticamente. El segundo problema es una generalización del primero y trata de encontrar la mínima raíz para un conjunto de funciones unidimensionales definidas analíticamente.

Existen varias formas de formular el primer problema, como por ejemplo:

Dada una función real $f : x \in S \subset \mathbb{R} \to \mathbb{R}$; $S = [\underline{s}, \overline{s}]$, $\underline{s} \leq \overline{s}$, donde S es el dominio o la región de búsqueda del problema, determinar:

$$x^* = \min\{x \ : \ f(x) = 0, x \in S\} \tag{2.1}$$

o

$$\begin{aligned} x_u &= \min\{x \ : \ f(x) \cdot f(\underline{s}) \leq 0, x \in S\}, \text{ y calcular} \\ x_l &= \max\{x \ : \ f(x) \cdot f(\underline{s}) \geq 0, x < x_u\} \end{aligned} \tag{2.2}$$

La mayoría de los algoritmos que fueron propuestos por otros autores se diseñaron para encontrar todas las raíces de la función en un intervalo determinado, pero no para encontrar sólo la mínima raíz. Sin embargo, existen muchas aplicaciones donde sólo la mínima raíz contiene información de interés para el problema a tratar [5, 6, 24, 86, 113, 119, 188] y por lo tanto es conveniente desarrollar algoritmos más eficientes para resolver este problema. Algunos de los métodos que se pueden encontrar en la literatura son :

- El método más simple está basado en una técnica de rejilla (*GRID*), que produce una serie de muestreos, empezando en el margen izquierdo y distanciados un tamaño constante ϵ, hasta que dos muestras consecutivas de la función ($f(x)$ y $f(x+\epsilon)$) tengan distinto signo [6]. Hay que tener cuidado con este tipo de métodos ya que si la distancia de muestreo es grande, es posible saltarse una raíz y si es demasiado pequeña el número de iteraciones puede ser muy grande.

- La segunda aproximación consiste en usar algoritmos de búsqueda local para encontrar raíces de ecuaciones y obtener así una convergencia rápida al punto x^* [B.6.1]. El principal inconveniente de estos métodos es que la convergencia no está asegurada cuando la función $f(x)$ tiene muchos máximos y mínimos. Además, si la función $f(x)$ tiene varias raíces, diferentes elecciones de las condiciones iniciales pueden dar lugar a soluciones diferentes para la ecuación $f(x) = 0$.

  Uno de los métodos más utilizados es el *método de Newton clásico* (Sección 1.3.4, página 19). Este método genera una secuencia de valores x_i que convergen a x^*, bajo la suposición de que $f(x)$ es localmente lineal en x^*. Aunque la convergencia de este tipo de métodos suele ser cuadrática, no se puede asegurarse cuando $f(x)$ tiene muchos máximos y mínimos en S [146]. Además, su convergencia depende del punto inicial elegido. Por ejemplo, el *método de Newton* no converge para la función $f(x) = x^3 - x$ si el punto inicial es $x_0 = 1/\sqrt{5}$.

- El tercer método propuesto por Daponte et al. [36, 37, 164] y Kalra y Barr [86], se encuentra dentro de los algoritmos de Ramificación y Acotación que hacen uso de las derivadas Lipschitz. En [36, 37, 164] se proponen diferentes soluciones para resolver problemas en el campo de la Ingeniería Electrónica. En el artículo de Karla y Barr [86] se encuentran aplicaciones en el campo de la computación gráfica y la visualización.

- El cuarto método para resolver (2.2) consiste en el uso de algoritmos generales de Optimización Global basados en estrategias de Ramificación y Acotación y en Aritmética de Intervalos, como los propuestos en [71, 88, 153].

  El método de Mitchell [119] se basa en el algoritmo de intervalos introducido por Moore [125]. Mitchell distingue dos etapas: la primera etapa consiste en aislar las raíces de la función, encontrando el conjunto de subintervalos donde se sabe con certeza que existe una sola raíz; en la segunda etapa se realiza un refinamiento en el que se reduce el tamaño de los intervalos producidos en la etapa inicial, usando técnicas de búsqueda local, como el *método de Newton*.

El segundo problema considerado en este capítulo es una generalización del problema 2.1. Existen varias aplicaciones donde es necesario el calcular la mínima raíz de un conjunto de p funciones, en vez de en una única función, es decir, encontrar la mínima de las raíces mínimas de varias funciones [37, 86, 119]. Un ejemplo de estas aplicaciones aparece en el el campo de la computación gráfica donde la escena a visualizar está compuesta por un conjunto de objetos definidos por sus superficies. Una de las operaciones fundamentales a realizar es la de calcular la intersección entre los rayos de luz y las superficies. La visualización de la escena implica que para cada rayo solo se necesita conocer una de las intersecciones de ese rayo con todos los objetos (la más cercana al punto de observación) [86, 119]. Matemáticamente el problema puede formularse como:

Dado un rayo, representado paramétricamente por un punto inicial $a = (a_x, a_y, a_z)$, un vector director V y un conjunto de p superficies, $f_i(x, y, z) = f_i(a_x + tV_x, a_y + tV_y, a_z + tV_z) = f_i(t)$; i.e. $f_i : t \in R \rightarrow R$; $t \in S = [\underline{s} = 0, \overline{s}]$, $i = 1, \ldots, p$, determinar:

$$t^* = \min\{t \in S : f_i(t) = 0; \quad \forall i,\ i = 1, \ldots, p\} \tag{2.3}$$

En la Sección 2.2 se presentan los nuevos algoritmos para resolver el problema (2.1). A partir de los resultados obtenidos con estos algoritmos, en la Sección 2.3 se desarrollarán nuevos métodos para resolver el problema (2.3).

2.2 Algoritmo para encontrar la mínima raíz en una función unidimensional

Nuestra propuesta de algoritmo para la resolución del problema (2.1) está basada en las técnicas de Ramificación y Acotación y Aritmética de Intervalos, e incorpora las ideas

descritas en [36, 37, 164], pero sin la sobrecarga computacional producida por el cálculo de derivadas, con lo que puede aplicarse también a problemas no diferenciables.

A continuación se definen las reglas básicas de los algoritmos de Ramificación y Acotación (Sección 1.5) utilizadas para este problema:

Definición 2.2.1 *Se define la Regla de División de forma que el intervalo seleccionado, $X \subseteq S$ se divide usando el método tradicional de bisección, generando dos subintervalos de igual tamaño que se almacenan en una lista de trabajo en un orden no decreciente basado en $\underline{x}$, i.e., el subintervalo con mayor $\underline{x}$ primero.*

Definición 2.2.2 *La Regla de Selección establece el siguiente intervalo a procesar, como aquel que se encuentra en la cabeza de la lista de trabajo, que está organizada mediante un esquema LIFO (Last-In-First-Out). Por lo tanto, el algoritmo siempre trabaja en el intervalo activo más a la izquierda.*

Definición 2.2.3 *La Regla de Acotación está basada en la evaluación de la función de inclusión, F, de f, obtenida mediante la Aritmética de Intervalos.*

En esta tesis, la función de inclusión usada es la extensión natural a intervalos [124, 152, 90] (véase la Sección 1.2.2, página 8).

Definición 2.2.4 *Para el problema del primer corte con cero, la Regla de Eliminación Tradicional se basa en eliminar los intervalos $X \subseteq S$ que cumplen que $0 \notin F(X)$, porque entonces es seguro que $0 \notin f(X)$.*

Definición 2.2.5 *La Regla de Terminación establece que los intervalos $X \subseteq D : w(X) \leq \epsilon$, que fueron seleccionados pero no rechazados, i.e, $0 \in F(X)$, se almacenen en la llamada lista final en vez de en la lista de trabajo, perteneciendo así a la solución final. El parámetro de entrada ϵ determina la precisión requerida para la solución del problema. El algoritmo finaliza cuando la lista de trabajo se encuentra vacía.*

Se proponen cuatro algoritmos que siguen el esquema anterior, en los que se ha usado la siguiente notación para sus parámetros de entrada:

S : El dominio del problema, también llamado la región de búsqueda.
F : La extensión natural a intervalos de f.
L : La lista de trabajo con prioridad.
Q : La lista final que contiene los intervalos $X : w(X) \leq \epsilon$ y $0 \in F(X)$.
ϵ : El criterio de terminación. Si $w(X) \leq \epsilon$, X se almacena en Q.

El Algoritmo 2.2.1 (Roots Finder, (RF)) es similar al propuesto por Mitchell [119], con la diferencia de que *RF* no computa derivadas y por lo tanto no aplica el *Test de Monotonía*. El algoritmo comienza inicializando la lista final, Q, al conjunto vacío y la

Algoritmo 2.2.1 : Roots Finder

```
1  funct RF(S, F, ε, L, Q) ≡
2     Q := {∅}; L := {S}                          Inicialización de las listas final y de trabajo
3     while (L ≠ {∅})
4           X := Cabeza(L)
5           L := L − {X}                          X ←pop L
6           if (0 ∈ F(X))                         Regla de eliminación tradicional
7              if (w(X) ≤ ε)                      Regla de Terminación
8                 Q := Q + {X}                    X pertenece a la solución final
9           else
10                Subdivide(X, X_l, X_r)          X_l = [x, m(X)], X_r = [m(X), x̄]
11                L := L + {X_r} + {X_l}          L ←push X_r, L ←push X_l
12    return Q
```

lista de trabajo, L, a la región de búsqueda (línea 2). El algoritmo termina cuando no existen más intervalos en la lista de trabajo para ser procesados (línea 3). El siguiente intervalo a procesar está determinado por la *Regla de Selección* (línea 4). La *Regla de Acotación* se aplica en la línea 6. Si el intervalo no puede ser eliminado por la *Regla de Eliminación Tradicional* (línea 6), ni cumple la *Regla de Terminación* (línea 7), se divide usando la *Regla de División* (línea 10). Los subintervalos generados (X_r y X_l) se almacenan en la lista de trabajo en el orden establecido por la *Regla de Selección* (línea 11).

Además de la *Regla de Eliminación Tradicional* en el algoritmo original de Mitchell se eliminan aquellos intervalos $X : 0 \notin F'(X)$ y $\underline{[F(\overline{x}) \cdot F(\underline{x})]} > 0$, es decir, es seguro que la función es monótona y completamente positiva o negativa en X. En las comparaciones numéricas realizadas se usará el Algoritmo 2.2.1 para medir la eficiencia de los nuevos métodos.

El Algoritmo 2.2.2 (Minimal Root Finder (MRF)) [22, 20], al contrario que el Algoritmo 2.2.1, está diseñado para detectar sólo la mínima raíz de una función unidimensional. Esta tarea se lleva a cabo mediante la eliminación de la lista de trabajo, L, de todos aquellos intervalos, X_i de los que se sabe que no contienen la mínima raíz. Con este propósito, se han añadido al Algoritmo 2.2.1 varias mejoras especificas para resolver el problema (2.1), obteniendo como resultado el Algoritmo 2.2.2. Estas mejoras se detallan a continuación.

La siguiente proposición determina el error cometido cuando se establece como cierto que $F([x, x]) = 0 \Rightarrow f(x) = 0$ y su utilización para descartar subregiones de búsqueda donde no se encuentra la mínima raíz.

Proposición 2.2.1

Dada una función $f : S \subset \mathbb{R} \to \mathbb{R}$ y F, la función de inclusión de f obtenida mediante Aritmética de Intervalos, si para un punto $x \in X \subseteq S$ se tiene que $0 \in F(x)$, entonces se

Algoritmo 2.2.2 : Minimal Root Finder

```
1  funct MRF(S, F, ε, L, Q) ≡
2    Q := {∅}; L := {S}                          Inicialización de las listas final y de trabajo
3    if (0 ∈ F(s))
4      Q = Q + [s, s]
5      return Q                                  El algoritmo finaliza con la solución s
6    if (F̄(s) < 0)                               Precondición
7      F se cambia por − F
8    while (L ≠ {∅})
9          X := Cabeza(L)
10         L := L − {X}                          X ←pop L
11         if (0 ∉ F(X))                         Regla de eliminación tradicional
12           if (Q ≠ {∅})
13             L := {∅}
14       else                                    0 ∈ F(X)
15           if (w(X) ≤ ε)                       Regla de Terminación
16             Q := Q + {X}                      X pertenece a la solución final
17         else
18             Subdividir(X, X_l, X_r)           X_l = [x, m(X)], X_r = [m(X), x̄]
19             L := L + {X_r} + {X_l}            L ←push X_r, L ←push X_l
20             if (F(x̄_l) ≤ 0)                   Test del Punto Negativo
21               Eliminar todos los X_i ∈ L : x_i ≥ x̄_l   Análogo a L := {X_l}
22   return Q
```

puede concluir que la mínima raíz de f en S no se encuentra a la derecha del punto x. El posible error cometido se encuentra en el intervalo $[0, w(F(x))]$.

Demostración. Aunque sí es cierto, que debido a las sobreestimaciones que aparecen cuando se usa la Aritmética de Intervalos y al redondeo directo usado en la Aritmética Computacional, $0 \in F(x) \not\Rightarrow f(x) = 0$, también es verdad que con la *función de inclusión* usada no se puede obtener una precisión mayor, debido a que $w([x, x]) = 0$, siendo el valor $w(F(x))$ una cota superior del posible error cometido. ■

Por lo tanto, el encontrar un punto $x \in X \subseteq S$ con $0 \in F(X)$, nos permite rechazar la subregión de búsqueda a la derecha de x. En el caso particular en el que $0 \in F(\underline{s})$, entonces se puede concluir que la mínima raíz se encuentra en $\underline{s}$ (Algoritmo 2.2.2, línea 3), con lo que el algoritmo termina con la solución $Q = \{[\underline{s}, \underline{s}]\}$. Generalmente el posible error cometido es cercano a la precisión de la máquina.

Teorema 2.2.1 *Sea una función $f : S \subset \mathbb{R} \to \mathbb{R}$, F la función de inclusión de f obtenida mediante Aritmética de Intervalos y sea un algoritmo de Ramificación y Acotación*

definido por las reglas básicas establecidas en las definiciones 2.2.1 a 2.2.5, que hace uso de la Proposición 2.2.1 y que evalúa F puntualmente en el límite superior del subintervalo izquierdo (línea 20), generado a partir de la división del intervalo seleccionado. Si $0 \notin F(\underline{s})$ y la función se precondiciona de forma que $\underline{F}(\underline{s}) > 0$ (línea 7), entonces el encontrar un punto negativo, $\{x \in X \subseteq S : \overline{F}(x) \leq 0\}$, nos permite afirmar que la mínima raíz no se encuentra a la derecha de x, por lo que esta región puede eliminarse de la búsqueda.

Demostración. Denotamos por $A \subseteq S$, a la subregión de S que en un determinado momento del algoritmo permanece activa, es decir, $\{A = [\underline{a}, \overline{a}] = \bigcup X_i : 0 \in F(X_i)$ y $X_i \in L\}$. Al comienzo del algoritmo $A = S$ y debido al precondicionamiento impuesto, $\underline{F}(\underline{a}) > 0$. Si se encuentra un punto $\{y \in A : \overline{F}(y) \leq 0\}$, A se reduciría al intervalo $[\underline{a}, y]$. El intervalo A tiene que incluir una raíz debido a que los signos de $f(\underline{a})$ y $f(\overline{a})$ son opuestos, pudiéndose eliminar así la subregión de S a la derecha de y.

La única forma de que A posiblemente no contenga una raíz es si $\overline{F}(\underline{a}) < 0$, durante la ejecución del algoritmo. Esto es imposible que ocurra ya que el valor de $\underline{a}$ sólo puede incrementarse mediante la aplicación de la *Regla de Eliminación Tradicional*, en el intervalo $\{X : \underline{x} = \underline{a}\}$, para lo cual debe cumplirse que $0 \notin F(X)$. Esto implica que $f(\overline{x}) > 0$, actualizándose la nueva subregión activa al intervalo $A = [\overline{x}, \overline{a}]$. Por lo tanto, se puede concluir que durante toda la ejecución del Algoritmo 2.2.2 $f(\underline{a}) > 0$. ■

Nótese que la precondición impuesta en la línea 7 no cambia la posición de la mínima raíz de f en S.

Corolario 2.2.1 *En el algoritmo 2.2.2, si un punto $x \in X \subseteq S$ cumple que $0 \in F(x)$ (Proposición 2.2.1) o que $\overline{F}(x) \leq 0$ (Teorema 2.2.1), la región a la derecha de x puede descartarse de la búsqueda. Ambas condiciones pueden aunarse para obtener un nuevo criterio de eliminación: $\underline{F}(x) \leq 0$ (línea 20), al que denotaremos como Test del Punto Negativo.*

Aún añadiendo el *Test del Punto Negativo* a la *Regla de Eliminación Tradicional*, el Algoritmo 2.2.2 podría continuar la búsqueda hacia la derecha de un intervalo que: contiene la mínima raíz, ha alcanzado la *Regla de Terminación* ($w(X) \leq \epsilon$) y $\underline{F}(\overline{x}) > 0$. Para frenar esta búsqueda hacia la derecha, se fuerza la parada del algoritmo, mediante el vaciado de la lista de trabajo (línea 13), cuando existe algún intervalo en la lista final (línea 12) y el intervalo seleccionado ha sido rechazado (línea 11).

En las Figuras 2.1 y 2.3, se muestran ejemplos gráficos de la ejecución de los algoritmos *RF* y *MRF*. En ellas, cada caja representa un intervalo $[\underline{x}, \overline{x}]$ y los límites inferior y superior de la *función de inclusión* evaluada sobre ese intervalo, $[\underline{F}(X), \overline{F}(X)]$. Se pueden distinguir tres tipos de cajas entre los intervalos evaluados. La cajas de color gris representan a los intervalos que fueron evaluados y rechazados por la *Regla de Eliminación Tradicional*, las cajas donde sólo se dibuja el borde, representan los intervalos que fueron evaluados y divididos y las cajas de color negro representan los intervalos que alcanzaron la lista final Q. Los puntos negros de las gráficas representan las evaluaciones de $F(\underline{s})$ (algoritmo *MRF*,

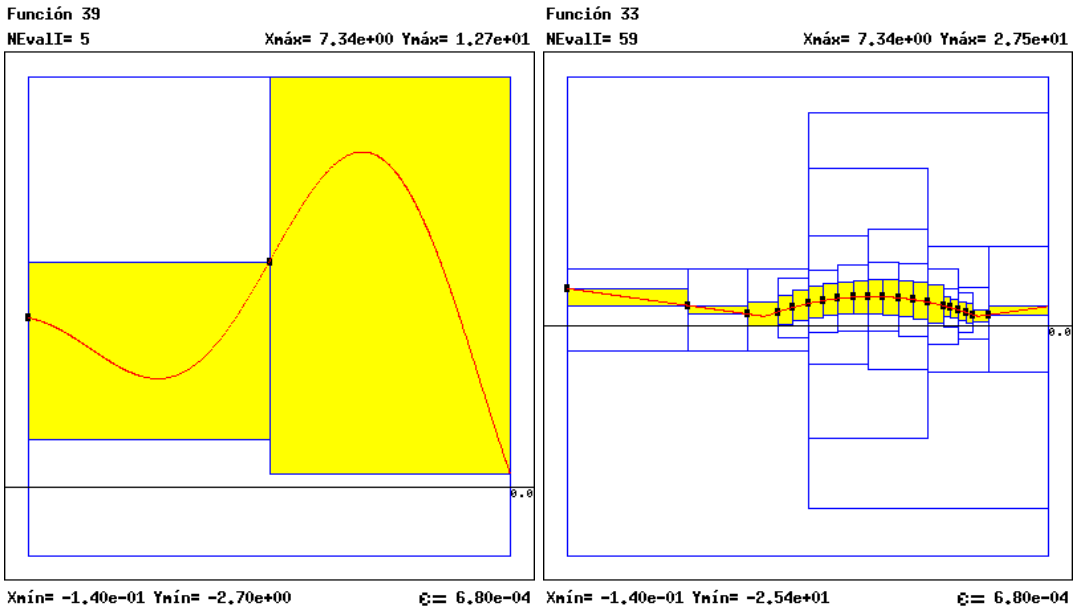


Figura 2.1: Resultados gráficos del algoritmo *MRF* para los problemas 39 y 33, descritos en la Tabla 2.1.

línea 6) y del *Test del Punto Negativo* (línea 20). El número de evaluaciones de la *función de inclusión* (NEvalI) y el parámetro de terminación (ϵ) se muestran en la parte superior e inferior de cada ejemplo.

La *Regla de Eliminación Tradicional* es capaz de eliminar gran cantidad del espacio de búsqueda, principalmente en funciones que no tienen ningún corte con cero en el dominio del problema, por lo que puede obtenerse una convergencia rápida a la solución. Como ejemplo de este comportamiento, la gráfica de la izquierda de la Figura 2.1 muestra una función en la que sólo se han realizado cinco evaluaciones de intervalos para determinar que no existe ningún corte con cero. En la gráfica pueden observarse claramente tres de los intervalos evaluados (uno dividido y dos rechazados). Las restantes evaluaciones puntuales se encuentran en $\underline{s}$, para establecer la precondición inicial (línea 6) y en el punto medio de S (línea 20), porque se aplicó el *Test del Punto Negativo*.

Tal como se comentó en la Sección 1.2.5 (página 14), el principal inconveniente de la Aritmética de Intervalos para obtener *funciones de inclusión* es que las inclusiones obtenidas son generalmente sobreestimaciones de los rangos reales de la función objetivo en un intervalo. Debido a esto, en algunas ocasiones la *Regla de Eliminación Tradicional* actúa sólo cuando los intervalos son suficientemente pequeños, como ocurre en la gráfica derecha de la Figura 2.1. En esta gráfica puede observarse también como $w(F(X))$ decrece conforme lo hace $w(X)$. Bajo estas circunstancias, el uso de mejores *funciones de inclusión* como las *expresiones centradas* y las *expresiones basadas en la pendiente* (Definiciones 1.2.5

y 1.2.6 respectivamente, página 15), podrían acelerar la convergencia del algoritmo.

En la Figura 2.3 se muestra la capacidad del *Test del Punto Negativo* para rechazar intervalos que no contienen la mínima raíz. En las gráficas superiores de la Figura 2.3 se comparan los algoritmos *RF* (superior-izquierda) y *MRF* (superior-derecha). Para el algoritmo *MRF* no se han dibujado los intervalos eliminados de la lista de trabajo (y no evaluados con Aritmética de Intervalos) por *Test del Punto Negativo*. Durante la ejecución del algoritmo la localización de los puntos que cumplen el *Test del Punto Negativo* (indicada mediante una flecha en la parte inferior de la gráfica) va moviéndose de derecha a izquierda. Si las nuevas ideas de eliminación no se usan (algoritmo *RF*), se determinarían todos los cortes con cero con la precisión establecida por el parámetro de terminación ϵ (gráfica superior-izquierda de la Figura 2.3).

Los criterios de selección y eliminación propuestos cooperan para obtener una convergencia rápida del algoritmo, evitando la búsqueda en L de todas las raíces de la función objetivo. El algoritmo siempre selecciona de L el intervalo que se encuentra más a la izquierda y por lo tanto, que no ha sido rechazado. De esta forma, se examinan los intervalos de izquierda a derecha y cualquier intervalo $X \in L$ no es evaluado a menos que se verifique, mediante la *Regla de Eliminación Tradicional*, que todos los intervalos generados a la izquierda de X no contienen ninguna raíz. Así se garantiza que la solución encontrada contiene la mínima raíz de la función f en el dominio S. Por otro lado, si A es la unión de todos los intervalos activos en cualquier momento del algoritmo *MRF*, el algoritmo trata de desplazar el punto $\overline{a}$ de derecha a izquierda, mediante el *Test del Punto Negativo*, para eliminar la máxima subregión de L y por lo tanto para acelerar la búsqueda.

2.2.1 Regla de División Adaptativa

Usando el algoritmo *MRF*, los valores de $F(X)$ y $F(\overline{x})$ se pueden conocer antes de dividir el intervalo X, que ha sido seleccionado y no rechazado. Teniendo en cuenta esta información, se va a proponer una *Regla de División* heurística basada en las siguientes ideas:

- El ancho de la *función de inclusión* ($w(F(X))$) está relacionado con el ancho del intervalo donde se calcula ($w(X)$). Esto es debido a que la función de inclusión obtenida con Aritmética de Intervalos es normalmente una función isótona (Definición 1.2.3) y dependiendo de si se utiliza la extensión natural de intervalos o extensión del valor medio de intervalos para obtener la *función de inclusión*, ésta será de orden uno o dos respectivamente (Teoremas 1.2.5 y 1.2.6, página 15). A veces es interesante que el tamaño de un subintervalo sea menor que $w(X)/2$ (sería el tamaño obtenido si se usa bisección), porque puede ocurrir que un intervalo que debería rechazarse porque $0 \notin f(X)$, no se rechace porque es suficientemente grande como para que $0 \in F(X)$. Por otro lado, el obtener subintervalos muy pequeños puede dar lugar a tener que realizar muchas divisiones, lo que haría que la convergencia del algoritmo fuese más lenta. En el método de división adaptativa se ha asumido que la posible localización de la raíz en el intervalo es aproximadamente proporcional a la asimetría de los límites de la *función de inclusión* respecto al cero (véase la Figura 2.2). Esta

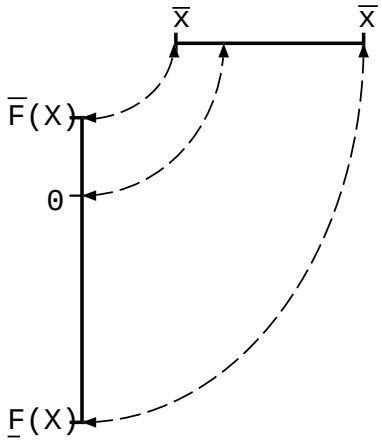


Figura 2.2: Heurística usada en la *Regla de División Adaptativa*.

idea es buena sobre todo cuando la subregión activa no tiene puntos estacionarios y contiene una sola raíz.

- Si $\overline{F}(X) << w(F(X))$, la mayoría de los puntos del intervalo X tienen un valor negativo de la función objetivo, por lo que la probabilidad de obtener una mejora con el *Test del Punto Negativo* es grande. Entonces, dividiendo el intervalo X por su parte izquierda, el subintervalo derecho, que tiene un tamaño mayor, posiblemente sería eliminado.
- Por otro lado, si $\overline{F}(X) >> w(F(X))$, la mayoría de los puntos del intervalo X tienen un valor positivo de la función objetivo, por lo que dividiendo por la parte izquierda o derecha del intervalo, uno (el mayor es preferible) o ambos subintervalos generados podrían ser eliminados por la *Regla de Eliminación Tradicional*. El intervalo será dividido por la parte izquierda o derecha, dependiendo del valor de $\overline{F}(\overline{x})$.

El método de división adaptativo propuesto es el siguiente:

$X_l = [\underline{x}, \underline{x} + \beta]$
y
$X_r = [\underline{x} + \beta, \overline{x}]$,
donde

$$\beta = \begin{cases} 0.33 \times w(X) & if\ 0.00 \leq \frac{\overline{F}(X)}{w(F(X))} \leq 0.33, \\ w(X) \times \frac{\overline{F}(X)}{w(F(X))} & if\ 0.33 < \frac{\overline{F}(X)}{w(F(X))} \leq 0.66, \\ 0.66 \times w(X) & if\ 0.66 < \frac{\overline{F}(X)}{w(F(X))} \leq 1.00 \end{cases}$$

Si $\overline{F}(X) - \overline{F}(\overline{x}) \leq \overline{F}(\overline{x}) - \underline{F}(X)$ y $0.66 < \frac{\overline{F}(X)}{w(F(X))} \leq 1.00$
entonces $\beta = 1 - \beta$

El algoritmo *MRF* con la nueva *Regla de División* adaptativa se denotará por *MRFda*. En el algoritmo *MRFda* las demás *Reglas básicas* son iguales que el Algoritmo 2.2.2. Un

Algoritmo 2.2.3 : Minimal Root Finder (reordered)

```
1  funct MRFro(S, F, ε, L, Q) ≡
2    Q := {∅}; L := {S}                          Inicialización de las listas final y de trabajo
3    if (0 ∈ F(s))
4      Q := Q + [s, s]
5      return Q                                  El algoritmo finaliza con la solución s
6    if (F(s) < 0)                               Precondición
7      F se cambia por − F
8    while (L ≠ {∅})
9          X := Cabeza(L)
10         L := L − {X}                          X ←pop L
11         Eliminado := True                     El intervalo actual podría ser eliminado
12         if (F(x) ≤ 0)                         Test del Punto Negativo
13           Eliminar todos los X_i ∈ L : x_i ≥ x   Análogo a L := {X}
14           Eliminado := False
15         elsif (0 ∉ F(X))
16               Eliminado := True               Regla de Eliminación Tradicional
17         if (Eliminado = True)
18           if (Q ≠ {∅})
19             L := {∅}
20       else                                    Eliminado=False
21           if (w(X) ≤ ε)                       Regla de Terminación
22             Q := Q + {X}                      X pertenece a la solución final
23         else
24           Subdividir(X, X_l, X_r)             X_l = [x, m(X)], X_r = [m(X), x]
25           L := L + {X_r} + {X_l}              L ←push X_r, L ←push X_l
26   return Q
```

ejemplo de la nueva *Regla de División* puede observarse en la gráfica inferior-izquierda de la Figura 2.3. Aunque en este ejemplo el algoritmo *MRFda* no es mejor que el algoritmo *MRF*, pruebas posteriores, donde se usará un gran número de funciones y una mayor precisión, probarán que en media, el uso de una *Regla de División* adaptativa permite obtener mejores resultados que la bisección.

2.2.2 Reordenación de las evaluaciones de la función de inclusión

El Algoritmo 2.2.2 sigue el orden tradicional de evaluaciones propuesto por Mitchell [119], sin el *Test de Monotonía*: primero se evalúa la *función de inclusión* en el intervalo seleccionado X ($F(X)$) y después se evalúa la *Función de Inclusión* en $\overline{x}_l$ ($F(\overline{x}_l)$). Sin embargo, para el problema de encontrar la mínima raíz, éste no es necesariamente el orden apro-

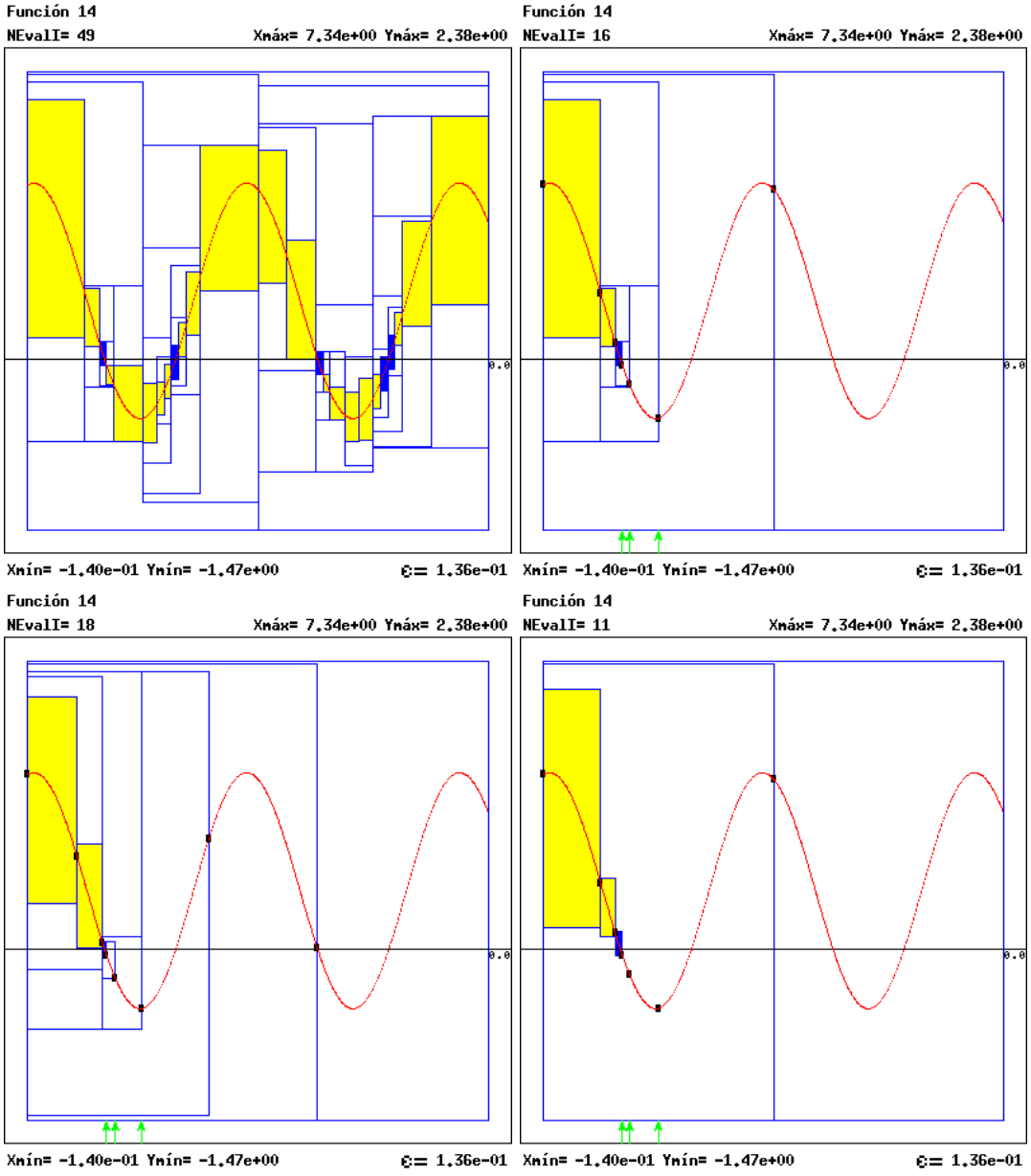


Figura 2.3: Resultados gráficos del los algoritmos *RF* (arriba-izquierda) y *MRF* (arriba-derecha), *MRFda* (debajo-izquierda) y *MRFro* (debajo-derecha), para el problema 14 (ver Tabla 2.1).

piado. En esta tesis se propone usar el orden opuesto al anterior, basándose en que si el punto $\overline{x}$ cumple el *Test del Punto Negativo*, entonces no es necesario evaluar $F(X)$ para

aplicar la *Regla de Terminación Tradicional*, porque es seguro que $0 \in F(X)$ (véase la demostración del Teorema 2.2.1) y el intervalo no será eliminado.

El algoritmo para encontrar la mínima raíz, desarrollado a partir del *MRF*, con una reordenación de las evaluaciones realizadas por el algoritmo de Mitchell, se describe en el Algoritmo 2.2.3 (*MRFro*). El algoritmo *MRFro* sólo evalúa $F(X)$ (línea 15), para aplicar la *Regla de Eliminación Tradicional*, cuando $\overline{x}$ no cumple el *Test del Punto Negativo* (línea 12).

Obsérvese que la división adaptativa no puede usarse en el algoritmo *MRFro* ya que la división adaptativa está basada en $F(X)$ y en el algoritmo *MRFro* existen intervalos X en los que $F(X)$ no ha sido evaluada. Por lo tanto la *Regla de División* utilizada en el algoritmo *MRFro* será la bisección tradicional. En la gráfica inferior-derecha de la Figura 2.3 se muestra un ejemplo de ejecución del algoritmo *MRFro*.

2.2.3 Resultados

En esta sección se demostrará que las distintas versiones del método propuesto son capaces de obtener soluciones muy precisas para el problema (2.1) con pocas evaluaciones de la *función de inclusión*.

La evaluaciones de los algoritmos *RF*, *MRF*, *MRFda* y *MRFro* se han realizado sobre un conjunto de cuarenta funciones de prueba, cuya descripción se muestra en la Tabla 2.1. Todas las funciones tienen varios máximos y mínimos, la mayoría tienen más de una raíz y doce de ellas tienen puntos de no diferenciabilidad en el espacio de búsqueda $S = [0.2, 7.0]$, que es el mismo para todas ellas. La Tabla 2.2 muestra los resultados obtenidos para un valor de la precisión requerida igual a $\epsilon = w(S) \cdot 10^{-15}$. En la última fila de la Tabla 2.2 se muestran los valores medios del número de evaluaciones de la *función de inclusión* para cada uno de los algoritmos. Estos resultados se han obtenido sobre un Pentium-II 266MHz con el Sistema Operativo Linux 2.0. Los programas fueron codificados en C y la Aritmética de Intervalos fue implementada con las rutinas BIAS [90]. Las notaciones usadas en las Tablas 2.1 y 2.2 son la siguientes:

N	Índice de la función (para futuras referencias).
MR	Mínima raíz de la función. Las funciones están ordenadas respecto a este valor, de menor a mayor y después las que no tienen raíces.
NR	Número de raíces en $S = [0.2, 7.0]$.
NM	Número de máximos y mínimos.
EF	Número de evaluaciones de la *función de inclusión*.
NQ	Número de intervalos en la lista final Q.
NS	Número de soluciones encontradas. Una solución está compuesta de uno o más intervalos consecutivos en Q.

Tabla 2.1: Descripción de las funciones de prueba. Las funciones marcadas con $\triangleright$ no son diferenciables en algunos puntos de $S = [0.2, 7.0]$.

	N	Función f(x)	MR	NR	NM
	1	$-e^{sin(3x)} + 2$	0.2553	7	9
	2	ln(3x)ln(2x)-0.1	0.2804	2	3
$\triangleright$	3	$\begin{cases} cos(5x) & x \leq 3/2\pi \\ cos(x) & \text{en otro caso} \end{cases}$	0.3142	8	10
	4	$-\sum_{k=1}^{5} ksin[(k+1)x+k] + 3$	0.4893	10	15
	5	$e^{-x}sin(2\pi x)$	0.5000	14	15
	6	$-\sum_{k=1}^{10} 1/(\lvert(k_i(x-a_i))^2\rvert + c_i) + 10$	0.6426	8	17
$\triangleright$		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)			
	7	$x + sin(5x)$	0.8209	2	13
	8	$-x + sin(3x) + 1$	1.0354	1	9
$\triangleright$	9	$\begin{cases} sin(3x) & x \leq 5 \\ 7x^2 - 14x - 100 & \text{en otro caso} \end{cases}$	1.0472	4	8
	10	$cos(x) + 2cos(2x)e^{-x}$	1.1407	2	4
	11	$-\sqrt{x}sin(x) + 1$	1.1748	3	4
	12	$(x^2 - 5x + 6)/(x^2 + 1) - 0.5$	1.2583	1	3
	13	$(3x - 1.4)sin(18x) + 1.7$	1.2655	34	42
	14	$sin(x)cos(x) - 1.5sin^2(x) + 1.2$	1.3408	4	7
	15	$(x+1)^3/x^2 - 7.1$	1.3646	2	3
$\triangleright$	16	$-\lvert xsin(x)\rvert + 1.5$	1.5034	5	6
$\triangleright$	17	$\begin{cases} x^2 + 0.5 & x \leq 1 \\ 5sin(2\pi x) + 1.5 & 1 < x \leq 3 \\ x^2 - 8x + 16.5 & \text{en otro caso} \end{cases}$	1.5485	4	8
	18	$cos(x) - sin(5x) + 1$	1.5708	6	13
$\triangleright$	19	$\begin{cases} cos(5x) & 3/2\pi \leq x \leq 5/2\pi \\ cos(x) & \text{en otro caso} \end{cases}$	1.5708	5	7
	20	$-x - sin(3x) + 1.6$	1.9686	1	9
	21	$xsin(x) + sin(10x/3) + ln(x) - 0.84x + 1.3$	2.9609	2	6
	22	$-0.5x^2ln(x) + 5$	3.0117	1	2
$\triangleright$	23	$x\lvert sin(x)\rvert - 0.02$	3.1352	4	6
	24	$2sin(x)e^{-x}$	3.1416	2	4
$\triangleright$	25	$\begin{cases} sin(x) & x \leq \pi \\ sin(5x) & \text{en otro caso} \end{cases}$	3.1416	7	9
	26	$\sqrt{x}sin^2(x)$	3.1416	2	6
$\triangleright$	27	$\begin{cases} sin(x) & sin(x) > cos(x) \\ cos(x) & \text{en otro caso} \end{cases}$	3.1416	2	6
	28	$2(x-3)^2 - e^{x/2} + 5$	3.2811	2	4
	29	$\begin{cases} sin(5x) + 2 & x \leq \pi \\ 5sin(x) + 2 & x > \pi \end{cases}$	3.5531	2	8
	30	$\sum_{k=0}^{5} kcos[(k+1)x+k] + 12$	4.7831	2	15
	31	$\sum_{k=1}^{5} -cos[(k+1)x] + 4$	6.1304	2	14
	32	$\lvert sin(x)^3cos(x)^3\rvert + 0.1$	-	-	10
$\triangleright$	33	$\begin{cases} -x + 4 & x \leq 3 \\ -8/9x^2 + 8x - 15 & 3 < x < 6 \\ x - 5 & \text{en otro caso} \end{cases}$	-	-	5
$\triangleright$	34	$\lvert sin(x) + cos(x)\rvert + 0.3$	-	-	6
$\triangleright$	35	$x + \lvert sin(x)cos(x)\rvert$	-	-	10
	36	$-e^{sin(x)} + 4$	-	-	4
	37	$e^{sin(3x)}$	-	-	9
	38	$2cos(x) + cos(2x) + 5$	-	-	6
	39	$-xsin(x) + 5$	-	-	4
	40	$-e^{-x}sin(2\pi x) + 1$	-	-	13

En la Tabla 2.2 no se han incluido resultados del método *GRID* ya que en el mejor de los casos (para la función número uno), el número de evaluaciones requeridas con $\epsilon = w(S) \cdot 10^{-15}$ es aproximadamente $8 \cdot 10^{12}$. En el peor de los casos, es decir, cuando no existe ningún corte con cero, el método de *GRID* necesita $w(S) \cdot 10^{15}$ evaluaciones de la función, ya que tiene que recorrer todo el espacio de búsqueda. Sin embargo, para las funciones sin ningún corte con cero (de la 32 a la 40), los distintos algoritmos aquí presentados resuelven el problema (2.1) en muy pocas iteraciones, gracias a la *Regla de Eliminación Tradicional*.

Comparando los algoritmos *RF* y *MRF*, se puede decir que sólo para las funciones 8, 20, 22 que sólo tienen una raíz y para las que no contienen ninguna raíz en la región de búsqueda, *RF* es mejor que *MRF*. Esto es debido a que *MRF* no obtiene beneficio de la evaluación de *Test del Punto Negativo*. Por otro lado, estos casos son los más fáciles de resolver. Para funciones con un número mayor de raíces, el *Test del Punto Negativo*, junto con la *Regla de Eliminación Tradicional*, hacen que el algoritmo trabaje más rápido, porque sólo se determina la mínima raíz.

Para el conjunto de funciones de prueba usados en esta tesis, el método *MRF* mejora en media al *MR* en más de un 70%.

Por otro lado, la nueva *Regla de División Adaptativa*, *MRFda*, permite acelerar la búsqueda con respecto al método *MRF*. La mejora en la reducción del número de evaluaciones de la *función de inclusión* está en el intervalo $[-7\%, 38\%]$, pero en media sólo se obtiene una mejora del 15%.

El algoritmo *MRFro*, que realiza una reordenación de las evaluaciones de la *función de inclusión*, es un 15% mejor que el *MRFda*, un 30% mejor que el *MRF* y un 80% mejor que el *RF*, respecto al número medio de evaluaciones, convirtiéndose así en el mejor de los algoritmos presentados. Además, no existe ninguna diferencia respecto al número de raíces encontradas por los algoritmos *MRF*, *MRFda* y *MRFro*, ni en la precisión de las soluciones (NQ) de los algoritmos *MRF* y *MRFro*. Entre el algoritmo *MRFda* y los algoritmos *MRF* y *MRFro*, las diferencias en la precisión de las soluciones (NQ) son inapreciables y se deben al uso de distintas *Reglas de División*.

Para comparar nuestra propuesta de algoritmo con el algoritmo de Mitchell, se ha incluido el *Test de Monotonía* en el algoritmo *MRFro*. En la Tabla 2.2 se muestran los resultados obtenidos por el algoritmo *MRFro* con el *Test de Monotonía* (*MRFro+*) y el algoritmo de Mitchell. Las ejecuciones se llevaron a cabo solamente en el subconjunto de funciones de prueba de la Tabla 2.1 sin puntos de no diferenciabilidad.

Al algoritmo de Mitchell se le añadió el criterio de terminación usado en el algoritmo *MRF*. En esta versión del algoritmo de Mitchell, sólo se evalúa $F(\underline{x}) \cdot F(\overline{x})$ si $0 \in F(X)$ y $0 \notin F'(X)$. Tal como se estableció en la Proposición 2.2.1, si $0 \in F(\overline{x})$ la subregión de búsqueda a la derecha de $\overline{x}$ se rechaza.

Puede observarse que para muchas funciones, el número de evaluaciones de la *función de inclusión* realizadas por el algoritmo *MRFro* (sin el *Test de Monotonía*) es menor que las realizadas por los algoritmos *MRFro+* y de *Mitchell*. Sin embargo, para la función número 15 el número de evaluaciones del algoritmo *MRFro* es casi cinco veces superior.

Tabla 2.2: Resultados numéricos para $\epsilon = w(S) \cdot 10^{-15}$.

	RF			*MRF*			*MRFda*			*MRFro*		
N	*EF*	*NS*	*NQ*	*EF*	*NS*	*NQ*	*EF*	*NS*	*NQ*	*EF*	*NS*	*NQ*
1	675	7	10	124	1	1	97	1	1	77	1	1
2	191	2	2	123	1	1	95	1	1	77	1	1
3	763	8	14	122	1	1	88	1	1	74	1	1
4	2207	10	39	209	1	1	196	1	2	161	1	1
5	1333	14	19	125	1	1	92	1	1	76	1	1
6	883	8	11	122	1	1	82	1	1	78	1	1
7	287	2	4	192	1	1	166	1	2	146	1	1
8	105	1	1	125	1	1	86	1	1	75	1	1
9	397	4	5	132	1	1	85	1	1	82	1	1
10	255	2	4	180	1	1	143	1	2	130	1	1
11	349	3	5	130	1	1	96	1	1	81	1	1
12	241	1	2	194	1	2	193	1	2	144	1	2
13	3137	34	54	135	1	1	119	1	1	91	1	1
14	519	4	8	128	1	1	111	1	1	79	1	1
15	1441	2	18	611	1	5	649	1	2	562	1	5
16	575	5	8	123	1	1	96	1	1	73	1	1
17	451	4	5	124	1	1	103	1	1	75	1	1
18	803	6	12	126	1	1	102	1	1	81	1	1
19	487	5	11	126	1	1	98	1	1	76	1	1
20	133	1	1	173	1	1	128	1	1	123	1	1
21	491	2	9	153	1	1	120	1	1	103	1	1
22	101	1	2	126	1	1	78	1	1	76	1	1
23	369	4	7	121	1	1	98	1	1	80	1	1
24	201	2	3	126	1	1	95	1	1	76	1	1
25	677	7	13	126	1	1	131	1	1	76	1	1
26	201	2	3	127	1	1	105	1	1	127	1	1
27	199	2	3	126	1	1	95	1	1	76	1	1
28	1051	2	13	205	1	2	202	1	2	155	1	2
29	199	2	2	127	1	1	100	1	1	77	1	1
30	265	2	5	199	1	2	169	1	1	152	1	2
31	209	2	4	140	1	1	119	1	1	92	1	1
32	1	0	0	2	0	0	2	0	0	2	0	0
33	39	0	0	59	0	0	65	0	0	59	0	0
34	1	0	0	2	0	0	2	0	0	2	0	0
35	1	0	0	2	0	0	2	0	0	2	0	0
36	1	0	0	2	0	0	2	0	0	2	0	0
37	1	0	0	2	0	0	2	0	0	2	0	0
38	1	0	0	2	0	0	2	0	0	2	0	0
39	3	0	0	5	0	0	5	0	0	5	0	0
40	1	0	0	2	0	0	2	0	0	2	0	0
M.	**481.1**			**124.5**			**105.5**			**88.2**		

	MRFro+			*Mitchell*		
N	*EF*	*NS*	*NQ*	*EF*	*NS*	*NQ*
1	80	1	1	126	1	1
2	83	1	1	125	1	1
3						
4	194	1	1	218	1	3
5	77	1	1	127	1	1
6						
7	100	1	1	131	1	1
8	75	1	1	127	1	1
9						
10	84	1	1	127	1	1
11	82	1	1	132	1	1
12	98	1	1	138	1	1
13	103	1	1	137	1	1
14	80	1	1	127	1	1
15	114	1	1	133	1	1
16						
17						
18	88	1	1	128	1	1
19						
20	112	1	1	163	1	1
21	104	1	1	146	1	1
22	76	1	1	128	1	1
23						
24	78	1	1	128	1	1
25						
26	179	1	1	127	1	1
27						
28	103	1	1	138	1	1
29	79	1	1	129	1	1
30	122	1	1	149	1	1
31	101	1	1	142	1	1
32						
33						
34						
35						
36	3	0	0	1	0	0
37	3	0	0	1	0	0
38	3	0	0	1	0	0
39	7	0	0	4	0	0
40	3	0	0	1	0	0
M.	**82.6**			**112.4**		

Comparando el algoritmo *MRFro+* y el de Mitchell puede observarse que en muchas ocasiones el algoritmo *MRFro+* realiza menos evaluaciones. Sólo para la función número 26 el algoritmo de Mitchell es claramente superior. Esto es debido a que aunque la función número 26 no tiene valores negativos, si tiene raíces en la región de búsqueda. El resto de funciones en las que el algoritmo de Mitchell es mejor, son las menos costosas ya que no contienen ninguna raíz. Por lo tanto, para las funciones sin raíces y para la función número 26, los intervalos sólo se eliminan por la *Regla de Eliminación Tradicional*. En

estos casos, el algoritmo *MRFro+* es más costoso porque debe realizar el *Test del Punto Negativo* antes de comprobar que $0 \notin F(X)$.

En el caso particular de la función número 4, el algoritmo de Mitchell obtiene tres intervalos en la lista final porque para todos ellos, $0 \in F(X)$ y $0 \in F'(X)$. Por lo tanto, $F(\underline{x}) \cdot F(\overline{x})$ no se evalúa. Sin embargo el algoritmo *MRFro+* obtiene un solo intervalo en la lista final ya que $0 \in F(\overline{x})$, siendo X el primer intervalo de los tres obtenidos por el algoritmo de Mitchell, produciendo así la finalización del algoritmo.

2.3 Algoritmo para encontrar la mínima raíz de un conjunto de funciones unidimensionales

Esta sección trata de analizar diferentes soluciones algorítmicas del problema (2.3). El modo más simple, pero más costoso, de resolver este problema consiste en calcular la mínima raíz para cada una de las funciones y de todas las soluciones encontradas quedarse con la menor. Sin embargo existen varios modos de mejorar esta solución. En esta sección se analiza una extensión del algoritmo *MRFro* que permite resolver el problema (2.3). La idea principal de nuestro algoritmo consiste en utilizar la información obtenida en alguna de las funciones para determinar la subregión de búsqueda en la que no se puede encontrar la solución, reduciendo así el número de intervalos activos para todas la funciones [19, 21]. Nuestra propuesta de algoritmo para resolver el problema (2.3) se describe en el Algoritmo 2.3.1, donde cada *función de inclusión* $F_i,\ i = 1, \ldots, p$, tiene asociada su propia lista de trabajo (L_i) y final (Q_i). El algoritmo se ejecuta hasta que todas la listas de trabajo se encuentren vacías.

Definición 2.3.1 *Se define el intervalo $A \subseteq S$ como la subregión de búsqueda activa en cualquier momento de la ejecución del Algoritmo 2.3.1, con $\underline{a} = \min\{\underline{x}_j,\ \forall X_j \in L_i,\ i = 1, \ldots, p\}$ y $\overline{a} = \max\{\overline{x}_j,\ \forall X_j \in L_i,\ i = 1, \ldots, p\}$*

En el Algoritmo 2.3.1, el intervalo A se inicializa a toda la región de búsqueda en la línea 3 y puede ser actualizado en cada iteración del algoritmo *MRFroEx* por los procedimientos $ActualizaA_{min}$ (líneas 16 y 22), que se describe en el Algoritmo 2.3.2, y $ActualizaA_{max}$ (línea 25), que se describe en el Algoritmo 2.3.3.

En cada iteración, el algoritmo trabajará sobre el intervalo X, seleccionado de entre el conjunto de intervalos X_i que se encuentran en la cabeza de alguna lista de trabajo y que verifiquen que $\underline{x}_i = \underline{a}$, asegurando así que la búsqueda se dirige hacia la mínima raíz. Si se tiene más de un intervalo X_i, verificando que $\underline{x}_i = \underline{a}$, la selección del intervalo a procesar se hace cíclicamente respecto de los índices de las listas que contienen dichos intervalos.

De forma análoga al algoritmo *MFRro*, si en el algoritmo *MRFroEx* existe alguna *función de inclusión*, F_i para la que $0 \in F_i(\underline{s})$, entonces el algoritmo acaba con la solución $[\underline{s}, \underline{s}]$, tal como se estableció en la Proposición 2.2.1 (línea 4).

Para poder aplicar el *Test del Punto Negativo* todas las *funciones de inclusión* se precondicionan de forma que $\underline{F}_i(\underline{s}) > 0$ (línea 8). De esta forma, si alguna función cumple

Algoritmo 2.3.1 : Algoritmo MRFro para un conjunto de p funciones unidimensionales

```
1  funct MRFroEx(S = [s, s̄], F_1, ... , F_p, ε, L_1, ... , L_p, Q_1, ... , Q_p)
2    Q_i := {∅}; L_i := {S}; i = 1, ... , p        Inicialización de las listas finales y de trabajo
3    A = [a, ā] := S                               Inicialización de la subregión activa de búsqueda
4    if (∃F_i, i = 1, ... , p : 0 ∈ F_i(s))
5      Q_i := Q_i + [s, s]
6      return Q = Q_1, ... , Q_p                   El algoritmo finaliza con la solución s
7    if (∃F_i, i = 1, ... , p : F̄(s) < 0)          Precondición
8      F_i se cambia por − F_i
9    while (∃L_j ≠ {∅}, j = 1, ... , p)
10         for L_i : L_i ≠ {∅} ∧ a = y : Y = Cabeza(L_i)
11           X := Cabeza(L_i)
12           L_i := L_i − {X}                                   X ←pop− L_i
13           Eliminado := False                  El intervalo actual no se ha eliminado
14           if (¬Evaluado(F_i(ā)) ∨ (Evaluado(F_i(ā)) ∧ F_i(ā) > 0))
15             if (0 ∉ F(X))                     Regla de Eliminación Tradicional
16               ActualizaA_min(a, L_1, ... , L_p, s̄);
17               Eliminado := True
18               if (Q_i) ≠ {∅}
19                 L_i := {∅}
20           if (Eliminado = False)
21             if (w(X) ≤ ε)                     Regla de Terminación
22               ActualizaA_min(a, L_1, ... , L_p, s̄);
23               Q_i := Q_i + {X}                X pertenece a la solución final
24           else
25               MejoraA_max := ActualizaA_max(ā, X)
26               Subdivide(X, X_l, X_r)          X_l = [x, m(X)], X_r = [m(X), x̄]
27               L_i := L_i + {X_r} + {X_l}      L_i ←push− X_r, L_i ←push− X_l
28               if (MejoraA_max)
29                 ActualizaListas(ā, L_1, ... , L_p)
30   return Q = Q_1, ... , Q_p
```

el *Test del Punto Negativo* ($\underline{F}(x) \leq 0$), entonces puede asegurarse que la mínima raíz se encuentra en algún punto $y : y \leq x$, pudiéndose actualizar la subregión de búsqueda activa, de forma que $\overline{a} = x$ (línea 3, Algoritmo 2.3.3). Así, la subregión a la derecha de x puede eliminarse de las listas de trabajo $L_i,\ i = 1, \ldots, p$. Esto se realiza eliminando los intervalos $X_j : \underline{x}_j \geq \overline{a}$ y reduciendo los intervalos $X_j : \overline{a} \in X_j$ a los intervalos $[\underline{x}_j, \overline{a}],\ \forall X_j \in L_i$ y $\forall L_i$ (Algoritmo 2.3.4).

Por otro lado, el valor de $\underline{a}$ puede cambiar sólo cuando la *Regla de Eliminación Tradicional* sea capaz de descartar todo el conjunto de intervalos X_i que verifiquen que $\underline{x}_i = \underline{a}$

Algoritmo 2.3.2 : Chequea si se ha actualizado $\underline{a}$

```
1  proc ActualizaA_min(a, L_1, ..., L_p, s) ≡
2     a := s
3     for i = 1...p
4        if (L_i ≠ {∅})
5           X := Cabeza(L_i)
6           if (x < a)
7              a := x
```

Algoritmo 2.3.3 : Intenta mejorar el valor de $\overline{a}$

```
1  funct ActualizaA_max(a, X) ≡
2     Mejora := False
3     if (F(m(X)) ≤ 0)                    Test del Punto Negativo
4        a := m(X)
5        Mejora := True
6     return Mejora
```

(línea 16), y/o dichos intervalos se almacenen en sus correspondientes listas finales (línea 22).

El Algoritmo *MRFroEx* realiza dos tipos de evaluaciones de las *funciones de inclusión*: $F_i(X)$ y $F_i(m(X))$. Cuando se selecciona un intervalo X de la lista de trabajo L_i, $F_i(X)$ sólo se evalúa si no se conoce el valor de $F_i(\overline{x})$ o si $F_i(\overline{x}) > 0$ (línea 14). Obsérvese que $\overline{x}$ fue el punto medio del intervalo padre de X. Si $F_i(\overline{x})$ fue evaluado y $F_i(\overline{x}) > 0$, $F_i(X)$ debe evaluarse para intentar aplicar la *Regla de Terminación Tradicional* (línea 15), porque $F(\underline{x} = \underline{a}) \cdot F(\overline{x}) > 0$ por el Teorema 2.2.1. Por otro lado, si no se conoce el valor de $F_i(\overline{x})$ es porque $\overline{x} = \overline{a}$. En este caso, el valor de $\overline{a}$ puede ser igual a $\overline{s}$, o cumplir que $\underline{F}_j(\overline{a}) \leq 0$, para algún $j \neq i$. Por lo tanto, el punto $\overline{a}$ ya cumplió el *Test del Punto Negativo* para otra *función de inclusión*, por lo que evaluar $F_i(\overline{x})$ no es de utilidad. Esto permite la reducción del número de evaluaciones.

Si el intervalo X no ha sido eliminado por la *Regla de Eliminación Tradicional* (línea 15), debe evaluarse $F_i(m(X))$ (línea 25), siempre que $w(X) > \epsilon$ (línea 21).

El algoritmo *MRFroEx* se ha diseñado como una extensión del algoritmo *MRFro* para trabajar sobre un conjunto de p funciones unidimensionales. De forma similar, se pueden extender los algoritmos *MRF* y *MRFda* para trabajar sobre p funciones unidimensionales (*MRFEx* y *MRFdaEx*, respectivamente).

Una posible mejora del Algoritmo 2.3.1 consiste en actualizar repetidamente el valor de $\overline{a}$ usando el mismo intervalo y la misma *función de inclusión*, hasta que el valor de $\overline{a}$ no cumpla el *Test del Punto Negativo*, en vez de mover las computaciones a la siguiente *función de inclusión disponible*. Incorporando esta idea al algoritmo *MRFroEx* se obtiene

Algoritmo 2.3.4 : Actualiza las listas de trabajo basándose en $\overline{a}$

```
1  proc ActualizaListas(ā, L_1, ..., L_p) ≡
2    for i = 1...p
3      if (L_i ≠ {∅})
4        Elimina los X_j ∈ L_i : x_j ≥ ā
5        if (L_i ≠ {∅})                    Existe un solo intervalo en la lista L_i
6          X := Cabeza(L_i)
7          X := [x, ā]
```

Algoritmo 2.3.5 : Intenta mejorar el valor de $\overline{a}$ y repite el proceso en [x,m(X)] si $\underline{F}(m(X)) \leq 0$

```
1  funct ActualizaA_max(ā, X) ≡
2    Mejora := False
3    while (w(X) > ε ∧ (F(m(X)) ≤ 0))
4          ā := m(X)
5          X := [x, ā]
6          Mejora := True
7    return Mejora
```

al nuevo algoritmo *MRFroEx-II*, mediante el cambio del Algoritmo 2.3.3 (Algoritmo 2.3.1, línea 25) por el Algoritmo 2.3.5.

2.3.1 Resultados

Las implementaciones de los algoritmos descritos anteriormente: *MRFEx*, *MRFdaEX*, *MRFroEX* y *MRFroEX-II*, han sido realizadas y probadas usando el conjunto de funciones de prueba mostradas en la Tabla 2.1. La Tabla 2.3 muestra el número de evaluaciones realizadas por los algoritmos *MRFEx*, *MRFdaEX*, *MRFroEX* y *MRFroEX-II* y la Figura 2.4 muestra una representación gráfica de los mismos resultados experimentales.

La Tabla 2.3 y la Figura 2.4 contienen resultados de tres series de experimentos. Todos los algoritmos se probaron para cuarenta valores diferentes de p (número de funciones en el conjunto, $p = 1, 2, 3, \ldots, 40$).

- En la primera serie, el conjunto de p funciones contiene las funciones $N = 1, \ldots, p$. Por ejemplo, los datos de la tercera columna ($N = 3$), implica que las funciones utilizadas fueron la 1, 2 y 3. Estos datos se muestran en la primera subtabla de la Tabla 2.3 y en la gráfica superior-izquierda de la Figura 2.4. Recuérdese que las funciones se encuentran ordenadas por su mínima raíz. Conforme se incrementa p, se puede observar un lento y casi lineal incremento en el número de evaluaciones de

Tabla 2.3: Resultados numéricos para los algoritmos MRFEx (columna A), MRFdaEx (columna B), MRFroEx (columna C) y MRFroEx-II (columna D). La precisión de los resultados es $\epsilon = w(S)\cdot 10^{-15}$, en todos los casos. La fila i contiene los resultados numéricos del conjunto de i funciones cuyo número se especifica en la columna N, desde la fila 1 a la i.

Ordenadas de 1 a 40				
N	**A**	**B**	**C**	**D**
1	124	97	77	77
2	137	106	92	88
3	147	105	101	97
4	149	111	104	106
5	155	117	109	104
6	161	123	115	108
7	167	129	121	112
8	157	130	112	107
9	161	134	116	111
10	165	138	120	115
11	167	144	122	117
12	171	148	126	121
13	175	152	130	125
14	177	156	132	127
15	181	162	136	131
16	183	164	138	133
17	185	166	140	135
18	187	168	142	137
19	189	170	144	139
20	193	174	148	143
21	197	180	152	147
22	199	182	154	149
23	201	184	156	151
24	203	186	158	153
25	205	188	160	155
26	207	190	162	157
27	209	192	164	159
28	211	194	166	161
29	213	196	168	163
30	217	200	172	167
31	219	204	174	169
32	221	206	176	171
33	223	208	178	173
34	225	210	180	175
35	227	212	182	177
36	229	214	184	179
37	231	216	186	181
38	233	218	188	183
39	235	220	190	185
40	237	222	192	187

Ordenadas de 40 a 1				
N	**A**	**B**	**C**	**D**
40	2	2	2	2
39	7	7	7	7
38	9	9	9	9
37	11	11	11	11
36	13	13	13	13
35	15	15	15	15
34	17	17	17	17
33	76	82	76	76
32	78	84	78	78
31	215	203	167	167
30	274	256	227	227
29	196	198	146	146
28	284	304	234	234
27	206	191	156	156
26	328	381	278	230
25	378	494	352	304
24	444	580	418	378
23	283	261	236	239
22	269	166	220	221
21	295	245	246	247
20	308	216	258	258
19	269	230	220	221
18	343	334	317	295
17	289	274	242	245
16	292	277	245	248
15	779	831	730	731
14	347	347	229	300
13	356	344	309	307
12	414	439	366	361
11	350	332	303	301
10	395	390	349	352
9	331	319	283	285
8	343	313	295	297
7	381	389	333	335
6	303	257	256	256
5	256	237	209	210
4	348	319	302	306
3	251	215	205	209
2	251	230	206	211
1	257	232	213	219

Ordenación aleatoria				
N	**A**	**B**	**C**	**D**
19	126	98	76	76
4	215	191	166	164
6	221	197	172	168
39	223	199	174	170
24	227	203	178	172
12	231	219	183	176
2	153	132	106	96
29	155	134	108	98
37	157	136	110	100
11	161	139	114	104
26	163	141	116	106
34	165	143	118	108
27	167	145	120	110
17	171	147	124	114
21	177	153	130	118
28	179	155	132	120
22	181	157	134	122
5	187	142	140	126
15	193	146	146	130
36	195	148	148	132
9	183	150	137	137
7	187	154	141	141
8	191	156	145	145
40	193	158	147	147
31	195	160	149	149
32	197	162	151	151
1	205	163	160	155
30	207	165	162	157
23	209	167	164	159
33	211	169	166	161
35	213	171	168	163
3	213	175	169	165
18	215	177	171	167
25	217	179	173	169
16	219	181	175	171
13	221	183	177	173
20	223	185	179	175
14	225	187	181	177
38	227	189	183	179
10	229	191	185	181

la *función de inclusión*. Esto es debido principalmente a las mejoras producidas en el valor de $\overline{a}$, que después de unas pocas iteraciones, es lo suficientemente pequeño para que la subregión de búsqueda activa no contenga ninguna raíz en la mayoría de las funciones de prueba. Estas funciones se volverán inactivas en pocas iteraciones

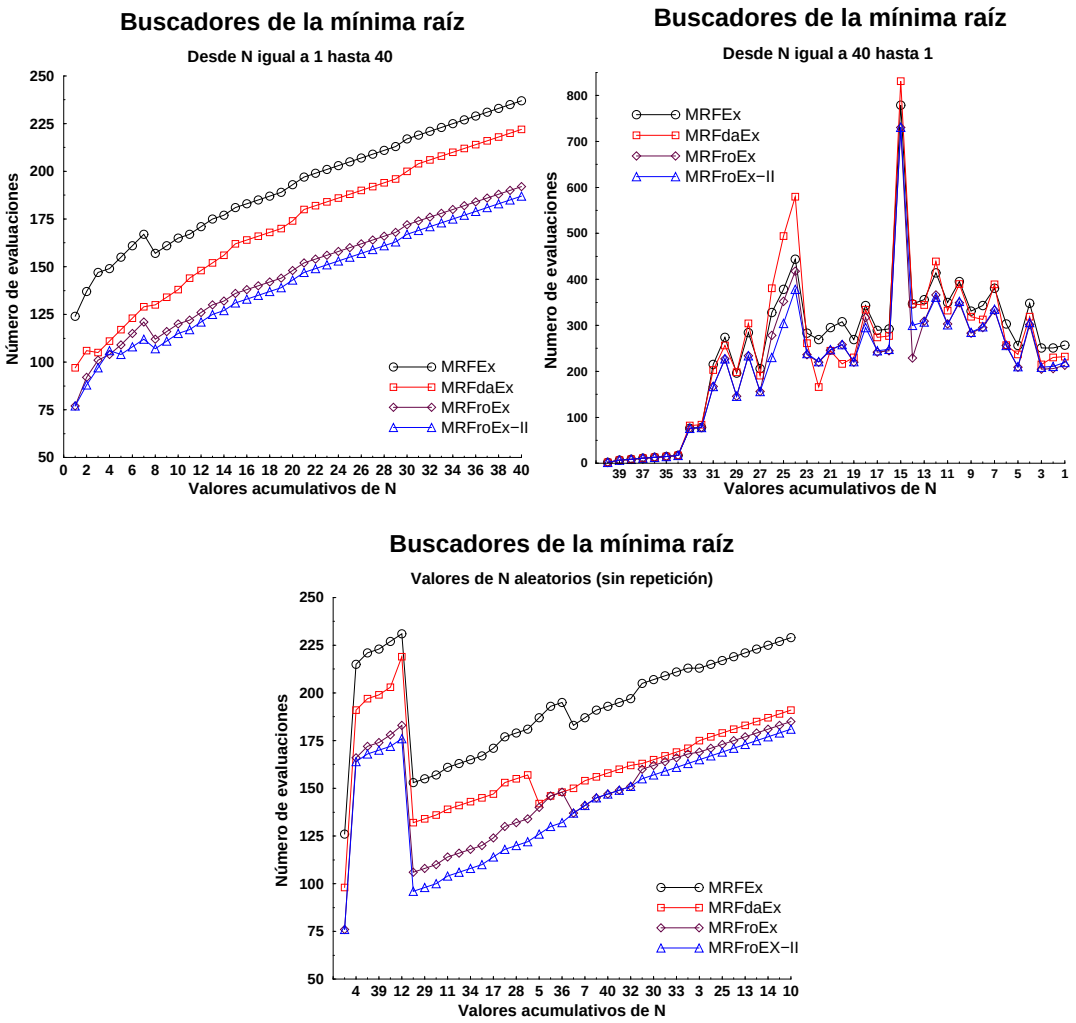


Figura 2.4: Representaciones gráficas de los resultados mostrados en la Tabla 2.3.

debido a la *Regla de Eliminación Tradicional*.

- En la segunda serie, el conjunto de funciones se ha construido en un orden decreciente de N, empezando siempre por el valor $N = 40$. Por ejemplo, el conjunto que contiene tres funciones (fila 3, $N = 38$), incluye las funciones $N = 40$, 39 y 38. Los datos experimentales para esta serie se muestran en la segunda subtabla de la Tabla 2.3 y en la gráfica superior-derecha de la Figura 2.4. Ahora la última función de cualquier conjunto de p funciones contiene la mínima raíz. Por lo tanto, al final de los distintos algoritmos, sólo esta función permanece activa. Como en el caso anterior, el número de evaluaciones realizadas sobre la función con la mínima raíz tendrá un peso

importante en el resultado final. En los datos pueden identificarse dos casos particularmente significativos. Cuando la última función es la número 15 y cuando la última función es la número 24. En el primer caso, el numero de evaluaciones se incrementa considerablemente debido a que las sobreestimaciones producidas por la Aritmética de Intervalos no permiten eliminar intervalos hasta que estos sean suficientemente pequeños. Este número de evaluaciones puede reducirse considerablemente si se usa el *Test de Monotonía*, tal como se mostró en la Tabla 2.2. En el segundo caso, las funciones número 27, 26, 25 y 24 tienen el mismo valor de su mínima raíz en la región de búsqueda, por lo tanto, los algoritmos tienen que continuar trabajando en todas ellas, con el consiguiente incremento en el número de evaluaciones.

- En la tercera serie de experimentos, el conjunto de funciones de prueba se ordenó aleatoriamente, construyéndose cada conjunto de p funciones con las p primeras del orden aleatorio. Los datos experimentales se muestran en la tercera subtabla de la Tabla 2.3 y en la gráfica inferior de la Figura 2.4. Estos experimentos se han introducido debido a que normalmente la ordenación de las funciones, respecto a la localización de la mínima raíz, no se conoce de antemano. Como en los casos anteriores, el orden de las funciones y la localización de la función con la mínima raíz en el conjunto son importantes para la mejora del valor de $\overline{a}$ y por lo tanto, para la reducción del número de evaluaciones.

Comparando las cuatro versiones del algoritmo, puede observarse que para el conjunto de funciones de prueba las versiones *MRFroEx* y *MRFroEx-II* son mejores que la *MRFEx*. El algoritmo *MRFdaEx* es generalmente mejor que el *MRFEx* y peor que los algoritmos *MRFroEx* y *MRFroEx-II*. Entre los algoritmos *MRFroEx* y *MRFroEx-II* sólo se pueden notar pequeñas diferencias. Para la mayoría de las funciones, el algoritmo *MRFroEx-II* es ligeramente mejor que el *MRFroEx* y cuando el algoritmo *MRFroEx* mejora los valores del *MRFroEx-II*, las diferencias son menores de tres o cuatro evaluaciones de la *función de inclusión*.

En la Figura 2.5 se muestra gráficamente un ejemplo donde se comparan los métodos *MRFroEx* (columna izquierda) y *MRFroEx-II* (columna derecha), para las funciones 8, 27 y 16 de la Tabla 2.1 y un valor de $\epsilon = 0.34$. El método *MRFroEx* necesita 19 evaluaciones en total, mientras el método *MRFroEx-II* necesita sólo 15, debido a que realiza varias mejoras consecutivas del valor de $\overline{a}$ en la función número 8.

2.4 Resumen

En este capítulo se han propuesto nuevos algoritmos de Ramificación y Acotación basados en Aritmética de Intervalos para resolver dos problemas que se pueden aplicar a diversas áreas de la Ciencia y la Ingeniería. Para este tipo de algoritmos se puede hacer la siguiente reflexión:

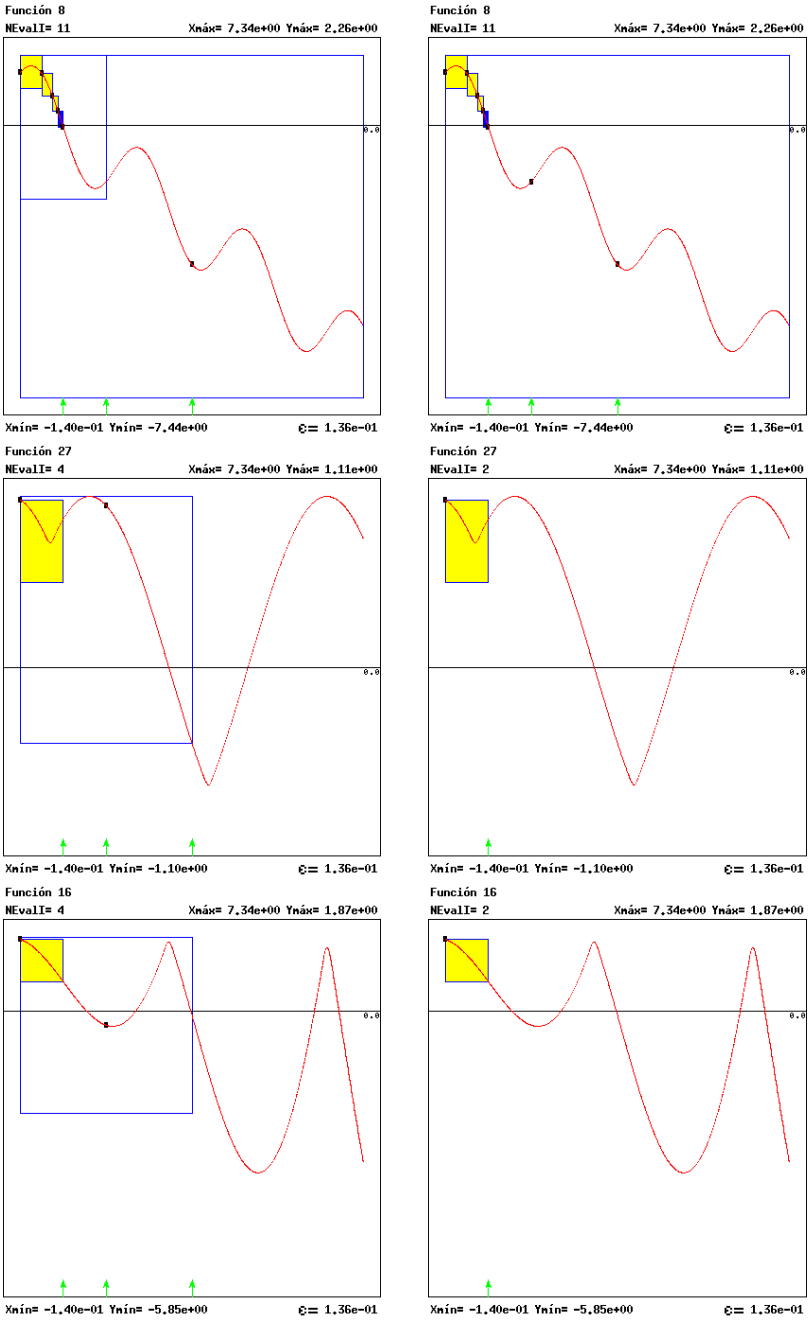


Figura 2.5: Algoritmos *MRFroEx* (columna izquierda) y *MRFroEx-II* (columna derecha), para las funciones 8, 27 y 16 de la Tabla 2.1 y un valor de $\epsilon = 0.34$.

La evaluación de la *función de inclusión* obtenida a partir de la Aritmética de Intervalos normalmente produce sobreestimaciones del rango real de una función debido a la definición de las operaciones de la Aritmética de Intervalos y también por el *redondeo directo* usado en la Aritmética Computacional. Normalmente esta sobreestimación decrece con la anchura del intervalo evaluado. Así, la información más precisa que se puede obtener es para los intervalos puntuales $[x, x]$. Por lo tanto, cuando $w(X) > 0$, es mejor tomar decisiones basadas en la evaluación de $F([x, x])$ que en $F(X)$. Los algoritmos tradicionales para encontrar la raíz mínima de una función unidimensional hacen uso de las evaluaciones de $F(X)$, $F'(X)$ y $F([x, x])$, en este orden. En esta tesis se ha probado que es mejor utilizar el orden inverso, principalmente cuando la función objetivo tiene al menos una raíz en la región de búsqueda (los otros casos suelen ser los menos costosos).

Para el problema de encontrar la mínima raíz en una función unidimensional esta reordenación proporcionó una mejora en el número de evaluaciones de un 30%, en promedio, para el conjunto de funciones de prueba. Incorporando el *Test de Monotonía*, las mejoras fueron del 26%. Para muchos casos, esta reordenación aportó mejores resultados, incluso sin usar el *Test de Monotonía*.

Se han propuesto algoritmos eficientes para resolver el problema de encontrar la mínima raíz de un conjunto de funciones unidimensionales, los cuales utilizan información obtenida en una función para reducir el espacio de búsqueda en las demás funciones. Incluso para estos algoritmos tan eficientes, una reordenación de los tipos de evaluaciones a realizar aportó en promedio un 20% de mejora.

Capítulo 3

Algoritmos secuenciales de Optimización Global basados en Aritmética de Intervalos y Ramificación y Acotación

En este capítulo se describirá un algoritmo básico de Optimización Global basado en el método de Ramificación y Acotación y Aritmética de Intervalos, de acuerdo a las reglas básicas de los algoritmos de Ramificación y Acotación descritas en la Sección 1.5.1 (página 26). En este trabajo se propone un nuevo parámetro (Sección 3.2) que estima la proximidad de un intervalo n-dimensional a la región de atracción de un mínimo, o la cantidad de región de atracción hacia un mínimo que contiene un determinado intervalo n-dimensional (también llamado caja). Este parámetro se ha utilizado para obtener información acerca del nivel de división a realizar sobre la caja seleccionada (Sección 3.3), obtener mejores criterios de selección (secciones 3.5 y 3.6) y para añadir nuevos criterios de eliminación (secciones 3.4 y 3.6).

3.1 Algoritmo básico de Optimización Global basado en Intervalos

Se quiere resolver el problema de Optimización Global, donde la única restricción es la de tener límites implícitos en la región de búsqueda. Es decir:

$$f^* = \min_{x \in S} f(x), \tag{3.1}$$

donde el intervalo n-dimensional $S \subseteq \mathbb{R}^n$ es la región de búsqueda y $f(x) : \mathbb{R}^n \to \mathbb{R}$ es la *función objetivo*. No sólo se está interesado en encontrar f^*, sino también el conjunto de puntos minimizadores globales, i.e, $X^* = \{x^* \in S : f(x^*) = f^*\}$.

Las distintas reglas de Ramificación y Acotación utilizadas para los algoritmos de Optimización Global basados en intervalos se detallan a continuación:

Regla de Acotación: La Aritmética de Intervalos provee un método natural y riguroso para obtener límites de una función en un intervalo. Puede usarse la extensión natural de intervalos, la Aritmética de Intervalos Extendida (Sección 1.2.3), *expresiones centradas* (Definición 1.2.5) o expresiones basadas en la pendiente (Definición 1.2.6). En este trabajo, la función de inclusión usada es la extensión natural a intervalos [124, 152, 90] (Sección 1.2.2, página 8).

Regla de División: Csendes y Ratz [35, 155] estudiaron cuatro estrategias para determinar en qué coordenada de la caja seleccionada se debía realizar una bisección. En estos trabajos, la coordenada, k, a dividir está determinada por la siguiente ecuación:

$$k = \min\left\{ j : j \in \{1, \ldots, n\} \text{ y } D(j) = \max_{i=1..n}\{D(i)\} \right\} \tag{3.2}$$

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 [125, 153, 167]:

$$D(i) = w(X_i) \tag{3.3}$$

Estrategia B: Hansen [71] propuso una estrategia heurística de forma que la coordenada elegida fuera aquella en la que f varía mas:

$$D(i) = w(F'(X_i)) \cdot w(X_i) \tag{3.4}$$

Estrategia C: En esta estrategia, Ratz [154] intenta disminuir la anchura de la *función de inclusión*:

$$D(i) = w(F'(X_i) \cdot (X_i - m(X_i))) \tag{3.5}$$

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 [68]:

$$D(i) = \begin{cases} w(X_i) & 0 \in X_i \\ w(X_i)/\min\{|\ x_i\ | : x_i \in X_i\} & \text{en otro caso} \end{cases} \tag{3.6}$$

Estrategia E: Esta estrategia es análoga a la **C**, pero usando información de la segunda derivada [155]:

$$D(i) = w\left((X_i - m(X_i)) \cdot \left(F'(m(X)) + \frac{1}{2}\sum_{j=1}^{n} (H_{ij}(X) \cdot (X_i - m(X_i)) \right) \right) \tag{3.7}$$

Para las funciones de prueba utilizadas, todas las estrategias fueron las mejores en algunos casos, pero las estrategias **B**, **C** y **E** se mostraron superiores, en media, a las estrategias **A** y **D**. Las mejoras se incrementaron con la dificultad de los problemas tratados. La estrategia **D** fue la peor y la **C** la mejor, seguida de cerca por la **B** y **E**. Los autores establecen que debido a su elevado coste, la estrategia **E** sólo está justificada si además se calculan las segundas derivadas para otros propósitos.

Basándose en las propuestas anteriores, Berner [7] establece que es mejor realizar una bisección de dos coordenadas a la vez. Para los problemas de prueba que utilizó, Berner obtuvo una mejora de un 20% respecto a la bisección de una sola coordenada. Hansen [71], propone 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 esta tesis la selección de la coordenada o coordenadas a dividir estará basada en $w(X_i)$, ya que es independiente de la diferenciabilidad de la función objetivo. Sobre cada coordenada se realizará una bisección u otros niveles de división, proponiéndose una *Regla de División* adaptativa.

Regla de Selección: En [62] y [7] se demuestra que, en la mayoría de los casos, la regla de selección *Primero el Mejor* (Sección 1.5.1, página 28), es más eficiente que la búsqueda en profundidad o en anchura, siendo la *Regla de Selección* que normalmente se usa en los algoritmos de Optimización Global basados en intervalos. Este criterio de selección se obtiene mediante una ordenación no decreciente respecto a $\underline{F}(X^i)$ de las cajas activas X^i, como primer criterio de ordenación y un orden decreciente con respecto a la edad de las cajas, para aquellas cajas con $\underline{F}(X^i) = \underline{F}(X^j),\ i \neq j$, como segundo criterio de ordenación [35]. Este tipo de ordenación es debido a que se supone que las cajas con un menor valor de $\underline{F}(X^i)$ tienen más probabilidad de contener un mínimo global. Denotaremos al menor de los valores obtenidos de $\underline{F}(X^i)$ como $\underline{f^*}$, ya que será un límite inferior del valor mínimo de la función objetivo en la región de búsqueda (f^*).

La estructura de datos más simple para almacenar las cajas activas es una lista enlazada ordenada según $\underline{F}(X^i)$. En vez de una lista enlazada, aquí se ha utilizado un árbol binario balanceado, ya que se mejora la complejidad de una inserción ordenada. En este árbol de trabajo, cada nodo usa una lista enlazada para almacenar las cajas con el mismo valor de $\underline{F}(X)$.

Regla de Terminación: Generalmente la *Regla de Terminación* está basada en los valores de $w(X)$ y/o $w(F(X))$. Una descripción más detallada de estos criterios de terminación puede encontrarse en [153]. En esta tesis se usará $w(X) < \epsilon$ como criterio para establecer que la caja X pertenece a la solución final y por lo tanto será almacenada en un árbol final, en vez de en el árbol de trabajo. De esta forma, el algoritmo termina cuando no quedan cajas en el árbol de trabajo.

Regla de Eliminación: La *Regla de Eliminación* básica y más simple es la denominada *Test de Corte* (*CutOffTest*). Denotaremos por $\overline{f^*}$ al límite superior del mínimo global f^*, obtenido mediante el *Test del Punto Medio* (Sección 1.3.1, página 18). El *Test de Corte* elimina las cajas activas, X^i, que cumplan la relación $\overline{f^*} < \underline{F}(X^i)$. Se puede usar algún método de búsqueda local para mejorar el valor de $\overline{f^*}$ y así poder aplicar el *Test de Corte* al mayor número posible de cajas.

Otros dispositivos aceleradores, como el test de monotonía, concavidad o el método de Newton de Intervalos (Sección 1.3), han demostrado ser unas herramientas eficientes para establecer criterios de eliminación. Utilizando este tipo de dispositivos el número de cajas evaluadas decrece considerablemente pero el esfuerzo computacional aplicado a cada caja se incrementa conforme se incrementa la dimensión de la función objetivo. La utilidad de estos tests decrece para los casos donde la función objetivo tiene un gran número de máximos y mínimos y cuando la sobreestimación del rango real de la función objetivo, que aparece en la *función de inclusión*, hace que estos dispositivos sean ineficientes, frenando la convergencia de los algoritmos que los usan. Normalmente, los dispositivos más costosos, como el Método de Newton de Intervalos, sólo se aplican en las fases finales de los algoritmos, donde se supone que las cajas a procesar sólo contienen un mínimo global y son lo suficientemente pequeñas para que las sobreestimaciones de la *función de inclusión* no sean importantes. El principal inconveniente de estos dispositivos aceleradores es que no pueden aplicarse si la función objetivo es no diferenciable.

El algoritmo básico de Optimización Global basado en intervalos se muestra en el Algoritmo 3.1.1. Su principal característica es que no tiene ningún requerimiento acerca de la diferenciabilidad de la función objetivo. Los parámetros de entrada del Algoritmo 3.1.1 son los siguientes:

S	: Región de búsqueda.
F	: Extensión natural a intervalos de la función objetivo f.
ϵ	: Criterio de parada ($w(X) < \epsilon$).
T	: Árbol de trabajo.
Q	: Árbol final.
$+/-$	: Denotan la introducción/extracción de cajas del árbol.

El algoritmo comienza inicializando: el límite superior de f^* a $\overline{F}(S)$, el árbol de trabajo a la región de búsqueda y el árbol final al conjunto vacío (líneas 2, 3 y 4, respectivamente). El algoritmo termina cuando el árbol de trabajo se encuentra vacío (línea 5). La *Regla de Selección*, basada en procesar primero la caja con un menor valor de $\underline{F}(X)$, se aplica al árbol de trabajo en la línea 6. A continuación, se evalúa el punto medio de la caja seleccionada para intentar mejorar el valor de $\overline{f^*}$ (línea 8). Si se ha mejorado el valor de $\overline{f^*}$, se aplica el *Test de Corte* a los árboles de trabajo y final (líneas 10 y 11). La *Regla de División* se aplica a la caja seleccionada en la línea 12. Finalmente, las subcajas generadas

Algoritmo 3.1.1 : Algoritmo básico de Optimización Global basado en intervalos.

```
1  funct OGBásico(S, F, ε, T, Q)
2    f̄* = F̄(S)                                   Límite superior de f*
3    T := {S}                                     Árbol de trabajo
4    Q := {∅}                                     Árbol Final
5    while (T ≠ {∅})
6          X := SelecionaCaja(T)                  F̲(X) = min{F̲(X^i), ∀X^i ∈ T}
7          T := T − {X};
8          f̄* := min{f̄*, F̄(m(X))}                 Actualización de f̄*
9          if (f̄* = F̄(m(X)))                      Se ha mejorado el valor de f̄*
10           T := TestdeCorte(T, f̄*)              ∀X^i ∈ T, f̄* < F̲(X^i) ⇒ T := T − {X^i}
11           Q := TestdeCorte(Q, f̄*)
12         Divide(X, X^1, ..., X^s)
13         for i := 1 to s
14           if (f̄* < F̲(X^i) next_i               Test del Punto Medio
15           if (w(X^i) < ε) Q := Q + {X^i}       Almacenar X^i en Q
16         else T := T + {X^i}                    Almacenar X^i en T
17   return Q
```

se almacenan en el árbol de trabajo (línea 16) si no cumplen el *Test del Punto Medio* (línea 14) ni la *Regla de Terminación* (línea 15). Normalmente cuando se habla del *Test del Punto Medio*, no solo se hace referencia a la posible eliminación de una caja (línea 14), sino también a la evaluación del punto medio de la caja seleccionada (línea 8).

El algoritmo 3.1.1 es el resultado de diversas mejoras realizadas al algoritmo inicial de Moore [126] y Skelboe [167]. En el algoritmo Moore-Skelboe, el valor de $\overline{f^*}$ se actualizaba al menor valor de $\overline{F}(X)$ encontrado. Posteriormente Ichida y Fuji [84] incorporaron el *Test del Punto Medio*, mejorando así la actualización de $\overline{f^*}$. Hansen [69, 70] propuso usar una *Regla de Selección* basada en una búsqueda en anchura (Sección 1.5.1, página 28). Además del *Test del Punto Medio*, Hansen incorporó el *Test de Corte* cuando se obtenía un mejor valor de $\overline{f^*}$. A este tipo de algoritmo se le conoce como algoritmo de Hansen. En los códigos usados actualmente por Hansen [71] se usa una *Regla de Selección* basada en *Primero el Mejor*. En [153], se han analizado propiedades de convergencia de los algoritmos de Moore-Skelboe, Ichida-Fuji y Hansen, presentando además algunos experimentos con el algoritmo de Hansen. Una revisión más general de las distintas versiones de algoritmos de Optimización Global basados en intervalos puede encontrarse en [88].

En la Figura 3.1 se muestra la ejecución del Algoritmo 3.1.1 para la función $f(x) = x + sin(5x)$, sobre el espacio de búsqueda $S = [0.2, 2.0]$ y un valor de $\epsilon = 0.2$. Las gráficas siguen un orden de izquierda a derecha y de arriba a abajo conforme a la ejecución del Algoritmo 3.1.1 avanza. En las gráficas, una caja representa un intervalo $[\underline{x}, \overline{x}]$ y los límites inferior y superior de la *función de inclusión* evaluada sobre ese intervalo, $[\underline{F}(X), \overline{F}(X)]$.

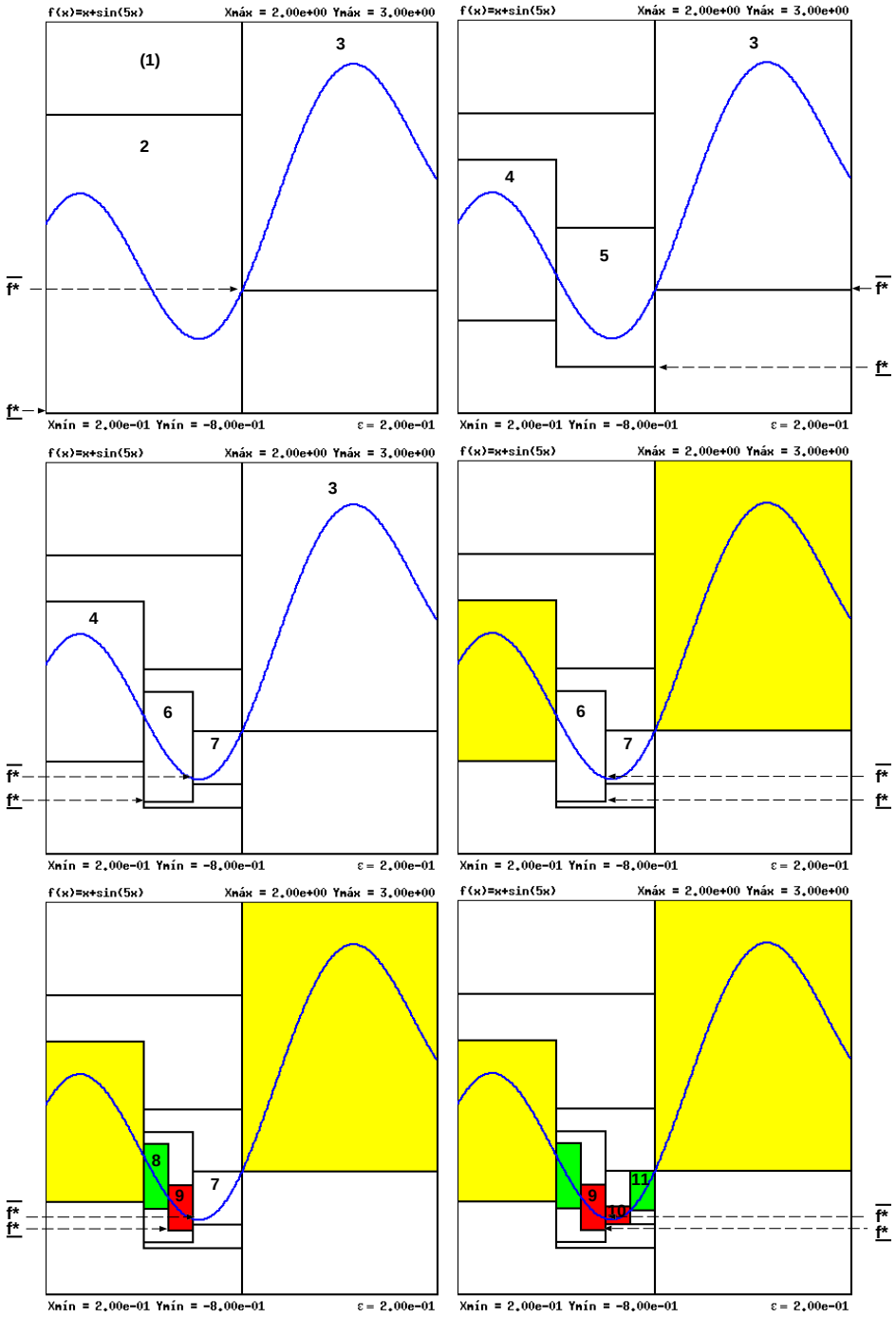


Figura 3.1: Ejemplo de ejecución del Algoritmo 3.1.1 sobre una función unidimensional.

Cada una de las cajas se ha enumerado siguiendo su orden de creación. Se ha dibujado también el intervalo $[\underline{f^*}, \overline{f^*}]$ que encierra el valor mínimo de la función en el intervalo S ($f^* \in [\underline{f^*}, \overline{f^*}]$). Las distintas iteraciones del Algoritmo 3.1.1 se detallan a continuación:

1. En la primera gráfica de la Figura 3.1 se muestra cómo se actualiza el valor de $\overline{f^*}$=$\overline{F}(m(X^1))$ (línea 8). El límite inferior de f^* ($\underline{f^*}$) se inicializa a $\underline{F}(X^1)$. La *Regla de División* utilizada es la bisección. El algoritmo divide el intervalo X^1 (línea 12) y almacena los subintervalos generados, X^2 y X^3, en el árbol de trabajo (línea 16).

2. Siguiendo la *Regla de Selección* (línea 6), se extrae el intervalo X^2 del árbol de trabajo para su procesamiento. En este caso la evaluación de $F(m(X^2))$ no mejora el valor de $\overline{f^*}$. Se divide el intervalo X^2, generando los intervalos X^4 y X^5 que se almacenan en el árbol de trabajo. El valor $\underline{f^*}$ se mejora como resultado de la evaluación de $F(X^5)$.

3. En la tercera gráfica, se selecciona el intervalo X^5, actualizándose el valor de $\overline{f^*}$ al evaluar $F(m(X^5))$. Esta mejora de $\overline{f^*}$ permite realizar el *Test de Corte* (línea 10), que elimina los intervalos X^4 y X^3 del árbol de trabajo. Esta eliminación se muestra en la cuarta gráfica mediante el cambio de color de las cajas eliminadas. Las cajas X^6 y X^7, generadas a partir de X^5, son almacenadas en el árbol de trabajo, actualizándose de nuevo el valor de $\underline{f^*} = \underline{F}(X^6)$.

4. En la quinta gráfica se selecciona X^6. En esta ocasión, no se mejora el valor de $\overline{f^*}$. Mediante la regla de división se generan las cajas X^8 y X^9. La caja X^8 se elimina directamente mediante el *Test del Punto Medio* (línea 14), mientras que la caja X^9 cumple la *Regla de Terminación*, por lo que se almacena en el árbol final (línea 15), mejorándose además el valor de $\underline{f^*}$.

5. El mismo proceso se realiza con la caja X^7, que es la única del árbol de trabajo, eliminándose la subcaja generada X^{11} por el *Test del Punto Medio* y almacenándose la subcaja X^{10} en al árbol final. Al encontrarse el árbol de trabajo vacío, el algoritmo termina devolviendo las cajas que se encuentran en Q.

De esta forma se obtiene una solución rigurosa, determinándose la situación de los puntos minimizadores por la unión de los intervalos que se encuentran en la árbol final ($X^* = X^9 \cup X^{10}$) y el rango donde se encuentra el valor mínimo de la función objetivo en la región de búsqueda ($f^* \in [\underline{f^*}, \overline{f^*}]$).

El algoritmo 3.1.1 será la base de trabajo para la mejora de algunas de las reglas de Ramificación y Acotación en las que está basado. La mayoría de estas mejoras están basadas en el parámetro $p\overline{f^*}(X)$ que se describe en la siguiente sección.

3.2 El parámetro $p\overline{f^*}(X)$

El parámetro $p\overline{f^*}(X)$ aparece ante la necesidad de obtener un estimador de la cantidad de región de atracción hacia un mínimo que contiene una determinada caja o la distancia de un mínimo a la región definida por una caja concreta. Esta información es muy importante porque la mayoría del trabajo computacional se hace alrededor de los puntos minimizadores de la función objetivo. En las siguientes secciones se usará este parámetro para reducir el número de evaluaciones de la *función de inclusión* realizadas por el Algoritmo 3.1.1, reduciendo así su tiempo de ejecución.

Definición 3.2.1 *Dada una función $f(x) : S \subset \mathbb{R}^n \to \mathbb{R}$ y $\overline{f^*}$ un límite superior de f^*, se define el parámetro $p\overline{f^*}$ de una caja $X \subseteq S$ como:*

$$p\overline{f^*}(X) = \frac{\overline{f^*} - \underline{F}(X)}{w(F(X))}, \tag{3.8}$$

donde $\overline{f^*}$ puede obtenerse mediante el uso de algoritmos de búsqueda local o durante la ejecución del Algoritmo 3.1.1, mediante el Test del Punto Medio. Como muestra la ecuación (3.8), el parámetro $p\overline{f^*}$ determina la distancia de $\overline{f^*}$ a $\underline{F}(X)$, normalizada por $w(F(X))$.

En la expresión (3.8), el denominador es positivo debido a las propiedades de las *funciones de inclusión* y a los redondeos directos usados por la Aritmética Computacional. Por otro lado, el valor $\overline{f^*} - \underline{F}(X)$ es siempre positivo o igual a cero, ya que de otra forma la caja X habría sido previamente eliminada por el *Test de Corte* o el *Test del Punto Medio*. Para algunos problemas poco comunes, pero importantes, el límite inferior de la *función de inclusión*, para la mayoría de la cajas relevantes, es igual al mínimo global de la función objetivo, inhibiendo la utilidad del parámetro $p\overline{f^*}$. Tales problemas aparecen por ejemplo cuando hay que determinar las raíces de una función $g(x)$ mediante la minimización de $f(x) = g(x)^2$, como en algunas fases de problemas de equilibrio [171].

En esta sección se mostrará la relación entre el valor de $p\overline{f^*}(X)$ y la proximidad de X a un punto $x^* \in X^*$. Nuestros experimentos muestran que para valores grandes de $p\overline{f^*}(X)$, la caja X está normalmente cerca de un mínimo local o global.

Como ejemplo preliminar se ha usado una función $f(x) : S \subset \mathbb{R}^n \to \mathbb{R}$ cuyos mínimos globales $(f(x_{g_1}), f(x_{g_2}), \dots)$ y locales $(f(x_{l_1}), f(x_{l_2}), \dots)$, se conocen de antemano. Los valores de $p\overline{f^*}(X)$ se han computado para todas la cajas disjuntas $X \subset S$ de tamaño $\delta = w(S)/d$, usando $\overline{f^*}{=}f^*$. También se ha calculado la distancia entre el centro de cada caja X y el mínimo global más cercano. Estas computaciones se han realizado para varios valores del tamaño de caja ($d = 16$, 32, 64, 128, 256 y 512).

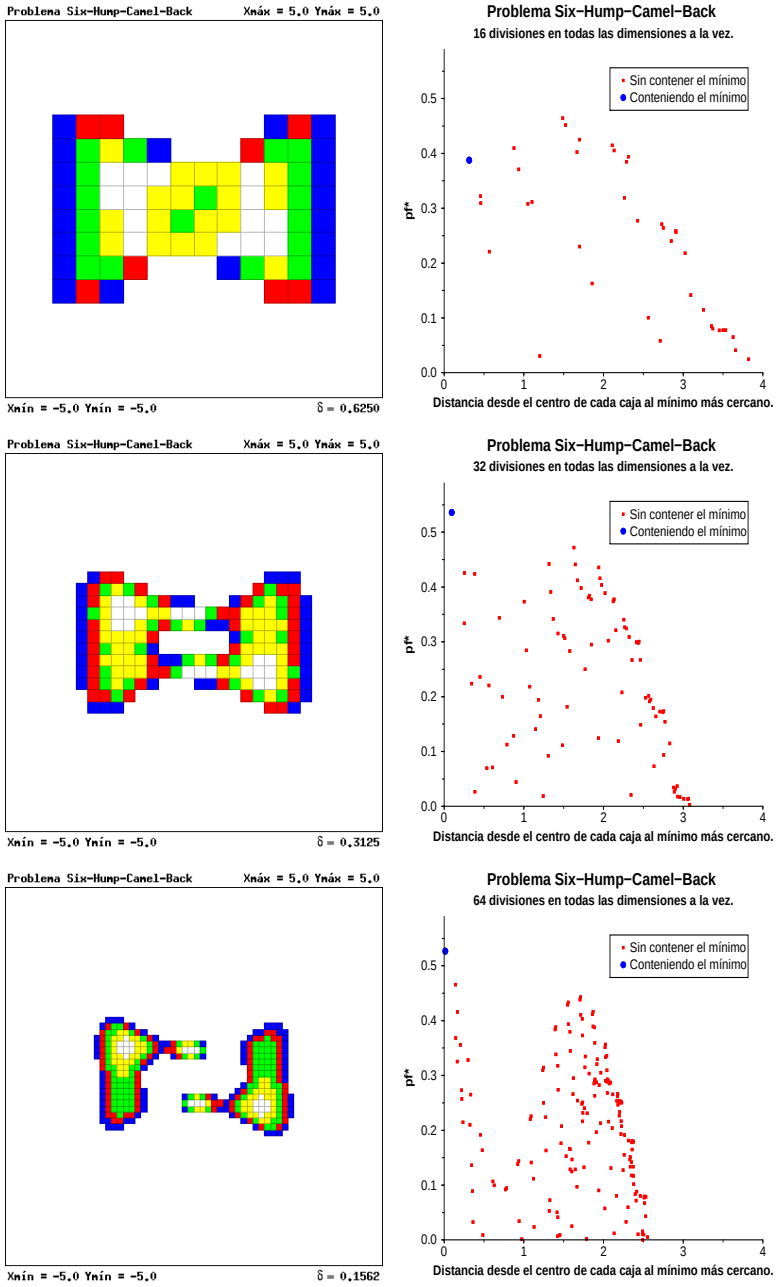


Figura 3.2: En la columna de la izquierda se muestran las cajas no eliminadas por el *Test del Punto Medio*. Las cajas más claras tienen un mayor valor de $\overline{pf^*}(X)$. En la columna de la derecha se representan los valores de $\overline{pf^*}(X)$ respecto a la distancia del centro de la caja X con el mínimo global más cercano.

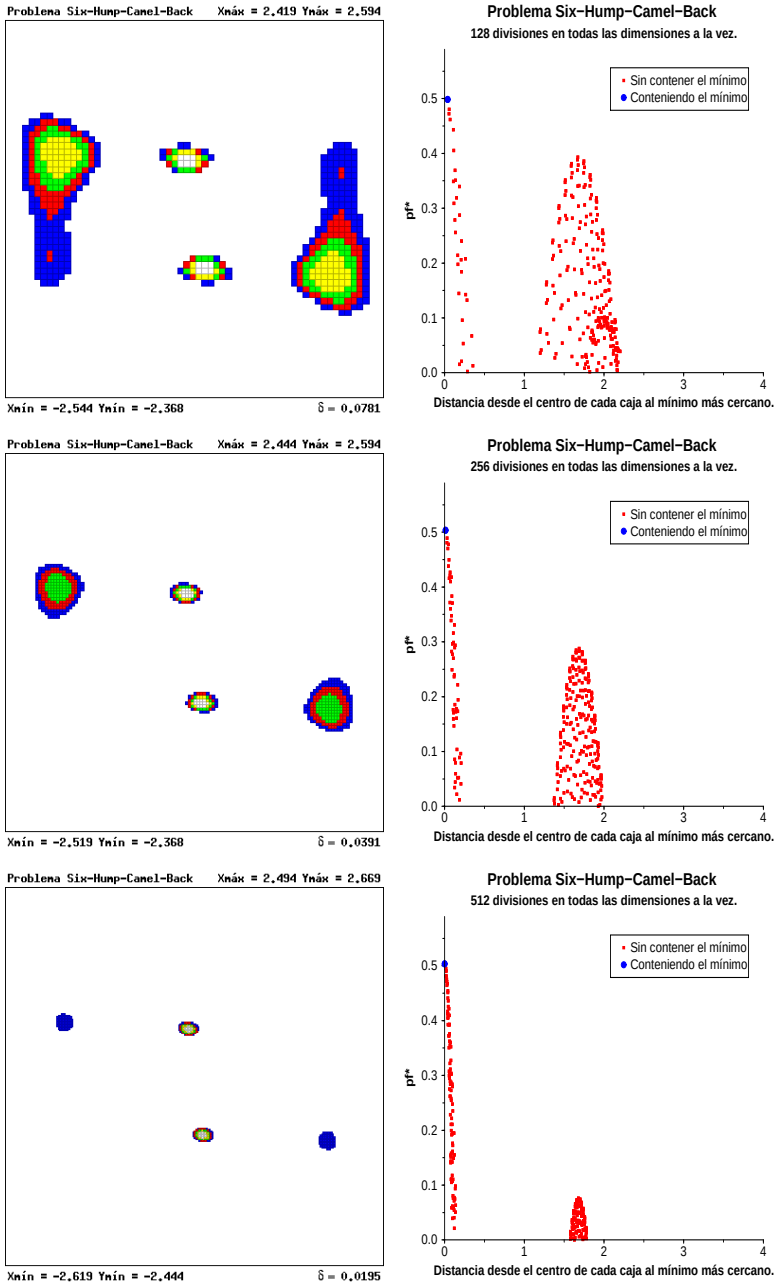


Figura 3.3: En la columna de la izquierda se muestran zooms de las cajas no eliminadas por el *Test del Punto Medio*. Las cajas más claras tienen un mayor valor de $\overline{pf^*}(X)$. En la columna de la derecha se representan los valores de $\overline{pf^*}(X)$ respecto a la distancia del centro de la caja X con el mínimo global más cercano.

Las Figuras 3.2 y 3.3 muestran los resultados de estas evaluaciones para la función bidimensional Six-Hump-Camel-Back (véase el Apéndice A). En las columnas de la izquierda de las Figuras 3.2 y 3.3 se muestran los valores de $p\overline{f^*}(X)$. Cada caja representa el intervalo bidimensional X y el valor de $p\overline{f^*}(X)$ viene dado por su color. Las cajas más claras tienen un mayor valor de $p\overline{f^*}(X)$. Las cajas con un valor de $p\overline{f^*}(X) < 0$ no se han dibujado, ya que serían eliminadas por el *Test del Punto Medio*.

En las columnas de la derecha de las Figuras 3.2 y 3.3 se ha representado los valores de $p\overline{f^*}(X)$ respecto a la distancia de $m(X)$ al mínimo global más cercano. Puede observarse que para valores pequeños de δ, las cajas X con los mayores valores de $p\overline{f^*}(X)$ (i.e. aquellas más claras), están cerca de un mínimo global o local de la función objetivo. Se ha obtenido el mismo resultado para todas las funciones de prueba utilizadas en este trabajo. Parece ser que $p\overline{f^*}$ puede considerarse como un parámetro que da información acerca de la localización de un mínimo global o local.

A continuación se estudiará si esta información tan fácil de obtener, acerca de la localización de los mínimos, es correcta:

Teorema 3.2.1 *Supóngase que para el problema de optimización (3.1), la función de inclusión $F(X)$ de $f(x)$ es isótona (Definición 1.2.3, página 14) y α-convergente (Definición 1.2.4, página 15) para unas constantes positivas α y K. Supóngase además que el parámetro $p\overline{f^*}(X)$ es menor que 1 para todas las subcajas X de la región de búsqueda S. Entonces un número arbitrariamente grande ($N > 0$) de intervalos seleccionados consecutivamente por el Algoritmo 3.1.1 pueden tener la propiedad de que [16]:*

1. *Ninguno de esos intervalos procesados tengan un punto estacionario (en este caso un mínimo).*

2. *Durante esta fase de la búsqueda, los valores de $p\overline{f^*}(X)$ de estos intervalos son máximos.*

Demostración. Considérese que bajo las condiciones anteriores existe una caja $Y \subset S$ con $\underline{F}(Y) > f^*$ (debería ser eliminada) y que Y no contiene ningún punto estacionario. Este tipo de cajas es fácil de encontrar en los problemas de Optimización.

Constrúyase la siguiente función de inclusión:

$$G(X) = [\underline{F}(X) - Dw(X)^{\alpha}, \overline{F}(X)] \tag{3.9}$$

para la caja Y y para las cajas Z que sean ascendentes o descendientes de Y y $G(X) = F(X)$ para las demás. Es fácil ver que $Z \subseteq Z'$ implica que $G(Z) \subseteq G(Z')$, i.e. G es isótona. Por otro lado, G es también α-convergente con las constantes α y $K + D$.

Considérese la iteración en la que la caja Y fue generada y X' fue seleccionada en la siguiente iteración, por lo tanto, $\underline{F}(X') \leq \underline{F}(Y)$. Usando la *función de inclusión* $G(X)$, puede establecerse el valor de D de tal forma que el Algoritmo 3.1.1 seleccione la caja Y en vez de X', siendo los siguientes N intervalos seleccionados descendientes de Y.

Nótese que conforme se incrementa el valor de D, el correspondiente valor de $p\overline{f^*}(Z)$ converge a 1. Denótese por $p\overline{f^*_G}$ al parámetro $p\overline{f^*}(Z)$ obtenido con la *función de inclusión* $G(X)$ y $p\overline{f^*_F}$ al obtenido con $F(X)$. Eligiendo un valor apropiado de D, es obvio que $\underline{G}(Z) < \underline{F}(X')$ y que $p\overline{f^*_G}(Z) > p\overline{f^*_F}(X')$, $\forall Z \subseteq Y$ (si $p\overline{f^*}(X') < 1$).

Por lo tanto, si se usa la *función de inclusión* $G(X)$ con la máxima constante D, la serie de N intervalos seleccionados a partir del intervalo Y cumplen las propiedades 1 y 2 del Teorema 3.2.1. ■

La consecuencia del Teorema 3.2.1 es, que incluso usando una *función de inclusión* isótona y α-convergente, la secuencia de intervalos seleccionados para su procesamiento puede estar lejos de los minimizadores globales durante un tiempo arbitrariamente largo.

Si se fuerza al Algoritmo 3.1.1 para que pare cuando se ha encontrado una sola caja que cumple la *Regla de Terminación*, entonces el resultado obtenido podría ser incorrecto.

La suposición de que el valor de $p\overline{f^*}(X)$ es menor que uno para todos los intervalos $X \subset S$ es cierta para la mayoría de los problemas prácticos. Puede no ser cierta sólo en los problemas donde el valor de la función objetivo es constante en todo un subintervalo $X \subseteq S$, siendo este valor igual al mínimo global y además $\overline{F}(X)=f^*$. En estos casos tan poco comunes, se obtendría rápidamente un resultado más que satisfactorio.

A partir de ahora se investigarán los valores de $p\overline{f^*}(X)$ de los intervalos generados por el Algoritmo 3.1.1. En la Figura 3.4 se han dibujado las cajas eliminadas y las insertadas en el árbol final durante la ejecución del algoritmo de Ramificación y Acotación, para tres funciones de prueba bidimensionales. De izquierda a derecha, las funciones usadas son GoldStein-Price (GP), Six-Hump-Camel-Back (C), y Levy 3 (L3), que tienen uno, dos y nueve mínimos globales, respectivamente (véase el Apéndice A). A diferencia de las cajas representadas en las Figuras 3.2 y 3.3, todas las cajas activas en un momento dado de la ejecución el Algoritmo 3.1.1 no tienen por qué tener el mismo tamaño. En la Figura 3.4, las cajas eliminadas se han representado en gris, mientras que las cajas blancas son aquellas que se almacenaron en el árbol final, determinando así la región solución. Como puede observarse en la Figura 3.4, la región solución disminuye conforme disminuye el valor de ϵ, definiendo cada vez mejor los mínimos globales. En la última fila se han dibujado las cajas de la fila anterior ($\epsilon = 10^{-3}$) que en cualquier momento de la ejecución del algoritmo alcanzaron un valor de $p\overline{f^*} > 0.4$. Como puede observarse, estas cajas se encuentran sobre los mínimos locales y globales.

3.3 División múltiple en algoritmos de Optimización Global basados en intervalos

Como se introdujo en la Sección 3.1, el método de división tradicional es la bisección de la coordenada más ancha [153]. De acuerdo con nuevos estudios [7, 34, 35], dividir varias coordenadas a la vez puede mejorar la eficiencia de las técnicas de Ramificación y Acotación aplicadas a la Optimización Global. Todos estos trabajos investigan o discuten exclusivamente una división múltiple fija y estática, es decir, el método de división elegido

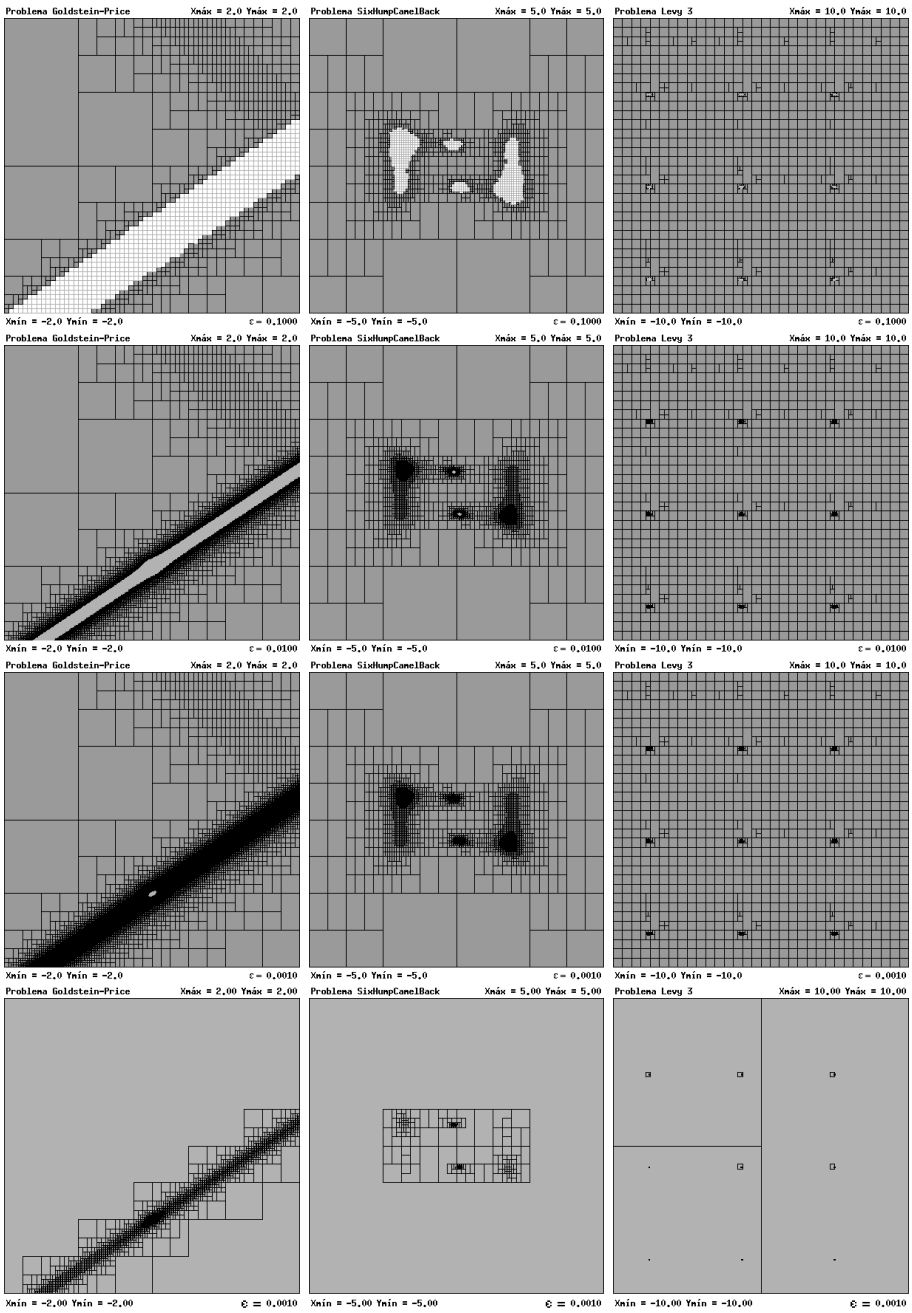


Figura 3.4: Ejecución del Algoritmo 3.1.1 para las funciones GP, C y L3. En las tres primeras filas se ha usado ϵ=10^{-1}, 10^{-2} y 10^{-3}, respectivamente. En la cuarta fila sólo se han dibujado las cajas de la tercera fila con $\overline{pf^*}(X) > 0.4$.

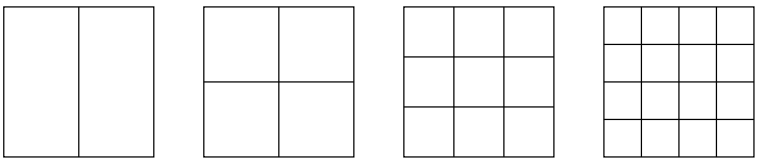


Figura 3.5: Métodos de división sobre un problema bidimensional. El método de división adaptativa se basa en (a), (b) y (d).

no variaba durante toda la ejecución del algoritmo. En esta sección se presenta una *Regla de División* adaptativa basada en varias estrategias de división, simples y regulares que se muestran en la Figura 3.5 [15, 17]. Aunque en la Figura 3.5 se presenta un ejemplo de problema bidimensional, éste debe extenderse al caso n-dimensional. Así, la gráfica 3.5(a) representa la bisección de la coordenada mayor y las gráficas 3.5(b), 3.5(c) y 3.5(d) realizan una, dos y tres divisiones en todas las dimensiones, respectivamente. Estas estrategias no hacen uso de información acerca de la función objetivo ni de sus derivadas. Por lo tanto, esta división adaptativa es apropiada para funciones diferenciables y no diferenciables. Para funciones diferenciables, nuestra propuesta podría mejorarse mezclándola con las propuestas de otros autores [7, 34, 158, 35].

Aunque se conozca el mínimo global de la función, el espacio de búsqueda de cualquier problema de Optimización, basado en Ramificación y Acotación, debe dividirse para determinar dónde se encuentra el conjunto de los puntos minimizadores globales X^*. La selección de la *Regla de División* a usar juega un papel importante respecto al coste computacional del algoritmo. En el caso particular del Algoritmo 3.1.1, si una caja se divide por una sola coordenada, el número de cajas generadas en una división es el menor posible. Si las cajas generadas se eliminan, el número de subcajas evaluadas será menor que el realizado por otro tipo de división. En algunos casos, el algoritmo que usa este método de bisección puede dar lugar a más evaluaciones de la *función de inclusión* que usando una división múltiple. Las relaciones entre la complejidad de ambos tipos de división se detallan en el siguiente Teorema:

Teorema 3.3.1 *El ahorro de evaluaciones de la función de inclusión causadas por el uso de una división múltiple en el Algoritmo 3.1.1 (línea 12) es [15]:*

1.- En el mejor de los casos, cerca de la mitad de las requeridas usando la bisección de una sola coordenada (bisección simple).

2.- En el peor de los casos, una división múltiple puede requerir un número de evaluaciones arbitrariamente grande comparado con el requerido si se usa una bisección simple.

Demostración.

1.- El número de evaluaciones de la *función de inclusión* usando una bisección simple está estrechamente relacionado con el número de nodos de un árbol binario de búsqueda,

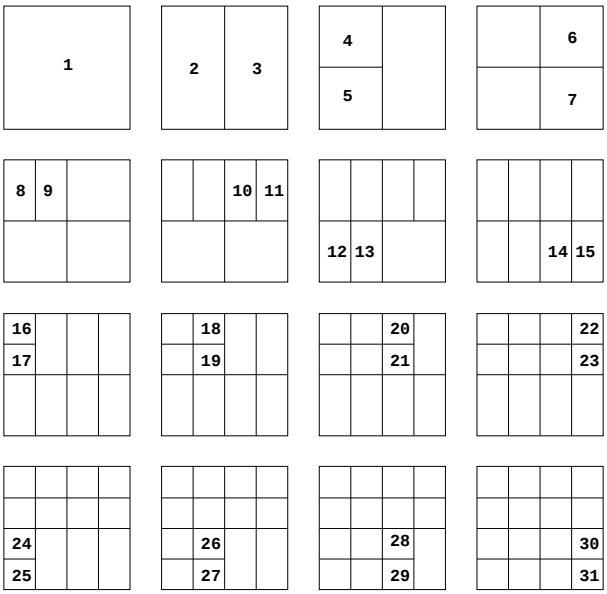


Figura 3.6: Posibles divisiones realizadas en un ejemplo bidimensional usando bisección de una sola coordenada.

es decir, con el número de subcajas generadas. El mejor caso respecto a la mejora del algoritmo ocurre cuando todos los intervalos de tamaño $2^{-\lfloor l/n \rfloor} w(S)$, para un nivel del árbol $l > 0$ y un problema en la región de búsqueda $S \in \mathbb{I}^n$, deben ser evaluados. El mismo planteamiento puede aplicarse a cualquier subárbol del árbol de búsqueda. Aquí, $\lfloor . \rfloor$ representa el mayor entero no mayor que el argumento.

Usando una bisección simple, si se completa el nivel l del árbol de búsqueda, sin eliminar ninguna caja, el número de evaluaciones de intervalos sería $2^0 + 2^1 + \ldots + 2^l = 2^{l+1} - 1$. Nótese que siempre se hace la evaluación de toda la región de búsqueda, S, en el paso 2 del Algoritmo 3.1.1. Si se realiza una división múltiple, como la representada en la Figura 3.5(d), de forma que se generan $s = 2^l$ subcajas a la vez, se realizarían $2^l + 1$ evaluaciones de intervalos, con un ahorro de $2^l - 2$ evaluaciones.

En la Figura 3.6, el nivel el árbol de búsqueda l es igual a cuatro. Usando una bisección simple, se necesitan realizar $2^{4+1} - 1 = 31$ evaluaciones en total, mientras que si se usa una división múltiple con $s = 2^4 = 16$, el número de evaluaciones de la *función de inclusión* es $2^4 + 1 = 17$. El ahorro es por tanto de $2^4 - 2 = 14$ evaluaciones. El porcentaje de ahorro viene dado por la expresión $(2^l - 2)/(2^{l+1} - 1)$, que converge a 0.5 cuando l tiende a infinito.

2.- Por otro lado, se realizarían computaciones innecesarias si una subcaja, que puede ser rechazada después de una bisección simple, se divide usando una división múltiple. El peor caso respecto a la complejidad de la división múltiple, comparada con la bisección simple, se da cuando se alcanza el nivel l del árbol de búsqueda, debido a la *Regla de*

Terminación, y en cada bisección simple se produce una subcaja que es eliminada. Así, usando una bisección simple se requieren $2l+1$ evaluaciones, mientras que con una división múltiple se requerirían $2^l + 1$ evaluaciones. El porcentaje de evaluaciones innecesarias debidas a una división múltiple, comparadas con las obtenidas mediante bisección simple, es $(2^l - 2l)/(2l + 1)$. Este porcentaje puede ser arbitrariamente alto para un valor de l suficientemente grande. ■

Para el Algoritmo 3.1.1, con una división múltiple del tipo (d), la mejora del rendimiento del algoritmo que podría obtenerse, comparado con el uso de una bisección simple, está cerca del 45%, mientras que el porcentaje de posibles evaluaciones innecesarias adicionales está sobre el 89%. En otras palabras, la posible pérdida de rendimiento es casi dos veces mayor que la posible mejora.

La consecuencia del Teorema 3.3.1 es, que es importante conocer qué método de división es más apropiado para cada fase de la resolución del problema. Nuestra propuesta de un método de división adaptativo se basa en resultados experimentales que muestran que usando una bisección simple, las cajas alrededor de los mínimos globales son similares a las que se obtendrían, si se hubiera utilizado un método de división múltiple, por lo que merecería la pena aplicarlo. Un ejemplo de estos experimentos se muestra en la Figura 3.7, donde se ha realizado un zoom sobre uno de los mínimos globales de las funciones Goldstein-Price, Six-Hump-Camel-Back y Levy3. Las cajas fueron generadas usando el Algoritmo 3.1.1, es decir, usando una bisección simple.

En las gráficas puede observarse que:

- Alrededor de un punto minimizador global se han generado muchas cajas, por lo que una división múltiple parece ser ventajosa frente a una bisección simple.

- Fuera de la región de atracción a un mínimo, se incrementa el número de cajas rectangulares, por lo que realizar una bisección simple parece lo más apropiado.

Para establecer el nivel de división se va a utilizar, el parámetro $p\overline{f^*}(X)$, como estimador de la distancia de un mínimo a la región definida por una caja concreta, y dos umbrales P_1 y P_2 del parámetro $p\overline{f^*}(X)$. Dependiendo de los valores de P_1, P_2 $(0 \leq \mathrm{P}_1, \mathrm{P}_2 \leq 1)$ y $p\overline{f^*}(X)$, la caja X se dividirá de la siguiente forma:

- Las cajas con $p\overline{f^*}(X) < \mathrm{P}_1$ se dividirán de forma análoga a la Figura 3.5(a) (Bisección simple).
- Las cajas con $\mathrm{P}_1 \leq p\overline{f^*}(X) < \mathrm{P}_2$ se dividirán en 2^n subcajas de forma análoga a la Figura 3.5(b).
- Las cajas con $p\overline{f^*}(X) \geq \mathrm{P}_2$ se dividirán en 4^n subcajas de forma análoga a la Figura 3.5(d).

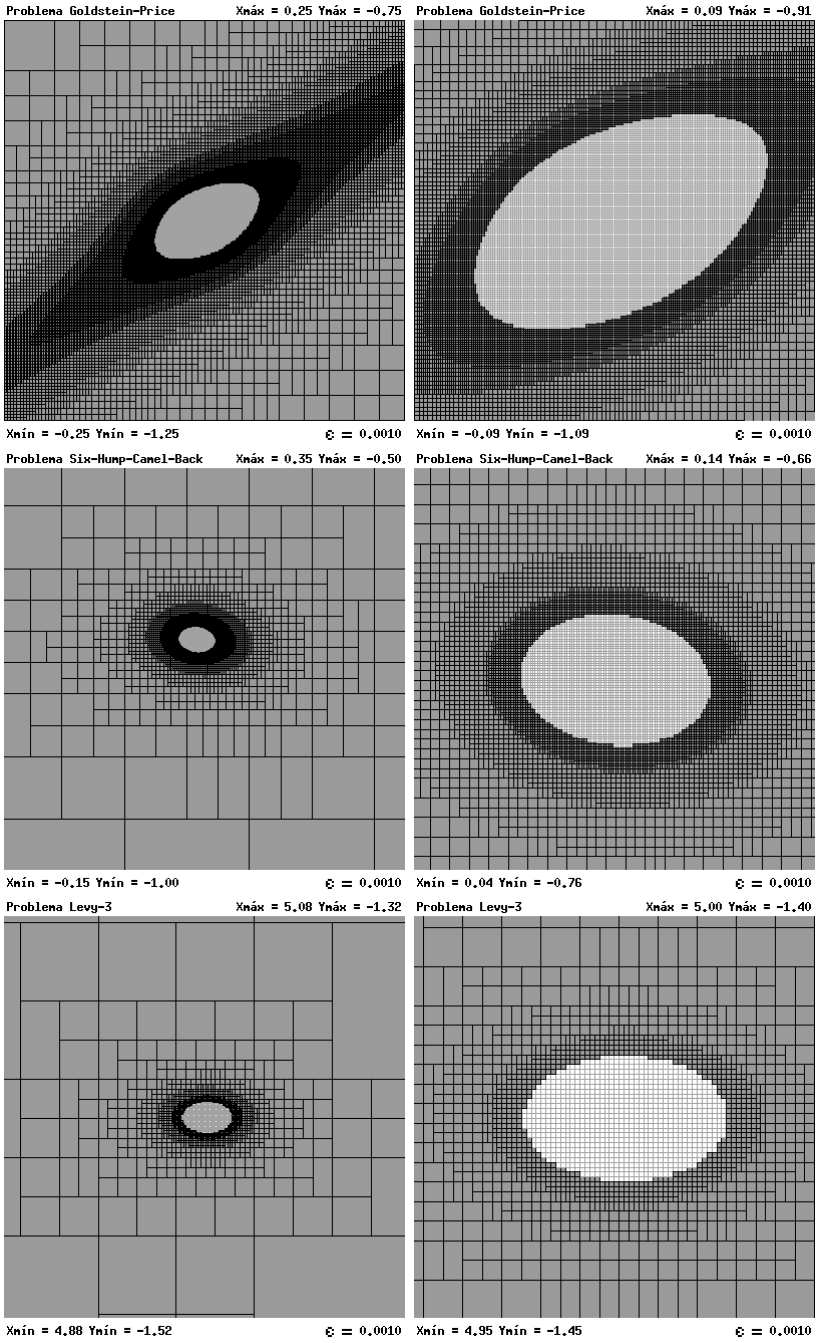


Figura 3.7: Zooms sobre un punto minimizador global de las funciones Goldstein-Price, Six-Hump-Camel-Back y Levy3.

3.3.1 Resultados numéricos

La pruebas numéricas se han llevado a cabo sobre un PC Pentium II (233 MHz y 256Mb de RAM), con el Sistema Operativo Linux. Los programas fueron codificados en C y se usaron las rutinas BIAS [90] para implementar la Aritmética de Intervalos. La unidad de tiempo estándar, tiempo de CPU necesario para evaluar 1000 veces la función real Shekel-10 en $(4.0, 4.0, 4.0, 4.0)^T$, es de 0.037 segundos.

Se han usado para estos experimentos siete funciones de prueba estándar dentro de la Optimización Global: Six-Hump-Camel-Back (abreviada como C), Goldstein-Price (GP), Hartman-3 (H3), Levy 3 (L3), Shekel-5 (S5), Shekel-7 (S7) y Shekel-10 (S10), que se describen en el Apéndice A.

También se han usado seis estrategias de división para el Algoritmo 3.1.1 (línea 12), que se describen a continuación:

1D : Siempre se divide de forma análoga a la (Figura 3.5(a)).
1DA : Siempre se divide de forma análoga a la (Figura 3.5(b)).
2DA : Siempre se divide de forma análoga a la (Figura 3.5(c)).
3DA : Siempre se divide de forma análoga a la (Figura 3.5(d)).
P_1P_2A : Se han optimizado los parámetros P_1 y P_2 para todas las funciones (P_1 y P_2 son los mismos para todos los problemas).
P_1P_2 : P_1 y P_2 se han optimizado para cada problema.

En la estrategia P_1P_2, los parámetros P_1 y P_2 se determinaron separadamente para cada problema mediante un algoritmo estocástico de Optimización Global, llamado SASS (Single Agent Stochastic Search) [170, 176, 23]. La función objetivo que se ha usado, intentando minimizar el número de evaluaciones de la *función de inclusión*, es $f(P_1, P_2) = EX(P_1, P_2) + Ex(P_1, P_2)$, donde $EX(P_1, P_2)$ y $Ex(P_1, P_2)$ devuelven el número de evaluaciones de $F(X)$ y de $F(m(X))$, respectivamente, realizadas por el Algoritmo 3.1.1, usando una *Regla de División* basada en los parámetros P_1 y P_2. La optimización de estos parámetros se ha realizado mediante un método estocástico porque es muy difícil, si no imposible, el obtener las funciones $EX(P_1, P_2)$ y $Ex(P_1, P_2)$ de forma analítica.

En el estudio realizado, los parámetros P_1 y P_2 se mantienen constantes durante toda la ejecución del Algoritmo. En trabajos futuros se investigará el uso dinámico de estos parámetros.

Se ha introducido el método P_1P_2A para intentar encontrar los valores P_1 y P_2 que minimicen el costo computacional de la solución para todos los problemas de prueba, de forma análoga al método P_1P_2. Las optimizaciones mencionadas para afinar los valores de los parámetros P_1 y P_2 requirieron una gran cantidad de computaciones, incluso para los algoritmos de Ramificación y Acotación relativamente más simples, ya que para una sola ejecución del algoritmo *SASS*, todos los problemas de prueba deben ser resueltos varios cientos de veces. Esta es la razón por la que sólo se ha usado un número pequeño de funciones de prueba.

En las Tablas 3.1 a 3.5 se ha usado la siguiente notación para las columnas:

CPU : Tiempo de ejecución en segundos.
EX : Número de evaluaciones de $F(X)$.
Ex : Número de evaluaciones de $F(m(X))$.
AF : Número de cajas en el árbol final.
AM : Número máximo de cajas en el árbol de trabajo.

En las Tablas 3.1 a 3.3, el valor de Ex es igual que el número de cajas extraídas del árbol de trabajo, es decir, el número de iteraciones, ya que sólo se evalúa $F(m(X))$ sobre las cajas seleccionadas.

En las Tablas 3.1 a 3.1, puede observarse que en la mayoría de los casos, los métodos de división 2DA y 3DA no mejoran los resultados obtenidos mediante una bisección simple, en términos de tiempo de CPU y número de evaluaciones de $F(X)$, pero en la mayoría de los casos necesitan un número menor de evaluaciones de $F(m(X))$. Esto es debido a que las versiones 2DA y 3DA generan un número mayor de cajas en cada división y por lo tanto más pequeñas que si se usa una bisección simple. La mayoría de las cajas generadas se eliminan directamente, por lo que el número de cajas seleccionadas del árbol de trabajo, antes de que se cumpla el criterio de terminación, es menor. En la Tabla 3.4 se muestra el promedio de los resultados obtenidos, donde se confirma este comportamiento.

Para los problemas S5, S7 y S10, que son simples de resolver, el método 1D es mejor que el 1DA porque en muchos casos, cuando una caja es dividida, una de las subcajas se elimina. Para las demás funciones, el método 1DA parece ser mejor que el 1D. Esto se debe a que generalmente se necesitan más divisiones para poder eliminar cajas.

El método P_1P_2 tiene en todos los casos el menor número de evaluaciones de intervalos de todas las variantes del algoritmo investigadas. La mejora es proporcional a la precisión requerida. El método P_1P_2A, aunque es peor que el método P_1P_2, da mejores resultados que los métodos estáticos en promedio.

3.3.2 Evaluación de los resultados numéricos

En la Tabla 3.4 se muestra un resumen de los resultados numéricos obtenidos en las Tablas 3.1, 3.2 y 3.3, para ver el comportamiento de la eficiencia con diferentes complejidades del problema. La Tabla 3.4 muestra los valores medios y la eficiencia relativa de los distintos métodos de división, comparando los resultados con los obtenidos con el método 1D. Estos valores medios representan el comportamiento del algoritmo para resolver problemas similares a los del conjunto de prueba.

Desde el punto de vista del usuario, lo más importante es el número de evaluaciones de la *función de inclusión*, puesto que los problemas reales son normalmente más complejos de resolver que los de prueba. Debido a las propiedades del algoritmo, siempre se necesitan más evaluaciones de $F(X)$ que de $F(m(X))$. Las consecuencias de los resultados pueden resumirse en:

1. Respecto a los valores de EX, las divisiones múltiples y estáticas, 2DA y 3DA, son definitivamente peores que el método básico 1D, mientras que el método 1DA

Tabla 3.1: Resultados numéricos de las seis variantes del Algoritmo 3.1.1 para $\epsilon = 10^{-1}$.

Problema	CPU	EX	Ex	AF	AM	P_1	P_2
C-1D	0.090	2731	1365	528	348		
C-1DA	0.060	2589	647	528	286		
C-2DA	0.110	5518	613	554	429		
C-3DA	0.120	6865	429	550	316		
C-P_1P_2A	0.063	2461	653	528	286	0.0405	1.0000
C-P_1P_2	0.047	2055	403	556	200	0.1854	0.3249
GP-1D	0.100	2537	1268	679	244		
GP-1DA	0.060	2037	509	679	202		
GP-2DA	0.070	2863	318	974	201		
GP-3DA	0.050	2209	138	682	120		
GP-P_1P_2A	0.057	1979	510	679	202	0.0405	1.0000
GP-P_1P_2	0.043	1591	286	680	186	0.1423	0.3085
H3-1D	0.060	727	363	118	62		
H3-1DA	0.030	553	69	154	36		
H3-2DA	0.100	2026	75	382	65		
H3-3DA	0.057	1153	18	189	55		
H3-P_1P_2A	0.030	549	97	154	36	0.0405	1.0000
H3-P_1P_2	0.030	543	73	154	36	0.0073	1.0000
L3-1D	0.560	2571	1285	72	544		
L3-1DA	0.360	1929	482	75	820		
L3-2DA	1.030	6562	729	99	723		
L3-3DA	0.773	4849	303	84	1711		
L3-P_1P_2A	0.357	1859	505	75	820	0.0405	1.0000
L3-P_1P_2	0.357	1865	484	75	833	0.0163	0.3417
S5-1D	0.007	79	39	16	12		
S5-1DA	0.010	113	7	16	26		
S5-2DA	0.020	406	5	16	115		
S5-3DA	0.050	1025	4	32	64		
S5-P_1P_2A	0.010	113	7	16	26	0.0405	1.0000
S5-P_1P_2	0.007	79	39	16	12	1.0000	1.0000
S7-1D	0.007	83	41	16	14		
S7-1DA	0.010	113	7	16	28		
S7-2DA	0.030	406	5	16	117		
S7-3DA	0.060	1025	4	47	75		
S7-P_1P_2A	0.010	113	7	16	28	0.0405	1.0000
S7-P_1P_2	0.007	83	41	16	14	1.0000	1.0000
S10-1D	0.010	87	43	16	14		
S10-1DA	0.010	113	7	16	32		
S10-2DA	0.037	406	5	16	128		
S10-3DA	0.080	1025	4	47	101		
S10-P_1P_2A	0.010	113	7	16	32	0.0405	1.0000
S10-P_1P_2	0.010	87	43	16	14	1.0000	1.0000

Tabla 3.2: Resultados numéricos de las seis variantes del Algoritmo 3.1.1 para $\epsilon = 10^{-2}$.

Problema	CPU	EX	Ex	AF	AM	P_1	P_2
C-1D	0.240	7919	3959	424	668		
C-1DA	0.200	8597	2149	424	1098		
C-2DA	0.240	13006	1445	894	955		
C-3DA	0.270	15665	979	438	944		
C-P_1P_2A	0.200	9073	1137	438	1030	0.0347	0.2851
C-P_1P_2	0.167	6633	1529	424	2032	0.1422	0.3379
GP-1D	1.760	43923	21961	12106	4413		
GP-1DA	0.960	32153	8038	12107	3087		
GP-2DA	1.203	53227	5914	16988	3041		
GP-3DA	1.840	92897	5806	16591	3353		
GP-P_1P_2A	0.660	26277	2592	12123	1438	0.0347	0.2851
GP-P_1P_2	0.647	25665	2648	12123	1450	0.1070	0.2853
H3-1D	1.267	15905	7952	2516	1296		
H3-1DA	0.560	10249	1281	2517	961		
H3-2DA	2.030	42229	1564	8208	1769		
H3-3DA	2.790	61249	957	7760	1112		
H3-P_1P_2A	0.560	10085	1488	2517	1152	0.0347	0.2851
H3-P_1P_2	0.550	9951	1247	2517	959	0.0176	0.3910
L3-1D	1.113	4963	2481	502	544		
L3-1DA	0.680	3633	908	502	820		
L3-2DA	1.120	7066	785	553	723		
L3-3DA	1.520	9857	616	1000	1711		
L3-P_1P_2A	0.940	5571	727	508	1711	0.0347	0.2851
L3-P_1P_2	0.640	3455	826	504	833	0.0227	0.3229
S5-1D	0.017	233	116	19	15		
S5-1DA	0.020	401	25	38	26		
S5-2DA	0.030	568	7	16	115		
S5-3DA	0.060	1281	5	38	64		
S5-P_1P_2A	0.040	859	63	45	64	0.0347	0.2851
S5-P_1P_2	0.017	233	116	19	15	1.0000	1.0000
S7-1D	0.030	313	156	40	33		
S7-1DA	0.030	433	27	56	28		
S7-2DA	0.087	1378	17	51	132		
S7-3DA	0.077	1281	5	56	75		
S7-P_1P_2A	0.050	873	42	61	75	0.0347	0.2851
S7-P_1P_2	0.027	415	32	56	28	0.0166	1.0000
S10-1D	0.040	331	165	40	31		
S10-1DA	0.037	465	29	56	32		
S10-2DA	0.140	1864	23	62	132		
S10-3DA	0.100	1281	5	56	101		
S10-P_1P_2A	0.070	891	37	58	101	0.0347	0.2851
S10-P_1P_2	0.037	427	31	56	32	0.0111	1.0000

Tabla 3.3: Resultados numéricos de las seis variantes del Algoritmo 3.1.1 para $\epsilon = 10^{-3}$.

Problema	CPU	EX	Ex	AF	AM	P_1	P_2
C-1D	1.457	42695	21347	6616	4892		
C-1DA	0.820	33413	8353	6616	4342		
C-2DA	0.850	44848	4983	7968	3260		
C-3DA	0.860	48945	3059	6642	2430		
C-P_1P_2A	0.620	29769	3535	6642	2522	0.0353	0.2705
C-P_1P_2	0.617	28469	4301	6642	2144	0.0489	0.3055
GP-1D	6.920	181125	90562	9517	17840		
GP-1DA	4.397	169209	42302	9517	15101		
GP-2DA	5.040	258688	28743	14489	11601		
GP-3DA	6.230	357873	22367	9529	14422		
GP-P_1P_2A	3.250	140349	24270	9517	9943	0.0353	0.2705
GP-P_1P_2	3.173	119933	27978	9518	15917	0.0967	0.3081
H3-1D	38.170	454569	227284	71893	36438		
H3-1DA	15.820	283409	35426	71894	28158		
H3-2DA	66.663	1379809	51104	225050	48435		
H3-3DA	25.767	557889	8717	73520	8166		
H3-P_1P_2A	13.683	260265	18548	73520	8575	0.0353	0.2705
H3-P_1P_2	13.480	262145	14791	73520	8586	0.0149	0.2712
L3-1D	10.430	44865	22432	7693	5896		
L3-1DA	6.207	32765	8191	7693	5017		
L3-2DA	10.890	67735	7526	13841	5773		
L3-3DA	13.320	87265	5454	15357	4557		
L3-P_1P_2A	5.250	30373	4385	7705	3110	0.0353	0.2705
L3-P_1P_2	5.220	30129	4397	7705	3103	0.0520	0.2676
S5-1D	0.047	627	313	70	66		
S5-1DA	0.040	769	48	70	58		
S5-2DA	0.147	3079	38	74	282		
S5-3DA	0.150	3329	13	70	71		
S5-P_1P_2A	0.050	1067	69	70	64	0.0353	0.2705
S5-P_1P_2	0.037	619	64	70	58	0.0257	1.0000
S7-1D	0.347	3743	1871	490	432		
S7-1DA	0.293	4801	300	523	329		
S7-2DA	1.040	17335	214	1447	520		
S7-3DA	1.470	24833	97	1006	151		
S7-P_1P_2A	0.433	6835	292	1006	152	0.0353	0.2705
S7-P_1P_2	0.223	3235	602	522	326	0.0842	1.0000
S10-1D	0.533	4607	2303	623	534		
S10-1DA	0.490	6161	385	643	362		
S10-2DA	1.517	20251	250	1692	643		
S10-3DA	2.400	32513	127	1143	175		
S10-P_1P_2A	0.630	7675	361	1143	172	0.0353	0.2705
S10-P_1P_2	0.350	3841	709	643	362	0.0906	1.0000

Tabla 3.4: Resumen de los resultados numéricos para los distintos tipos de división. Los valores medios de las distintas ejecuciones se muestran en las columnas (Med.). También se muestran los valores relativos comparados con los obtenidos con el tipo de división 1D.

		CPU		EX		Ex		AM	
ϵ	subd.	Med.	/1D	Med.	/1D	Med.	/1D	Med.	/1D
10^{-1}	1D	0.1191		1259		629		177	
	1DA	0.0771	65%	1064	84%	247	39%	204	116%
	2DA	0.1996	168%	2598	206%	250	40%	254	144%
	3DA	0.1700	143%	2593	206%	129	20%	349	197%
	P_1P_2A	0.0767	64%	1027	82%	255	41%	204	116%
	P_1P_2	0.0716	60%	900	72%	196	31%	185	105%
10^{-2}	1D	0.6381		10512		5256		1000	
	1DA	0.3553	56%	7990	76%	1780	34%	865	86%
	2DA	0.6929	109%	17048	162%	1394	27%	981	98%
	3DA	0.9510	149%	26216	249%	1196	23%	1051	105%
	P_1P_2A	0.3600	56%	7661	73%	869	17%	796	80%
	P_1P_2	0.2979	47%	6683	64%	918	17%	764	76%
10^{-3}	1D	8.2720		104604		52302		9443	
	1DA	4.0096	48%	75790	72%	13572	26%	7624	81%
	2DA	12.3067	149%	255964	245%	13265	25%	10073	107%
	3DA	7.1710	87%	158950	152%	5691	11%	4282	45%
	P_1P_2A	3.4166	41%	68048	65%	7351	14%	3505	37%
	P_1P_2	3.3000	40%	64053	61%	7549	14%	4357	46%

obtiene una mejor eficiencia en promedio. Esta mejora es mayor cuanto menor es el parámetro de parada ϵ (los problemas de prueba son más difíciles de resolver). Las dos variantes adaptativas (P_1P_2A y P_1P_2) mejoran aún más los resultados y el método a medida, P_1P_2, es siempre el mejor.

Esto indica que para el usuario merece la pena establecer los parámetros P_1 y P_2 del algoritmo para el conjunto de problemas a resolver (basándose en ejemplos pequeños pero representativos). La mejora del número de evaluaciones $F(X)$ esperada está, dependiendo de ϵ, entre un 18 y un 35% para el método P_1P_2A y entre un 28 y un 39% para el P_1P_2.

2. El número de evaluaciones de $F(m(X))$ se mejora con todas las nuevas variantes del algoritmo. Esto es debido a que la división múltiple permite eliminar más subcajas mediante el *Test del Punto Medio*, por lo que el número de cajas en las que se evalúa $F(m(X))$ es menor. Como el número de evaluaciones de $F(m(X))$ es igual al número de cajas extraídas del árbol de trabajo (y divididas si no alcanzan la *Regla de Terminación*), las mejoras son mayores que las conseguidas para las evaluaciones de $F(X)$, ya que éstas deben realizarse sobre todas las subcajas generadas. El método P_1P_2 no da el menor número de evaluaciones de $F(m(X))$ para $\epsilon = 10^{-1}$ y $\epsilon = 10^{-3}$. Estos resultados son acordes con la función optimizada para encontrar los

parámetros P_1 y P_2: $f(P_1, P_2) = EX(P_1, P_2) + Ex(P_1, P_2)$. Las mejoras obtenidas con los métodos P_1P_2A y P_1P_2 están entre un 59 y un 86% y entre un 69 y un 86%, respectivamente. El ahorro de evaluaciones crece conforme decrece el valor del parámetro de terminación ϵ.

3. El valor del tiempo de CPU requerido permite prever la mejora respecto EX y Ex. Así, la variante P_1P_2 es siempre la mejor. Para $\epsilon = 10^{-1}$ y $\epsilon = 10^{-2}$, los métodos P_1P_2A y 1DA obtienen tiempos de respuesta similares, aunque el número de evaluaciones de la *función de inclusión* realizadas por el método P_1P_2A es en promedio menor. Esto puede ser debido al bajo número de evaluaciones realizadas y al coste adicional que supone el cálculo del parámetro $p\overline{f^*}(X)$, que también se calcula mediante Aritmética de Intervalos, para evitar los errores de redondeo de la Aritmética Computacional. Para $\epsilon = 10^{-3}$ las diferencias entre el número de evaluaciones de la *función de inclusión* realizadas por los métodos P_1P_2A y 1DA son mayores, siendo el método P_1P_2A el segundo mejor, después del P_1P_2. La mejoras para los métodos P_1P_2A y P_1P_2 están entre un 36 y un 59% y entre un 40 y un 60%, respectivamente.

   Para $\epsilon = 10^{-3}$, el método 3DA obtiene un menor tiempo de ejecución que el método 1D, aunque el número de evaluaciones realizado es superior. Esto puede ser debido a que el tamaño máximo del árbol de trabajo en el método 3DA es menos de la mitad que el requerido por el método 1D, por lo que su gestión requiere menos tiempo.

4. El número máximo de nodos en el árbol de trabajo (AM) refleja los requerimientos de memoria de las distintas variantes del algoritmo. La principal pregunta para el usuario es si un problema puede resolverse con una capacidad de memoria dada. Aunque se puede realizar intercambio con memoria principal, su precio, en tiempo de CPU, es muy alto en muchos casos. Para $\epsilon = 10^{-1}$, los valores obtenidos para los métodos P_1P_2A y P_1P_2 son peores que para el 1D, pero las diferencias son pequeñas. Las nuevas estrategias de división adaptativa decrementan los requerimientos de memoria para $\epsilon = 10^{-2}$ y $\epsilon = 10^{-3}$. Las mejoras obtenidas con los métodos P_1P_2A y P_1P_2 están entre un 20 y un 63% y un 24 y un 54%, respectivamente.

En la Tabla 3.4 no se han incorporado los valores medios de las cajas en los árboles finales (AF) porque reflejan más las características especiales de cada problema y la eficiencia de las distintas variantes del algoritmo se ven poco afectadas por este valor.

El valor actualizado del límite superior del mínimo global $\overline{f^*}$, está siempre disponible y su evaluación es poco costosa. Sin embargo, en algunos casos, el usuario conoce el valor exacto del mínimo global de la función, por ejemplo, debido a un conocimiento previo de la estructura del problema. Esta información puede mejorar la eficiencia del algoritmo no sólo porque el *Test de Corte* sea más efectivo, sino también, porque las decisiones acerca del nivel de división a realizar son más precisas.

Tabla 3.5: Resumen de los resultados numéricos para los distintos tipos de división, conociendo el mínimo global de la función ($\overline{f^*} = f^*$). Los valores medios de las distintas ejecuciones se muestran en las columnas (Med.). También se muestran los valores relativos de los demás métodos de división, comparados con los obtenidos con el tipo de división 1D.

		CPU		EF		Ef		AM	
10^{-1}	$1D$	0.1124		1241		620		146	
	1DA	0.0709	63%	1042	84%	243	39%	109	75%
	2DA	0.1559	139%	2286	184%	220	36%	150	103%
	3DA	0.1610	143%	2543	205%	127	21%	97	67%
	P_1P_2A	0.0713	63%	1028	83%	183	30%	90	62%
	P_1P_2	0.0639	57%	867	70%	188	30%	92	63%
10^{-2}	$1D$	0.6190		10466		5232		964	
	1DA	0.3434	55%	7939	76%	1776	34%	739	77%
	2DA	0.6723	109%	16713	160%	1383	26%	763	79%
	3DA	0.9363	151%	26193	250%	1196	23%	723	75%
	P_1P_2A	0.2966	48%	6904	66%	924	18%	472	49%
	P_1P_2	0.2796	45%	6560	63%	862	16%	423	44%
10^{-3}	$1D$	8.1406		104571		52285		9423	
	1DA	3.9591	49%	75727	72%	13568	26%	7485	79%
	2DA	12.2567	151%	254747	244%	13244	25%	9668	103%
	3DA	7.0474	87%	156991	150%	5679	11%	4087	43%
	P_1P_2A	3.2781	40%	66981	64%	6918	13%	3343	35%
	P_1P_2	3.2124	39%	63280	61%	7501	14%	4159	44%

Todo el experimento se ha repetido para el caso en el que se conoce de antemano el valor de f^*. Los resultados de este experimento se muestran en la Tabla 3.5. Comparando estos resultados con los obtenidos en la Tabla 3.4, se puede observar la mejora prevista para los métodos estáticos (1D a 3DA) y dinámicos (P_1P_2A y P_1P_2). Todos los indicadores de la eficiencia son mejores que los de la Tabla 3.4 en términos absolutos. Las tendencia de los valores relativos respecto al método 1D es básicamente la misma, pero las diferencias en ambas direcciones son más pequeñas. De nuevo, la división adaptativa P_1P_2 es la mejor, seguida del método P_1P_2A. Las mejoras son mayores cuando se requieren resultados más precisos.

Haciendo un resumen general de los resultados de los experimentos numéricos, puede concluirse, que la estrategia de división adaptativa mejora sustancialmente los requerimientos de memoria y la complejidad computacional del Algoritmo 3.1.1. Siempre que sea posible, debería usarse el valor del mínimo global f^* para decidir el nivel de división a realizar, pero también se puede usar con éxito una aproximación de f^*, incluso la obtenida mediante el *Test del Punto Medio*. Si se pueden optimizar los valores de P_1 y P_2, entonces el método P_1P_2 es la mejor opción. Si esto no es posible, se pueden usar los valores obtenidos para la versión P_1P_2A del algoritmo como buenas elecciones.

Al igual que los estudios acerca de la selección de la mejor coordenada para ser dividida [8, 7, 35, 158], creemos que nuestros resultados sobre la mejora de la eficiencia del Algoritmo básico de Optimización Global basado en intervalos, se mantendrá válida para un rango más amplio de estos algoritmos, donde se usen conjuntos diferentes de dispositivos aceleradores y otros cambios algorítmicos como los discutidos en [7, 34, 35, 158, 88].

3.4 Criterio de Eliminación Heurístico en Algoritmos de Optimización Global basados en Intervalos

En la sección anterior se vio cómo se puede disminuir el número de evaluaciones necesarias para resolver problemas de Optimización global basados en intervalos cambiando la *Regla de División* utilizada, pero siempre manteniendo la rigurosidad de la soluciones obtenidas, es decir, se mantiene la propiedad de que la convergencia a los mínimos globales está garantizada.

En esta sección se propone una modificación del Algoritmo 3.1.1 basada en relajar la *Regla de Eliminación* de forma que el número de cajas eliminadas sea mayor, decrementándose así el número de cajas procesadas. Estos nuevos criterios de eliminación están motivados porque existen muchos problemas para los que los algoritmos de Optimización Global basados en intervalos tienen unos requerimientos de memoria muy grandes. Esto ocurre cuando la precisión requerida y/o la dimensión del problema son muy grandes, especialmente si no se usa información sobre las derivadas de la función objetivo. Para evitar estos problemas de memoria, Madsen y Zerchaninov presentaron una versión estocástica y por lo tanto heurística, de un algoritmo de Optimización Global con Ramificación y Acotación, pero los problemas a resolver debían ser diferenciables [112].

Cuando los requerimientos de memoria del Algoritmo 3.1.1 crecen demasiado y se decide eliminar algunas de las subcajas que se encuentran en el árbol de trabajo, con la consiguiente pérdida de rigurosidad, se plantea la siguiente cuestión: ¿cuál sería el mejor criterio para elegir dichas cajas?.

El criterio tradicional de selección está basado en $\underline{F}(X)$, porque las cajas con menor $\underline{F}(X)$ tienen más probabilidad de contener un mínimo global. Esto sería completamente cierto si $F(X) = f(X)$. Pero esto no es siempre cierto debido a las sobreestimaciones de $f(X)$ obtenidas en $F(X)$. Estas sobreestimaciones son proporcionales a $w(X)$, debido a la propiedad de α-convergencia de la Aritmética de Intervalos en la obtención de $F(X)$. Por estos motivos, una caja puede ser seleccionada por ser más grande que las demás, en vez de porque contenga realmente un mínimo global. Por lo tanto, no se puede usar un criterio de eliminación basado en rechazar aquellas cajas con un mayor valor de $\underline{F}(X)$.

3.4.1 El parámetro $p\overline{f^*}(X)$ como criterio de eliminación

A partir de los experimentos realizados en las Figuras 3.2 y 3.3, se puede observar que las cajas generadas a partir de una caja que contiene un mínimo global pueden clasificarse

en dos categorías: las que siguen conteniendo un mínimo global y las que no lo contienen. Para las últimas, el valor de $p\overline{f^*}$ decrece conforme decrece $w(X)$ y aumenta la distancia al mínimo global. Este fenómeno también ocurre para las cajas que se encuentran fuera de la región de atracción a un mínimo global.

En esta sección se estudiará el comportamiento de los valores del parámetro $p\overline{f^*}$ durante la ejecución del Algoritmo 3.1.1 y su uso, para determinar qué cajas son las mejores candidatas para ser eliminadas [16, 18].

En los siguientes experimentos, se ha usado un valor de $\overline{f^*} = f^*$ y una *Regla de División* basada en realizar una bisección de todas las coordenadas a la vez (1DA), ya que en la Sección 3.3 se mostró en la mayoría de los casos como el mejor método de división estático para el conjunto de problemas de prueba. En estos experimentos no se usará el método de división adaptativo, debido a que se pretende que los experimentos estén afectados por el menor número de parámetros posible.

En la Figura 3.8 se muestran los valores de $p\overline{f^*}(X)$ generados secuencialmente durante la ejecución del Algoritmo 3.1.1 para el problema Six-Hump-Camel-Back, con $\epsilon = 10^{-2}$ y $\epsilon = 10^{-4}$. Los valores de $p\overline{f^*}(X^i) < 0$ no se han dibujado, ya que las cajas X^i han sido eliminadas, bien por el *Test del Punto Medio*, o bien por el *Test de Corte*. En ambas gráficas de la Figura 3.8, las cajas se han clasificado en dos clases, dependiendo de si el valor de $p\overline{f^*}$ de su caja madre era mayor o menor que su propio valor de $p\overline{f^*}$. En las gráficas se muestran también los valores de $p\overline{f^*}$, para las cajas que contenían un mínimo global durante la ejecución del algoritmo.

En la Figura 3.8 puede observarse que en las primeras iteraciones del algoritmo, i.e., cuando las cajas procesadas son grandes y los mínimos globales no están todavía bien definidos, aparece una nube irregular de valores de $p\overline{f^*}$. Por otro lado, en las últimas iteraciones, cuando los mínimos globales están más definidos, aparecen formas regulares, como bandas, de los valores de $p\overline{f^*}$. También puede observarse que durante la ejecución del algoritmo, las cajas que contienen los mínimos globales tienen los valores de $p\overline{f^*}$ más altos.

En la Figura 3.9 se muestran varios ejemplos de los posibles comportamientos de $p\overline{f^*}$, para una caja y sus descendientes, tal como aparecen en nuestros experimentos. Las cajas que contienen un mínimo global se han representado mediante un círculo. La gráfica superior izquierda muestra los valores de $p\overline{f^*}$ de una caja y de sus descendientes en la primera etapa de ejecución del algoritmo. En esta etapa los valores de $p\overline{f^*}$ tienen un comportamiento irregular. La primera caja que contiene un mínimo global de la gráfica superior derecha es la última que contenía un mínimo global en la gráfica superior izquierda. Ahora se observa un comportamiento más regular de los valores de $p\overline{f^*}$ para sus descendientes. Solamente las cajas con los mayores valores de $p\overline{f^*}$ contienen un punto minimizador. En las gráficas inferiores se muestran algunas de las cajas generadas en las últimas iteraciones del algoritmo. En la gráfica inferior izquierda, la primera caja contiene un mínimo global y sólo sus hijas y nietas que tienen los valores de $p\overline{f^*}$ más altos siguen conteniendo un mínimo global. Obsérvese también que las diferencias entre los valores de $p\overline{f^*}$ son muy pequeñas (< 0.01). En la gráfica inferior derecha, la caja inicial no contiene ningún mínimo

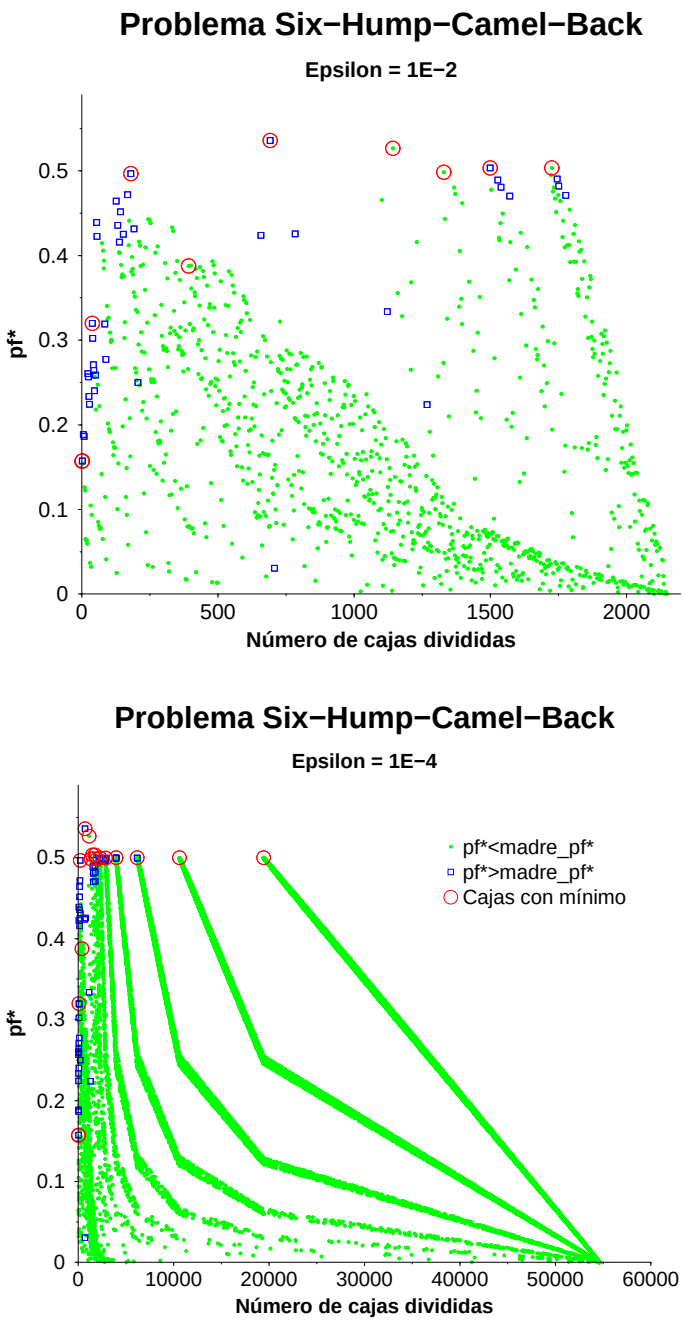


Figura 3.8: Valores de $\overline{pf^*}(X)$ durante la ejecución del Algoritmo 3.1.1, con una *Regla de División* 1DA y $\overline{f^*} = f^*$, para el problema Six-Hump-Camel-Back.

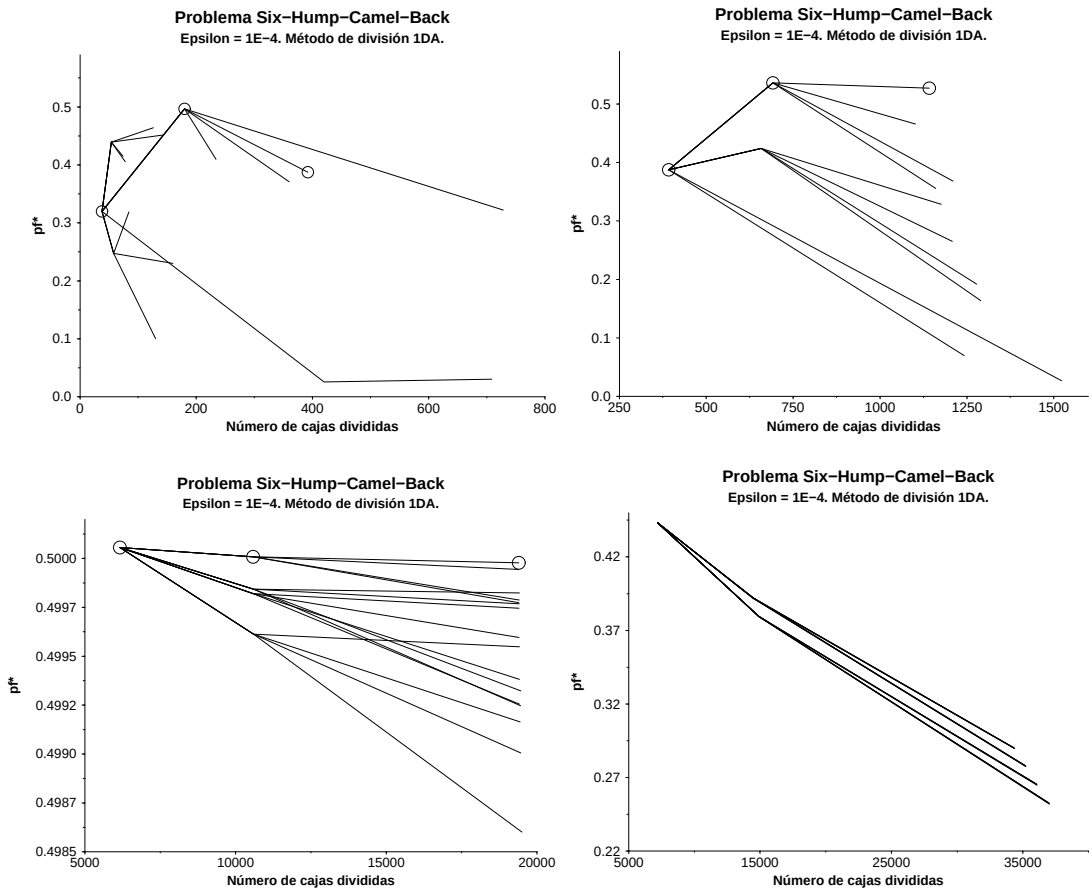


Figura 3.9: Valores de $p\overline{f^*}(X)$ de las descendientes (hijas y nietas) de algunas cajas en diferentes fases de ejecución del Algoritmo 3.1.1 para el problemas Six-Hump-Camel-Back con $\epsilon = 10^{-4}$.

global, decrementándose significativamente los valores de $p\overline{f^*}$ para las cajas hijas y nietas. Experimentos similares realizados sobre los distintos problemas de prueba han mostrado los mismos resultados.

A partir de los experimentos realizados se pueden distinguir tres etapas en la ejecución del algoritmo, a las que llamaremos inicial, intermedia y final.

La etapa final se caracteriza por un comportamiento regular del parámetro $p\overline{f^*}$; las cajas procesadas están en la región de atracción de un mínimo global y por lo tanto los respectivos valores de $p\overline{f^*}(X)$ dependerán de la cercanía de X a un mínimo global y del tamaño de X. Basándonos en los experimentos anteriores, es claro que en esta etapa, si existe una caja X con un decremento en el valor de $p\overline{f^*}(X)$, por encima de un umbral, con respecto al valor de $p\overline{f^*}(X')$ de su caja madre, entonces la caja X es una buena candidata para ser eliminada.

Algoritmo 3.4.1 : Algoritmo básico de Optimización Global basado en intervalos.

```
1  funct OGBásico(S, f, F, ε, T, Q)
2    f̄* = F̄(S)                                   Límite superior de f*
3    T := {S}                                     Árbol de trabajo
4    Q := {∅}                                     Árbol Final
5    while (T ≠ {∅})
6          X := SelecionaCaja(T)                  F̲(X) = min{F̲(X^i), ∀X^i ∈ T}
7          T := T − {X};
8          f̄* := min{f̄*, F̄(m(X))}                 Actualización de f̄*
9          if (f̄* = F̄(m(X)))                      Se ha mejorado el valor de f̄*
10           T := TestdeCorte(T, f̄*)              ∀X^i ∈ T, f̄* < F̲(X^i) ⇒ T := T − X^i
11           Q := TestdeCorte(Q, f̄*)
12         Divide(X, X^1, ..., X^s)
13         for i := 1 to s
14           if (f̄* < F̲(X^i) next;                Test del Punto Medio
15           InsertaCajaArbol(X, T, Q, ε)
16   return Q
```

En la etapa intermedia las cajas sólo pueden eliminarse si han habido dos decrementos consecutivos del valor de $p\overline{f^*}$, es decir, $p\overline{f^*}(X) < p\overline{f^*}(X') < p\overline{f^*}(X'')$, donde X, X' y X'' son la caja actual, su caja madre y su caja abuela, respectivamente.

La etapa inicial, si existe, se caracteriza por un decremento el valor de $p\overline{f^*}$ desde una caja madre a su hija. Esta etapa termina cuando aparece un incremento en el valor de $p\overline{f^*}$.

Nuestra propuesta de algoritmo, teniendo en cuenta todas las consideraciones anteriores, establece un criterio heurístico de eliminación basado en las etapas inicial e intermedia. La etapa final no se tiene en cuenta, ya que los requerimientos acerca del valor de $p\overline{f^*}$ están incluidos en los de la etapa intermedia y se han usado estos últimos para las etapas intermedia y final. Si a partir de la etapa inicial se producen dos decrementos consecutivos en los valores de $p\overline{f^*}$ y el decremento total es mayor que un umbral, que denotaremos por Dc, entonces la caja actual se elimina. El algoritmo básico de Optimización Global basado en intervalos, mostrado en el Algoritmo 3.1.1, puede reescribirse cambiando las líneas 15 y 16 por una llamada al procedimiento $InsertaCajaArbol(X, T, Q, \epsilon)$, tal como se muestra en el Algoritmo 3.4.1.

En el procedimiento $InsertaCajaArbol$, que se muestra en el Algoritmo 3.4.2, la caja actual no se inserta en el árbol de trabajo o en el árbol final, si cumple el nuevo criterio de eliminación heurístico.

Los dos siguientes teoremas caracterizan las propiedades de convergencia del Algoritmo 3.4.1-3.4.2:

Algoritmo 3.4.2 : InsertaCajaArbol con el nuevo criterio de eliminación heurístico.

```
1  proc InsertaCajaArbol(X, T, Q, ε)
2    var static Comienzo := Falso
3      if ( Comienzo = Falso ∧ pf*(X) > pf*(X') )
4        then Comienzo := Cierto
5      if ( Comienzo = Cierto ∧ pf*(X) < pf*(X')
6          ∧ pf*(X') < pf*(X'')
7          ∧ pf*(X'') − pf*(X) > Dc )
8        then return                          Nuevo criterio de eliminación
9        if (w(X) < ε)
10         then Q := Q + {X};                 Almacenar X en Q
11          else T := T + {X};                Almacenar X en T
12   end
```

Teorema 3.4.1 *Para cada problema de Optimización Global (3.1) existe un número infinito de funciones de inclusión que aseguran la convergencia del Algoritmo 3.4.1-3.4.2 al conjunto de puntos minimizadores globales cuando se conoce f^*.*

Demostración.

Considérese la clase de funciones de inclusión:

$$F(X) = [\underline{f}(X) - a\,w(f(X)), \overline{f}(X) + b\,w(f(X))] \tag{3.10}$$

donde $a, b \geq 0$ son unas constantes dadas y $\underline{f}(X)$ es el mínimo del rango de $f(x)$ en X. $F(X)$ es una *función de inclusión* de $f(x)$ puesto que $x \in X$ implica que $f(x) \in F(X)$. Para una función continua $f(x)$, la *función de inclusión* así definida, tendrá la propiedad de convergencia cuadrática. El caso ideal en el que $a = b = 0$ no es usual, sin embargo puede ocurrir que $a = 0$, incluso en las implementaciones realizadas en un computador donde la representación de datos es finita; por ejemplo, en los casos en los que la función objetivo tiene la forma $f(x) = (g(x))^2 + c$, siendo c una constante.

Considérese ahora un intervalo X en la región de búsqueda S que contiene un punto minimizador global x^*. Como $\overline{f^*} = f^*$, el parámetro $p\overline{f^*}(X)$ estará definido por:

$$p\overline{f^*}(X) = \frac{a}{1 + a + b} \tag{3.11}$$

Entonces, el valor de $p\overline{f^*}(X)$ es independiente de las características del intervalo X para la secuencia de cajas que convergen al mínimo global, permaneciendo el valor de $p\overline{f^*}(X)$ constante, con lo que las nuevas ideas de eliminación heurística no pueden aplicarse.

La secuencia de intervalos seleccionados para su proceso puede incluir intervalos que no contengan puntos minimizadores globales. Tales casos pueden ocurrir cuando una subcaja es relativamente grande y así, las sobreestimaciones del límite inferior de $f(X)$ obtenidas

en $\underline{F}(X)$ (cuando $a > 0$) pueden causar que esta caja tenga el menor valor de $\underline{F}(X)$, de todas las cajas almacenadas en el árbol de trabajo. No obstante, la secuencia de cajas seleccionadas no puede incluir una subsecuencia que converja a un punto $\hat{x}$ con $f(\hat{x}) > f^*$, puesto que para esa secuencia de cajas, $w(X^i)$ y también $w(F(X^i))$ convergen a cero, por lo que la relación $\underline{F}(X^i) \leq f^*$ sólo se puede mantener para un número finito de elementos de la secuencia.

Para el caso $a = 0$, todas las cajas activas son, exclusivamente, aquellas que contienen un punto minimizador global, puesto que sólo para esas cajas $\underline{F}(X)$ puede alcanzar el valor del límite inferior mínimo de f^*. ■

De acuerdo con el Teorema 3.4.1 existe una clase bastante grande de funciones de inclusión que aseguran la convergencia del Algoritmo 3.4.1-3.4.2. El siguiente teorema establece los casos en los que el Algoritmo 3.4.1-3.4.2 no converge a los puntos minimizadores globales.

Teorema 3.4.2 *Considérese el problema de Optimización Global (3.1), con un único mínimo global x^*. Entonces existe una función de inclusión isótona y α-convergente $F(X)$ de $f(x)$ tal que el Algoritmo 3.4.1-3.4.2 no converge a x^*.*

Demostración.

Considérese un problema que cumple las condiciones establecidas en el teorema y las cuatro primeras cajas de la secuencia de cajas seleccionadas: X^1, X^2, X^3 y X^4, que converge a x^*. Se asume que la *función de inclusión* original $G(X)$ es isótona y α-convergente. Tales *funciones de inclusión* obviamente existen y se pueden generar, por ejemplo, mediante la extensión natural a intervalos [153].

El valor de $p\overline{f^*}(X)$ puede establecerse a cualquier valor entre cero y uno decrementando el límite inferior de la *función de inclusión* o incrementando el límite superior. Si la *función de inclusión* se altera sólo para las cuatro primeras cajas seleccionadas, es decir, $F(X) = G(X)$ para las demás cajas, entonces la propiedad de α-convergencia se mantiene. De forma similar se puede mantener la propiedad de isotonicidad si se establecen los nuevos valores de $F(X^i)$ de forma que $F(X^i) \supseteq F(X^{i+1})$, $i = 1, 2, 3$ y 4. Si se establece $F(X)$ de forma que $p\overline{f^*}(X^1) > p\overline{f^*}(X^2)$ y que $p\overline{f^*}(X^4) < p\overline{f^*}(X^3) < p\overline{f^*}(X^2)$ y $p\overline{f^*}(X^2) - p\overline{f^*}(X^4) > Dc$, entonces la caja X^4 sería eliminada y de esta forma el Algoritmo 3.4.1-3.4.2 no podría converger al punto $x^* \in X^4$. Como para estos cuatro intervalos seleccionados se cumple que $\underline{F}(X^i) \leq \underline{G}(X^i)$, los intervalos serían seleccionados en el mismo orden que si se hubiera usado $G(X)$. ■

Aunque el Teorema 3.4.2 investiga sólo el caso en el que existe un solo mínimo global en la región de búsqueda, se puede usar el mismo razonamiento para problemas con un número grande (finito) de puntos minimizadores.

La demostración del Teorema 3.4.2 indica también, que aunque puede haber problemas reales en los que el Algoritmo 3.4.1-3.4.2 no converge a lo mínimos globales, este comportamiento requiere una secuencia de valores de la *función de inclusión* en las cajas

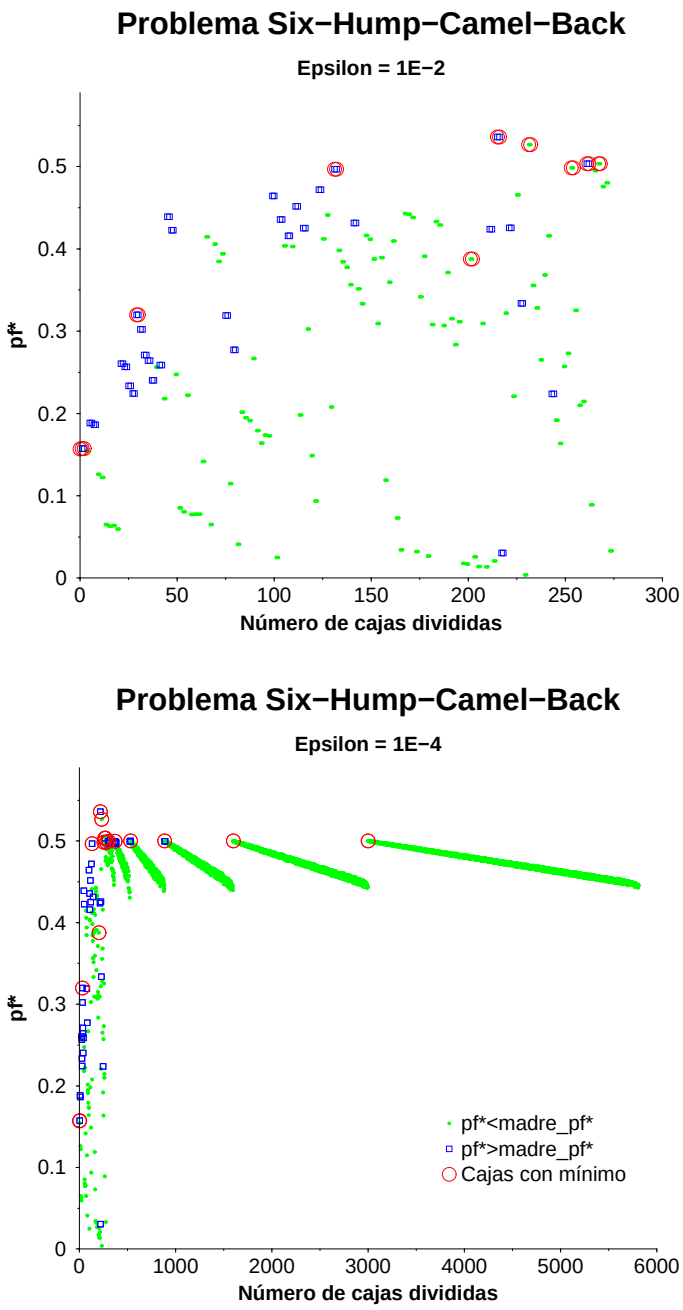


Figura 3.10: Valores de $\underline{pf^*}(X)$ durante la ejecución del Algoritmo 3.4.1-3.4.2, con una *Regla de División* 1DA, $\overline{f^*} = f^*$ y $Dc = 0.04$, para el problema Six-Hump-Camel-Back.

que contienen el mínimo global, que es raro que ocurra para valores relativamente grandes de Dc y más, cuanto mayor es el valor de Dc. Nótese que para un valor de $Dc = 1$ el Algoritmo 3.4.1-3.4.2 sería equivalente al Algoritmo (riguroso) 3.1.1.

La Figura 3.10 muestra el comportamiento del Algoritmo 3.4.1-3.4.2 con un valor de $Dc = 0.04$, para el problema de prueba Six-Hump-Camel-Back. Comparando las Figuras 3.8 y 3.10, puede observarse que el coste computacional (número de evaluaciones de la *función de inclusión*) del Algoritmo 3.4.1-3.4.2 es, aproximadamente, un 90% menor que el del Algoritmo 3.1.1.

3.4.2 Resultados numéricos

Las pruebas numéricas se han desarrollado sobre el mismo conjunto de problemas de prueba que en la Sección 3.3.1 y el mismo entorno de trabajo. En las Tablas 3.6 y 3.7 se ha añadido, a la notación usada en las Tablas 3.1 a 3.3, la cabecera NCM, que indica el número de cajas con algún punto minimizador global durante la ejecución del algoritmo. Este valor se ha usado para chequear si con los nuevos criterios de eliminación heurísticos se elimina alguna caja que contiene un punto minimizador.

En la Tabla 3.6 se muestran los resultados rigurosos del Algoritmo 3.1.1. La propiedad más importante a destacar es, que el número de cajas que contienen puntos minimizadores globales durante la ejecución del algoritmo es muy pequeño, comparado con el número de cajas evaluadas (EX) y con el número de cajas extraídas del árbol de trabajo (Ex). Estas diferencias crecen con la precisión requerida. Los valores de NCM son un límite inferior de EX y lo que se busca son algoritmos en los que el valor de EX se acerque lo más posible a los valores de NCM, sin eliminar cajas que contengan el mínimo global.

Para el problema Hartman-3 (véase el Apéndice A), el Algoritmo 3.1.1 se ejecutó sólo con una precisión de $\epsilon = 5 \cdot 10^{-4}$, porque para $\epsilon = 10^{-4}$ el computador se quedaba sin memoria disponible. Ésta es también la razón de porqué el conjunto de funciones de prueba tienen una baja dimensionalidad.

En la Tabla 3.7 se muestran los resultados obtenidos mediante el uso del nuevo criterio de eliminación heurístico. Comparando las Tablas 3.6 y 3.7, puede observarse que en general, todos los valores han mejorado. Solamente para los casos del problema S5 con $\epsilon = 10^{-1}$ y $\epsilon = 10^{-2}$, y los problemas S7 y S10 con $\epsilon = 10^{-1}$, todos los valores son iguales y para S5 y S10 con $\epsilon = 10^{-2}$, todos los valores son iguales, excepto para el número de cajas en el árbol final (AF), que es menor con el nuevo algoritmo. Se puede afirmar que la eficiencia del nuevo algoritmo es un 80% mejor que la del Algoritmo básico en problemas computacionalmente costosos, obteniéndose de esta forma mejores resultados para unos recursos computacionales fijos.

En la Tabla 3.8 se muestran, abreviados como Med.1, la media del tiempo de CPU, del número de evaluaciones de $F(X)$ (EX) y de $F(m(X))$ (Ex) de los datos presentados en la Tabla 3.6. También se muestran los valores relativos (abreviados como Rel.Med.2 (=Med.2/Med.1) de la Tabla 3.7, comparados con los de la Tabla 3.6. Puede observarse que la mejora crece con la precisión y que para la mayor precisión usada, $\epsilon = 10^{-4}$, la

Tabla 3.6: Resultados numéricos para el Algoritmo 3.1.1 con un método de división 1DA y f^* conocido.

Probl.	ϵ	CPU	EX	Ex	AF	AM	NCM
C	10^{-1}	0.060	2589	647	528	280	13
	10^{-2}	0.190	8597	2149	424	856	19
	10^{-3}	0.790	33413	8353	6614	4322	27
	10^{-4}	5.507	218477	54619	52876	34988	33
GP	10^{-1}	0.053	2037	509	678	198	19
	10^{-2}	0.920	32145	8036	12100	3069	31
	10^{-3}	4.300	169201	42300	9512	14535	43
	10^{-4}	19.487	689085	172271	135096	90773	59
H3	10^{-1}	0.030	465	58	99	29	4
	10^{-2}	0.560	10233	1279	2505	924	7
	10^{-3}	15.78	283409	35426	71892	28048	10
	$5 \cdot 10^{-4}$	48.267	858545	107318	202894	85812	11
L3	10^{-1}	0.330	1865	466	68	254	59
	10^{-2}	0.657	3633	908	501	291	86
	10^{-3}	6.047	32765	8191	7692	5009	122
	10^{-4}	46.06	247441	61860	61280	40448	149
S5	10^{-1}	0.003	113	7	1	1	7
	10^{-2}	0.010	161	10	1	1	10
	10^{-3}	0.037	705	44	45	27	14
	10^{-4}	0.453	9377	586	1100	419	17
S7	10^{-1}	0.010	113	7	3	1	7
	10^{-2}	0.030	385	24	9	15	10
	10^{-3}	0.290	4609	288	480	200	14
	10^{-4}	7.980	125281	7830	21293	6304	17
S10	10^{-1}	0.010	113	7	3	1	7
	10^{-2}	0.037	417	26	18	16	10
	10^{-3}	0.470	5985	374	601	252	14
	10^{-4}	12.513	157681	9855	26260	8014	17

mejora es del 88%. Todos los mínimos globales fueron localizados en todos los casos.

El nuevo criterio de eliminación heurístico se ha probado también en el problema EX2, tomado de Csendes y Ratz [35], que es una simplificación de la estimación paramétrica de un problema real [142] y cuya formulación se encuentra en el Apéndice A. Se ha introducido este problema para resaltar el uso del nuevo criterio de eliminación en problemas donde los dispositivos aceleradores más avanzados, como el método de Newton basado en intervalos, tienen muchas sobreestimaciones en la *función de inclusión* y por lo tanto su efectividad no es tan buena como la deseada. Debido a estas sobreestimaciones, se ha cambiado el criterio de terminación de $w(X) \leq \epsilon$ a $w(F(X)) \leq \epsilon$. Este criterio de terminación es el usado en

Tabla 3.7: Resultados numéricos para el Algoritmo 3.4.1-3.4.2 con un método de división 1DA, f^* conocido y $Dc = 0.04$.

Probl.	ϵ	CPU	EX	Ex	AF	AM	NCM
C	10^{-1}	0.023	1053	263	2	78	13
	10^{-2}	0.027	1101	275	22	78	19
	10^{-3}	0.090	3533	883	712	406	27
	10^{-4}	0.613	23253	5813	5646	3414	33
GP	10^{-1}	0.030	1045	261	304	93	19
	10^{-2}	0.100	3921	980	83	250	31
	10^{-3}	0.187	6781	1695	909	451	43
	10^{-4}	1.780	60321	15080	14265	8441	59
H3	10^{-1}	0.010	225	28	44	18	4
	10^{-2}	0.190	3345	418	903	271	7
	10^{-3}	1.817	33025	4128	3321	2475	10
	$5 \cdot 10^{-4}$	3.327	59593	7449	8289	4324	11
L3	10^{-1}	0.300	1709	427	61	251	59
	10^{-2}	0.557	3097	774	235	251	86
	10^{-3}	1.537	8409	2102	910	684	122
	10^{-4}	6.040	32433	8108	6654	4090	149
S5	10^{-1}	0.003	113	7	1	1	7
	10^{-2}	0.010	161	10	1	1	10
	10^{-3}	0.033	705	44	18	27	14
	10^{-4}	0.323	6689	418	545	321	17
S7	10^{-1}	0.010	113	7	3	1	7
	10^{-2}	0.030	385	24	5	15	10
	10^{-3}	0.257	4081	255	318	183	14
	10^{-4}	2.073	32737	2046	1255	1396	17
S10	10^{-1}	0.010	113	7	3	1	7
	10^{-2}	0.037	417	26	7	16	10
	10^{-3}	0.383	4913	307	357	216	14
	10^{-4}	2.680	33905	2119	1408	1411	17

[35] y permite obtener resultados más fiables cuando las sobreestimaciones de la *función de inclusión* son considerables, lo que incrementa también el número de evaluaciones de intervalos requeridas. Se ha introducido este problema para mostrar cómo el algoritmo también puede trabajar con otros métodos de división (1D, bisección de la coordenada mayor) y sin conocer de antemano el mínimo global f^*.

La Tabla 3.9 contiene los resultados numéricos obtenidos con el algoritmo básico, sin el nuevo criterio de eliminación. El problema para un valor de $\epsilon = 0.1$ no pudo ser resuelto debido a los enormes requerimientos de memoria. La ejecución del programa se paró después de más de una semana sin haber finalizado. Se usó cerca de 1Gb de memoria de intercambio, además de los 256 Mb de memoria RAM. Ésta es la razón obvia por la que

Tabla 3.8: Resumen de los resultados numéricos. Se muestran los valores medios (abreviados como Med.1) de los datos de la Tabla 3.6. También se muestran los valores relativos (abreviados como Rel.Med.2 = Med.2 / Med.1) de los valores medios de los datos en la Tabla 3.7 comparados con los de la Tabla 3.6.

	CPU		EX		Ex	
ϵ	Med.1	Rel.Med.2	Med.1	Rel.Med.2	Med.	Rel.Med.2
10^{-1}	0.0709	77.72%	1042	59.88%	243	58.85%
10^{-2}	0.3434	39.57%	7939	22.36%	1776	20.16%
10^{-3}	3.9591	15.53%	75727	11.59%	13568	9.91%
10^{-4}	20.0381	12.00%	329412	10.80%	59139	9.91%

Tabla 3.9: Resultados numéricos para el problema EX2 con el algoritmo básico de Optimización Global (Algoritmo 3.1.1, página 63), sin conocer f^*, usando como *Regla de División* una bisección simple (1D) y como *Regla de Terminación* $w(F(X)) \leq \epsilon$.

ϵ	CPU	EX	Ex	AF	AM	NCM	$[\underline{f^*}, \overline{f^*}]$
0.50	53.39	377767	188883	55029	29495	25	[0.08584908, 0.21462488]
0.10	-	-	-	-	-	-	-

Tabla 3.10: Resultados numéricos para el problema EX2 con el nuevo criterio heurístico de eliminación (Algoritmo 3.4.1-3.4.2), sin conocer f^*, usando como *Regla de División* una bisección simple (1D), como *Regla de Terminación* $w(F(X)) \leq \epsilon$ y $Dc = 0.004$.

ϵ	CPU	EX	Ex	AF	AM	NCM	$[\underline{f^*}, \overline{f^*}]$
0.50	11.57	83153	41576	6374	5915	25	[0.08584908, 0.21462488]
0.10	37.66	267701	133850	10193	23799	35	[0.17129792, 0.21273191]
0.05	58.53	415505	207752	9548	33013	40	[0.19055822, 0.21258066]
0.01	178.75	1250593	625296	133600	70992	63	[0.20757242, 0.21246471]

no se dan resultados para valores menores de ϵ.

La Tabla 3.10 muestra los resultados numéricos del nuevo algoritmo. Los resultados comparables, para $\epsilon = 0.5$, indican una mejora cercana al 80% en la mayoría de los índices. Un hecho, aún más importante, es que el algoritmo con el nuevo criterio de eliminación es capaz de resolver el problema para una precisión aceptable, lo que era imposible para el algoritmo básico. Aunque nuestro algoritmo es mucho más simple, los resultados son comparables, y para algunos índices mejores, que los obtenidos en [7, 35].

Los resultados numéricos de las Tablas 3.9 y 3.10 confirman que los resultados obtenidos

en los problemas estándar de prueba son válidos para problemas mucho más difíciles de resolver. Otro aspecto importante es que se ha obtenido una mejora cercana a un orden de magnitud en los indicadores de la eficiencia y que se mantuvo la rigurosidad del algoritmo básico: todas las cajas que contenían puntos minimizadores globales no fueron eliminadas por el nuevo algoritmo. Aunque todos los mínimos globales de los problemas usados fueron encontrados, esto no puede probarse para todos los casos posibles, de acuerdo con nuestros resultados teóricos. Otra técnica para mantener la rigurosidad del algoritmo puede consistir en almacenar las cajas eliminadas por el nuevo algoritmo en un fichero para su posterior chequeo. También debe tenerse en cuenta que el nuevo algoritmo puede aplicarse a problemas diferenciables y no diferenciables, donde los dispositivos aceleradores presentados en la Sección 1.3, de los que no se ha hecho uso, no pueden aplicarse.

3.5 Nuevos criterios de selección basados en $p\overline{f^*}(X)$

En las secciones anteriores se han observado las siguientes características de los algoritmos de Optimización Global basados en intervalos y en el uso del parámetro $p\overline{f^*}(X)$ para su mejora:

- La *Regla de Selección* basada en elegir para su procesamiento aquella caja X con un valor menor de $\underline{F}(X)$ del árbol de trabajo, está motivada principalmente por:
  - Parece razonable pensar que la caja X con un menor valor de $\underline{F}(X)$ tiene más probabilidad de contener un mínimo global, aunque como se comentó en la Sección 3.4 (página 84), esto no es siempre cierto. Bajo esta suposición, el aplicar el *Test del Punto Medio* sobre esta caja podría aportar una mejora del valor de $\overline{f^*}$. Con esta mejora se conseguiría, por un lado, la posible eliminación de algunas cajas gracias al *Test de Corte* y por otro lado, reducir el intervalo $[\underline{f^*}, \overline{f^*}]$ que contiene el mínimo global, f^*.
  - Al seleccionar y dividir la caja X con un menor valor de $\underline{F}(X)$ se consigue aumentar el valor del límite inferior de f^* ($\underline{f^*}$), si la *función de inclusión* es isótona.
- Debido a las sobreestimaciones de $f(X)$ obtenidas por $F(X)$, el número de cajas activas alrededor del mínimo global puede ser muy grande. Este número dependerá principalmente de las distintas combinaciones de: las sobreestimaciones obtenidas, del tamaño de las cajas activas y de la pendiente en las cercanías de los puntos minimizadores globales, para cajas pequeñas.

  De las consideraciones anteriores se puede deducir :
  - Es importante encontrar rápidamente el valor f^* o la mejor aproximación que se pueda obtener de él ($\overline{f^*}$). Obsérvese que si f^* es conocido, el número de cajas evaluadas es independiente de la *Regla de Selección* usada, cuando se quieren encontrar todos los puntos minimizadores globales.

  - No es menos importante el hecho de que se realicen muchas evaluaciones de la *función de inclusión* alrededor de los mínimos globales, ya que esto puede ocurrir incluso si se conoce f^*.

- El parámetro $p\overline{f^*}$ ha demostrado ser un buen estimador de la cercanía de una caja al mínimo global o de cuanta región de atracción a un mínimo contiene una determinada caja. Por lo tanto, parece razonable usar una *Regla de Selección* basada en el parámetro $p\overline{f^*}$. A continuación se estudia este tipo de *Regla de Selección*.

3.5.1 Mejor Primero basado en $p\overline{f^*}(X)$

En la Figura 3.11 se muestra la ejecución del algoritmo básico (Algoritmo 3.1.1), con una *Regla de División* basada en bisección de todas la coordenadas (1DA), una *Regla de Selección* basada en escoger primero la caja con el mayor valor de $p\overline{f^*}(X)$ y sin conocer el valor de f^*.

En la gráfica superior se muestra la ejecución del algoritmo para el problema Glodstein-Price. Para este ejemplo, el algoritmo tarda en seleccionar las cajas que contienen un mínimo global. Esto es debido a que, aunque el método de selección usado sea del tipo *Primero el Mejor* ($p\overline{f^*}$), este se comporta realmente como una búsqueda en profundidad (Sección 1.5.1, página 28), ya que el valor de $p\overline{f^*}$ crece porque se reduce el tamaño de las cajas que se encuentran cerca de, o contienen, un mínimo. La desventaja de las búsquedas en profundidad es que si se entra en una rama del árbol de búsqueda que no lleva a la solución óptima, se tarda mucho en salir de ella. Esto es lo que ocurre en el problema Goldstein-Price, ya que sólo tiene un mínimo global y en la división de toda la región de búsqueda (la primera división), el mínimo se encuentra compartido en el borde de dos de las cuatro cajas generadas, por lo que la región de atracción a ese mínimo se ha dividido entre esas dos cajas, decreciendo el valor de $p\overline{f^*}$. Esta bajada de $p\overline{f^*}$ permite que una de las cajas generadas que no contiene el mínimo global tenga el mayor valor de $p\overline{f^*}$, debido a la región de atracción hacia un mínimo local. La búsqueda continúa sobre la región del mínimo local mientras los valores de $p\overline{f^*}$ de las cajas generadas en esta región sean mayores que alguno de los valores de $p\overline{f^*}$ de las cajas que contienen el mínimo global. El principal problema de esta fase consiste en que se trabaja sólo sobre la región que contiene un mínimo local, siendo imposible obtener, mediante el *Test del Punto Medio*, un valor de $\overline{f^*}$ que permita eliminar cajas que contengan este mínimo local. Se obtendrá un valor de $\overline{f^*}$ menor que el del mínimo local, cuando el algoritmo se ejecute sobre la región de atracción del mínimo global. Llegado este momento, ya se habrán evaluado muchas cajas alrededor del mínimo local e incluso algunas de ellas pueden haber alcanzado la *Regla de Terminación*, por lo que habrá que eliminarlas del árbol final mediante el *Test de Corte*.

En la gráfica superior de la Figura 3.11 puede observarse que una vez que se sale de la región de atracción al mínimo local y se selecciona una caja que contiene el mínimo global, la siguiente secuencia de cajas seleccionadas converge al mínimo global, con crecimientos continuos en los valores de $p\overline{f^*}$. Una vez que se alcanza la *Regla de Terminación* para la

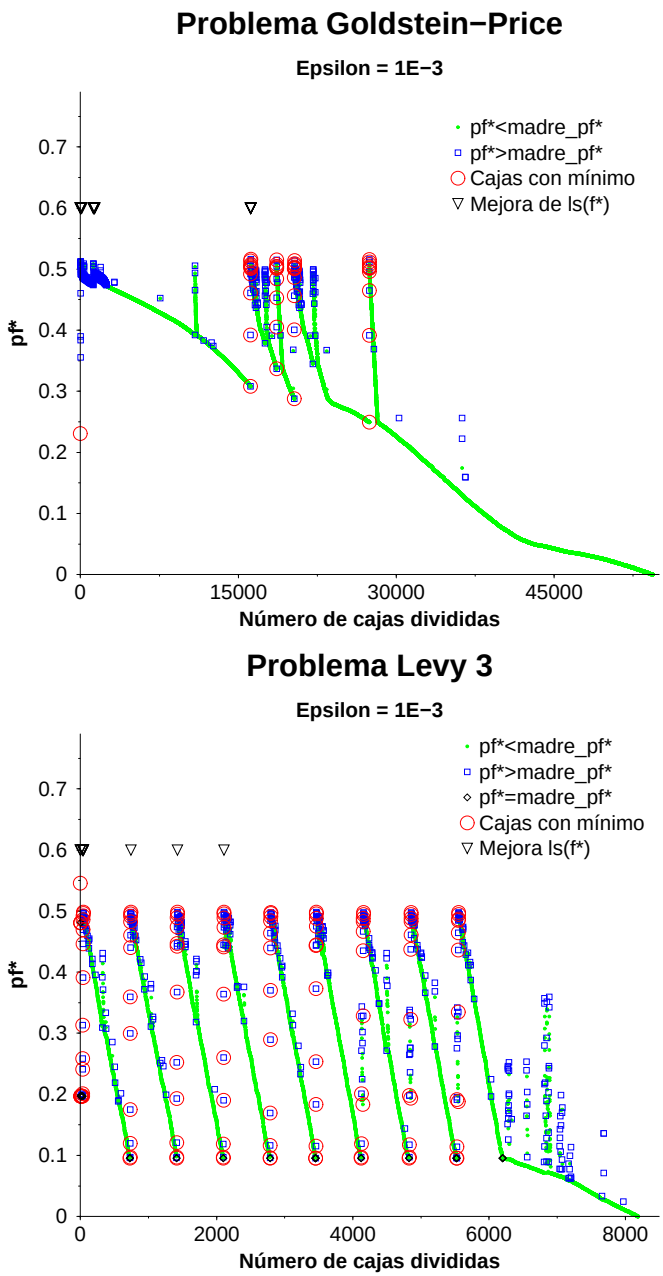


Figura 3.11: Valores de $\overline{pf^*}(X)$ durante la ejecución del Algoritmo 3.1.1, realizando bisección en todas las coordenadas (1DA), sin conocer f^* y seleccionando la caja X con mayor valor de $\overline{pf^*}(X)$ primero.

caja que contiene el mínimo global en esta secuencia, el algoritmo continúa trabajando sobre las cajas que se generaron en este proceso, hasta que una nueva caja que contenga el mínimo global tenga un mayor valor de $p\overline{f^*}$. Pueden observarse cuatro secuencias de cajas seleccionadas que convergen hacia el mínimo global. Esto es debido a que, aunque sólo existe un mínimo global, éste es compartido por cuatro cajas. También se observan varias secuencias de cajas seleccionadas con algún crecimiento de $p\overline{f^*}$ que no contienen un mínimo global, por lo que puede deducirse que convergen hacia mínimos locales.

Una vez que todas las posibles cajas que contienen un mínimo global han sido procesadas, hay que evaluar las cajas que restan y que no han podido ser eliminadas por el *Test de Corte*. Para la gran mayoría de estas cajas, el valor de $p\overline{f^*}$ decrece con su tamaño.

En la gráfica inferior de la Figura 3.11 se muestra la ejecución del algoritmo anterior para el problema Levy 3. En este caso, el algoritmo converge rápidamente hacia uno de los mínimos globales. Se ha introducido esta gráfica porque en ella puede verse claramente:

- La convergencia hacia los nueve mínimos globales.
- El trabajo que se realiza alrededor de un mínimo global, después de alcanzar la *Regla de Terminación* para la(s) caja(s) que contiene(n) el mínimo global.
- El trabajo final sobre las cajas restantes que no han sido eliminadas y que pueden contener algún mínimo local.

3.5.2 Modelo de selección híbrido basado en $p\overline{f^*}(X)$

Tal como se vio en la Sección 1.5.1 (página 29), el modelo de selección híbrido trata de obtener las ventajas del método de *Búsqueda en Profundidad* y de *Primero el Mejor*. Mediante el uso de una búsqueda en profundidad se pretende obtener rápidamente un mejor valor de $\overline{f^*}$. Para que esta búsqueda en profundidad tenga éxito, hay que usar dos buenos criterios de selección:

- El primero establece la caja inicial a partir de la cual se realizará una búsqueda en profundidad. Esta selección se realiza normalmente mediante una estrategia *Primero el Mejor*.
- Dentro de la búsqueda en profundidad hay que establecer cuál de las cajas generadas, a partir de la seleccionada, es la que se elige para continuar la búsqueda en profundidad.

Un fallo en la selección de la caja inicial en la que se aplicará una búsqueda en profundidad tendrá un coste que dependerá de la profundidad de la búsqueda y del nivel de división. Para problemas con pocas dimensiones, usando una bisección de sólo dos o tres coordenadas y con la precisión usada en los ejemplos ($\epsilon = 10^{-3}$), una búsqueda en profundidad tiene muy poco peso en el valor del número final de evaluaciones de intervalos. Para estos casos, dependiendo del valor de $\overline{f^*}$, puede que incluso no se alcance la profundidad establecida por la *Regla de Terminación*. Para problemas dimensionalmente complejos y

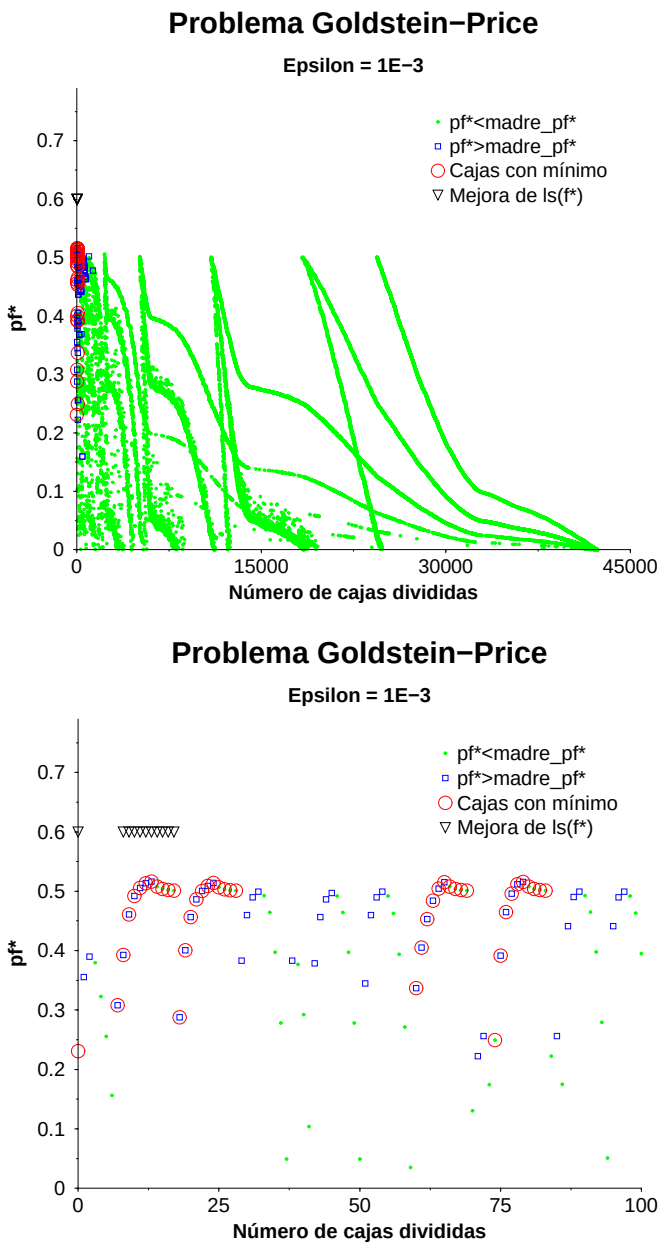


Figura 3.12: Valores de $p\overline{f^*}(X)$ durante la ejecución del Algoritmo 3.1.1, realizando bisección en todas las coordenadas (1DA), sin conocer f^* y usando un método de selección híbrido, basado en *Primero el Mejor* $w(X) \cdot p\overline{f^*}(X)$, seguido de una búsqueda en profundidad basada en $p\overline{f^*}(X)$. En la gráfica inferior se muestran las cien primeras iteraciones de la gráfica superior.

grandes requerimientos de precisión en las soluciones, el costo de una búsqueda en profundidad fallida debe tenerse en cuenta, principalmente cuando no se han encontrado anteriormente buenos valores de $\overline{f^*}$.

En la fase de selección *Primero el Mejor*, se ha elegido aquella caja X, con un valor mayor de $w(X)\cdot p\overline{f^*}(X)$. De esta forma se eligen cajas grandes, para comenzar la búsqueda en profundidad en niveles altos del árbol de búsqueda y de entre las cajas de mayor tamaño se eligen aquellas que tienen un mayor valor de $p\overline{f^*}(X)$. El criterio de selección en la fase de *Búsqueda en Profundidad* es más fácil de establecer porque las cajas candidatas tienen todas el mismo tamaño. De esta forma, la caja X seleccionada es aquella con un mayor valor de $p\overline{f^*}(X)$.

En la gráfica superior de la Figura 3.12 se muestra la ejecución del Algoritmo 3.1.1 para la función Goldstein-Price, con la nueva regla de selección híbrida, sin conocer f^* y realizando una bisección de todas las coordenadas de la caja seleccionada (1DA). Para este ejemplo, cuando se usa el nuevo criterio de selección, todas las cajas que contenían un mínimo global en cualquier momento de la evaluación del Algoritmo 3.1.1, son ahora evaluadas al principio del nuevo algoritmo, lo que permite obtener rápidamente el mejor valor de $\overline{f^*}$ que se obtendría con el algoritmo básico. Las ejecuciones realizadas sobre otros problemas de prueba han mostrado resultados similares.

En la gráfica inferior de la Figura 3.12 se muestran los valores de $\overline{f^*}$ de las cien primeras cajas seleccionadas. La primera búsqueda en profundidad realizada converge hacia un mínimo local, tal como ocurrió en la gráfica superior de la Figura 3.11, pero en este caso, el número de evaluaciones de intervalos realizado sobre la región de atracción al mínimo local es casi inapreciable, debido al nuevo criterio de selección híbrido. La siguiente búsqueda en profundidad se realiza sobre una de las cajas que contiene el mínimo global. Esta búsqueda en profundidad converge hacia el mínimo global, obteniendo rápidamente el mejor valor posible de $\overline{f^*}$. Debido a esto, no aparecen mejoras de $\overline{f^*}$ después de este punto en la ejecución del algoritmo. Las siguientes búsquedas en profundidad se realizan sobre: otra caja con el mínimo global, varias búsquedas sobre mínimos locales, otras dos búsquedas sobre cajas con mínimos globales, etc. Si se hubiera decidido parar el algoritmo cuando se llevan ejecutadas cien iteraciones, los resultados hubieran sido los correctos. El problema estriba en saber cuál es el momento apropiado en que el algoritmo debe parar, de forma que no se pierdan cajas que contengan el mínimo global, o si se pierden, que sea porque el mínimo global se compartía con otras cajas que no han sido eliminadas. El problema de decidir cuándo sería conveniente parar el algoritmo, si se usa una *Regla de Selección* híbrida, sigue siendo un problema abierto en el que hay que seguir investigando.

La ventaja del modelo híbrido frente al basado en *Primero el Mejor* $p\overline{f^*}(X)$, es que permite evaluar el *Test del Punto Medio* de una manera más o menos equitativa entre las subregiones de búsqueda. Así se obtiene una información más global de las evaluaciones puntuales de la *función de inclusión*, evitando una búsqueda intensiva sobre zonas de atracción a mínimos locales, tal como ocurría en la gráfica superior de la Figura 3.11. Además, no se pierde la ventaja de obtener un buen valor de $\overline{f^*}$, gracias a las búsquedas en profundidad realizadas.

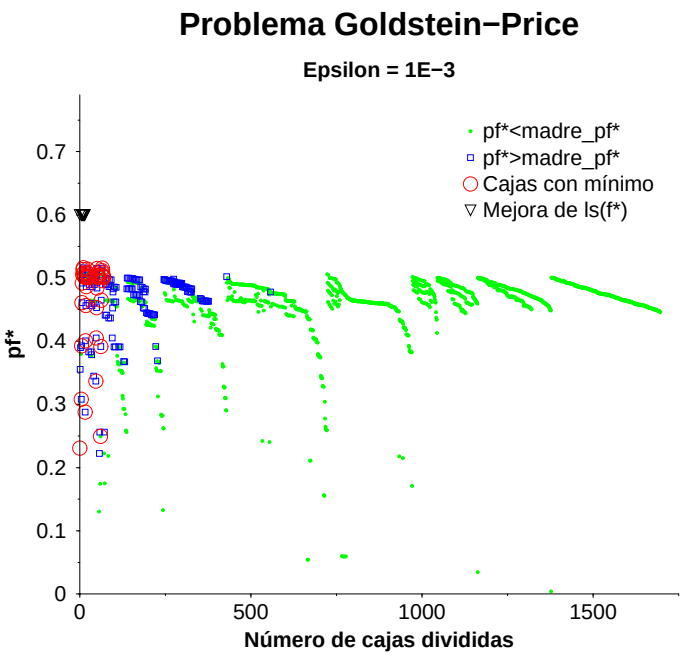


Figura 3.13: Valores de $p\overline{f^*}(X)$ durante la ejecución del Algoritmo 3.1.1, realizando una bisección de todas las coordenadas (1DA), sin conocer f^* y usando un método de selección híbrido basado en *Primero el Mejor* $w(X) \cdot p\overline{f^*}(X)$, seguido de una búsqueda en profundidad basada en $p\overline{f^*}(X)$ y un criterio de eliminación heurístico con $Dc = 0.04$.

Regla de Selección híbrida y el nuevo criterio de eliminación heurístico

Con la *Regla de Selección* híbrida se consigue obtener un buen límite superior de f^* rápidamente, pero se siguen haciendo muchas evaluaciones alrededor de los mínimos. Para reducir el número de iteraciones, se puede usar la *Regla de Eliminación* heurística (Sección 3.4.1), con la consiguiente pérdida de rigurosidad del algoritmo.

En la Figura 3.13 se muestran los valores de $p\overline{f^*}(X)$ cuando se añade el criterio de eliminación heurístico a la *Regla de Selección* híbrida. Ahora no sólo se obtiene rápidamente un buen límite superior de f^*, sino que también se reduce considerablemente el número de iteraciones del algoritmo.

Aunque el criterio de eliminación heurístico permite reducir el número de evaluaciones de intervalos, no es suficiente en la mayoría de los casos, cuando se requiere una gran precisión en las soluciones. Esto es debido a que el número de cajas evaluadas crece exponencialmente con la precisión requerida. Tanto en la Figura 3.13, como en la gráfica inferior de la Figura 3.10 (página 91), puede observarse que el número de cajas que no cumplen el criterio de eliminación heurístico se incrementa exponencialmente conforme se ejecuta el algoritmo. Por lo tanto, si se quieren resolver problemas con mucha precisión, hay que establecer un límite en el crecimiento del árbol de búsqueda.

3.6 Algoritmo de Optimización Global basado en Intervalos con Restricciones Computacionales

Tal como se comentó en la Sección 3.4, existen problemas que, debido a su dimensionalidad y/o a la precisión requerida en las soluciones, son impracticables para los computadores de uso común en la actualidad, si se intentan resolver con el Algoritmo 3.1.1. En esta sección se presenta un algoritmo que por un lado tiene en cuenta las limitaciones de almacenamiento de los computadores actuales y por otro lado hace uso de las ideas mostradas en las secciones anteriores, para intentar mantener la convergencia del algoritmo hacia los mínimos globales [13].

La idea principal consiste en establecer en cada iteración un número máximo de cajas que se pueden seleccionar para su división y así asegurar que el algoritmo termina dentro de los requerimientos de memoria establecidos. Para ello, cuando se rebasa este número, hay que utilizar algún tipo de criterio de eliminación heurístico, como el presentado en la Sección 3.4, con lo que se podría perder la rigurosidad en los resultados.

En este nuevo algoritmo se propone usar el parámetro $p\overline{f^*}$, para determinar qué cajas son las seleccionadas: aquellas cajas X^i con un mayor valor de $p\overline{f^*}(X^i)$ serán seleccionadas y divididas. Es importante que las cajas X^i tengan la misma forma y tamaño. De esta forma, los valores de $p\overline{f^*}(X^i)$ se calculan bajo las mismas condiciones y se realizan unas comparaciones más equitativas de los mismos.

Esto está en contraposición con los métodos de selección: *Primero el Mejor* $\underline{F}(X)$, *Primero el Mejor* $p\overline{f^*}(X)$ y el modelo híbrido, ya que este tipo de métodos generan cajas a distintos niveles del árbol de búsqueda y por lo tanto, de distinto tamaño y forma. En el nuevo algoritmo se aplicará una *Búsqueda en Anchura*. Aún así, el nuevo algoritmo usará las ideas subyacentes los métodos de selección: *Primero el Mejor* $p\overline{f^*}(X)$ y el modelo híbrido. Obsérvese que como todas las cajas que se pueden seleccionar para su división deben tener el mismo tamaño, sólo hay que tener en cuenta el valor de $p\overline{f^*}(X)$, mientras que el modelo de selección híbrido también había que tener en cuenta $w(X)$. Usando el modelo de selección híbrido se dividía la mejor caja y de las cajas generadas se continuaba sobre aquella que tenía un mayor valor de $p\overline{f^*}(X)$. La *Regla de Selección* del nuevo algoritmo puede verse también como un modelo híbrido, en el que en vez de continuar sólo sobre la caja más prometedora, en términos de $p\overline{f^*}(X)$, se continúa sobre un número fijo de ellas, que es determinado por el usuario. Para llevar a cabo esta estrategia, hay que distinguir entre el conjunto de cajas a dividir y el conjunto de cajas generadas en las distintas divisiones. De entre las cajas candidatas a ser divididas se selecciona un número que no sea mayor que un umbral establecido, basándose en *Primero el Mejor* $p\overline{f^*}(X)$ y se dividen. En la siguiente iteración las cajas generadas pasan a ser las candidatas a ser divididas. Este proceso se repite hasta que no existan cajas para dividir. El umbral o número máximo de cajas seleccionadas para su división, dependerá de los requerimientos de memoria del problema y del grado de convergencia hacia los mínimos globales que se quiera establecer. Es claro que con un número mayor de cajas a dividir, las evaluaciones del *Test del Punto*

Medio se realizan sobre una mayor región de búsqueda y se tiene una mayor probabilidad de convergencia.

Las cajas que se ha decidido apartar de la ejecución del algoritmo pueden tratarse de dos formas:

1. Almacenarlas temporalmente para procesarlas después de la ejecución normal del algoritmo. La ventaja de este método es que se asegura la rigurosidad de las soluciones y la desventaja es que se mantienen los problemas debidos a las limitaciones de memoria.
2. Eliminarlas directamente, con lo que se pierde la rigurosidad del método, a costa de acelerar la ejecución del algoritmo y cumplir los requerimientos de memoria impuestos. En este caso se supone que el número máximo establecido de cajas a dividir en cada iteración es suficiente para que esta búsqueda en anchura, con limitación del número de cajas en cada nivel del árbol de búsqueda, converja a la solución.

En esta tesis se ha optado por el segundo criterio. El algoritmo de Optimización Global basado en intervalos con restricciones computacionales se muestra en el Algoritmo 3.6.1, el cual ha sido desarrollado a partir del Algoritmo 3.1.1.

Los parámetros de entrada del Algoritmo 3.6.1 son los siguientes:

S	: Región de búsqueda.
F	: Extensión natural a intervalos de la función objetivo f.
ϵ	: Criterio de parada ($w(X) < \epsilon$).
T_0	: Árbol de trabajo.
T_1	: Árbol temporal.
Q	: Árbol final.
N_m	: Número máximo de cajas seleccionadas en cada iteración para su división.
$+/-$	: Denotan la introducción/extracción de cajas del árbol.

Las principales diferencias entre el Algoritmo 3.6.1 y el Algoritmo 3.1.1 son: que se usa un árbol temporal (T_1) para almacenar las cajas generadas a partir de las seleccionadas del árbol de trabajo (T_0) y que la regla para seleccionar cajas del árbol de trabajo está basada en *Primero el Mejor* $p\overline{f^*}(X)$. Aquellas cajas que tengan un valor igual de $p\overline{f^*}(X)$ se almacenan siguiendo un orden no decreciente del valor de $\overline{F}(X)$ y si tienen el mismo valor de $\overline{F}(X)$ en un orden decreciente con respecto a la edad de las cajas. Mediante el uso de un árbol temporal nos aseguramos que las cajas que se encuentran en el árbol temporal tienen el mismo tamaño y forma, ya que se generaron con la misma *Regla de División* a partir de cajas que se encontraban en el árbol de trabajo, que a su vez tenían el mismo tamaño y forma.

Al igual que en el Algoritmo 3.1.1, en el Algoritmo 3.6.1 se inicializa: el árbol de trabajo con la región de búsqueda, el árbol final con conjunto vacío y el límite superior del mínimo global con el límite superior de la evaluación de la *función de inclusión* sobre toda

Algoritmo 3.6.1 : Algoritmo de Optimización Global basado en Intervalos con Restricciones Computacionales.

```
1  funct MemLim(S, F, ε, T0, T1, Q, Nm)
2    f̄* = F̄(S)                                   Límite superior de f*
3    T0 := {S}                                    Árbol de trabajo
4    T1 := {∅}                                    Árbol temporal
5    Q := {∅}                                     Árbol Final
6    while (T0 ≠ {∅})
7          NSelec := 0                            Número de cajas seleccionadas de T0
8          while (T0 ≠ {∅} ∧ NSelec < Nm)
9                X := SelecionaCaja(T0)           pf̄*(X) = max{pf̄*(X^i), ∀X^i ∈ T0}
10               T0 := T0 − {X};
11               NSelec := NSelec + 1
12               f̄* := min{f̄*, F̄(m(X))}           Actualización de f̄*
13               if (f̄* = F̄(m(X)))               Se ha mejorado el valor de f̄*
14                  T0 := TestdeCorte(T0, f̄*)     ∀X^i ∈ T0, f̄* < F̲(X^i) ⇒ T0 := T0 − {X^i}
15                  T1 := TestdeCorte(T1, f̄*)
16                  Q := TestdeCorte(Q, f̄*)
17               Divide(X, X^1, ..., X^s)
18               for i := 1 to s
19                   if (f̄* < F̲(X^i) nextᵢ       Test del Punto Medio
20                   if (w(X^i) < ε) Q := Q + {X^i}   Almacenar X^i en Q
21                 else T1 := T1 + {X^i}          Almacenar X^i en T1
22         T0 := T1                               Las posibles cajas en T0 son eliminadas
23         T1 := {∅}
24   return Q
```

la región de búsqueda. El árbol temporal se inicializa al conjunto vacío. El bucle interior del algoritmo (línea 8) se ejecuta mientras el árbol de trabajo tenga cajas y el número de cajas seleccionadas de él sea menor que la variable N_m. Al salir de este bucle, las cajas que se encuentren en el árbol de trabajo son eliminadas. Entonces, el árbol temporal se convierte en el árbol de trabajo (línea 22) y el nuevo árbol temporal se inicializa al conjunto vacío (línea 23). El bucle externo (línea 6) se ejecuta mientras haya cajas en el árbol de trabajo, es decir, si se almacenaron cajas en el árbol temporal durante la ejecución del bucle interno, o lo que es lo mismo, si todas las cajas generadas no cumplen la *Regla de Terminación* ni de *Eliminación*.

En el algoritmo hay que resaltar que cuando se mejora el valor de $\overline{f^*}$ (línea 13), el *Test de Corte* se aplica, no sólo al árbol de trabajo y al final, sino también al temporal (líneas 14, 15 y 16).

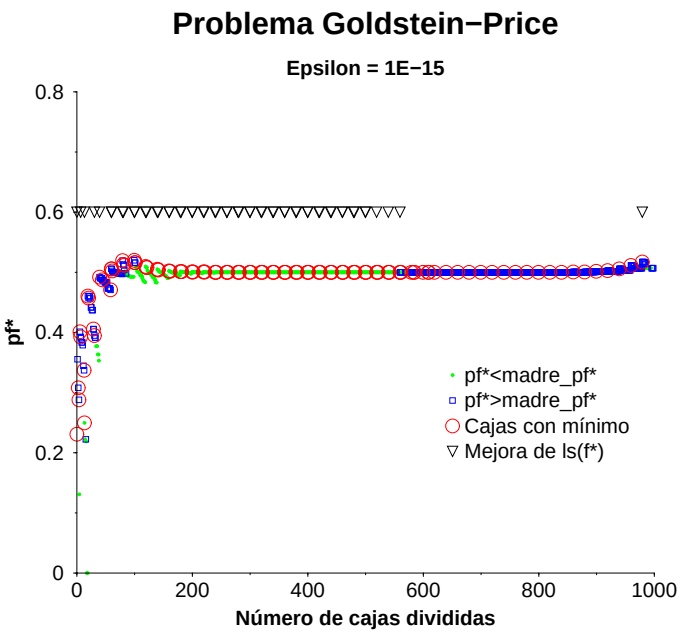


Figura 3.14: Valores de $p\overline{f^*}(X)$ durante la ejecución del Algoritmo 3.6.1, realizando bisección en todas las coordenadas (1DA), sin conocer f^* y $N_m = 20$.

3.6.1 Resultados numéricos

Las pruebas numéricas se han desarrollado en el mismo entorno de trabajo que en la Sección 3.4.2, es decir, sobre un PC Pentium II (233 MHz y 256Mb de RAM) con el Sistema Operativo Linux. Los programas fueron codificados en C y se usaron las rutinas BIAS [90], para implementar la Aritmética de Intervalos. Se ha ampliado a treinta el número de funciones de prueba, las cuales se describen en el Apéndice A. Se han podido incluir problemas dimensionalmente más complejos gracias a que con el nuevo algoritmo se puede establecer las limitaciones de memoria en la resolución de los mismos. Con este amplio número de funciones de prueba se pretende demostrar la generalidad del nuevo método.

La Figura 3.14 muestra gráficamente los valores de $p\overline{f^*}(X)$ de las cajas seleccionadas y divididas por el Algoritmo 3.6.1 para el problema Goldstein-Price. En este ejemplo se ha usado un criterio de terminación $w(X) \leq 10^{-15}$. Este valor de ϵ es el máximo que se puede establecer para un Pentium-II, ya que las cajas generadas al dividir una caja de este tamaño, mantienen el tamaño de la caja madre debido al redondeo directo usado en la Aritmética Computacional (Sección 1.2.6, página 17). Para este experimento se ha usado un valor de $N_m = 20$, que permite obtener la solución $[\underline{f^*}, \overline{f^*}] = [2.999999999999257, 3.000000000000050]$, con unas 5000 evaluaciones de F_{GP}, aproximadamente (3996 en X y 999 en $m(X)$). El número de cajas finales fue de 80, de

Tabla 3.11: Resultados numéricos para el Algoritmo 3.6.1 realizando una bisección de las dos dimensiones más anchas de la caja seleccionada a la vez, sin conocer f^* y $\epsilon = 10^{-8}$.

Probl.	CPU	N_m	EX	Ex	$[\underline{f^*}, \overline{f^*}]$
GP	0.04	10	1101	275	[2.999993965029506, 3.000000000000092]
S10	0.21	10	2161	540	[-10.536409821845988, -10.536409816692032]
S7	0.17	10	2153	538	[-10.402940571469488, -10.402940566818650]
S5	0.24	20	3801	950	[-10.153199679953214, -10.153199679058211]
H6	1.29	40	12549	3137	[-3.322368039956182, -3.322368011415509]
H3	0.12	10	1565	391	[-3.862782202989112, -3.862782147820753]
L3	0.23	10	1141	285	[-176.541795200601854,-176.541793136744104]
L5	0.24	10	1113	278	[-176.137580072493620,-176.137578001628754]
L8	0.11	10	977	244	[0.000000000000000, 0.000000000000000]
C	0.04	10	1141	285	[-1.031628487150618, -1.031628453489877]
C3	0.24	400	9149	2287	[-0.000000000000000, 0.000000000000000]
G2	0.04	10	621	155	[-0.000000000000000, 0.000000000000001]
G10	40.76	200	145141	36285	[-0.000000000000000, 0.000000000000004]
BR2	0.07	10	1125	281	[0.000000000000000, 0.000000000000000]
RB	0.03	10	1065	266	[0.000000000000000, 0.000000000000000]
SRB	0.02	10	617	154	[0.000000000000000, 0.000000000000000]
P	0.02	10	985	246	[0.000000000000000, 0.000000000000000]
TR	0.01	10	685	171	[0.000000000000000, 0.000000000000000]
S3.1	0.04	10	1305	326	[0.000000000000000, 0.000000000000000]
MT	0.02	10	949	237	[-0.000000000000000, 0.000000000000000]
EX1	0.06	10	973	243	[-1.076182009447070, -1.076182007071511]
EX2	18.04	600	147001	36750	[0.212459791854624, 0.212459838694385]
BR	0.06	10	1137	284	[0.397887357729738, 0.397887357729745]
KW	11.76	600	110849	27712	[0.000307485694808, 0.000307485987806]
R4	0.06	10	1141	285	[-0.106891343149305, -0.106891341408143]
R8	1.60	10	5493	1373	[0.000000000000000, 0.000000000000000]
HM3	3.11	200	19965	4991	[-24.062499142799162, -24.062498884334261]
HM4	3.13	100	16033	4008	[-36.093748714198512, -36.093748326501391]
BL	0.03	10	1101	275	[0.000000000000000, 0.000000000000000]
CHI	0.08	10	1237	309	[-43.315862131678990, -43.315862072142579]

las cuales una contenía el mínimo global.

En la Tabla 3.11 se muestran los resultados de la evaluación del Algoritmo 3.6.1 para las treinta funciones de prueba mencionadas y un criterio de terminación $\epsilon = 10^{-8}$. El método de división utilizado consiste en realizar una bisección de las dos coordenadas más anchas de la caja seleccionada. El valor de N_m se ha elegido aproximadamente, para que al menos uno de los mínimos globales se encuentre en el conjunto de cajas finales y poder establecer así los requerimientos de memoria necesarios para obtener al menos una solución para cada una de las funciones de prueba utilizadas.

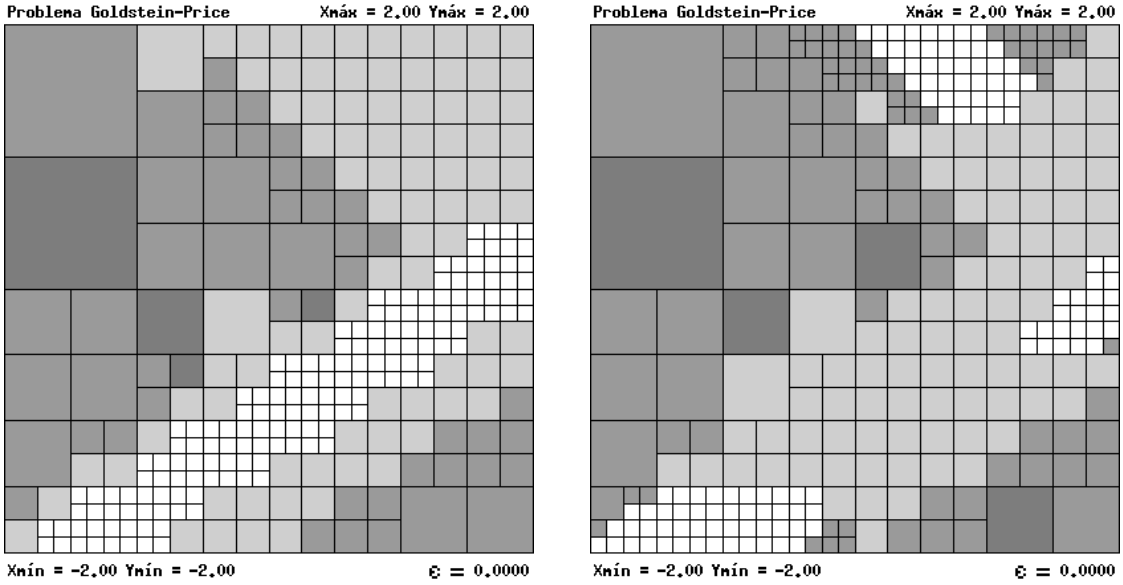


Figura 3.15: Ejecución de cinco iteraciones del Algoritmo 3.6.1 con $N_m = 40$ y una *Regla de Selección* basada en $p\overline{f^*}(X)$ (izquierda) y en el límite inferior de $F(X)$ (derecha).

A partir de los datos obtenidos puede deducirse que la mayoría de funciones de prueba pueden resolverse con unos requerimientos de memoria muy pequeños, comparados con la cantidad de memoria RAM que normalmente tiene cualquier computador personal. Por ejemplo, para la función 5-dimensional $EX2$, en la que se usó $N_m = 600$, el número máximo de cajas en el árbol temporal, con una bisección de las dos dimensiones mayores de las cajas seleccionadas, es en el peor de los casos de: 4 cajas generadas por 600 cajas seleccionadas = 2400 cajas.

Con este tipo de algoritmo no sólo se reduce la cantidad de memoria necesaria, sino que el número de evaluaciones de la *función de inclusión* se acerca al mínimo posible. Por ejemplo, para la función GP con $N_m = 40$ y $\epsilon = 10^{-8}$, se obtuvieron las cuatro cajas finales que contienen el mínimo global con 5296 evaluaciones totales de intervalos. El número mínimo de evaluaciones de intervalos, para obtener todas las soluciones, es igual al número de cajas que en cualquier momento de la ejecución del algoritmo contienen un mínimo global. Para el ejemplo anterior, en el que se realizaron 5296 evaluaciones, este número de cajas fue de 111.

Desde el punto de vista del usuario la selección del valor de N_m puede hacerse de forma incremental hasta llegar a un valor suficientemente grande en el que no se produzcan mejoras en el valor de $\overline{f^*}$. Un índice claro de que el valor de N_m es insuficiente es que el árbol final se encuentre vacío.

¿Qué pasaría si en vez de usar el mayor valor de $p\overline{f^*}(X)$, se usa el menor valor de $\underline{F}(X)$ como criterio de selección?.

La gráfica izquierda de la Figura 3.15 muestra la ejecución de cinco iteraciones del bucle externo del Algoritmo 3.6.1, con una *Regla de Selección* basada en *Primero el Mejor* $p\overline{f^*}(X)$, mientras que en la gráfica de la derecha se ha utilizado una *Regla de Selección* basada en *Primero el Mejor* $\underline{F}(X)$. Las cajas blancas representan las cajas activas, mientras que las cajas grises representan las cajas eliminadas. Desde el nivel de gris oscuro al claro las cajas fueron eliminadas por el *Test de Corte*, el Test del Punto Medio y el nuevo criterio de eliminación usado el Algoritmo 3.6.1, respectivamente. Como puede observarse en la Figura 3.15 usando una *Regla de Selección* basada en primero el menor valor de $\underline{F}(X)$ las cajas que contenían el mínimo global fueron eliminadas, lo que no ocurrió cuando se usó una *Regla de Selección* basada en primero el mayor $p\overline{f^*}(X)$.

Estos mismos resultados se repitieron para varias de las funciones de prueba utilizadas, donde sólo incrementando el valor de N_m se alcanzaban los mismos resultados que con una *Regla de Selección* basada en $p\overline{f^*}(X)$. Sin embargo, para otras funciones, incluso usando una *Regla de Selección* basada en escoger primero la caja con el menor valor de $\underline{F}(X)$, se obtenían los mismos resultados con el mismo valor de N_m.

3.7 Resumen

En este capítulo se ha demostrado que una buena elección de las reglas básicas de los algoritmos de Ramificación y Acotación permite una mejora sustancial en los indicadores de la eficiencia. El nuevo parámetro $p\overline{f^*}(X)$ se ha mostrado como un buen criterio para determinar cuanta región de atracción hacia un mínimo contiene una determinada subregión de búsqueda. Este parámetro se ha utilizado para establecer una nueva *Regla de División* adaptativa, que ha aportado una mejora del 60% en tiempo de CPU, del 40% en el numero de nodos de árbol de búsqueda inspeccionados y un 50% en cuanto a los requerimientos de memoria, para una precisión de $\epsilon = 10^{-3}$ en los problemas tratados, en comparación con el método de bisección tradicionalmente utilizado en algoritmos de Optimización Global basados en intervalos. Este parámetro también se ha utilizado para establecer una nueva *Regla de Selección* híbrida, que permite alcanzar rápidamente un buen valor del límite superior del mínimo global.

También se ha mejorado la *Regla de Eliminación* gracias al nuevo parámetro $p\overline{f^*}(X)$: en la primera propuesta, se ha obtenido una mejora del 80%. La segunda propuesta está motivada por los grandes requerimientos de memoria que necesitan los problemas a resolver cuando son dimensionalmente grandes o se necesita mucha precisión en los resultados. La nueva *Regla de Eliminación*, junto con una *Regla de Selección* basada en un modelo híbrido, han permitido resolver un mayor número de problemas, con más precisión y con unos requerimientos mínimos de memoria.

Capítulo 4

Algoritmos paralelos de Optimización Global basados en Aritmética de Intervalos y Ramificación y Acotación

En este capítulo se realizará una introducción a las arquitecturas paralelas y se describirán algunas de las características más destacables de los distintos modelos, exponiendo las medidas de rendimiento que normalmente se usan para evaluar la eficiencia de los algoritmos paralelos. Se realizará una introducción de la metodología a seguir para paralelizar algoritmos secuenciales, donde uno de los problemas que pueden aparecer es el desbalanceo de la carga computacional asociada a cada uno de los procesadores. En algunos casos, estos problemas deben ser tratados mediante técnicas de balanceo dinámico de la carga. También se tendrán en cuenta las posibles decisiones a tomar para la implementación de algoritmos paralelos de Ramificación y Acotación y las anomalías que se pueden encontrar. Posteriormente se describirán, de forma resumida, las implementaciones paralelas de algoritmos de Optimización Global basados en intervalos realizadas por otros autores. Finalmente se propone un nuevo criterio de balanceo de la carga en algoritmos paralelos de Optimización Global basados en intervalos y se presentan los resultados de las implementaciones paralelas realizadas, mostrando los indicadores de eficiencia de los nuevos algoritmos sobre problemas de prueba computacionalmente poco costosos.

4.1 Algunas nociones sobre paralelismo

El rendimiento de los computadores convencionales (mono-procesador) se ha incrementado sustancialmente en poco tiempo, doblándose la frecuencia del reloj en el procesador cada 18 meses. El aumento de las prestaciones del procesador hace necesario un aumento en el rendimiento de la Aritmética Computacional y una disminución de los tiempos de acceso

a memoria. Con estas mejoras, el número de problemas resueltos mediante un computador convencional es mayor, pero aún existe un gran número de problemas computacionalmente complejos que no pueden ser resueltos por dichos computadores, o cuyo tiempo de resolución sobre computadores mono-procesador sigue siendo muy alto. Un ejemplo de este tipo de problemas lo constituyen los algoritmos de Optimización Global basados en Aritmética de Intervalos y Ramificación y Acotación, especialmente en aquellos casos en los que se desea obtener una solución rigurosa, principalmente cuando las funciones a optimizar son no diferenciables.

Para intentar resolver un mayor número de problemas complejos existen dos tipos de computadores de alto rendimiento, los computadores vectoriales y los computadores paralelos [79, 136]. En los computadores vectoriales se obtiene una ganancia en velocidad mediante el uso de varias cadenas ("pipelines") de unidades funcionales en la Unidad Aritmético Lógica (ALU) de la Unidad Central de Proceso (CPU). Se entiende por "pipeline" el desglosar las operaciones aritméticas, como la obtención de datos, comparación de exponentes, desplazamientos de mantisa, normalización etc, y su ejecución paralela en diferentes componentes de la ALU, para muchos datos homogéneos (por ejemplo, la suma de los componentes de dos vectores). Usando computadores vectoriales se pueden obtener ganancias en velocidad considerables para muchos problemas, con las herramientas adecuadas (compiladores de procesamiento vectorial). En general, estos computadores no son adecuados para su uso en la resolución de problemas de Optimización Global ya que, generalmente, las funciones a evaluar son no lineales. No obstante, con los computadores vectoriales se obtiene una gran eficiencia si se realizan muchas operaciones lineales (operaciones sobre vectores o matrices).

A partir de aquí se hablará de computadores paralelos y su uso eficiente para la resolución de problemas computacionalmente costosos. Esta sección describe algunas de las nociones fundamentales sobre las arquitecturas paralelas existentes, cómo evaluar el rendimiento de los algoritmos paralelos y los modelos de paralelización de algoritmos secuenciales.

4.1.1 Clasificación de las Arquitecturas Paralelas

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 entre los mismos, consumen tiempo. Las medidas de rendimiento de los algoritmos paralelos serán tratadas en el apartado 4.1.2.

Los computadores paralelos se clasifican de acuerdo a diferentes criterios:

SECUENCIA DE INSTRUCCIONES Y DATOS

La clasificación de los computadores paralelos a partir de las secuencias de las instrucciones y de los datos es debida a Flynn [53]. Se diferencian las siguientes clases de computadores:

SISD (Simple Instrucción / Simple Dato): Este tipo de computador es el computador serie estándar según Neumann. Los computadores de este tipo tienen una secuencia simple de instrucciones que operan sobre una secuencia simple de datos. Sólo se dispone de una CPU. Los algoritmos usados en este tipo de computadores son llamados algoritmos secuenciales. Una de las mejoras incorporadas en los computadores SISD es el paralelismo a nivel de instrucción ("pipeline"), lo que permite reducir el número de ciclos de reloj necesarios para la ejecución de un conjunto de instrucciones. Ejemplos de computadores SISD son los computadores personales o las estaciones de trabajo (naturalmente con una sola CPU).

SIMD (Simple Instrucción / Múltiples Datos): Todos los elementos de proceso del computador paralelo realizan las mismas instrucciones de un solo programa y las ejecutan sobre diferentes datos de entrada. Sobre un computador paralelo del tipo SIMD con $p = n \cdot m$ elementos de proceso se pueden sumar dos matrices $n \times m$ con una sola suma en punto flotante por elemento de proceso. Por lo tanto el tiempo total para realizar la suma de las dos matrices, vendrá determinado por el tiempo necesario para realizar una suma en punto flotante. Los computadores SIMD tienen normalmente un gran número de elementos de proceso, debido a su estructura simple, siendo reconocidos normalmente como sistemas de procesamiento paralelo masivo. Ejemplos de computadores SIMD son la Connection Machine CM-2 y los MasPar MP-1 y MP-2.

MISD (Múltiples Instrucciones / Simple Dato): Existen pocos ejemplos de esta clase de computadores en la actualidad. Este modelo está basado en la ejecución de diferentes instrucciones (programas) sobre el mismo conjunto de datos. Normalmente se usan como sistemas tolerantes a fallos, donde debe garantizarse la respuesta del sistema, o en máquinas de propósito especial, como por ejemplo, para aplicar varios filtros de frecuencia a una misma señal.

MIMD (Múltiples Instrucciones / Múltiples Datos): Cada uno de los procesadores del computador paralelo ejecuta su propio programa y por lo tanto pueden operar, independientemente unos de otros, sobre datos diferentes. Una máquina MIMD también puede programarse para operar como si perteneciese a cualquiera de las otras categorías. Frecuentemente se usan computadores estándar (computadores personales o estaciones de trabajo) como nodos individuales del computador paralelo, haciendo uso de las nuevas tecnologías del mercado. Un aspecto importante en los computadores paralelos son las comunicaciones entre los nodos del mismo. La red de interconexión suele ser uno de los componentes más caros del sistema. Ejemplos de computadores MIMD son el Intel Paragon, IBM SPx, el Cray T3D.

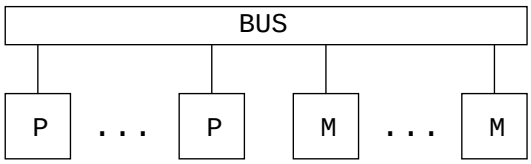


Figura 4.1: Bus de datos (M = memoria; P = Procesador).

TIPO DE ALMACENAMIENTO

Los computadores paralelos se pueden clasificar dependiendo de como esté estructurada la memoria:

Memoria compartida: Los procesadores del computador paralelo comparten una memoria central de almacenamiento que es accesible por todos los procesadores mediante una red de interconexión. Pueden aparecer conflictos de acceso a memoria, en particular, cuando el número de procesadores es alto. Así, por ejemplo, se introducen retardos si dos procesadores quieren escribir en la misma posición de memoria. La red de interconexión entre los procesadores y la memoria puede ser un bus (Figura 4.1). Tanto la memoria principal como el bus tienen un ancho de banda fijo que debe repartirse entre los distintos procesadores, lo que limita la escalabilidad de este tipo de sistemas. Para reducir el acceso a memoria principal y mejorar el rendimiento, se introdujeron memorias caches, locales a cada procesador. Un factor importante que se debe mantener en los computadores paralelos de memoria compartida es la coherencia de cache. Para permitir un mayor paralelismo en el acceso a memoria principal, la memoria principal puede constituirse por un conjunto de módulos a los que se accede mediante una red de interconexión que enrute las peticiones de los distintos procesadores hacia los distintos módulos de memoria, como por ejemplo una red crossbar-switch (Figura 4.3(a)). De esta forma, pueden realizarse varias comunicaciones procesador-memoria en paralelo. Ejemplos de computadores de memoria compartida son el C.mmp y el NYU Ultracomputer.

Desde el punto de vista del programador, este modelo es el más simple, debido a que la comunicación entre los distintos procesos paralelos se hace implícitamente como resultado de las operaciones de acceso a memoria.

Memoria distribuida: Muchos de los computadores paralelos de hoy en día usan memoria distribuida. Cada procesador tiene su propia memoria local, a la cual no pueden acceder otros procesadores directamente. Las ventajas de este modelo frente al de memoria compartida son: (i) Cada procesador puede usar todo el ancho de banda para el acceso a su memoria local. (ii) Es más escalable; el tamaño del sistema sólo está limitado por el tipo de red de interconexión utilizada para conectar los procesadores entre sí. (iii) No hay problemas de coherencia de cache. La programación de este tipo de sistemas es más costosa, ya que el intercambio de datos entre procesa-

dores se hace mediante el paso de mensajes, a través de una red de interconexión. Las operaciones más utilizadas en el paso de mensajes son *enviar* y *recibir*. De una forma simple, *enviar* significa crear un mensaje con datos locales y un identificador del proceso al que se le enviará dicho mensaje. *Recibir* implica reservar memoria local para almacenar los datos del mensaje transmitido. El paso de mensajes introduce además dos tipos de sobrecarga: el producido por el tiempo necesario para construir un mensaje y enviarlo, y el debido a que el proceso receptor debe interrumpir su ejecución para tratar el mensaje recibido. En este modelo, los programadores deben tener en cuenta dónde se almacenan los datos, lo que introduce el concepto de *localidad*, en el diseño de algoritmos paralelos.

Un computador paralelo de memoria distribuida puede estar compuesto por procesadores heterogéneos, que difieran en su velocidad de proceso (y en la velocidad de las comunicaciones), y programarse mediante librerías de paso de mensajes (por ejemplo PVM [60] o MPI [169]). Un ejemplo de este tipo de organización, parecida a la que se ha usado en la implementación de los algoritmos paralelos diseñados en este trabajo, consta de conjunto de estaciones de trabajo con una red de interconexión Ethernet o Fast-Ethernet. Su principal desventaja es la baja velocidad de la red de interconexión, comparada con la velocidad de cálculo de los procesadores [117]. La principal diferencia entre un conjunto, o cluster, de estaciones de trabajo y los computadores paralelos, como por ejemplo el SP2 de IBM, es que los últimos están provistos una red de interconexión mucho más rápida. En estos últimos también se pueden usar las rutinas de paso de mensaje de las librerías PVM o MPI, con lo que se consigue una alta portabilidad para este tipo de programas paralelos. Ejemplos de computadores paralelos con memoria distribuida son la Connection Machine 5 (CM5), el Computing Surface 2 de Meiko (CS2), Paragon de Intel con i860, NCube y otros.

Memoria Compartida-Distribuida: En este modelo la memoria está distribuida físicamente en cada uno de los procesadores, pero existe un soporte hardware para permitir un espacio de direcciones globales, e incluso para la gestión de la coherencia entre las caches locales de cada nodo. Esta filosofía auna la sencillez de la programación de un modelo de memoria compartida y la escalabilidad que proporcionan los sistemas de memoria distribuida. Los computadores paralelos con memoria compartida-distribuida se pueden organizar en dos grandes bloques:

Computadores NUMA (Non Uniform Memory Access): Con espacio de direcciones físico estático, entre las que encontramos arquitecturas sin coherencia de cache, como el Cray T3D con procesadores DEC-Alpha 21064, con coherencia parcial de cache, como el Cray T3E con el mismo procesador, o con coherencia de cache CC-NUMA (Cache Coherent Non Uniform memory Access) como el Origin 2000 de SGI, con procesadores R10000.

Computadores COMA (Cache Only Memory Architecture): Con espacio de di-

recciones físico dinámico, donde la memorias distribuidas son convertidas en caches. En este caso particular de computadores NUMA no hay jerarquía de memoria en cada procesador. Esto ocurre en la KSR-1 de Kendall Square Research y en la arquitectura I-ACOMA propuesta por investigadores de la Universidad de Illinois [178].

Simular un computador de paso de mensajes con p procesadores, sobre uno de memoria compartida con el mismo número de procesadores es fácil, ya que sólo hay que dividir el espacio de memoria compartida entre los p procesadores. Un procesador envía un mensaje a otro escribiendo en la partición de memoria del otro procesador. Sin embargo, emular un computador de memoria compartida sobre uno de paso de mensajes es costoso, ya que acceder a la memoria de otro procesador requiere enviar y recibir mensajes.

TOPOLOGÍA: RED DE INTERCONEXIÓN

El factor más importante a tener en cuenta en el diseño de sistemas paralelos es el conjunto de enlaces que utilizan los procesadores, las memorias y dispositivos de Entrada/Salida para comunicarse. Estos enlaces definen la red de interconexión o topología de una máquina, de la cual dependerá cómo los procesadores intercambian datos y a qué coste. Tanto a nivel físico como lógico, el aumento del número de procesadores del sistema, debe ir acompañado de un grafo de interconexión que permita transmisiones de datos entre ellos, en un tiempo razonable [150]. En estas condiciones, normalmente es necesario que los algoritmos paralelos se adapten a la topología de la máquina donde se va a realizar la ejecución. Sin embargo, la tendencia actual lleva hacia sistemas que tratan de realizar una simulación eficiente de cualquier tipo de red.

Matemáticamente una red de interconexión se puede ver como un modelo de grafo G=(V,A) dirigido o no: los elementos a comunicar están localizados en los vértices (nodos) del grafo recursivo, potencialmente infinito, y se comunican a través de los arcos (aristas). El uso práctico de estas estructuras está limitado, tanto por restricciones de cableado, como por principios de diseño y fabricación. Para tener una medida de la efectividad de las redes en la implementación de algoritmos paralelos eficientes sobre el hardware real se han introducido las siguientes definiciones:

Grado : Número de aristas por nodo. Dos nodos son vecinos si existe una arista que los conecta. El grado de un nodo está definido por el número de vecinos que tiene. Para obtener un sistema escalable, es mejor si el número de aristas por vértice es una constante independiente del número de nodos.

Diámetro : El diámetro de una red es la distancia más larga entre dos nodos cualesquiera. Los diámetros bajos, al menos logarítmicos con respecto al número de procesadores, son mejores, ya que la complejidad de los algoritmos paralelos que requieren comunicaciones entre pares de vértices arbitrarios, es menor.

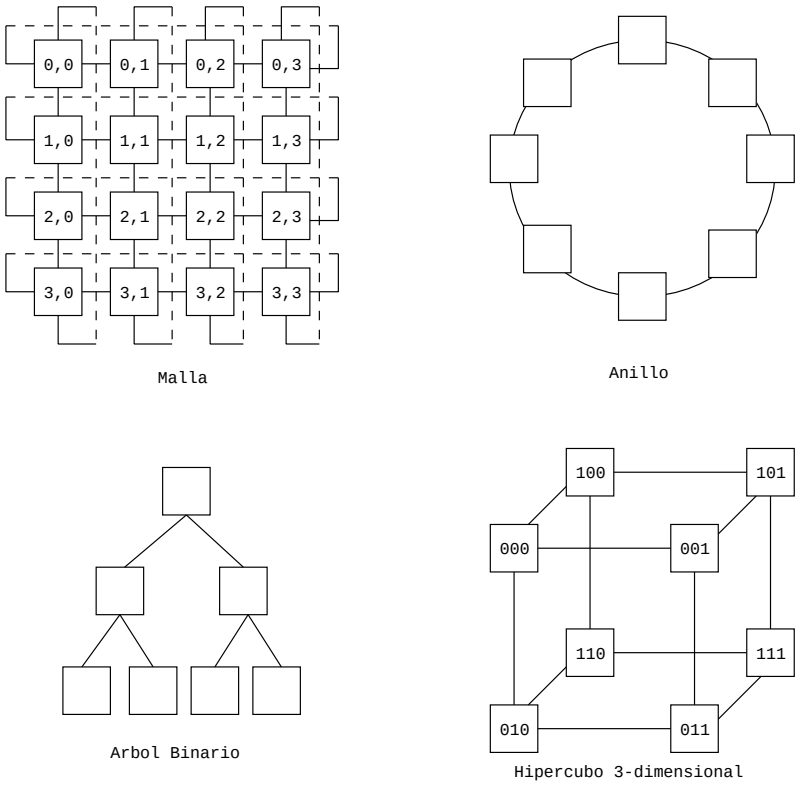


Figura 4.2: Topologías de redes estándar en computadores paralelos.

Ancho : Es el mínimo número de aristas que deben ser eliminadas con el fin de dividir la red en dos mitades (ancho de bisección). Son mejores las redes con un ancho grande, ya que permite mayores alternativas de enrutado.

En los ejemplos siguientes, los enlaces entre nodos son punto a punto, es decir, no son buses que se comparten entre varios nodos, como el mostrado en la Figura 4.1.

Algunas de la topologías estáticas más utilizadas son las representadas en la Figura 4.2, que serán descritas brevemente a continuación:

Malla : En una malla rectangular de $n \times n$ procesadores (se establece cuadrada por simplicidad), los nodos están organizados en un plano (generalizando hiperplanos de dimensión q). Cada nodo (i, j) se comunica con los nodos $(i \pm 1, j)$ e $(i, j \pm 1)$. El grado de cada vértice es cuatro, constante para cualquier número de procesadores, el diámetro es igual a $2(n-1)$ y el ancho es igual a n. Este tipo de red ha sido utilizado en computadores como el Maspar MP-1 y en el Intel Paragon XP/S.

Anillo : En un anillo de procesadores, cada nodo (i) tiene como vecinos a $((i+1) \mod n)$ e $((i-1) \mod n)$. Tiene un grado de 2, un diámetro de $n/2$ y un ancho de 2. No ha sido utilizada como organización en ningún tipo de máquina paralela de propósito general. Sin embargo es muy adecuada en los códigos de *cadena de montaje* (*pipeline*) con realimentación.

Árbol binario : La topología de árbol binario de nivel k utiliza $2^{k+1} - 1$ procesadores formando, como su nombre indica, un árbol binario. Cada nodo interior puede comunicarse con sus dos hijos y cada nodo distinto del raíz lo puede hacer con su padre. Su diámetro es $2(k-1)$ pero su ancho, igual a 1, es muy bajo. Al igual que la topología en anillo, debido a su pequeña bisección, suelen ocurrir cuellos de botella y no es usado como organización general.

Hipercubo : Esta topología consta de 2^k procesadores formando un hipercubo de dimensión k. Si se denotan los nodos con los índices $0, 1, \ldots, 2^k - 1$, dos vértices son adyacentes, si sus etiquetas difieren exactamente en un único bit. El diámetro de un hipercubo k-dimensional es k y su ancho 2^{k-1}. Este tipo de configuración tiene un diámetro bajo y un ancho grande. Sus buenos resultados quedan desvirtuados por su grado no constante k, que limita su escalabilidad. Esta deficiencia se ha intentado soslayar con topologías de hipercubos conectados en ciclos, en los que en cada vértice se sitúa un anillo de procesadores. Se ha usado como topología de propósito general, cuando el número de procesadores se encuentra dentro de unos límites, como por ejemplo en la Connection Machine CM-200.

De Brujin : Una red de Brujin está formada por 2^k nodos. Si cada nodo se identifica por su representación binaria $b_{k-1}b_{k-2}\ldots b_1b_0$, los dos nodos alcanzables directamente mediante arcos (aristas dirigidas) tienen las direcciones siguientes: $b_{k-2}\ldots b_1b_0, 0$ y $b_{k-2}\ldots b_1b_0, 1$. Ésta es la red que mejor comportamiento muestra con un grado de 2, y por lo tanto un factor constante en el número de enlaces, un diámetro k y una bisección $2^k/k$. Los procesadores del Triton/1, un procesador paralelo desarrollado en la Universidad de Karlsruhe, están conectados mediante este tipo de organización [75].

Los dos últimos ejemplos de redes de interconexión, que se van a describir en esta sección, son ejemplos de redes de interconexión que pueden reconfigurarse dinámicamente. Este tipo de redes, basadas en conmutadores de enlaces, se han usado tradicionalmente

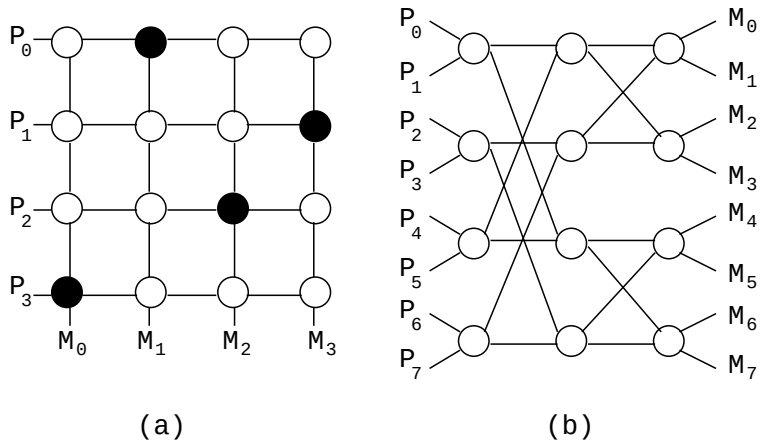


Figura 4.3: (a) Crossbar switch. (b) Red Butterfly.

para conectar procesadores con módulos de memoria, pero también se usan para conectar procesadores entre sí [11]. Los conmutadores se configuran dinámicamente para poder conectar los nodos emisor y receptor. De esta forma, la información se comunica del nodo emisor al receptor, a través de uno o más conmutadores, pudiendo realizar al mismo tiempo varias de estas comunicaciones:

Crossbar switch : Un ejemplo se muestra en la Figura 4.3(a). En este ejemplo, el crossbar se configura dinámicamente para conectar un procesador con una memoria. Si cada procesador se quiere conectar con una memoria diferente, entonces no existen conflictos de acceso a memoria. Si dos procesadores quieren acceder a la misma memoria, uno debe bloquearse hasta que el crossbar se reconfigure. Un crossbar tiene un diámetro corto, pero una escalabilidad muy pobre. Por ejemplo, para p procesadores y p memorias se necesitan p^2 conmutadores. Para añadir otro procesador y otra memoria se necesitan añadir $2p - 1$ conmutadores. Ejemplos de computadores paralelos que usan un crossbar switch son el Cray Y-MP y el Fujitsu VPP 500.

Red Bayan : Es una red multietapa de conmutadores, que tiene el mismo número de entradas y de salidas, con conmutadores de $m \times m$. Ejemplos de redes Bayan son las redes Omega y las redes Butterfly, ambas con conmutadores de 2×2. Un ejemplo de una red Butterfly se muestra en la Figura 4.3(b). El diámetro de una red Butterfly de k entradas es $\log_2 k$ y se necesitan $O(k \cdot \log_2 k)$ conmutadores, por lo que en este tipo de redes, aumentar el número de entradas y de salidas es menos costoso que en una red basada en un crossbar switch, pero el número de conexiones que se pueden realizar en paralelo es menor. Un ejemplo de computador paralelo basado en una red Butterfly es el BBN Butterfly.

Una descripción más general de los distintos tipos de redes de interconexión y su modo de funcionamiento puede encontrarse en [44].

4.1.2 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 el tiempo requerido para su resolución en 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 para obtener la aceleración producida por el algoritmo paralelo, frente al secuencial, se denota por la ganancia de velocidad o *"speed-up"*, que se define como:

Definición 4.1.1 *Se define la ganancia de velocidad o "speed-up"S(p) como la relación entre el tiempo total de resolución del algoritmo en un computador secuencial, t_{sec}, y el tiempo total requerido para resolverlo en un computador paralelo con p procesadores, $t_{par}(p)$:*

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

Si se calcula la relación entre la ganancia en velocidad obtenida y el número de procesadores, se obtiene la eficiencia del algoritmo:

Definición 4.1.2 *La eficiencia $E(p)$ representa la ganancia de velocidad por procesador:*

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

Comentarios:

1. Para el cálculo de la ganancia en velocidad debe usarse el mejor algoritmo serie conocido, por lo tanto, aquel que necesita un menor t_{sec}. Frecuentemente, el mejor algoritmo serie no es único, o el mejor algoritmo serie no se ha implementado todavía, por lo que su medida no es posible. Una técnica de uso común es la de usar, como algoritmo serie, el algoritmo paralelo sobre un solo procesador.

2. 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 Ramificación y Acotación que tratamos en esta tesis, esto puede deberse a que en el algoritmo serie se examinan 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.

3. La caracterización exacta de un algoritmo paralelo eficiente no es fácil. Muchos factores pueden contribuir a limitar la ganancia de la 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* [3]:

   Definición 4.1.3 *La Ley de Amdahl establece que la parte de código intrínsecamente secuencial limita la ganancia en velocidad de un algoritmo paralelo. Sea s la fracción de operaciones de un algoritmo paralelo que deben realizarse de forma secuencial ($0 \leq s \leq 1$). La Ley de Amdahl establece que la máxima ganancia en velocidad $S_m(p)$ que se puede conseguir mediante un modelo paralelo de p procesadores ejecutando dicho algoritmo es:*

$$S_m(p) \leq \frac{1}{s + \frac{1-s}{p}}$$

   Es decir, para un problema de tamaño fijo, cada algoritmo paralelo es ineficiente asintóticamente con el número de procesadores.

   Otra causa del aumento del tiempo de ejecución provocado por el código secuencial es la debida al retardo que se produce en la activación de los demás procesadores, cuando acaba el código secuencial. Este fenómeno se conoce con el nombre de *efecto árbol*.

4.1.3 Paralelización de algoritmos serie

Cuando se aborda el problema de la paralelización o implementación paralela de un algoritmo, en algunas ocasiones, el algoritmo paralelo, o la mejor solución paralela, puede ser muy diferente de la sugerida por el algoritmo serie. Para el diseño de algoritmos paralelos puede seguirse una metodología que divide la fase de diseño en cuatro etapas [54]:

Partición: La computación a realizar y los datos necesarios para esta computación se dividen en diversas tareas. Las diferentes estrategias a seguir pueden ser:

Descomposición de Datos : Primero se descomponen los datos y después se les asocia la computación necesaria.

Descomposición Funcional : Primero se descompone la computación en diferentes tareas, estableciéndose los datos que cada tarea tiene asociados.

En esta etapa se ignoran aspectos prácticos, tal como el número de procesadores en la arquitectura final, centrando la atención en reconocer las partes paralelas del algoritmo.

Comunicación: Se determinan las comunicaciones necesarias para coordinar la ejecución de las distintas tareas, definiendo estructuras y algoritmos de comunicación adecuados. Distintas opciones de los tipos de comunicaciones a realizar son:

Local/Global: En las comunicaciones locales, cada tarea se comunica con un conjunto pequeño de otras tareas. Las comunicaciones globales implican que cada tarea se comunique con otras muchas o con todas las demás.

Estructurada/ No Estructurada: En las comunicaciones estructuradas, una tarea y sus vecinas forman una estructura regular, como un árbol o una malla. En las comunicaciones no estructuradas la representación de las comunicaciones puede ser un grafo arbitrario.

Estáticas/Dinámicas: Las comunicaciones son estáticas cuando las tareas que intervienen en una comunicación no cambian durante el tiempo de ejecución. Las tareas que deben comunicarse entre sí mediante estructuras de comunicación dinámicas, se determinan por los datos computados en tiempo de ejecución y pueden ser muy variables.

Síncronas/Asíncronas: En las comunicaciones síncronas, los productores (tareas que generan datos) y consumidores (tareas que utilizan los datos generados por los productores) se ejecutan de una forma coordinada, esperándose unos a otros para intercambiar datos. En las comunicaciones asíncronas no existe una coordinación entre productor y consumidor, ya que los productores no pueden determinar cuándo los consumidores pueden requerir datos, por lo que los consumidores tienen que pedir datos explícitamente a los productores.

Aglomeración: Las tareas y las estructuras de comunicación, definidas en las dos primeras etapas del diseño, se evalúan respecto a los requerimientos de funcionamiento y costes de la implementación en una arquitectura particular. Si es necesario las tareas pueden combinarse en tareas mayores para mejorar el rendimiento o reducir costes de desarrollo. También hay que determinar si merece la pena tener datos y/o computación replicados. El número de tareas obtenido en esta fase, aunque reducido, debe ser suficiente para el número de procesadores a utilizar.

Proyección: Cada tarea se asigna a un procesador intentando maximizar el uso del procesador y minimizar los costes de comunicación. La proyección puede especificarse estáticamente o determinarse en tiempo de ejecución, mediante estrategias de balanceo dinámico de la carga. En computadores de memoria distribuida que usan paso de mensajes, las tareas que se comunican frecuentemente se sitúan en procesadores cercanos, o incluso en el mismo procesador. Por otro lado, las tareas que pueden ejecutarse en paralelo deben situarse en distintos procesadores para explotar su paralelismo. Ambas decisiones, en algunas ocasiones, entran en conflicto.

En la realización de un algoritmo paralelo a partir de uno serie se pueden usar dos alternativas:

Paralelismo implícito

Este tipo de paralelismo es más fácil para el programador, porque se deja al compilador que realice la mayor parte de la paralelización. De esta forma, los programas existentes pueden paralelizarse casi sin ser cambiados. Existen lenguajes de programación y compiladores que permiten realizar un paralelismo implícito como es el caso del HPF (High Performance FORTRAN). Éstos extienden los lenguajes de programación tradicionales, como el FORTRAN 90, con sentencias paralelas (por ejemplo con el comando FORALL). Desafortunadamente, la conversión automática de programas secuenciales, a versiones paralelas eficientes, es muy difícil porque el compilador debe analizar y comprender las dependencias entre las distintas partes del código secuencial, para asegurar una proyección eficiente en el computador paralelo. El compilador debe dividir el programa secuencial y analizar las dependencias entre las distintas partes. El análisis de dependencias es complicado debido a las estructuras de control como bucles, sentencias de salto condicional y llamadas a subrutinas, además de que normalmente existen muchas formas de escribir un programa secuencial, para una determinada aplicación. Por estos motivos, en la mayoría de los casos, la intervención del programador es necesaria.

Paralelismo explícito

En el modelo de paralelismo explicito, es el programador el encargado de determinar las opciones concretas en cada una de las etapas del diseño de algoritmos paralelos. Un escenario común está compuesto por computadores paralelos formados por nodos que son procesadores que se usan en estaciones de trabajo de altas prestaciones, o incluso las propias estaciones, conectadas mediante una red de comunicación rápida. Ejemplo de estos computadores son el SP2 de IBM con nodos RS/6000 o el Cray T3E con nodos basados en los Alpha de DEC. El número medio de nodos está entre 10 y 100. En este tipo de computadores, las comunicaciones entre nodos se realizan normalmente mediante el uso de paso de mensajes. En cuanto a los algoritmos, éstos se caracterizan por usar normalmente una descomposición de datos, donde se realiza una mezcla de grandes fases de cálculo, con posibles intercambios de datos entre procesadores, y en algunas ocasiones, fases de balanceo de la carga.

4.1.4 Balanceo de la carga

El objetivo de un programa paralelo es realizar una buena división del problema, para balancear la carga computacional entre los procesadores y minimizar la necesidad de comunicación entre procesos. Normalmente las técnicas de balanceo de la carga han sido propuestas para su uso en algoritmos paralelos basados en la descomposición de datos que se ejecutan sobre computadores de memoria distribuida. Esta descomposición de datos puede realizarse de dos formas:

Balanceo estático de la carga: Podemos hablar de una distribución de la carga estática, si la carga computacional del problema se puede conocer antes de resolverlo. La solución más simple es la de dividir el problema en p subproblemas que tengan aproximadamente la misma carga computacional, resolviendo cada uno de estos subproblemas en cada procesador, intentando también minimizar las comunicaciones. La principal ventaja del balanceo estático de la carga es que no produce ninguna sobrecarga computacional en tiempo de ejecución.

Balanceo dinámico de la carga: Se aplica a problemas donde la carga computacional se crea y/o se destruye dinámicamente, durante la ejecución del algoritmo paralelo. Hay que intentar balancear la carga, es decir, que la carga computacional de los subproblemas tratados en cada procesador sea aproximadamente igual, evitando que haya procesadores ociosos o en espera de trabajo, mientras otros tienen mucho trabajo por realizar. Para resolver este tipo de problemas se usan estrategias dinámicas de distribución de la carga computacional. Estas estrategias producen un incremento de la carga computacional del programa paralelo en tiempo de computación. Por este motivo, hay que establecer un compromiso entre un balanceo óptimo de la carga y la sobrecarga producida por el balanceo.

Generalmente los algoritmos de balanceo dinámico de la carga consisten en cuatro componentes:

Medida de la carga: La carga computacional se cuantifica normalmente mediante un índice positivo, que toma el valor cero cuando un procesador está ocioso, e incrementa su valor, cuando se incrementa la carga computacional. Este índice es una estimación de la carga computacional y se basa en algunos parámetros medibles, como la cantidad y el tamaño de las operaciones a realizar. Debido a que la medida de la carga se hace frecuentemente, su cálculo debe ser muy eficiente, por lo que normalmente se usan heurísticas para su estimación.

Los problemas tratados en esta tesis están basados en algoritmos de Ramificación y Acotación, que tienen como principal inconveniente, la necesidad de realizar un balanceo dinámico de la carga. Para estos algoritmos, la carga de un procesador se define mediante una función de carga, c, que depende del número de subproblemas locales y de la calidad de los mismos. Si se denota por p_i al procesador i y se usa como estimador de la calidad de una subregión de búsqueda, $X^i,\ i = 1, \cdots, k$, el valor de $\underline{F}(X^i)$, entonces algunas de las posibles funciones de la carga asociada al procesador p_i pueden ser:

$$c(p_i) = k \tag{4.1}$$

$$c(p_i) = \min_{j=1..k} \{\underline{F}(X^j)\} \tag{4.2}$$

$$c(p_i) = \sum_{j=1}^{k} (\overline{f^*} - \underline{F}(X^j))^2 \tag{4.3}$$

$$c(p_i) = \sum_{j=1}^{k} e^{\overline{f^*} - \underline{F}(X^j)} \tag{4.4}$$

Intercambio de información: Para tomar decisiones sobre la necesidad de balanceo, un procesador tiene que conocer la carga de los demás procesadores. Este intercambio de información puede hacerse de forma centralizada o distribuida. En la versión distribuida, cada procesador debe tener información de la carga de los procesadores vecinos, si se establece una estrategia local, o una información de todos los procesadores, si se usa una estrategia global. Normalmente la estrategia global es impracticable para sistemas con un gran número de procesadores. En el modelo centralizado un procesador está dedicado a recoger y mantener la información de la carga del sistema. Normalmente este procesador tiene la responsabilidad de tomar las decisiones para balancear la carga en los otros procesadores. Las estrategias centralizadas obtienen un buen rendimiento en sistemas de pequeña escala. En sistemas de una escala mayor, el procesador dedicado al balanceo de la carga suele ser un cuello de botella para las comunicaciones. Para remediar este problema se han propuesto estructuras jerárquicas de este tipo de procesadores dedicados.

Hay que establecer un compromiso entre tener un valor actualizado de la carga de los demás procesadores y reducir el número de comunicaciones. Las estrategias más utilizadas son las siguientes:

Por demanda : Un procesador pregunta a los demás sobre su carga antes de iniciar una operación de balanceo.

Periódicamente : Los procesos se intercambian periódicamente información sobre su carga.

Por cambio de carga : Cuando el estado de la carga de un procesador cambia un cierto grado, el procesador comunica el cambio a los demás.

Además de la información de la carga actual, se pueden realizar nuevos balanceos de la carga a partir de los resultados de operaciones de balanceo previas.

También existen algoritmos que distribuyen la carga de forma aleatoria, en tiempo de ejecución, sin ningún tipo de información sobre la carga del sistema.

Regla de inicialización: Indica cuándo debe inicializarse una operación de balanceo. En esta decisión se debe sopesar la ventaja de inicializar el balanceo, frente a la sobrecarga computacional que éste conlleva. Este tipo de decisiones suelen ser heurísticas, debido a que se basan en el último valor conocido de la carga computacional de los demás procesadores, que normalmente es impreciso. El balanceo de la carga puede inicializarse por un procesador con mucha carga computacional (inicializada por emisor), por un procesador con poca carga

computacional (inicializada por receptor) o periódicamente. En las versiones inicializadas por emisor y receptor hay que establecer unos umbrales máximos y mínimos de la carga computacional, a partir de los cuales debe inicializarse la operación de balanceo. Estos umbrales dependerán de la estrategia establecida para el intercambio de información sobre la carga computacional.

Operación de balanceo: La operación de balanceo está definida por la siguientes reglas:

Regla de localización: Determina el conjunto de procesadores que van a intervenir en el balanceo de la carga, definiendo de esta forma el dominio de balanceo.

Regla de distribución: Determina cómo distribuir la carga en el dominio de balanceo.

Regla de selección: Determina qué trabajos forman la carga que se va a distribuir.

Ejemplos concretos de algoritmos de balanceo estático y dinámico de la carga pueden encontrarse en [110, 186].

4.2 Algoritmos de Ramificación y Acotación paralelos

Los algoritmos de Ramificación y Acotación (*Branch and Bound)* se utilizan para encontrar soluciones óptimas mediante la generación de arboles de búsqueda, donde a menudo, una búsqueda exhaustiva en todo el árbol es impracticable. Están basados en una descomposición sucesiva del problema original en subproblemas más pequeños y disjuntos, hasta encontrar la solución óptima, evitando visitar algunos subproblemas en los cuales se sabe que no se encuentra dicha solución. El procesamiento paralelo ha sido considerado ampliamente como una fuente adicional de mejora en la eficiencia de la búsqueda, ya que los problemas que se generan durante la ejecución de los algoritmos de Ramificación y Acotación son disjuntos y se pueden resolver simultanea e independientemente, pudiéndose usar un conjunto de procesadores para resolver concurrentemente varios subproblemas en cada iteración.

Los algoritmos de *Ramificación y Acotación* difieren de otras aplicaciones de la Ciencia y la Ingeniería en que:

- Los conjuntos de datos son irregulares y de distinto tamaño.
- La carga de trabajo generada no es uniforme.
- En algoritmos paralelos de *Ramificación y Acotación* el trabajo realizado puede depender del número de procesadores que se usen.

Por estos motivos la programación se convierte en un trabajo preciso en el que hay que tener en cuenta las siguientes posibilidades:

Sobrecarga de la búsqueda: Los algoritmos de búsqueda paralelos tienen sobrecargas debido a diferentes motivos. Éstas incluyen sobrecargas en las comunicaciones y tiempos ociosos debido a desbalanceo de la carga y a conflictos de acceso a memoria, si se usan estructuras de datos compartidas. Así, si las implementaciones paralelas y secuenciales del algoritmo realizan la misma cantidad de trabajo, es normal que la ganancia en velocidad para p procesadores $S(p) < p$. Sin embargo, la cantidad de trabajo realizada en la implementación paralela a menudo difiere de la realizada por la versión secuencial, principalmente, porque se exploran diferentes partes del árbol de búsqueda T.

Definición 4.2.1 *Sea W el trabajo realizado por el algoritmo de búsqueda secuencial y W_p el realizado por la versión paralela con p procesadores. Se define el factor de sobrecarga de la búsqueda del sistema paralelo como la relación entre el trabajo realizado por la versión paralela y la secuencial, o W_p/W [65].*

Así, el límite superior de $S(p)$ viene dado por $p \cdot W/W_p$. En la práctica, la ganancia de velocidad puede ser menor, debido a otras sobrecargas de la paralelización. En la mayoría de los algoritmos de búsqueda paralelos el factor de *sobrecarga de la búsqueda* es mayor de uno. Sin embargo, en algunos casos es menor que uno, dando lugar a ganancias de velocidad superlineales. Si el *factor de sobrecarga* es menor que uno, en media, entonces el algoritmo serie no es el más rápido para resolver el problema.

Decisión entre comunicación y computación: En los algoritmos de Ramificación y Acotación paralelos hay que tomar la decisión clásica entre comunicación y computación. Como se mencionó anteriormente la *sobrecarga de la búsqueda* para muchos algoritmos es mayor que uno, lo que implica que la implementación paralela realiza más trabajo que la secuencial. Se puede reducir la *sobrecarga de la búsqueda* para estas implementaciones a costa de incrementar las comunicaciones y viceversa. Por ejemplo, si la región de búsqueda se divide estáticamente entre los procesadores y estos realizan una búsqueda independiente, por lo tanto sin comunicaciones, estos realizaran en conjunto una búsqueda mayor que en la versión secuencial del algoritmo. Esta *sobrecarga de la búsqueda* puede reducirse, en mayor o menor medida, dependiendo de las comunicaciones realizadas entre procesadores. Si en los algoritmos de Ramificación y Acotación paralelos, los procesadores se dedican a buscar en distintas ramas de T, un parámetro importante a comunicar es la mejora encontrada en el límite superior de la solución, $\overline{f^*}$. Este intercambio de información permitirá a más procesadores hacer uso de la *Regla de Eliminación* y reducir así la *sobrecarga de la búsqueda*, pero aumentando la *sobrecarga de las comunicaciones*.

Escalabilidad y algoritmos escalables: La ganancia de velocidad no se incrementa linealmente con el número de procesadores si se usa la versión paralela para resolver un problema de tamaño fijo. En estos casos, la línea de ganancia de velocidad tiende

a saturarse. Este fenómeno es cierto para todos los sistemas paralelos y es descrito mediante la *Ley de Amdahl* (Definición 4.1.3, página 121).

Para muchos algoritmos paralelos, si se incrementa el tamaño del problema, W, con un número fijo de procesadores, p, entonces la eficiencia se incrementa, aproximándose a uno, porque la sobrecarga total del proceso paralelo crece menos que W. Para estos algoritmos paralelos la eficiencia puede mantenerse a un determinado valor (entre 0 y 1) con el incremento del número de procesadores, si el tamaño del problema también se incrementa. Estos algoritmos son calificados como escalables en [65]. El porcentaje en el que el tamaño del problema debe crecer, con respecto al número de procesadores, para mantener la Eficiencia fija, determina el grado de escalabilidad de los algoritmos paralelos y está determinado por la *función de isoeficiencia* [93, 94].

Como el reparto del trabajo entre los procesadores es una parte importante del algoritmo de Ramificación y Acotación paralelo, el modo de almacenamiento de la estructura de datos D, que contiene los subproblemas activos, es un parámetro a considerar. Otro parámetro a determinar es cuándo se accede a la estructura de datos D. Este acceso puede hacerse en momentos arbitrarios durante la ejecución, o los procesadores deben esperarse unos a otros, es decir, el algoritmo puede ser síncrono o asíncrono. La taxonomía presentada en [32] está basada en estos parámetros. En [39] se consideran otros parámetros adicionales, como la unidad de trabajo y la posibilidad de interrumpir un procesador durante su ejecución. A continuación se hace una descripción más detallada de estos parámetros, siguiendo las referencias anteriormente mencionadas.

Memoria compartida versus distribuida: Hay que establecer cómo se almacenan los subproblemas activos. Los extremos de la posible elección son el modelo de memoria compartida, denominado SDOM (Shared Data-Object Model) en [32] y el modelo de memoria distribuida, denominado DDM (Distributed Data Model) por los mismos autores. Este parámetro está estrechamente relacionado con la elección de una arquitectura particular (Sección 4.1.1, página 114).

- En el modelo distribuido, cada procesador mantiene su propia estructura de datos local D_i. Los procesadores se comunican mediante intercambios de subproblemas entre las estructuras de datos locales, que se organizan como listas ordenadas de problemas. Cuando se selecciona un nodo del subárbol de búsqueda local, éste se obtiene de la estructura de datos local D_i y los generados a partir de él son almacenados en D_i. La simplicidad de este modelo puede dar lugar a pensar que los algoritmos paralelos de Ramificación y Acotación son, en este caso, sencillos: cada procesador realiza una búsqueda secuencial en partes diferentes y disjuntas del árbol de búsqueda T. Sin embargo, no se conoce de antemano la estructura de T, y como los problemas son generados y seleccionados de un modo no predecible, en los algoritmos de Ramificación y Acotación paralelos aparecen irregularidades durante la búsqueda. Es probable que el algoritmo distribuido produzca una sobrecarga en la búsqueda, si se compara con

el algoritmo secuencial, ya que una región de T, que no se considere en el caso secuencial, puede ser asignada a un procesador para su evaluación. Además, un procesador puede quedarse sin trabajo al quedarse su estructura de datos local vacía. Las medidas a tomar para evitar estos problemas están basadas en técnicas de balanceo dinámico de la carga (véase la Sección 4.1.4), donde la carga computacional debe redistribuirse de forma equitativa entre los procesadores. Esta carga computacional puede definirse de varias formas, pero siempre teniendo en cuenta los subproblemas almacenados en las estructuras de datos locales, D_i. Varios experimentos han mostrado que este balanceo de la carga debe basarse en una función heurística h (Sección 1.5.1, página 27), de forma que se tenga en cuenta la calidad o coste de los subproblemas [1, 4, 61, 87, 93, 109].

- En el modelo de memoria compartida existe una sola estructura de datos D, que contiene el conjunto de nodos activos de T, almacenados como una lista ordenada de problemas. En este caso no se tiene en cuenta la estructura del árbol de búsqueda, evitando así los problemas inherentes a la irregularidad del mismo. Los procesadores deben comunicarse a través de la estructura de datos global, D [104]. En todo momento los procesadores mantienen D en un estado consistente con todos los subproblemas activos. La ventaja es clara: como D contiene todos los subproblemas a resolver, los procesadores pueden trabajar sobre los subproblemas más prometedores. De este modo se realiza un balanceo de la carga cualitativo y cuantitativo. La desventaja es que los procesadores tienen que realizar accesos frecuentes a la estructura de datos D, que se encuentra en la memoria compartida. El acceso a esta estructura de datos se convierte en un cuello de botella, cuando el número de procesadores es suficientemente grande.

Por lo anteriormente mencionado, los algoritmos de búsqueda tienen como estructura de almacenamiento lógica las estructuras de datos globales. Éstas se usan normalmente en máquinas de memoria compartida. Implementaciones directas de estas técnicas de búsqueda en computadores basados en paso de mensajes, dan a menudo como resultado un aumento de los costes de comunicación para acceder a las estructuras de datos compartidas. Para reducir las comunicaciones pueden usarse estructuras de datos distribuidas, pero estas estructuras pueden producir un incremento en la *sobrecarga de la búsqueda*. Los computadores de memoria compartida no son muy escalables, particularmente si todos los procesadores tienen la misma accesibilidad a toda la memoria global. Por este motivo, la mayoría de los computadores de memoria compartida están basados en una arquitectura de memoria distribuida, con hardware especial para soportar un direccionamiento global, como por ejemplo el Cray T3D [33]. Aunque el acceso a las memorias remotas está soportado por hardware, su coste excede el coste de un acceso a memoria local en dos ordenes de magnitud, por lo que, incluso en estos sistemas, las estructuras de datos necesarias para computadores basados en paso de mensajes son útiles.

Los dos modelos SDOM y DDM pueden verse como los extremos de un amplio espectro, existiendo otros modelos intermedios. Por ejemplo, los procesadores pueden dividirse en grupos, donde cada grupo se comporta de acuerdo al modelo de memoria compartida y entre grupos se aplica el modelo de memoria distribuida.

Unidad de trabajo: Otro parámetro a elegir es la unidad de trabajo. Hasta ahora se ha asumido implícitamente que D, global o local, era actualizada cada vez que un subproblema ha sido evaluado. También puede permitírsele a un procesador realizar una búsqueda limitada a partir de un subproblema, es decir, generar un subárbol de pocos niveles a partir del nodo seleccionado del árbol de búsqueda, y sólo entonces actualizar D con los nodos hojas de ese subárbol. De esta forma decrece la necesidad de realizar comunicaciones, pero al mismo tiempo decrece la calidad de los datos almacenados en D. Éste es otro ejemplo de decisión entre computación y comunicación.

Modo de sincronización: Un procesador tiene dos opciones cuando completa su unidad de trabajo: esperar a que los demás procesadores también completen su unidad de trabajo, o continuar sobre otra unidad de trabajo. Esperar a los demás procesadores puede ser favorable, especialmente en un modelo de memoria compartida, ya que los problemas seleccionados por los procesadores en cada iteración son los mejores, debido a que un procesador tiene un conocimiento completo sobre el estado del problema a la hora de seleccionar una nueva unidad de trabajo. De esta forma las implementaciones síncronas limitan la cantidad de trabajo innecesario que se pueda realizar, pero también produce estados de espera en los procesadores. En el caso asíncrono, un procesador puede no tener conocimiento de la información generada por otro procesador. Ésto puede dar lugar a resolver subproblemas cuya resolución no sería necesaria en un modelo síncrono. Por otro lado, las ventajas que ofrece son: un mayor potencial de ganancia de velocidad que el modelo síncrono, se solapan comunicación y computación y no hay puntos de sincronización para acceder a la estructura de datos.

Modo de interrupción: Para habilitar el uso de la información generada por otros procesadores, un procesador tiene que detectar la actualización de D. Existen dos modos básicos en los que un procesador se da cuenta de una actualización: el procesador chequea D en intervalos regulares de tiempo (normalmente en modelos SDOM) o es interrumpido cuando D es actualizada (normalmente en modelos DDM). Si se recibe una interrupción, el procesador deja de trabajar sobre la unidad de trabajo actual, chequea D y decide si continúa trabajando sobre la unidad de trabajo actual, o si realiza otra tarea que supuestamente es más apropiada. En el último caso, la unidad de trabajo actual se almacena en D.

La elección de estos parámetros no es siempre fácil. Los parámetros dependen de factores como el tipo de problema, es decir, la especificación de las reglas básicas de Ramificación

y Acotación, del ejemplo concreto del problema (el tamaño del conjunto de nodos activos puede variar mucho) y la arquitectura utilizada, ya que no todas las arquitecturas soportan unas especificaciones arbitrarias de los parámetros, los protocolos de comunicación pueden ser rápidos o lentos, etc.

4.2.1 Anomalías de los algoritmos de Ramificación y Acotación paralelos

La naturaleza heurística de los Algoritmos de Ramificación y Acotación aparece cuando en la selección del siguiente subproblema a tratar hay que decidir, entre varios con la misma prioridad, 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 los problemas, es difícil, si no imposible, el 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^* [97, 98] (Notas históricas pueden verse en [147]). 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 [55].

Definición 4.2.2 *Se define $I(p)$ como el número de iteraciones llevadas a cabo con p procesadores en un problema de Ramificación y Acotación. Se define la ganancia de velocidad con respecto al número de iteraciones realizado $S_I(p) = I(1)/I(p)$ [32].*

Generalizando, cuando se usa un modelo de comunicación síncrono, la ganancia de velocidad $S_I(p)$ está determinada solamente por la dificultad del problema (tamaño y forma de T), 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 $1 < S_I(p) < p$, como se vio anteriormente, principalmente cuando el número de procesadores es grande respecto a la dificultad del problema. Las anomalías de la ganancia de velocidad son de dos tipos: *Anomalías Aceleratorias*, donde $S_I(p) > p$ y *Anomalías Detrimentales*, donde $S_I(p) < 1$ [97, 98, 106]. Es obvio que se desea preservar las anomalías aceleratorias y evitar las detrimentales. Dentro de 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 Ramificación y Acotación síncronos [10, 97, 98, 99, 55, 105, 106, 114]. El caso asíncrono se ha tratado en [29, 30, 38], siempre en el modelo SDOM, debido a las dificultades que presenta el estudio del caso asíncrono en un modelo DDM.

Condiciones para prevenir anomalías detrimentales

Estas condiciones aparecen en algoritmos que usan una estrategia de selección *Primero el Mejor*, para encontrar una única solución óptima. En [97, 98] se demostró, que para algoritmos de *Ramificación y Acotación* paralelos síncronos, el número de iteraciones realizadas por el algoritmo secuencial no se incrementaba en la versión paralela, con $p > 1$, si la función de acotación (Definición 1.5.1, página 27) $li(v) \neq f^*$, no siendo v un nodo solución [106]. 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, si la función de prioridad heurística h es inyectiva y además es 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 1.5.6, página 31). Bajo estas condiciones los nodos divididos por el algoritmo secuencial, también llamados *nodos primarios*, no pueden ser eliminados por un test de dominancia [38]. El uso de una regla de prioridad heurística no ambigua es una condición suficiente, 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 [106].

En [114] 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*.

Condiciones para preservar anomalías aceleratorias

Las anomalías aceleratorias son una consecuencia del hecho de que los algoritmos de *Ramificación y Acotación* 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 [105, 106] 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 $f(v_1) < f(v_2) \Rightarrow li(v_1) < li(v_2),\ \forall v_1, v_2 \in V$.

Observando las condiciones para prevenir anomalías detrimentales, el que la función h sea completamente consistente con la función li puede dar lugar también a anomalías detrimentales. En ejemplos prácticos las anomalías detrimentales pueden estar escondidas detrás de anomalías aceleratorias.

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 [25]. Resultados sobre anomalías mostraron que árboles de búsqueda densos con *caminos críticos mínimos* cortos producían mejores rendimientos, usando algoritmos paralelos, que para árboles poco densos con *caminos críticos mínimos* largos [97]. 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* [97, 31], el problema del *Viajante de Comercio* [141, 147, 149], el problema de *Cobertura de Vértices* [183], para los problemas: de la *Mochila*, *Programación Entera* y *Cobertura de Vértices* [184] y para problemas de *Asignación Cuadrática* [107, 137, 160].

4.3 Paralelizaciones existentes de algoritmos de Optimización Global basados en intervalos

En esta sección se describirán los métodos de paralelización usados en los pocos algoritmos paralelos de Optimización Global basados en intervalos y Ramificación y Acotación existentes y sus resultados:

4.3.1 Henriksen y Madsen

El problema de tener varias listas de trabajo distribuidas desaparece si se usa una estrategia *maestro-esclavo*, donde todas la cajas están almacenadas en la lista de trabajo del procesador *maestro*, al igual que $\overline{f^*}$. En este modelo, implementado por Henriksen y Madsen [73, 74], el procesador *maestro* envía cajas a los procesadores *esclavos* y recoge de éstos los resultados (véase la Figura 4.4). Inicialmente la región de búsqueda S se divide en $p-1$ cajas, siendo $p-1$ el número de procesadores *esclavos*. No existe comunicaciones entre los procesadores *esclavos*. Debido a esto, las mejoras de $\overline{f^*}$, obtenidas por un procesador *esclavo*, se comunican a los demás procesadores *esclavos* mediante el procesador *maestro*.

Una desventaja de este método es que el procesador *maestro* tiene mucho trabajo para gestionar las peticiones y devoluciones de resultados de los procesadores *esclavos*. Las comunicaciones con el procesador *maestro* pueden llegar a ser un cuello de botella, si el número de procesadores *esclavos* crece por encima de un umbral. Otro inconveniente es que la longitud máxima de la lista está determinada por la memoria del procesador *maestro*;

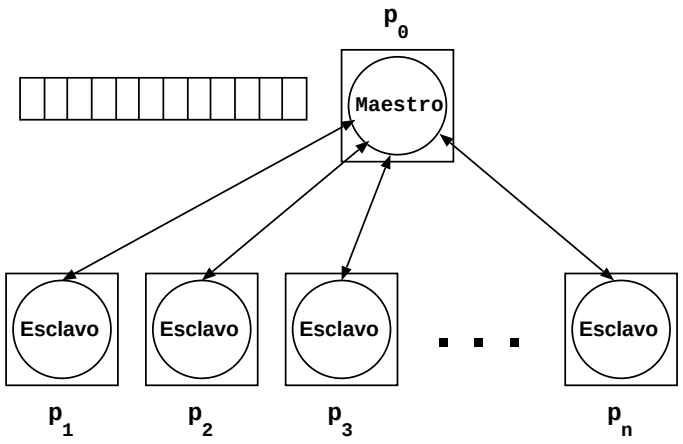


Figura 4.4: Esquema del método *maestro-esclavo*.

la memoria de los procesadores *esclavos* casi no se usa. Henriksen y Madsen aplicaron una estrategia de selección basada en una *Búsqueda en Profundidad* según el tamaño de las cajas, ya que sus requerimientos de memoria son menores que otras estrategias. El procesador *esclavo* se queda con una de las cajas generadas, resultado de la ultima división y envía la otra al procesador *maestro*, con lo que se reducen las comunicaciones, ya que sólo debe pedir cajas al procesador *maestro* cuando las dos cajas generadas se eliminan. Para obtener una buena eficiencia usando una *Búsqueda en Profundidad* hay que conocer de antemano un buen $\overline{f^*}$. En su algoritmo, Henriksen y Madsen usan el mínimo global, f^*, para inicializar $\overline{f^*}$, pero este valor normalmente se desconoce antes de ejecutar el algoritmo. Henriksen y Madsen proponen usar un método de Optimización Local para obtener una buena estimación inicial de f^*.

En [73] se comparan los resultados numéricos para dos funciones de prueba, con el mismo algoritmo, pero con distintas reglas de selección: usando una búsqueda basada en *Primero el Mejor* $\underline{F}(X)$ y una *Búsqueda en Profundidad*. Los resultados de la *Búsqueda en Profundidad* siempre fueron mejores, obteniendo una ganancia de velocidad de 28 y 16.5 con 1+32 procesadores, para cada una de las funciones, mientras que usando una búsqueda basada en *Primero el Mejor* $\underline{F}(X)$ se obtuvieron unas ganancias de velocidad de 16 y 3, para las mismas funciones. Henriksen y Madsen argumentan que si se conoce f^*, la *Búsqueda en Profundidad* es mejor debido principalmente a que el tiempo de gestión de la lista de trabajo es menor.

4.3.2 Eriksson y Lindström

Eriskson y Lindström proponen diferentes estrategias descentralizadas de balanceo dinámico, inicializadas por emisor y receptor, teniendo en cuenta la cantidad y la calidad de las cajas que migran. Los experimentos se realizaron sobre un Intel iPSC/2 [49], donde los

procesadores se organizaron formando un anillo. En cada procesador se ejecutaron dos procesos: un organizador y un *esclavo*. El proceso *esclavo* sólo pide y recibe cajas del organizador para su evaluación. El proceso organizador usa una estrategia de selección de la lista de trabajo basada en *Primero el Mejor* $\underline{F}(X)$. Además, el proceso organizador está encargado del balanceo de la carga y de detectar la terminación del algoritmo paralelo. Si se encuentra un mejor $\overline{f^*}$, el proceso *esclavo* lo comunica a todos los demás procesos *esclavos* de los otros procesadores.

Existen varias implementaciones, dependiendo de si el organizador pide o da cajas :

Organizador-Pide: El organizador pide cajas de un subconjunto de nodos cuando no tiene cajas o le quedan pocas. Este esquema está dirigido por la cantidad de cajas en la lista de trabajo local. Si algún procesador se queda sin cajas, envía una petición de cajas al siguiente en el anillo. Las peticiones son reenviadas hacia adelante por los procesadores, incorporando la petición de cada nuevo procesador, que tampoco tenga cajas. En su implementación un procesador con bastante trabajo no envía las cajas por el anillo, sino directamente al nodo que pidió trabajo y a los que en el camino del mensaje tampoco tenían cajas. Esta conexión directa se realiza gracias a que se usó una red de comunicaciones *cut-through* [44].

Organizador-Da: El organizador manda cajas a nodos con menos carga, sin petición previa de los nodos receptores. El organizador debe tener alguna información de la carga del sistema. Este esquema está dirigido por la calidad de las cajas en la lista de trabajo local. En el modelo Organizador-Da, se usaron los siguientes esquemas:

Distribución aleatoria: Si el número de cajas en la lista local excede un determinado umbral fijo, el proceso organizador envía la caja más prometedora de la lista a otro procesador elegido aleatoriamente.

Distribución aleatoria adaptativa: Intenta mejorar la anterior. El receptor informa al emisor si la caja recibida se inserta al principio de su lista o no (si ahora es la más prometedora o no). Si el emisor recibe un mensaje afirmativo, decrementa su umbral con lo que distribuye cajas más frecuentemente. Si recibe un mensaje negativo, incrementa dicho umbral.

Híbrido: Es una mezcla de los dos anteriores.

En la función con peor comportamiento de las seis de prueba, la eficiencia presentada por Eriksson y Lindström está por encima del 60% hasta 16 procesadores. Para otras funciones la eficiencia está por encima del 80% hasta 32 procesadores. Para 64 procesadores el comportamiento es variable con la función y con el tipo de esquema usado, siendo la distribución aleatoria adaptativa la que aportó mejores resultados.

4.3.3 Moore, Hansen y Leclerc

Moore, Hansen y Leclerc usaron una estrategia parecida a la la Eriksson y Lindström, donde todos los procesadores ejecutaban el mismo programa, pero usaban un conjunto de estaciones de trabajo conectadas mediante una red Ethernet [122]. Cuando un procesador se queda sin cajas en su lista de trabajo, se elige otro procesador aleatoriamente, para mandarle una petición de cajas y si este contesta que no puede mandar ninguna, se manda el mismo mensaje al que tiene un índice de procesador siguiente. Un procesador con cajas que recibe una petición, devuelve la mitad de su lista de trabajo, pero nunca un número mayor que un umbral establecido, para evitar mensajes largos. Este balanceo de la carga está dirigido por la cantidad de cajas en la lista de trabajo y no por su calidad. El método de selección utilizado se basa en una *Búsqueda en Anchura*. Para la función de prueba en la que se mostraron resultados de eficiencia se alcanzó una ganancia de velocidad de 170 con 32 procesadores, pero esto fue debido a que el algoritmo secuencial era ineficiente.

En un artículo posterior de Leclerc [103], usando un algoritmo secuencial muy parecido al de Moore *et al* [122], pero aplicando una estrategia *Primero el Mejor* $\underline{F}(X)$ y un balanceo dirigido por la calidad de las cajas, se obtuvo una mejora de un 78% en la resolución del problema anterior, comparado con los resultados obtenidos con el algoritmo de Moore *et al* [122]. La resolución de este problema en la nueva versión paralela del algoritmo sólo alcanzó la mitad de la ganancia de velocidad lineal.

4.3.4 Berner

Sonja Berner [8] intenta combinar en su algoritmo las ventajas de los modelos centralizado y distribuido (véase la Figura 4.5). En esta implementación, cada procesador tiene su propia lista de trabajo y usa una estrategia *Primero el Mejor* $\underline{F}(X)$ para seleccionar cajas de la lista de trabajo. Al contrario que en el modelo distribuido, se elige un procesador para actuar como *mediador centralizado*. Este procesador no trabaja sobre las cajas, sino que espera peticiones de los demás procesadores para enviarles cajas. El *mediador centralizado* establece un umbral, *max*, que cambia dinámicamente dependiendo del tamaño de su lista de trabajo y de las peticiones recibidas. Este umbral se usa para asegurarse que no se queda sin cajas y para no almacenar demasiadas. Los procesadores con un número de cajas mayor que *max* envían algunas de ellas al *mediador centralizado*. Se eligen la segunda, cuarta, etc, de la lista de trabajo para ser enviadas. La lista se encuentra ordenada por los valores no decrecientes de $\underline{F}(X)$. De esta forma se mandan cajas prometedoras pero también se mantienen cajas prometedoras en la lista de trabajo. Si un procesador encuentra un valor mejor de $\overline{f^*}$ debe comunicarlo a los demás procesadores. Además de estas comunicaciones hay que sumar las debidas a la actualización por parte del *mediador centralizado* de la variable *max*, que deben conocer los demás procesadores.

La idea del *mediador centralizado* fue tomada de Smith y Schnabel [168], donde se usó para paralelizar un método clásico de Optimización Global. En [168] se probaron también otros modelos de balanceo de la carga, obteniéndose los mejores resultados con

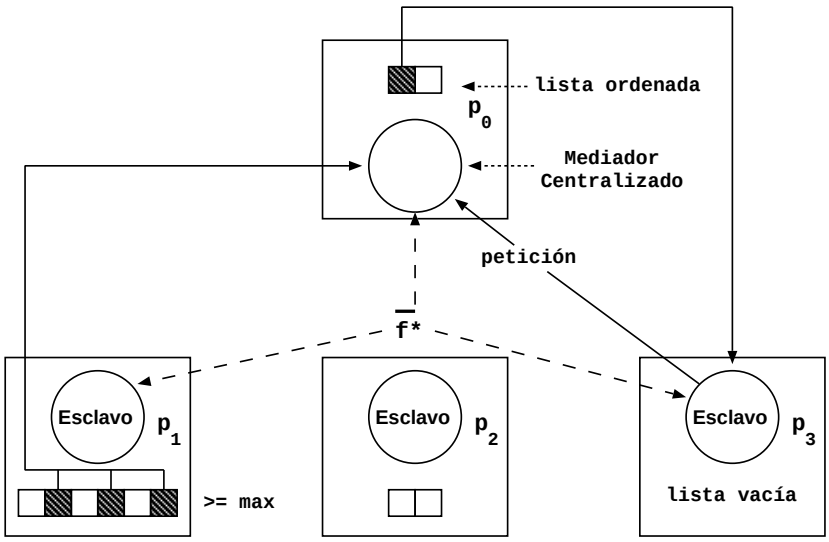


Figura 4.5: Estructura de comunicación con un *mediador centralizado*.

el modelo basado en el *mediador centralizado*.

En esta estrategia el *mediador centralizado* realiza menos trabajo que el procesador que actúa como *maestro* en una estrategia *maestro-esclavo*, además de aprovechar la memoria de los demás procesadores. Comparada con una estrategia distribuida, cuando un procesador se queda sin trabajo, no existe la necesidad de pedir cajas a varios procesadores. El principal inconveniente es que el *mediador centralizado* no realiza cómputo, sólo control, pero esta pérdida de cómputo se ve compensada por la reducción de las comunicaciones.

El algoritmo anterior es una simplificación del algoritmo real usado por Berner en el que introduce las siguientes mejoras:

- En la fase inicial, donde cada procesador obtiene su subregión de búsqueda a partir de la región inicial, cada procesador inicializa una búsqueda local desde el punto medio de su caja con lo que se obtiene un mejor valor de $\overline{f^*}$, por lo que se descartarán más cajas gracias al *Test del Punto Medio* y *de Corte*. Después de obtener un buen $\overline{f^*}$, se realiza un balanceo de la carga síncrono para realizar una buena distribución de las cajas. Esto mejora el algoritmo para funciones con un tiempo de ejecución alto, pero no para aquellas menos costosas, ya que el obtener un buen $\overline{f^*}$ no se ve compensado por el tiempo necesario para realizar la búsqueda local y el balanceo síncrono.

- Para obtener un buen balanceo, en cuanto a la calidad de las cajas, y mantener un nivel relativamente bajo en las comunicaciones, cuando un procesador encuentra un mejor valor de $\overline{f^*}$ lo comunica a los demás procesadores, junto con el valor $\underline{F}(X)$ de su mejor caja. Normalmente, los procesadores que encuentran mejores valores de $\overline{f^*}$ tienen mejores cajas, por lo que otro procesador que reciba esta información y

vea que la calidad de sus cajas es suficientemente baja, respecto a la del procesador emisor, le pedirá cajas para mejorar la calidad de su lista de trabajo.

- Algunas veces el *mediador centralizado* puede tener menos cajas que peticiones. Debido a que el trabajo necesario para investigar completamente una caja no se conoce de antemano, Berner propone que el *mediador centralizado* divida algunas cajas, en vez de dejar a los procesadores esperando.

El algoritmo implementado por Berner se ejecutó sobre una Connection Machine CM5 con 32 procesadores. En su trabajo, Berner usó un amplio número de problemas de prueba que clasificó en tres categorías: pequeños ($t_{sec} < 100$ sg), medianos (100 sg $\leq t_{sec} < 1000$ sg) y grandes (1000 sg $\leq t_{sec} < 17000$ sg). Para los problemas grandes se observó que la ganancia de velocidad crece con la complejidad de los problemas usados, alcanzando para algunos problemas una ganancia de velocidad superlineal. Para los problemas medianos, con 16 procesadores, sólo dos problemas de los siete examinados no alcanzaron una ganancia de velocidad superlineal. Para 32 procesadores la eficiencia obtenida estaba entre el 43% y el 115%. En cuatro de los siete problemas se alcanzó una ganancia de velocidad superlineal. Para problemas pequeños, la eficiencia no fue tan buena. En el peor de los casos se obtuvo una eficiencia del 18% con 32 procesadores, pero el tiempo secuencial de este problema era de 4.7 sg.

La principal conclusión de Berner es que los problemas con un solo mínimo global son poco paralelizables, principalmente cuando las cajas que no contienen el mínimo global pueden eliminarse rápidamente. Por otro lado, si existen al menos p mínimos globales y éstos se encuentran en cajas distintas, cuando el nivel del árbol de búsqueda es pequeño, entonces es posible realizar una buena paralelización, ya que el número de cajas es suficiente para mantener a todos los procesadores ocupados.

4.4 Nuevo criterio de balanceo de la carga en algoritmos paralelos de Optimización Global basados en intervalos

Las implementaciones paralelas que han sido comentadas en este capítulo, se podrían caracterizar globalmente porque la heurística utilizada para obtener un buen balanceo de la carga computacional se basa en la evaluación de los valores del limite inferior de la función de inclusión en cualquier intervalo X ($\underline{F}(X)$). Este valor ha sido utilizado como un estimador de la carga computacional asociada a cualquier hoja del árbol de subproblemas. Además, la mayoría de las implementaciones hacen uso de dispositivos aceleradores tales como el test de monotonía, el test de concavidad, o el *Método de intervalos de Newton*, con lo cual se restringe el conjunto de posible funciones sobre las que puede ser aplicado, al conjunto de funciones diferenciables.

Los resultados experimentales del Capítulo 3 han mostrado que el parámetro $p\overline{f^*}(X)$ es un buen estimador de la cantidad de región de atracción hacia un mínimo que contiene una determinada caja X, el cual está directamente relacionado con el trabajo que hay que

realizar en dicha caja. Por este motivo, en esta tesis se propone como estimador de la carga computacional de una caja X, el valor del producto $w(X)\cdot\overline{pf^*}(X)$ [12, 14]. Este estimador tiene también en cuenta el tamaño de la caja, siguiendo la heurística de que habrá que realizar más trabajo computacional sobre las cajas grandes con un valor de $\overline{pf^*}(X)$ alto, que en las cajas pequeñas, que se encuentren cerca del criterio de terminación, y con un valor pequeño de $\overline{pf^*}(X)$.

En los algoritmos de Ramificación y Acotación, la primera decisión a considerar es cómo y con qué problemas se inicializan cada uno de los EP. En [72] los métodos de inicialización estudiados fueron:

Raíz : La raíz del árbol de búsqueda se asigna a un solo EP, que comienza la ejecución del algoritmo de Ramificación y Acotación. Los subproblemas generados a partir de la raíz del árbol de búsqueda se distribuyen a los otros EPs, de acuerdo con el esquema de balanceo de la carga aplicado en el algoritmo. Los otros EPs, al recibir los subproblemas, actúan de igual forma. Después de un tiempo, todos los EPs deben tener subproblemas en los que trabajar. La ventaja de esta estrategia es que no hay que introducir en el algoritmo un código específico para la etapa de inicialización. El principal inconveniente es que muchos de los EPs están ociosos, en espera de subproblemas.

Enumerativa : Cada EP comienza la ejecución del algoritmo partiendo del problema total, es decir, de la raíz de árbol, hasta obtener al menos p subproblemas en su lista de trabajo. Entonces cada EP^i, $i = 1, \ldots, p$, continúa trabajando sobre el subproblema X^i, eliminando los demás. Si se crearon k subproblemas, con $k > p$, el EP^i continuará trabajando sobre los problemas X^l : $(l \mod p) = i - 1$. La principal ventaja del método *enumerativo* frente al *raíz* es que no se realiza ningún tipo de comunicación entre los EPs. Sin embargo, se realiza un trabajo redundante, porque todos los EPs procesan los mismos subproblemas, realizando una evaluación idéntica.

Selectiva : Este método evita las desventajas de la inicialización *enumerativa*. Al contrario que la inicialización *enumerativa*, en vez de generar todo el árbol de búsqueda hasta una cierta profundidad, cada EP genera sólo una rama del árbol. Los caminos del árbol de búsqueda que cada EP toma son distintos y se dirigen hacia el subproblema apropiado para inicializar el EP correspondiente. Cada EP tiene un número que lo identifica, $EPid$. Si denotamos por nEP al número de EPs que en un momento dado toman el mismo camino en el árbol de búsqueda y $size$ el número de subproblemas generados en cada división, entonces el elemento de proceso $EPid$ selecciona el subproblema denotado por g, con:

$$g = ((EPid - 1) \mod size) + 1 \tag{4.5}$$

En la siguiente iteración, se actualizan las variables siguiendo las ecuaciones:

$$\begin{aligned} delta &= \begin{cases} 1, & g \leq (nEP \mod size) \\ 0, & \text{en otro caso} \end{cases} \\ EPid &= \lceil EPid/size \rceil \\ nEP &= \lfloor nEP/size \rfloor + delta \end{aligned} \tag{4.6}$$

La etapa de inicialización termina cuando $size \leq nEP$. Entonces, cada EP toma un número de subproblemas igual a $\lfloor size/nEP \rfloor$. El último EP toma el resto de los subproblemas. La ventaja de una inicialización selectiva frente a la enumerativa es que el trabajo redundante es menor. Sin embargo, los EPs no se inicializan con los mejores subproblemas, ya que unas ramas del árbol se generan independientemente de las otras. Por este motivo, un EP puede quedarse sin trabajo rápidamente al terminar la fase de inicialización, si sus subproblemas iniciales son eliminados cuando se recibe un mejor valor de $\overline{f^*}$, generado en otro EP.

Directa : Este modo de inicialización evita la construcción explícita del árbol de búsqueda. Debido a que cada EP debe inicializarse con un subproblema distinto, todos los subproblemas iniciales se toman directamente de una cierta profundidad del árbol de búsqueda, en vez de comenzar la búsqueda por la raíz. Esta profundidad se elije de forma que el número de subproblemas en esta profundidad sea mayor que el número de EPs. De esta forma, cada EP genera directamente un subproblema como mínimo. Si existen más subproblemas que EPs, entonces algunos EPs deben tener más de un subproblema inicial para asegurar la correcta ejecución del algoritmo. Esta estrategia tiene, como en los dos métodos anteriores, la ventaja de que no se realizan comunicaciones en la etapa de inicialización, pero además cada EP no realiza un trabajo redundante. Sin embargo, al igual que en la inicialización *selectiva*, cada EP no trabaja sobre los mejores subproblemas iniciales.

Los análisis y experimentos realizados en [72] demuestran que el método de inicialización *directa* aporta mejores resultados. Su principal inconveniente es que algunos EPs pueden quedarse ociosos, debido a la eliminación de sus problemas iniciales, hasta que el balanceo de la carga dinámico les permita obtener nuevos subproblemas. El número de subproblemas iniciales eliminados dependerá del problema concreto a tratar y del número de EPs, por lo que no se puede establecer que la inicialización *directa* sea la mejor estrategia en el caso general.

A continuación se describen y analizan los tres modelos de balanceo de la carga computacional que se proponen en este trabajo, los cuales definen las tres implementaciones paralelas del problema de Optimización Global mediante Ramificación y Acotación que se estudian en esta tesis y que hacen uso de una estrategia de inicialización directa.

4.4.1 Modelo estático

En el modelo estático cada *EP* aplica el Algoritmo 3.1.1 (página 63) con una *Regla de División* basada en realizar una bisección de todas las coordenadas a la vez, a sus cajas iniciales. Los *EPs* sólo intercambian las mejoras obtenidas en el valor de $\overline{f^*}$. Este modelo de balanceo de la carga se ha implementado para resaltar la insuficiencia de un modelo de balanceo estático en los problemas de prueba usados.

4.4.2 Modelo semi-estático

El modelo semi-estático está basado en la idea del *mediador centralizado*. En este modelo se pueden distinguir dos etapas a partir de la inicialización *directa*:

Primera etapa:

En esta etapa se persiguen dos objetivos: por un lado, encontrar un buen límite superior del mínimo de la función en la región de búsqueda ($\overline{f^*}$) y por otro lado, dividir la región de búsqueda de una manera eficiente, para poder realizar posteriormente un buen balanceo de la carga. Para llevar a cabo estos objetivos, cada procesador *esclavo* aplica una *Regla de Selección Híbrida* (Sección 3.5.2, página 99) sólo en las cajas iniciales; las cajas generadas a partir de las cajas iniciales no se tratan en esta etapa. Si cada *EP* tiene más de una caja inicial, es posible que se elimine alguna de las cajas iniciales, si en la búsqueda en profundidad realizada sobre otra caja inicial se ha encontrado un valor de $\overline{f^*}$, que permita aplicar la *Regla de Eliminación*. Por este motivo, de entre las cajas iniciales se elige primero la que consideramos más prometedora, es decir, la que tiene un mayor valor de $p\overline{f^*}(X)$, ya que $w(X)$ es igual para todas, permitiendo así obtener una mejora de $\overline{f^*}$ rápidamente.

En la Figura 4.6 se muestra un ejemplo de ejecución de esta primera etapa para la función Goldstein-Price. En todas las gráficas presentadas, el número de *EPs* es fijo e igual a cuatro. La principal diferencia entre las gráficas es el número de cajas iniciales, asignadas a cada *EP*, sobre las que se realiza una búsqueda en profundidad. Con estas gráficas se pretende resaltar cómo el nivel de división del espacio de búsqueda depende del número de cajas iniciales y cómo se realiza una mayor división de las zonas de atracción hacia mínimos globales y locales, cuanto mayor es el número de cajas iniciales. La principal desventaja de aumentar el número de cajas iniciales es que la sobrecarga de la búsqueda también aumenta. Por lo tanto, hay que establecer un número de cajas iniciales, suficiente para realizar una buena división del espacio de búsqueda, pero que a la vez no incremente demasiado la sobrecarga de la búsqueda.

En la gráfica superior izquierda se han establecido los niveles de grises que indican el valor aproximado de $p\overline{f^*}(X)$ en cada caja. Las cajas rechazadas no han sido dibujadas. En el gráfico superior derecho cada uno de los cuatro *EPs* se inicializa con el número mínimo de cajas posibles. Si se realiza una bisección de todas las coordenadas a la vez, el valor

mínimo de cajas iniciales vendrá determinado por la relación:

$$2^{nl} \geq p, \tag{4.7}$$

donde l representa el nivel del árbol de búsqueda alcanzado y n la dimensión del problema. Para el caso de la función bidimensional Goldstein-Price y cuatro procesadores *esclavos*, el número de cajas iniciales es claramente de cuatro. Como puede observase en la gráfica superior derecha de la Figura 4.6, el uso del número mínimo de cajas iniciales puede dar lugar a desbalanceo de la carga en esta etapa, por lo que usar un número mayor de cajas iniciales parece aconsejable. Este desbalanceo se produce porque las búsquedas en profundidad realizadas sobre las cajas iniciales localizadas en la parte inferior del espacio de búsqueda, convergen hacia el mínimo global, mientras que las búsquedas en profundidad realizadas sobre las cajas iniciales de la mitad superior del espacio de búsqueda no llegan ni siquiera a alcanzar el criterio de terminación, que para los ejemplos mostrados en la Figura 4.6 es de $\epsilon = 10^{-2}$.

En la gráfica inferior izquierda se ha aumentado el número de cajas iniciales por EP a cuatro. Cuando el número de cajas iniciales es mayor que el número de EPs, se utiliza una distribución de datos cíclica por filas de las cajas iniciales. Este aumento de cajas iniciales permite una mejor división de la región de atracción al mínimo global y realizar algunas divisiones sobre regiones de atracción a mínimos locales.

En la gráfica inferior derecha, el numero de cajas iniciales se ha incrementado a dieciséis por EP. Como puede observarse, la división del espacio de búsqueda es aún mejor.

A partir de estos experimentos, se ha decidido que si el número mínimo de cajas iniciales se alcanza para un nivel del árbol de búsqueda l (ecuación (4.7)), entonces se usará un valor del nivel del árbol de búsqueda l_{ini}, con:

$$l_{ini} = l + 2 \tag{4.8}$$

La primera etapa acaba una vez que los procesadores *esclavos* terminan las búsquedas en profundidad sobre sus cajas iniciales y mandan las subcajas generadas al *mediador centralizado*.

En otras implementaciones, como las realizadas por Henriksen y Madsen [73, 74] y Berner [8], se obtiene un buen valor de $\overline{f^*}$ mediante un método de Optimización Local. La ventaja de búsqueda en profundidad basada en el valor de $p\overline{f^*}$, utilizada en esta etapa, comparada con los métodos de Optimización Local basados en evaluaciones puntuales de la función objetivo, es, que además de encontrar un buen valor de $\overline{f^*}$, permite avanzar en la ejecución del algoritmo de Ramificación y Acotación, realizando una buena partición del árbol de búsqueda, que posteriormente permite realizar un buen balanceo de la carga.

Segunda Etapa.

En la segunda etapa, el *mediador centralizado*, después de recibir las subcajas de todos los *esclavos*, hace una estimación de la carga de trabajo total:

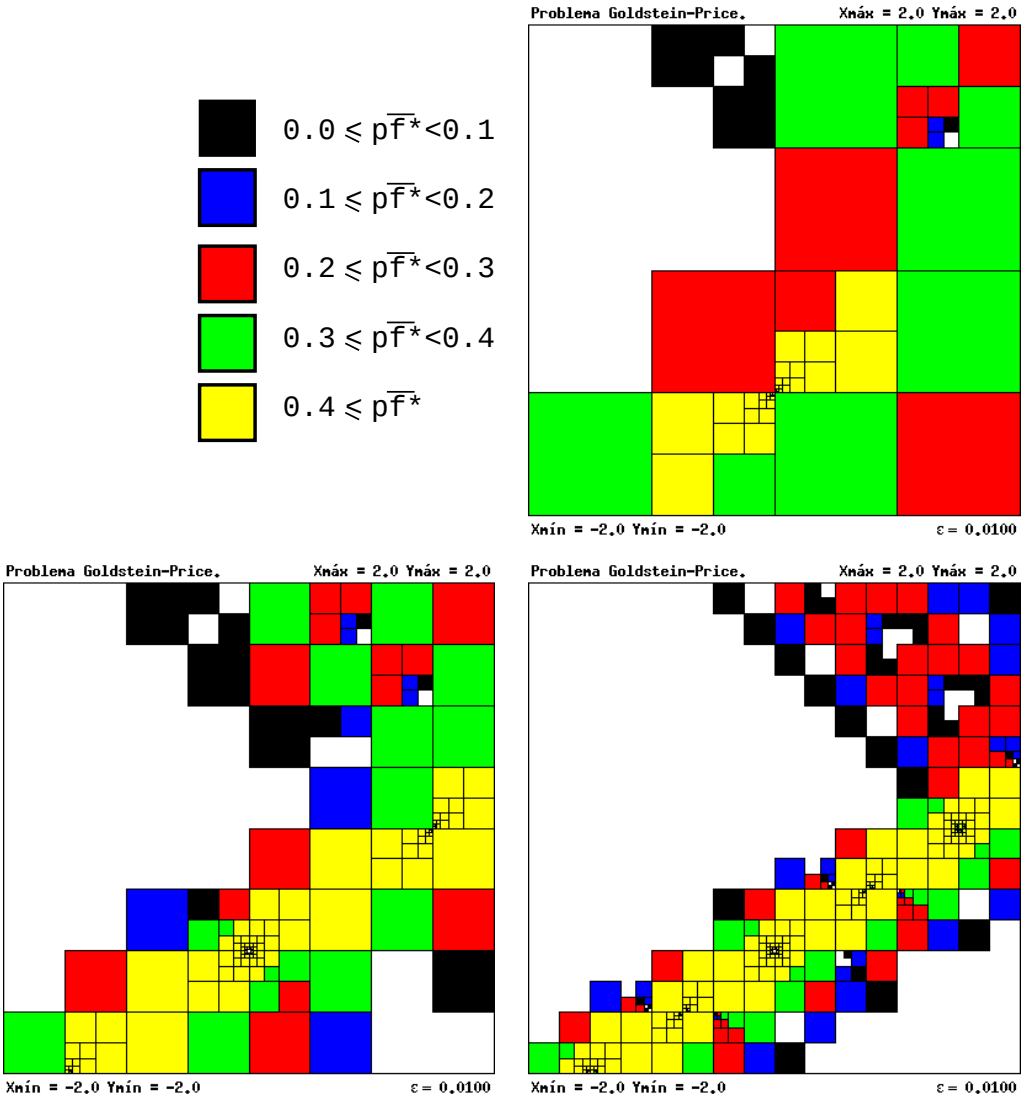


Figura 4.6: Primera etapa del modelo semi-estático sobre la función GP, para 4 *esclavos*, 1, 4 y 16 cajas iniciales por *esclavo* y $\epsilon = 10^{-2}$.

$$CT_t = \sum_i p\overline{f^*}(X^i) \cdot w(X^i), \ \forall X^i \in T. \tag{4.9}$$

A partir de la carga total estimada, el *mediador centralizado* distribuye un conjunto de cajas con una carga computacional aproximadamente igual a CT_t/p, entre los procesadores *esclavos*. Este conjunto de cajas se establece asignándole al procesador *esclavo* con menor

carga asignada, la caja con una carga tal, que haga que la carga total de ese procesador se aproxime lo más posible a CT_t/p, realizándose un balanceo de la carga cuantitativo y cualitativo. Esto se repite mientras queden cajas sin asignar.

Cuando los procesadores *esclavos* reciben las nuevas cajas, aplican el Algoritmo 3.1.1 sobre las cajas recibidas. Suponiendo que en la etapa inicial se ha encontrado el mejor valor de $\overline{f^*}$, el método de selección usado en la segunda etapa no va a variar el número de evaluaciones de la *función de inclusión*. En esta etapa se continuará usando una *Regla de Selección Híbrida*, ya que simplifica la implementación del algoritmo y al estar basada en una búsqueda en profundidad, también simplifica la gestión de las estructura de datos local, reduciendo así el tiempo de ejecución en cada procesador *esclavo*.

El algoritmo paralelo termina cuando todos los procesadores *esclavos* envían al procesador central sus cajas finales. Por lo tanto, en este modelo sólo se hace una redistribución dinámica de la carga.

4.4.3 Modelo dinámico

El modelo dinámico es similar al modelo semi-estático, pero ahora en vez de aplicar una sola vez la estrategia dinámica de balanceo de la carga, ésta se realiza las veces que el *mediador centralizado* necesite. En este modelo, siempre se aplica el Algoritmo 3.1.1 con una *Regla de Selección Híbrida*. El número de cajas iniciales por EP se determina de la misma forma que en el modelo semi-estático. Cuando un procesador *esclavo* no tiene más cajas en su estructura de datos local, se lo comunica al *mediador centralizado*. Si el *mediador centralizado* tampoco tiene cajas para servir, pide cajas a los demás procesadores *esclavos*. Los procesadores *esclavos* que tienen cajas para procesar, mandarán al *mediador centralizado* la segunda, cuarta, etc, que están ordenadas por $w(X) \cdot p\overline{f^*}(X)$. Una vez que el *mediador centralizado* recibe las cajas, da a los procesadores *esclavos* en espera de trabajo, un conjunto de cajas con una carga aproximadamente igual a $CT_t/(p-1)$. La carga total se ha dividido entre $p-1$ y no entre p, porque se supone que al menos un procesador tiene cajas para procesar y de esta manera la carga de trabajo que se manda a cada procesador *esclavo*, en espera, es mayor. Este valor permanece constante hasta que el procesador central tenga que volver a pedir más cajas.

El programa termina cuando, ni el *mediador centralizado*, ni los procesadores *esclavos* tienen cajas para procesar y el *mediador centralizado* recoge las cajas finales de los procesadores *esclavos*.

Las principales diferencias entre los modelos presentados son:

1. El modelo estático es un modelo distribuido mientras que los modelos semi-estático y dinámico son centralizados.
2. En el modelo estático, la sobrecarga debida a las comunicaciones se produce sólo por la comunicación de la mejora del valor de $\overline{f^*}$, pero existe un fuerte desbalanceo de la carga, mientras que el modelo dinámico tiene una mayor sobrecarga debido a las comunicaciones, con un alto grado de balanceo de la carga.

3. El modelo semi-estático puede clasificarse como una mezcla del modelo estático y del dinámico.

4.4.4 Resultados numéricos

Los modelos estático, semi-estático y dinámico, del algoritmo paralelo de Optimización Global basado en intervalos, han sido implementados usando la librería de paso de mensajes PVM-3.4.0-1, sobre un multicomputador de memoria distribuida llamado CAESARG [B.2.2][117], con $1 + p$ elementos de proceso, que consta de los siguientes componentes:

- Hub Fast Ethernet
- Máquina Front-End
  - Pentium-II 233 MHz
  - 256 Mb RAM
  - 4Gb UltraWide SCSI HDD
  - Tarjeta 10/100 Fast Ethernet
- Siete nodos
  - Pentium-MMX 200 MHz
  - 32 Mb RAM
  - Tarjeta 10/100 Fast Ethernet
  - Sin HDD

Se ha utilizado esta máquina paralela, de bajas prestaciones, si se compara, por ejemplo, con una Origin 2000, basada en el microprocesador R10000, o un Cray T3E, basado en el microprocesador Alpha 21064, porque el paquete BIAS [90], que proporciona las subrutinas para realizar operaciones de Aritmética de Intervalos, utilizado en las implementaciones, sólo está disponible para los microprocesadores IBM RS/6000, Sparc 680x0 y PC 80386 o superior (con coprocesador). Para los microprocesadores anteriores, el paquete BIAS proporciona las subrutinas ensamblador que permiten el intercambio entre los modos de redondeo especificados en el estándar IEEE-754. Otros computadores paralelos, como por ejemplo el Cray T3D, aunque están basados en la normativa IEEE-754, no permiten al usuario el intercambio entre los distintos modos de redondeo [B.4.2]. También existen microprocesadores con un formato especifico para representar los datos, proporcionado por el fabricante. Los ejemplos más representativos de estos computadores son: los Cray: X-MP, Y-MP, C-90 y T-90, los IBM: /370 y 3090, y el DEC VAX [B.4.1].

El principal inconveniente del multicomputador CAESARG es la baja velocidad de la red de comunicaciones y que el ancho de banda de la red es compartido por todos los procesadores. Debido al pequeño número de EPs de que dispone, se decidió utilizar una implementación del balanceo de la carga centralizada.

En las implementaciones de los algoritmos se ha intentado reducir la sobrecarga de las comunicaciones a costa de incrementar la sobrecarga de la búsqueda. Para ello se ha utilizado un modelo de comunicación asíncrono y en los modos de balanceo semi-estático y dinámico, no se realizan comunicaciones hasta que se termine la búsqueda en profundidad realizada sobre la caja seleccionada. Esto último es debido a que en las etapas iniciales del algoritmo, es normal que casi todos los procesadores encuentren mejoras de $\overline{f^*}$, las cuales deben comunicarse a los otros procesadores *esclavos*. La sobrecarga de la búsqueda que se puede producir por una búsqueda en profundidad innecesaria, realizando una bisección en todas las dimensiones, será menor o igual a:

$$2^n(lmax - lini), \tag{4.10}$$

donde l_{ini} es el nivel del árbol de búsqueda alcanzado en la fase de inicialización directa, que está determinado por las ecuaciones (4.8) y (4.7), y l_{max} es el número máximo de niveles del árbol que se puede alcanzar y está determinado por:

$$w(X)/2^{l_{max}} \leq \epsilon \tag{4.11}$$

Para la función Goldstein-Price sobre la región de búsqueda $[-2,2]^2$, y $\epsilon = 10^{-3}$, $l_{max} = 12$. Para cuatro procesadores, $l_{ini} = 3$, por lo que se evaluarían 36 cajas como mucho, antes de comunicar, o chequear la posible recepción, de una mejora de $\overline{f^*}$.

Debido al modo de comunicación asíncrono utilizado, que hace que la ejecución del algoritmo paralelo sea no determinista, en las Tablas 4.1 a 4.3 y en la Figura 4.7 se muestran la media de los valores obtenidos en cinco ejecuciones de las distintas versiones del algoritmo paralelo, para el tiempo de CPU consumido, ganancia en velocidad, desbalanceo (DBL), número de evaluaciones de $F(X)$ y de $F(m(X))$, para un valor de $\epsilon = 10^{-3}$. Los problemas de prueba utilizados fueron: GP, H3, L3 y C, que se detallan en el apéndice A. Debido al bajo coste computacional requerido para hacer una evaluación de estas funciones, lo que las hace inapropiadas para su resolución en paralelo, se multiplicó por cien el tiempo necesario para la evaluación de cada una de las funciones de inclusión. Aún así, los tiempos de evaluación de las cuatro funciones que se usan en este tesis como funciones de prueba, se encuentran en el intervalo $[10^{-4}, 10^{-2}]$ segundos, obtenidos en un Pentium-MMX 200 MHz.

La medida de desbalanceo usada se define como:

$$DBL = (I_{max} - I_{av})/I_{av} \in [0, p-1], \tag{4.12}$$

donde I_i es el número de evaluaciones de la *función de inclusión*, hechas por el EP_i, es decir, el número de nodos visitados del árbol de búsqueda, $I_{max} = \max\{I_i\}$ y $I_{av} = \sum_i I_i/m,\ 1 \leq i \leq m$.

En la Tabla 4.1 y la gráfica de la Figura 4.7 que muestra la ganancia de velocidad para el modelo estático, puede observarse que para la función tri-dimensional H3, con un solo mínimo global, una distribución estática de la carga es absolutamente ineficiente,

Tabla 4.1: Resultados numéricos para el método estático $\epsilon = 10^{-3}$.

p	1	3	5	7
Tiempo de CPU (sg.)				
GP	135.93	73.55	93.16	39.41
H3	1162.24	1155.72	1155.45	1147.26
L3	487.84	271.47	157.96	110.54
C	21.42	10.18	10.17	10.08
Ganancia de velocidad				
GP	1.00	1.85	1.46	3.45
H3	1.00	1.01	1.01	1.01
L3	1.00	1.80	3.09	4.41
C	1.00	2.10	2.11	2.13
DBL				
GP	0.000	0.607	2.470	1.094
H3	0.000	1.980	3.964	5.946
L3	0.000	0.655	0.617	0.550
C	0.000	0.416	1.337	2.253
Evaluaciones de $F(X)$				
GP	169209	169403	169289	169396
H3	283409	283678	283798	283944
L3	32765	32764	32760	32760
C	33413	33412	33667	33661
Evaluaciones de $F(m(X))$				
GP	42302	42349	42318	42345
H3	35426	35458	35473	35492
L3	8191	8190	8186	8186
C	8353	8352	8412	8411

obteniéndose un valor del desbalanceo de la carga cercano al máximo posible. Sin embargo, para la función bidimensional L3, con nueve mínimos globales, que están bien distribuidos en la región de búsqueda, una distribución estática de la carga permite obtener unos resultados que no son demasiado ineficientes.

En la Tabla 4.2 y la Figura 4.7, se muestran los resultados para el algoritmo con un balanceo de la carga semi-estático. Se han diferenciado los valores del desbalanceo de la carga para la primera y segunda etapa. Comparando los resultados obtenidos con los del modelo estático, puede observarse que existe una sobrecarga en la búsqueda ligeramente mayor en el modelo semi-estático, producida por el uso de una *Regla de Selección Híbrida*, frente a una *Regla de Selección* basada en *Primero el Mejor* $\underline{F}(X)$ del modelo estático, y porque en el modelo semi-estático no se comunica la mejora de $\overline{f^*}$ hasta terminar una búsqueda en profundidad. Los resultados obtenidos con el modelo semi-estático prueban que un solo balanceo dinámico de la carga basado en el nuevo estimador de la carga

Tabla 4.2: Resultados numéricos para el método semi-estático. $\epsilon = 10^{-3}$

p	1	3	5	7
Tiempo de CPU (sg.)				
GP	158.27	58.36	33.18	23.44
H3	1262.12	478.92	325.43	320.88
L3	488.85	172.92	134.70	86.21
C	24.93	11.81	6.63	5.25
Ganancia de velocidad				
GP	1.00	2.71	4.77	6.75
H3	1.00	2.64	3.88	3.93
L3	1.00	2.83	3.63	5.67
C	1.00	2.11	3.76	4.75
DBL				
GP	0.000	0.076-0.098	0.004-0.038	0.008-0.025
H3	0.000	0.075-0.138	0.084-0.299	0.274-0.774
L3	0.000	0.130-0.037	0.191-0.333	0.096-0.186
C	0.000	0.190-0.421	0.021-0.307	0.027-0.463
Evaluaciones de $F(X)$				
GP	169213	169280	171413	171457
H3	283409	283888	283968	284048
L3	32765	33080	35392	35360
C	33449	33500	36360	36376
Evaluaciones de $F(m(X))$				
GP	42303	42304	41829	41840
H3	35426	35422	35432	35442
L3	8191	8254	7824	7816
C	8362	8359	8066	8070

computacional, permite mejorar, no sólo el balanceo de la carga y la ganancia de velocidad del algoritmo paralelo, sino también la eficiencia del algoritmo, en cuanto al tiempo de CPU requerido. Para la resolución de los problema tratados en este trabajo, las mejoras oscilan entre un 22% para la función L3 y un 72% para la función H3.

Nótese que en la Figura 4.7 la escala para el desbalanceo es diferente en cada gráfica, decreciendo significativamente desde el modelo estático al dinámico.

En la Tabla 4.3 y la Figura 4.7 se muestran los resultados para el algoritmo que aplica un modelo dinámico de balanceo de la carga. Puede observarse que la sobrecarga de la búsqueda del modelo dinámico y la obtenida en el modelo semi-estático son muy parecidas. También se observa que el desbalanceo de la carga está por debajo de $3 \cdot 10^{-2}$ para los problemas de prueba utilizados. Con una sobrecarga en la búsqueda baja y un buen balanceo de la carga computacional, la ganancia de velocidad del modelo dinámico sólo puede estar limitada si existen sobrecargas importantes en las comunicaciones. Los

Tabla 4.3: Resultados numéricos para el método dinámico. $\epsilon = 10^{-3}$

p	1	3	5	7
Tiempo de CPU (sg.)				
GP	127.44	43.76	25.92	18.43
H3	1149.80	384.95	230.56	164.60
L3	487.98	164.70	104.32	74.10
C	20.50	6.94	4.51	3.20
Ganancia de velocidad				
GP	1.00	2.91	4.92	6.92
H3	1.00	2.99	4.99	6.99
L3	1.00	2.96	4.68	6.58
C	1.00	2.95	4.55	6.41
DBL				
GP	0.000	0.021	0.006	0.007
H3	0.000	0.001	0.007	0.007
L3	0.000	0.022	0.021	0.023
C	0.000	0.001	0.010	0.012
Evaluaciones de $F(X)$				
GP	169213	169280	171415	171457
H3	283409	283888	283950	284020
L3	32765	33082	35392	35360
C	33449	33528	36363	36376
Evaluaciones de $F(m(X))$				
GP	42303	42304	41829	41840
H3	35426	35422	35429	35438
L3	8191	8254	7824	7816
C	8362	8366	8066	8070

resultados muestran en para las funciones de prueba, la ganancia de velocidad está muy cerca del número de procesadores *esclavos* utilizados, observándose un pequeño decremento de la ganancia de velocidad, conforme se incrementa el número de procesadores *esclavos*, para las funciones L3 y C. En cuanto a la mejora del tiempo de CPU requerido para la resolución de los problema tratados, comparado con el método semi-estático, se encuentra entre un 14% para la función L3 y un 49% para la función H3 y comparado con el modelo estático las mejoras oscilan entre un 33% y un 86%.

Una evaluación global de los resultados de ganancia de velocidad y de desbalanceo para los tres métodos, muestra que la versión semi-estática mejora el balanceo de la carga y la ganancia de velocidad de la versión estática, pero la versión dinámica obtiene mejores resultados con valores de desbalanceo diez veces menores que los semi-estáticos. Por lo tanto, se puede concluir que el método de balanceo dinámico es imprescindible para obtener buenos resultados de eficiencia en los algoritmos de Optimización Global basados

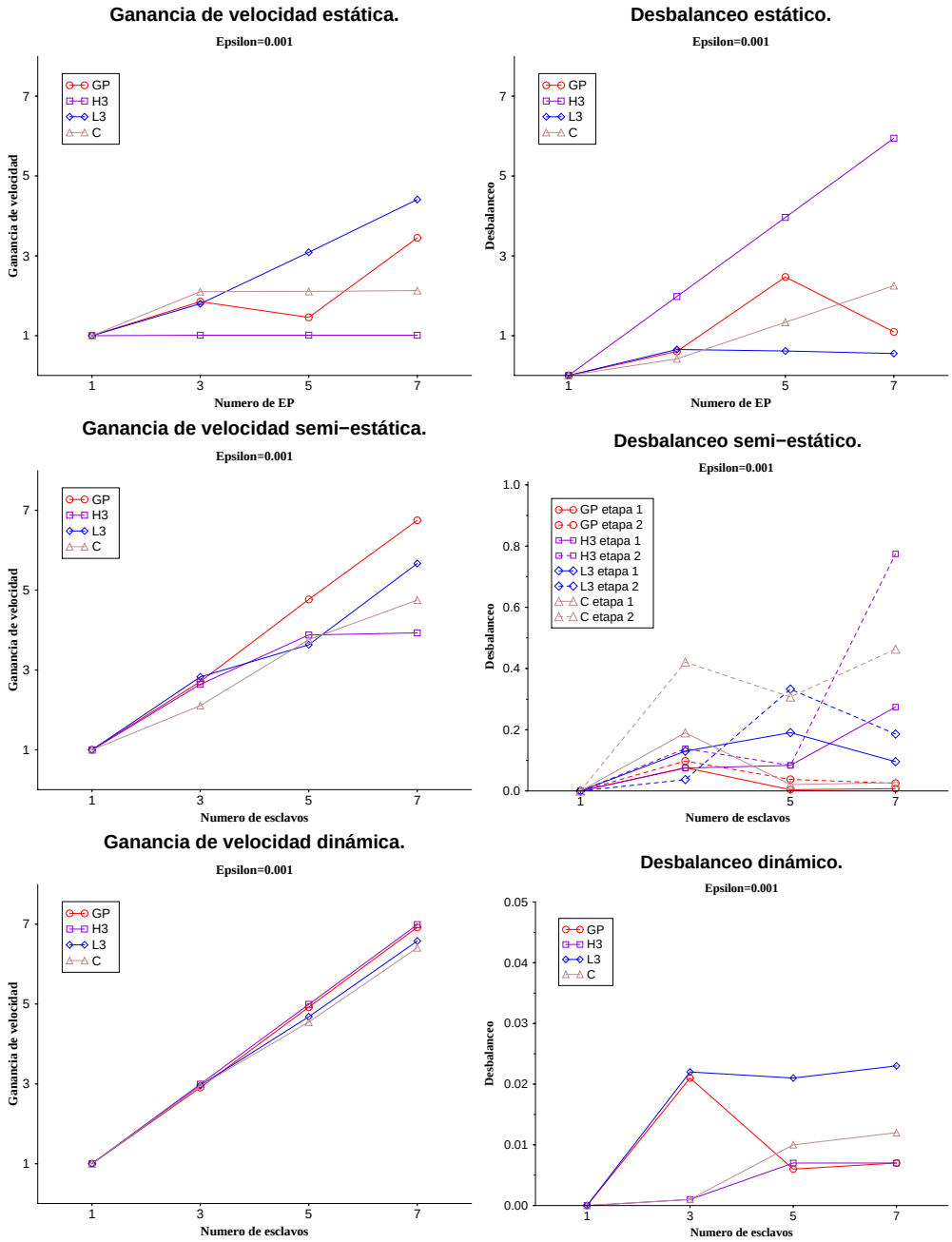


Figura 4.7: Resultados numéricos de la ganancia de velocidad y del desbalanceo de la carga.

en intervalos. También hay que resaltar que la sobrecarga de la búsqueda obtenida en los modelos semi-estático y dinámico es muy baja. Por otro lado, el nuevo estimador de la carga computacional propuesto, permite realizar buenas decisiones para obtener un buen balanceo de la carga en este tipo de algoritmos paralelos.

4.5 Resumen

En este capítulo se ha resaltado la dificultad de desarrollar algoritmos paralelos, principalmente cuando requieren estrategias de balanceo dinámicas. No solo hay que tener en cuenta las características de la arquitectura que finalmente se va a utilizar, sino que también hay que tomar decisiones sobre el tipo de implementación que se va a realizar, donde las posibles opciones normalmente entran en conflicto. Si además hay que realizar un balanceo dinámico de la carga, el número de parámetros que hay que adaptar a las exigencias de un problema concreto aumenta, aumentando así la dificultad de encontrar un algoritmo paralelo óptimo.

En los algoritmos de Ramificación y Acotación paralelos, hay que intentar realizar un buen balanceo de la carga, reduciendo el número de comunicaciones. Un factor importante es la utilización de un buen estimador de la carga computacional de un determinado problema. El estimador propuesto en este capítulo permite obtener un buen balanceo de la carga y unos buenos indicadores de la eficiencia en algoritmos de Optimización Global paralelos basados en intervalos, sobre un conjunto de problemas de prueba, computacionalmente poco costosos.

Conclusiones y principales aportaciones

En esta tesis se ha realizado un estudio de los algoritmos de Ramificación y Acotación y de los métodos de Optimización Global basados en intervalos.

El concepto de algoritmo de Ramificación y Acotación está basado en unas reglas básicas de actuación: *Selección*, *Acotación*, *División*, *Eliminación* y *Terminación*. Estas reglas básicas fueron descritas en la sección 1.5 de forma general ya que los algoritmos de Ramificación y Acotación pueden aplicarse a un gran número de problemas.

Los algoritmos de Optimización Global basados en intervalos hacen uso de las técnicas de Ramificación y Acotación para la búsqueda inteligente de un mínimo en el dominio del problema. La base de estos algoritmos es el uso de una *Regla de Acotación* basada en la obtención de una *función de inclusión* por medio de la Aritmética de Intervalos. El uso de unas buenas reglas de Ramificación y Acotación para el problema a resolver hacen que la eficiencia de tales algoritmos se incremente sustancialmente. Los problemas tratados en esta tesis han sido:

- Encontrar la mínima raíz en funciones unidimensionales.
- Encontrar el mínimo de una función n-dimensional, generalmente no lineal, con la única restricción de tener límites implícitos en la región de búsqueda.

Para ambos casos se han propuesto nuevas reglas de Ramificación y Acotación que han permitido la mejora de estos algoritmos.

Para el problema de encontrar la mínima raíz en funciones unidimensionales se han propuesto unas nuevas reglas de *Selección* y *Eliminación*. La nueva *Regla de Eliminación* está basada en una nueva estrategia denominada el *Test del Punto Negativo*. Estas reglas cooperan para obtener solamente la mínima raíz de una función unidimensional, o de un conjunto de funciones unidimensionales, con algoritmos simples, rigurosos y rápidos. También se ha demostrado que una reordenación de la secuencia lógica de los procesos que componen el algoritmo, permite acelerar el proceso de convergencia, es decir, incrementa la eficiencia computacional del método.

Para el problema general de encontrar el mínimo de una función n-dimensional se han presentado las reglas de Ramificación y Acotación de uso más común en los algoritmos

actuales. También se ha presentado un nuevo parámetro, $p\overline{f^*}(X)$, que se ha mostrado como un buen estimador del grado de proximidad, de cualquier subintervalo de la región de búsqueda, a la región de atracción hacia un mínimo. Basándonos en el parámetro $p\overline{f^*}(X)$, se han propuesto las siguientes modificaciones de las reglas de Ramificación y Acotación en algoritmos de Optimización Global basada en intervalos:

- *Regla de División*: El método más común de división es la bisección de una o varias coordenadas del espacio de búsqueda. Una vez establecido el tipo de división, ésta permanece fija durante toda la ejecución del algoritmo. Nuestra propuesta de *Regla de División* adapta el nivel de división a realizar sobre la subregión de búsqueda X, dependiendo del valor del parámetro $p\overline{f^*}(X)$. Comparando el nuevo tipo de división dinámico con otros estáticos, los resultados obtenidos fueron siempre mejores y cuanto mayor era la precisión requerida en las soluciones, mayores fueron las mejoras obtenidas con la nueva propuesta.

- *Regla de Selección*: Se ha presentado una nueva *Regla de Selección Híbrida* basada en el parámetro $p\overline{f^*}(X)$, que permite obtener en las primeras etapas del algoritmo el mejor valor del límite superior del mínimo global. Este hecho reduce el costo computacional del algoritmo, debido a que el conocimiento del mejor valor del límite superior, permite eliminar de la lista de trabajo un mayor número de subintervalos, que no serán evaluados. Cuando se alcanza este valor, el tipo de *Regla de Selección* utilizada no cambiará el número de evaluaciones de la *función de inclusión*, ya que sólo se seleccionarán los nodos críticos del árbol de búsqueda. Con el método usual de selección, este límite superior normalmente no es conocido hasta llegar a las fases finales del algoritmo.

- *Regla de Eliminación*: Los algoritmos de Optimización Global basados en intervalos se caracterizan principalmente por la rigurosidad de los resultados obtenidos. Las nuevas reglas de eliminación propuestas permiten reducir considerablemente el costo computacional del algoritmo, pero a costa de perder esta rigurosidad.

  La primera *Regla de Eliminación* propuesta está basada en el valor del parámetro $p\overline{f^*}(X)$ para la subregión actual y el de sus ascendentes. Esta *Regla de Eliminación* ha permitido una mejora del 80% en los problemas tratados, sin la perdida de ninguno de los mínimos globales.

  La segunda propuesta de *Regla de Eliminación* está motivada por la imposibilidad de resolver algunos problemas que requieren enormes cantidades de memoria. Con la nueva regla no solo se han resuelto problemas de mayores dimensiones sino que se ha podido alcanzar una mayor precisión en las soluciones, con un coste muy bajo en cuanto a la memoria requerida (casi despreciable). Con la nueva *Regla de Eliminación* no solo se reducen los requerimientos de memoria, sino que también se disminuye considerablemente el número de evaluaciones de la *función de inclusión*, acercándose este valor al mínimo posible, en la mayoría de los casos.

También se ha abordado el problema de la Optimización Global basada en intervalos desde una perspectiva paralela. Tras una revisión de las estrategias existentes en la paralelización de algoritmos de Optimización Global basados en intervalos, donde el balanceo dinámico está determinado por la cantidad de las subregiones de búsqueda, X^i, que cada procesador tiene pendientes para su procesamiento y en la calidad de las mismas, medida por el valor de $\underline{F}(X^i)$, se ha presentado un nuevo estimador de la carga computacional que puede contener una determinada subregión de búsqueda y se ha utilizado una búsqueda basada en un modelo de selección híbrido, que además de permitir encontrar rápidamente un buen límite superior del mínimo de la función, también realiza una buena partición del espacio de búsqueda. Ambas propuestas permiten obtener un balanceo de la carga más equitativo en los algoritmos paralelos de Optimización Global basada en intervalos. Nuestras propuestas paralelas han sido evaluadas sobre un sistema basado en una arquitectura de memoria distribuida con procesadores Pentium, con una red de interconexión Fast-Ethernet, que ha sido desarrollado por el grupo de investigación "Supercomputación-Algoritmos" del Departamento de Arquitectura de Computadores y Electrónica de la Universidad de Almería.

Consideramos que el trabajo realizado en esta tesis es un primer paso de entrada al complejo mundo de la Optimización Global y sus aplicaciones. Muchos aspectos teóricos y prácticos, tanto en la vertiente secuencial como en la paralela, deberán ser abordados en el futuro. Sin ser exhaustivos, nuestro interés a corto plazo se centrará en los siguientes aspectos:

- Inclusión de los aceleradores tradicionales basados en la evaluación del gradiente, mediante diferenciación automática o expresiones analíticas, en nuestras propuestas algorítmicas.

- También se incluirán expresiones que permiten obtener mejores *funciones de inclusión*, como las basadas en la pendiente (*slope forms*) y las centradas (*centered forms*).

- Desde el punto de vista del paralelismo, el análisis de modelos distribuidos puede ampliar las perspectivas de implementación de los algoritmos tratados en esta tesis. Además, los métodos propuestos deberán evaluarse sobre máquinas de memoria compartida-distribuida.

- Por último, forman parte de los proyectos de futuro, el tratamiento de aplicaciones reales científicas o industriales, mediante los métodos propuestos en esta tesis, lo que permitirá dar consistencia a las nuevas versiones de los algoritmos, que se han diseñado para poder abordar problemas extremadamente complejos desde un punto de vista computacional.

Apéndice A

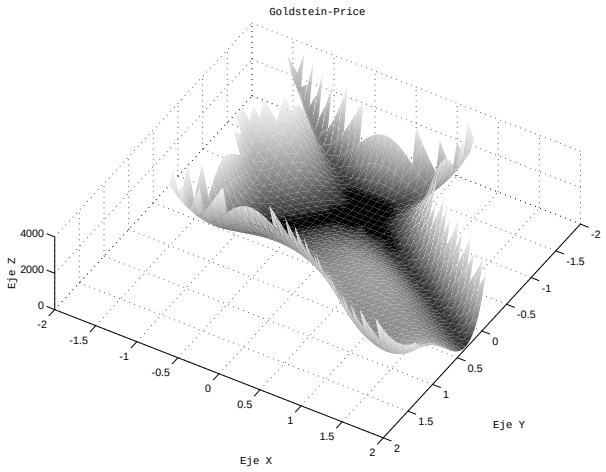
Funciones test.

Goldstein-Price (GP) [177]:

Ecuación de la función:

$$f_{GP}(x) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \times [30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$$

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:

$$F_{GP}(X) = (1 + (Y_1 + 1)^2(19 - 14Y_1 + 3Y_1^2)) \times (30 + Y_2^2(18 - 16Y_2 + 3Y_2^2))$$
$$\text{con } Y_1 = X_1 + X_2, \quad Y_2 = 2X_1 - 3X_2$$

Shekel 5 (S5) [177]:

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 $f \cong -1/c_i$ están aproximadamente en a_i.

Shekel 7 (S7) [177]:

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 $f \approxeq -1/c_i$ están aproximadamente en a_i.

Shekel 10 (S10) [177]:

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.

Hartman 3 (H3) [177]:

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) [177]:

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)

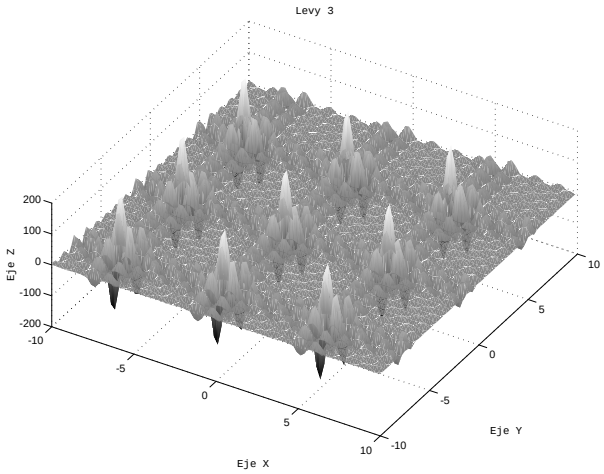
Dominio (S): $0 \leq x_i \leq 1,\ i = 1,..,6$
Anchura del rango real: $w(f(S)) \simeq 3.3$
Mínimo global:

$$f^* = -3.32236801$$
$$x^* = (0.2016895, 0.1500106, 0.4768739, 0.2753324, 0.31165161, 0.65730053)^T$$

Levy No. 3 (L3) [185]:

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:

$$f^* = -176.54179313$$

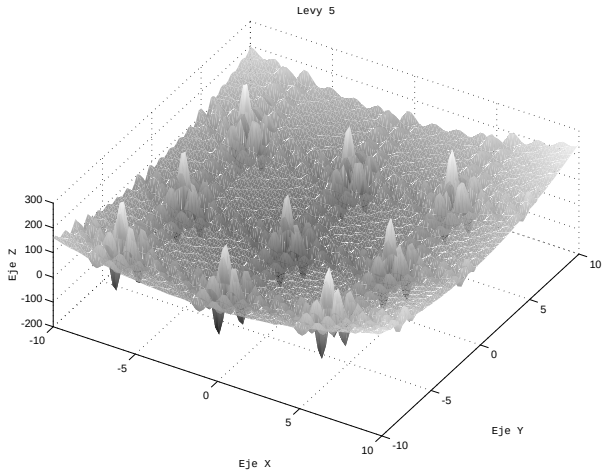
$$X^* = \begin{cases} (4.97647760,\ 4.85805687)^T \\ (4.97647760,\ -1.42512842)^T \\ (4.97647760,\ -7.70831373)^T \\ (-1.30670770,\ 4.85805687)^T \\ (-1.30670770,\ -1.42512842)^T \\ (-1.30670770,\ -7.70831373)^T \\ (-7.58989301,\ 4.85805687)^T \\ (-7.58989301,\ -1.42512842)^T \\ (-7.58989301,\ -7.70831373)^T \end{cases}$$

Levy No. 5 (L5) [185]:

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) [185]:

Ecuación de la función:

$$\begin{aligned} f_{L8}(x) =& \sin^2(\pi y_1) + \sum_{i=1}^{2}(y_i - 1)^2(1 + 10\sin^2(\pi y_{i+1})) \\ &+ (y_3 - 1)^2 \\ &\text{con } y_i = 1 + (x_i - 1)/4 \end{aligned}$$

Dominio (S): $-10 \leq x_i \leq 10,\ i = 1, 2$
Anchura del rango real: $w(f(S)) \simeq 160$
Mínimo global: $f^* = 0,\ x^* = (1, 1, 1)^T$

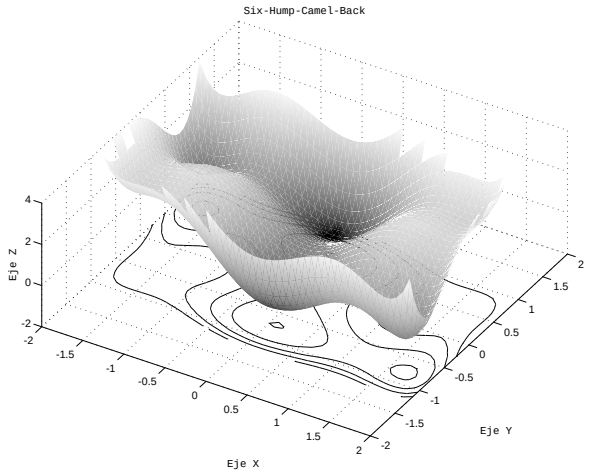
Six Hump Camel Back (C) [177]:

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:

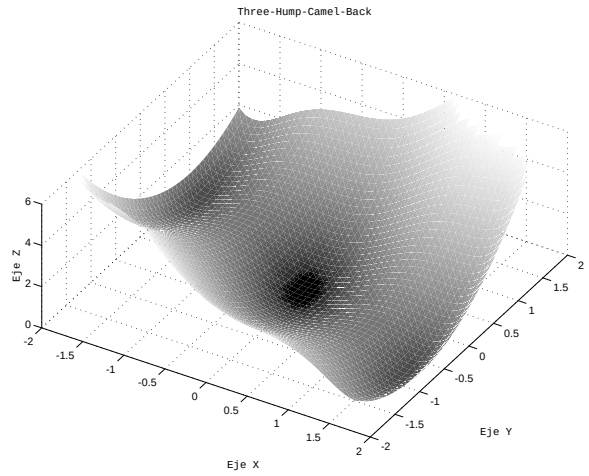
$$\begin{aligned} f^* &= -1.03162845 \\ X^* &= \begin{cases} (\ 0.08984201, -0.71265640)^T \\ (-0.08984201,\ 0.71265640)^T \end{cases} \end{aligned}$$



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) [41]:



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): $-5 \leq x_i \leq 5,\ i = 1, 2$

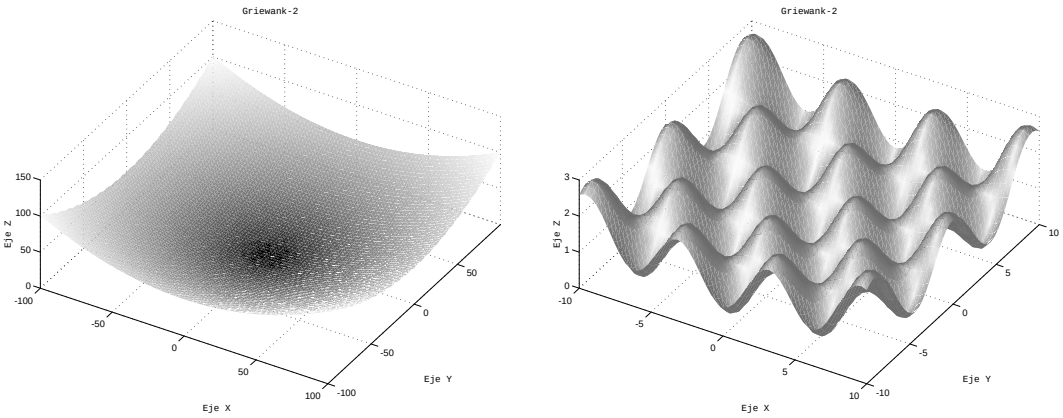
Anchura del rango real: $w(f(S)) \simeq 2000$
Mínimo global: $f^* = 0,\ x^* = (0,0)^T$

Griewank 2 (G2) [177]:

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 (G10) [177]:

Ecuación de la función:

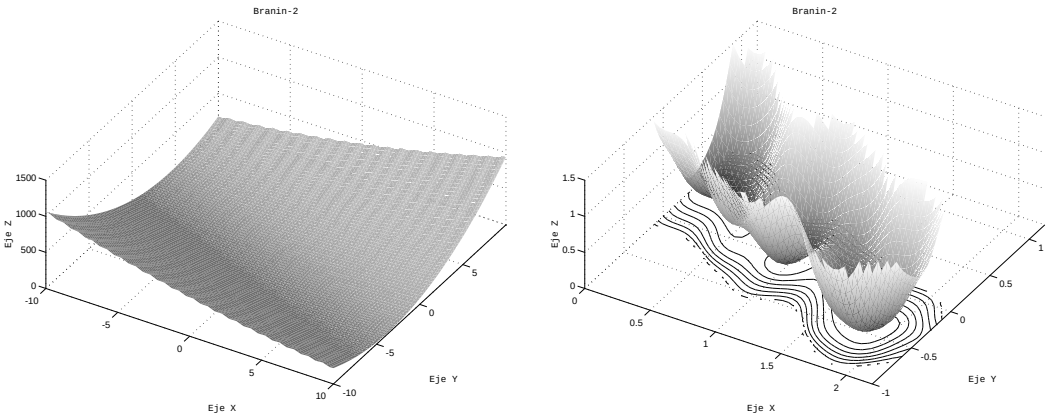
$$f_{G10}(x) = \sum_{i=1}^{10} \frac{x_i^2}{4000} - \prod_{i=1}^{10} \cos(\frac{x_i}{\sqrt{2}}) + 1$$

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.

Branin 2 (BR2) [41]:

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:

$$f^* = 0$$

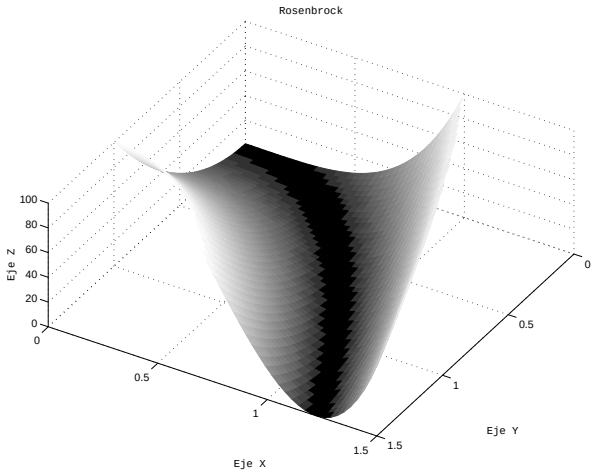
$$X^* = \begin{cases} (1,\ 0)^T \\ (0.148696,\ 0.402086)^T \\ (0.402537,\ 0.287408)^T \\ (1.59746,\ -0.287408)^T \\ (1.85130,\ -0.402086)^T \end{cases}$$

Rosenbrock (RB) [41]:

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: $f^* = 0,\ x^* = (1, 1)^T$

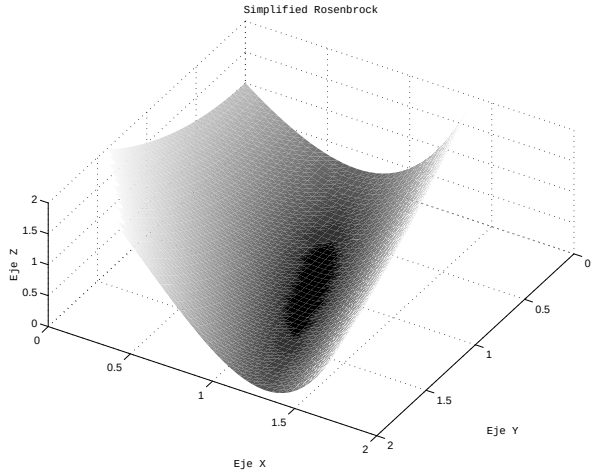


Simplified Rosenbrock (SRB) [41]:

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.

Price (P) [42]:

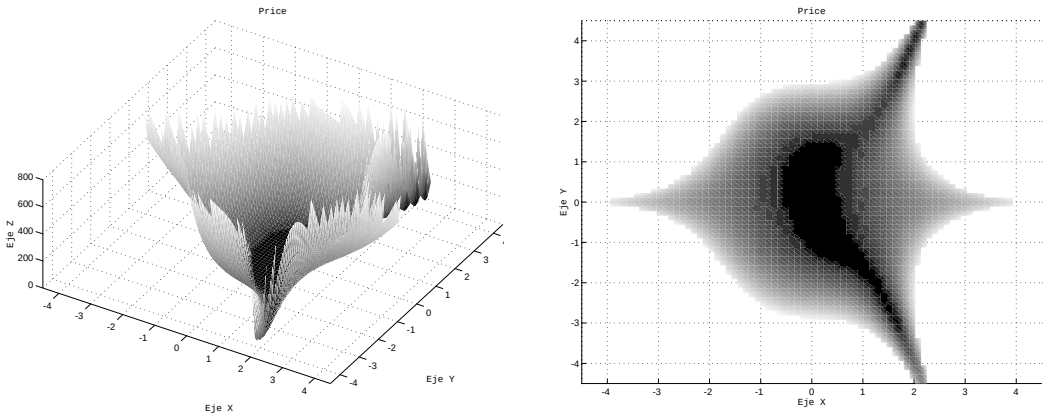
Ecuación de la función:

$$f_P(x) = (2x_1^3 x_2 - x_2^3)^2 + (6x_1 - x_2^2 + x_2)^2$$

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:

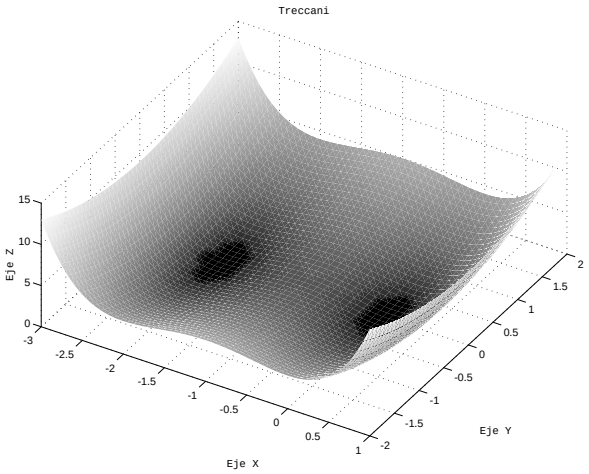
$$F_P(X) = \left(X_2\left(2X_1^3 - X_2^2\right)\right)^2 + \left(X_2\left(1 - X_2\right) + 6X_1\right)^2$$

Treccani (TR) [41]:

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".

Schwefel No. 3.1 (S3.1) [185]:

Ecuación de la función:

$$f_{S3.1}(x) = \sum_{i=1}^{3} \left[\left(x_1 - x_i^2\right)^2 + \left(x_i - 1\right)^2 \right]$$

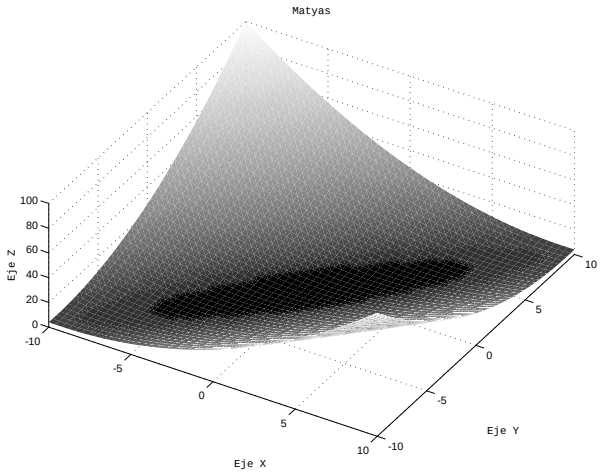
Dominio (S): $-10 \leq x_i \leq 10,\ i = 1, .., 3$
Anchura del rango real: $w(f(S)) \simeq 3.7 \cdot 10^4$
Mínimo global: $f^* = 0,\ x^* = (1, 1, 1)^T$

Matyas (MT) [185]:

Ecuación de la función:

$$f_{MT}(x) = 0.26(x_1^2 + x_2^2) - 0.48 x_1 x_2$$

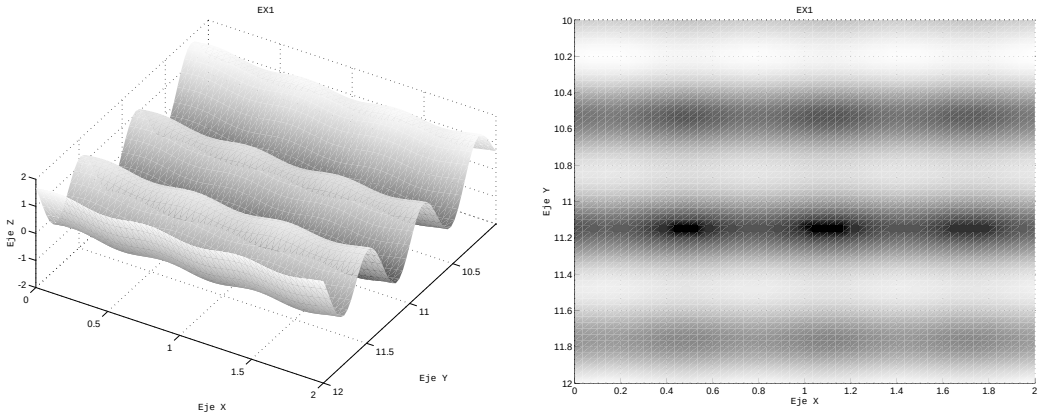
Dominio (S): $-10 \leq x_i \leq 10,\ i = 1, 2$
Anchura del rango real: $w(f(S)) \simeq 100$
Mínimo global: $f^* = 0,\ x^* = (0, 0)^T$



Función de inclusión:

$$F_{MT}(X) = 0.26(X_1 - X_2)^2 + 0.04X_1X_2$$

EX1 (EX1) [35]:



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 (EX2) [35]:

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:

$$F_{EX2}(X) = \sum_{i=1}^{6} \left(yre_i - \left(X_1 + \frac{X_2}{\omega_i^{X_3}} \right) \right)^2 + \left(yim_i - \left(\omega_i X_4 - \frac{X_5}{\omega_i^{X_3}} \right) \right)^2$$

$$yre_i = (5.0,\ 3.0,\ 2.0,\ 1.5,\ 1.2,\ 1.1)$$
$$yim_i = (-5.0,\ -2.0,\ -1.0,\ -0.5,\ -0.2,\ -0.1)$$
$$w_i = (0.05\pi,\ 0.1\pi,\ 0.15\pi,\ 0.2\pi,\ 0.25\pi,\ 0.3\pi)$$

Observación:

El termino $|\cdot|^2$ para números complejos puede calcularse simplemente como la suma de los cuadrados de los términos real e imaginario.

Branin (BR) [177]:

Ecuación de la función:

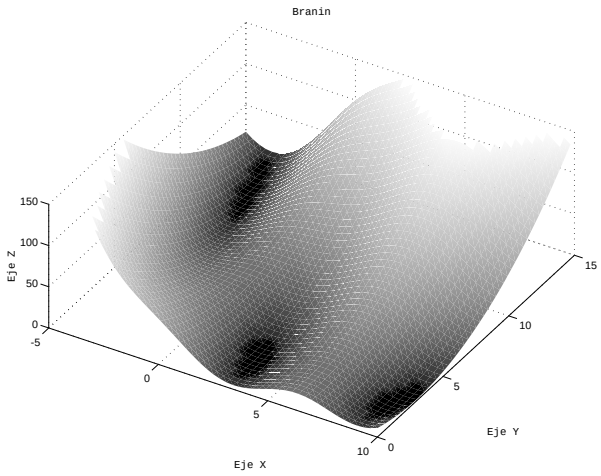
$$f_{BR}(x) = \left(x_2 - \frac{5.1}{4\pi^2} x_1^2 + \frac{5}{\pi} x_1 - 6 \right)^2 + 10 \left(1 - \frac{1}{8\pi} \right) cos(x_1) + 10$$

Dominio (S): $[-5, 10] \times [0, 15]$

Anchura del rango real: $w(f(S)) \simeq 300$

Existen 3 mínimos globales con:

$$f^* = 0.397887$$
$$X^* = \begin{cases} (\text{-}3.14159,\ 12.275)^T \\ (3.14159,\ 2.275)^T \\ (9.42478,\ 2.475)^T \end{cases}$$



Kowalik (KW) [185]:

Ecuación de la función:

$$f_{KW}(x) = \sum_{i=1}^{11} \left(a_i - x_1 \frac{b_i^2 + b_i x_2}{b_i^2 + b_i x_3 + x_4} \right)^2$$

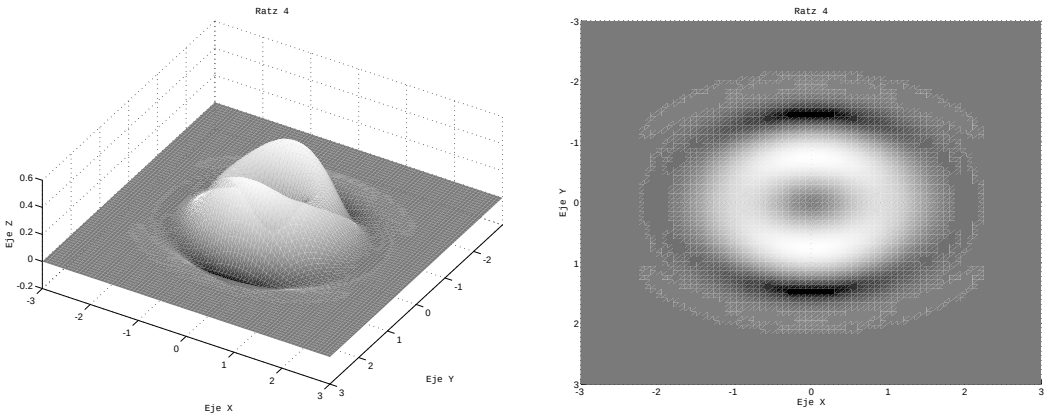
Coeficientes:

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:

$$f^* = 3.074859878 \cdot 10^{-4}$$
$$x^* = (0.192833452982, 0.19083623878, 0.12311729627, 0.13576598998)^T$$

Ratz No. 4 (R4) [158]:



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. 8 (R8) [158]:

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): $[-10,10]^9$
Anchura del rango real: $w(f(S)) \simeq 4.4 \cdot 10^5$
Mínimo global:

$$f^* = 0.0$$
$$x^* = (1,1,1,1,1,1,1,1,[-10,10])^T$$

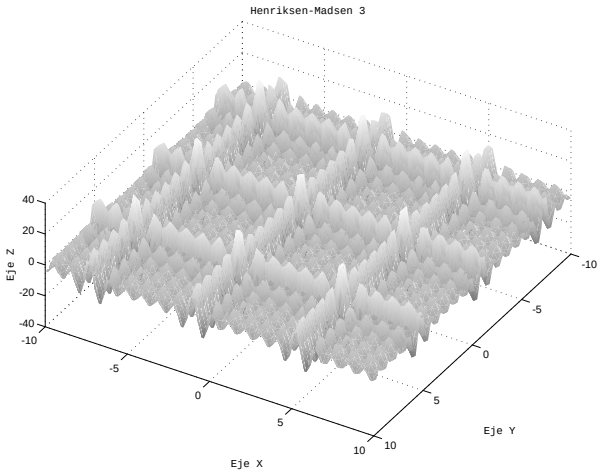
Observación: Existen infinitos mínimos globales ya que $X_9^* = [-10,10]$.

Henriksen-Madsen No. 3 (HM3) [74, 7]:

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) [74, 7]:

Ecuación de la función:

$$f_{HM4}(x) = -\sum_{i=1}^{3}\sum_{j=1}^{5} j\sin((j+1)x_i + j)$$

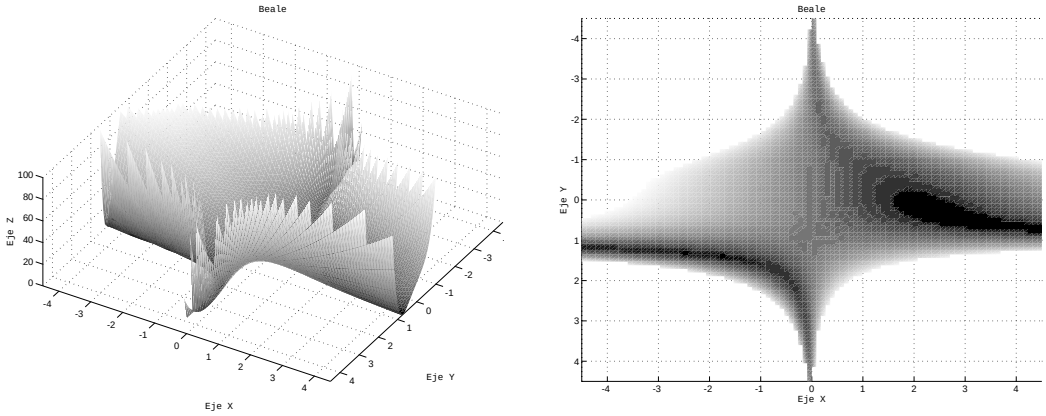
Dominio (S): $-10 \leq x_i \leq 10,\ i = 1,..3$
Anchura del rango real: $w(f(S)) \simeq 80$
Mínimo global: $f^* = -36.09374832,\ x^* = (0.4913908, 0.4913908, 0.4913908)^T$

Beale (BL) [185]:

Ecuación de la función:

$$f_{BL}(x) = (1.5 - x_1 + x_1x_2)^2 + (2.25 - x_1 + x_1x_2^2)^2 + (2.625 - x_1 + x_1x_2^3)^2$$

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:

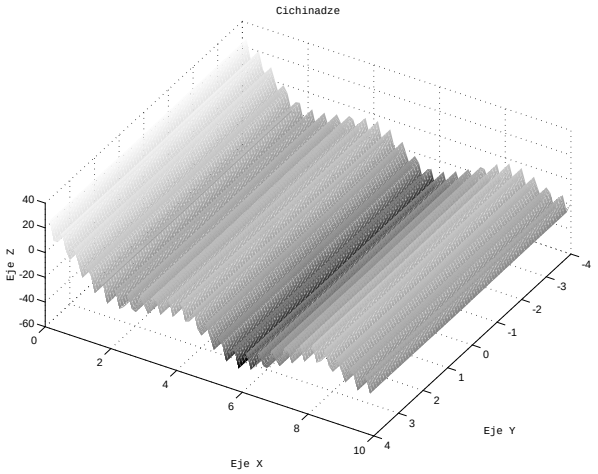
$$F_{BL}(X) = (1.5 + X_1(X_2 - 1))^2 + \left(2.25 + X_1(X_2^2 - 1)\right)^2 + \left(2.625 + X_1(X_2^3 - 1)\right)^2$$

Chichinadze (CHI) [42]:

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:

$$F_{BL}(X) = X_1(X_1 - 12) + 11 + 10\cos(\frac{\pi}{2}X_1) + 8\sin(5\pi X_1) - \frac{1}{\sqrt{5}}\exp\left(-\frac{\left(X_2 - \frac{1}{2}\right)^2}{2}\right)$$

Apéndice B

Direcciones de interés en Internet

B.1 Aritmética de Intervalos

B.1.1. **Errores de la Aritmética Computacional:**
http://www.math.psu.edu/dna/455.f96/disasters.html

B.1.2. **Interval Computations:**
http://cs.utep.edu/interval-comp/main.html

B.1.3. **Interval Methods:**
http://solon.cma.univie.ac.at/~neum/interval.html

B.2 Arquitecturas y Algoritmos Paralelos

B.2.1. **Parallel Architectures and Algorithms :**
http://miro.ualm.es/parallel.html

B.2.2. **Network of workstations :**
http://miro.ualm.es/caesarg.html

B.3 Diferenciación Automática

B.3.1. **Automatic Differentiation:**
http://solon.cma.univie.ac.at/~neum/glopt/related.html#autodiff

B.4 IEEE-754

B.4.1. **IEEE-754 status:**
http://www.cs.berkeley.edu/~wkahan/ieee754status/

B.4.2. **Cray Assembler for MPP:**
http://www.ciemat.es:8080/library/t3e/2510_2.2/8319

B.5 Optimización Global

B.5.1. **A Survey of GO Methods:**
http://www.cs.sandia.gov/opt/survey/

B.5.2. **Global (and Local) Optimization:**
http://solon.cma.univie.ac.at/~neum/glopt.html

B.5.3. **Decision Tree for Optimization Software:**
http://plato.la.asu.edu/guide.html

B.5.4. **List of Useful Optimization Software:**
http://ucsu.colorado.edu/~xu/software.html

B.5.5. **Continuous GO Software: A Brief Review**
http://plato.la.asu.edu/gom.html

B.5.6. **The Optimization Technology Center:**
http://www-c.mcs.anl.gov/home/otc/

B.5.7. **Global Optimization Page:**
http://cad.bu.edu/go/

B.5.8. **Mathematical Optimization:**
http://csep1.phy.ornl.gov/mo/mo.html

B.5.9. **Michael Trick's Operations Research Page:**
http://mat.gsia.cmu.edu/index.html

B.5.10. **Links to Global Optimization:**
http://miro.ualm.es/optimization.html

B.6 Problemas de corte con cero

B.6.1. **Numerical methods for finding roots :**
http://daisy.uwaterloo.ca/~kghare/numeric/index.html

B.7 Problemas de Optimización Combinatoria

B.7.1. **TSPBIB Home Page :**
http://www.ing.unlp.edu.ar/cetad/mos/TSPBIBhome.html

Bibliografía

[1] T. Abdelrahman and T. Mudge. Parallel branch and bound algorithms on hypercube multiprocessors. In *1988 ACM Conf. on LISP and Funct. Prog.*, pages 1492–1499. ACM Press, 1988.

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

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

[4] G. Ananth, V. Kumar, and P. Pardalos. Parallel processing of discrete optimization problems. *Encyclopedia of Microcomputers*, 13:129–147, 1993.

[5] Baoxin Li, A.G. Law, H. Raafat, P.H. Nguyen, and Y.-F. Yan. Eigenvalues of tridiagonal matrices: An alternative to Givens' methods. *Computers Math. Applic.*, 19(4):89–94, 1990.

[6] D. Bedrosian and J. Vlach. Time-domain analysis of networks with internally controlled switches. *IEEE Transactions on Circuits Syst.*, CAS-39(3):199–212, 1992.

[7] S. Berner. New results in verified global optimization. *Computing*, 57(4), 1996.

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

[9] 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.

[10] 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.

[11] L.G. Casado and I. García. Reconfiguración dinámica de una red de transputers. Technical Report TR-N2, Departamento de Arquitectura de Computadores y Electrónica, Universidad de Almería, 1997.

[12] 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, February 1998.

[13] L.G. Casado and I. García. Fast interval global optimization algorithm for limited computational resources. In *Proceedings of the International Workshop on Global Optimization, GO'99*, Firence, Italy, October 1999.

[14] 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, University of Madeira, Funchal, Portugal, February 1999.

[15] L.G. Casado, I. García, and T. Csendes. Adaptive multisection in interval methods for global optimization. *Submitted for publication.*

[16] L.G. Casado, I. García, and T. Csendes. A heuristic rejection criterion in interval global optimization algorithms. *Submitted for publication.*

[17] L.G. Casado, I. García, and T. Csendes. Adaptive multisection in interval global optimization. In *Volume of extended abstracts IMACS/GAMM International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics (SCAN'98)*, pages 22–25, Budapest, Hungary, September 1998.

[18] L.G. Casado, I. García, and T. Csendes. A heuristic rejection criterion in interval global optimization. In *Volume of extended abstracts IMACS/GAMM International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics (SCAN'98)*, pages 18–20, Budapest, Hungary, September 1998.

[19] L.G. Casado, I. García, and Ya. D. Sergeyev. Interval algorithms for finding the minimal root in a set of multidimensional functions. *Submitted for publication.*

[20] L.G. Casado, I. García, and Ya. D. Sergeyev. Interval branch and bound algorithm for finding the First-Zero-Crossing-Point in one-dimensional functions. *Accepted for publication in Reliable Computing.*

[21] L.G. Casado, I. García, and Ya. D. Sergeyev. A fast algorithm for finding the first-zero-crossing in a set of signals based on interval arithmetic. In *Volume of extended abstracts IMACS/GAMM International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics (SCAN'98)*, pages 48–51, Budapest, Hungary, September 1998.

[22] L.G. Casado, I. García, and Ya. D. Sergeyev. Interval branch and bound algorithm for finding the First-Zero-Crossing-Point in one-dimensional functions. In 16^{th} *Eu-*

ropean Conference on Operations Research, pages 99–100, Brussels, Belgium, July 1998.

[23] L.G. Casado, I. García, P.G. Szabo, and T. Csendes. Equal circles packing in a square II. – New results for up to 100 circles using the TAMSASS-PECS algorithm. *Submitted for publication.*

[24] J.J. Clark. Authenticating edges produced by zero-crossing algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI*, 11:43–57, 1989.

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

[26] 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.

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

[28] 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.

[29] 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.

[30] 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.

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

[32] 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.

[33] Cray Research. *Cray T3D System Architecture Overview*, 1993.

[34] A.E. Csallner, T. Csender, and M.C. Markót. Multisection in interval branch-and-bound methods for global optimization. *Submitted for publication. Available at http://www.inf.u-szeged.hu/∼csendes/publ.html.*

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

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] L.C.W. Dixon and G.P. Szego, editors. *Towards Global Optimization*. North-Holland Publishing Company, 1975.

[42] L.C.W. Dixon and G.P. Szego, editors. *Towards Global Optimization 2*. North-Holland Publishing Company, 1978.

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

[44] J. Duato, S. Yalamanchi, and L.Ñi, editors. *Interconnection networks: an engineering approach*. IEEE Computer Society, 1997.

[45] P.S. Dwyer. Computation with approximate numbers. *Linear Computations*, pages 11–34, 1951.

[46] W.L. Eastaman. Linear programming with pattern constraints. Technical Report BL 20, The Computation Laboratory, Harvard University, Cambridge, 1958.

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

[48] T.G.W. Epperly. *Global Optimization of Noncovex nonlinear Programs using Parallel Branch and Bound*. PhD thesis, University of Wisconsin-Madison, 1995.

[49] J. Eriksson and P. Lindström. A parallel interval method implementation for global optimization using dynamic load balancing. *Reliable Computing*, 1(1):77–91, 1995.

[50] C.A. Floudas and P.M. Pardalos. *A collection Test Problems for Constrained Global Optimization Algorithms*, volume 455 of *Lectures Notes in Computer Science*. Springer, 1990.

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

[52] C.A. Floudas and P.M. Pardalos, editors. *State of the Art in Global Optimization. Computational Methods and Applications*, volume 7 of *Nonconvex optimization and its applications*. Kluwer Academic Publishers, 1996.

[53] M.J. Flynn. Some computer organizations and their effectiveness. *IEEE Trans. Comp.*, 1972.

[54] I. Foster. *Designing and Building Parallel Programs*. Addison-Wesley, 1995.

[55] B. Fox, Lenstra J, A. Rinnoy Kan, and L. Schrage. Branching from largest upper bound: folklore and facts. *Europ. Journal of Op. Res.*, pages 191–194, 1978.

[56] I. Garcia, P.M. Ortigosa, L.G. Casado, G.T. Herman, and S. Matej. A parallel implementation of the controlled random search algorithm to optimize an algorithm for reconstruction from projections. In *IIIrd Workshop on Global Optimization*, pages 28–32, Szeged, Hungary, December 1995.

[57] I. García, P.M. Ortigosa, L.G. Casado, G.T. Herman, and S. Matej. Multidimensional optimization in image reconstruction from projections. *Developments in Global Optimization*, pages 289–300, 1997.

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

[59] M.R. Garey and D.S. Johnson. *Computer and Intractability*. W.H. Freeman and Company, 1979.

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

[61] M. Gengler and G. Coray. A parallel best-first B&B with synchronization phases. In *Parallel Processing: CONPAR 92 VAPP V, LNCS 634*, pages 515–526. Springer-Verlag, 1992.

[62] 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.

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

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

[65] A.Y. Grama and V. Kumar. A survey of parallel search algorithms for discrete optimization problems. *ORSA Journal of Computing*, 7(4):365–385, 1995.

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

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

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

[69] E. Hansen. Global optimization using interval analysis – the one dimensional case. *J. Optim. Theor. and Appl.*, 29:331–344, 1979.

[70] E. Hansen. Global optimization using interval analysis – the multidimensional case. *Numerische Mathematik*, 34:247–270, 1980.

[71] E. Hansen. *Global Optimization Using Interval Analysis*, volume 165 of *Pure and applied mathematics.* Marcel Dekker, Inc, New York, 1992.

[72] D. Henrich. Initialization of parallel branch-and-bound algorithms. In *Second International Workshop on Parallel Processing for Artificial Intelligence.* Elsevier, 1994.

[73] T. Henriksen and K. Madsen. Parallel algorithms for Global Optimization. *Interval Computations*, 3(5), 1992.

[74] 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.

[75] C.G. Herter, T.M. Warschko, W.F. Tichy, and M. Philippsen. Triton/1: A massively-parallel mixed-mode computer designed to support high level languages. In *International Parallel Processing Symposium, Proc. of 2nd Workshop on Heterogeneous Processing*, pages 65–70, 1993.

[76] 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.

[77] R. Horst and P.M. Pardalos, editors. *Handbook of Global Optimization*, volume 2 of *Noncovex optimization and its applications.* Kluwer Academic Publishers, 1995.

[78] R. Horst and H. Tuy. *Global Optimization. Deterministic Approaches.* Springer-Verlag, third edition, 1996.

[79] K. Hwang and F. Briggs. *Computer Architecture and Parallel Processing.* McGraw-Hill, New York, 1984.

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

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

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

[83] T. Ibaraki. *Enumerative approaches to combinatorial optimization*, volume I and II. J.C. BALTZER AG, Basel-Switzerlan, 1987.

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

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

[86] D. Kalra and A. H. Barr. Guaranteed ray intersections with implicit surfaces. *Computer Graphics*, 23(3):297–306, 1989.

[87] R. Karp and Y. Zhang. Randomized parallel algorithms for backtrack search and branch-and-bound computations. *Journal of the ACM*, 40(3):765–789, 1993.

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

[89] 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.

[90] O. Knüppel. BIAS-Basic Interval Arithmetic Subroutines. Technical Report 93.3, University of Hamburg, 1993.

[91] R. Krawczyk and K. Nickel. The centered form in interval arithmetics: Quadratic convergence and inclusion isotonicity. *Computing*, 28(2):117–137, 1982.

[92] U.W. Kullish and W.L. Miranker. The arithmetic of the digital computer: A new approach. *SIAM Rev.*, 1(28):1–40, 1986.

[93] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms.* The Benjamin/Cummings Publishing Company, 1994.

[94] V. Kumar and A. Gupta. Analyzing scalability of parallel algorithms and architectures. *Journal of Parallel and Distributed Computing (special issue on scalability)*, 22(3):379–391, 1994.

[95] 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.

[96] 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.

[97] T. Lai and S. Shani. Anomalies in parallel branch-and-bound algorithms. *Communications of the ACM*, 27:594–602, 1984.

[98] T. Lai and A. Sprague. Performance of parallel branch-and-bound algorithms. *IEEE Transactions on Computers*, C-34(10):962–964, 1985.

[99] 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.

[100] A.H. Lang and A.G. Doig. An automatic method for solving discrete programming problems. *Econometrica*, 28:497–520, 1960.

[101] E.L. Lawer and D.E. Wood. Branch and bound methods: A survey. *Operation Research*, 14:699–719, 1966.

[102] A.P. Leclerc. *Efficient and Reliable Global Optimization.* PhD thesis, Ohio State University, 1992.

[103] A.P. Leclerc. Parallel interval global optimization and its implementation in c++. *Interval Computations*, 3:148–163, 1993.

[104] L.W. Levelt, M. Kaashoek, H. Bal, and A. Tanenbaum. A comparison of two paradigms for distributed shared memory. *Software – Practice and Experience*, 22(11):985–1010, 1992.

[105] 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.

[106] G. Li and B. Wah. Coping with anomalies in parallel branch-and-bound algorithms. *IEEE Transactions on Computers*, C-35(6):568–573, 1986.

[107] 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.

[108] 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.

[109] R. Lüling and B. Monien. Load balancing for distributed branch and bound algorithms. Technical report, University of Paderborn, Germany, 1992.

[110] R. Lüling, B. Monien, and F.Ramme. A study on dynamic load balancing algorithms. Technical Report PC^2/TR-001-92, Paderborn. Center for Parallel Computing, 1992.

[111] 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.

[112] K. Madsen and S. Zertchanninov. A new branch and bound method for global optimization. Technical Report IMM 05/98, Technical University of Demmark, Copenhagen, 1998.

[113] S. Mallat. Zero crossing of a wavelet transform. *IEEE Transactions on Information Theory*, 17(4):1019–1033, 1991.

[114] B. Mans and C. Roucairol. Performances of parallel branch and bound algorithms with best-first search. *Discrete Applied Mathematics*, 66(1):57–76, 1996.

[115] S. Markov. Some applications on extended interval arithmetic to interval iterations. *Computing (Suppl.)*, 2:69–84, 1980.

[116] S. Markov. On interval arithmetic and its applications. In *Proceedings of the 5th Symposium on computer Arithmetic*, U. Michigan, MI, USA, 1981. IEEE.

[117] J.A. Martínez, V.G. Ruíz, L.G. Casado, J.J. Fernández, and I. García. Evaluación de un multiprocesador con alta calidad/precio: Una solución para laboratórios docentes. In *IX Jornadas de Paralelismo*, pages 153,157, Donostia, 1998.

[118] Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, 1996.

[119] D.P. Mitchell. Robust ray intersection with interval arithmetic. In *Graphics Interface'90*, pages 68–74, 1990.

[120] L.G. Mitten. Branch and bound methods: general formulation and properties. *Operation Research*, 18:24–34, 1970.

[121] J. Mockus. *Bayesian Approach to Global Optimization*. Kluwer Academic, 1989.

[122] R. Moore, E. Hansen, and A. P. Leclerc. Rigorous methods for Global Optimization. In C.A. Floudas and P.M. Pardalos, editors, *Recent Advances in Global Optimization*. Princenton University Press, 1992.

[123] R. E. Moore. Methods and applications of interval analysis. In *SIAM*, Philadelphia, 1979.

[124] R.E. Moore. *Interval Arithmetic and Automatic Error Analysis in Digital Computing*. PhD thesis, Stanford University, October 1962.

[125] R.E. Moore. *Interval analysis*. Prentice-Hall, 1966.

[126] R.E. Moore. On computing the range of values of a rational function of n variables over a bounded region. *Computing*, 16:1–15, 1976.

[127] R.E. Moore. A test for existence of solutions to linear systems. *SIAM Journal of Numerical Analysis*, 14:611–615, 1977.

[128] R.E. Moore and S.T. Jones. Safe starting regions for iterative methods. *SIAM Journal of Numerical Analysis*, 14:1051–1065, 1977.

[129] T. Morin and R. Marsten. Branch-and-bound strategies for dynamic programming. *Operations Research*, 24(4):611–627, 1976.

[130] B.A. Murtagh and P.A. Saunders. Minos 5.1 user's guide. Technical Report SOL 83-20R, Dept. Operations Res., Stanford University, 1987.

[131] K. G. Murty and S.N. Kabadi. Some NP-Complete problems in Quadratic and Nonlinear Programming. *Mathematical Programming*, 39:117–130, 1987.

[132] 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.

[133] A. Neumaier. *Interval Methods for Systems of Equations*. Cambridge University Press, Cambridge, 1990.

[134] A. Neumaier. Second-order sufficient optimality conditions for local and global nonlinear programming. *Journal of Global Optimization*, pages 141–151, 1996.

[135] M. Novoa. Theory of preconditioners of interval Gauss-Seidel methods. Dept.of Math, U.of Southwestern Louisiana, 1993.

[136] J.M. Ortega. *Introduction to Parallel and Vector Solution of Linear Systems*. Plenum Press, New York, 1988.

[137] P.M. Pardalos and J. Crouse. A parallel algorithm for quadratic assignment problem. In *Proceedings Supercomputing*, pages 351–360. ACM Press, 1989.

[138] P.M. Pardalos and J.B Rosen. *Constrained Global Optimization: Algorithms and Applications*. Lectures Notes in Computer Science. Springer, 1987.

[139] P.M. Pardalos and G. Schnitger. Checking local optimality in constrained quadratic programming is NP-hard. *Operations Research Letters*, 7(1):33–35, 1988.

[140] P.M. Pardalos and S.A. Vavasis. Quadratic programation with one negative eigenvalue is NP-hard. *Journal of Global Optimization*, 1(1):15–22, 1991.

[141] J. Pekny and D. Miller. A parallel branch and bound algorithm for solving large asymmetric traveling salesman problems. *Mathematical Programming*, 55:17–33, 1992.

[142] F. Peták, Z. Hantos, Á. Adamicza, and B. Daróczy. Partitioning of pulmonary impedance: modeling vs. alveolar capsule approach. *J. Appl. Physiol.*, 75:513–521, 1993.

[143] A.T. Phillips and J.B. Rosen. A parallel algorithm for constrained concave quadratic global minimization. *Journal of Optimization Theory and Applications*, 42:421–448, 1990.

[144] J. D. Pintér, editor. *Global Optimization in Action*, volume 6 of *Noncovex optimization and its applications*. Kluwer Academic Publishers, 1996.

[145] J.D. Pintér. Continuous global optimization software: A brief review. *Newsletter of the Mathematical Programming Society*, 52:1–8, 1996.

[146] W.H. Press, Y.B.P. Flanner, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes: The art of Scientific Computing*. Cambridge University Press, 1986.

[147] E. Pruul, G. Nemhauser, and R. Rushmeier. Branch-and-bound and parallel computation: A historical note. *Op. Res. Letters*, 7(2):65–69, 1988.

[148] I. Quesada and I.E. Grrossmann. A global optimization algorithm for linear fractional and bilinear programs. *Journal of Global Optimization*, 6:39–76, 1995.

[149] M.J. Quinn. Analysis and implementation of branch and bound algorithms on a hypercube multicomputer. *IEEE Transactions on Computers*, 39(3):384–387, 1990.

[150] M.J. Quinn. *Parallel Computing: Theory and Practice*. McGraw-Hill, 1994.

[151] M.J. Quinn and N. Deo. An upper bound for the speedup of parallel best-first branch-and-bound algorithms. *BIT*, 26:35–43, 1990.

[152] H. Ratschek and J. Rokne. *Computer methods for the range of functions.* Ellis Horwood Lmt.(John Willey & Sons), 1984.

[153] H. Ratschek and J. Rokne. *New computer methods for global optimization.* Ellis Horwood Lmt.(John Willey & Sons), 1988.

[154] D. Ratz. *Automatische Ergebnisverifikation bei globalen Optimierungsproblemen.* PhD thesis, University of Karlsruhe, 1992.

[155] D. Ratz. On branching rules in second-order branch-and-bound methods for global optimization. *Scientific Computing and Validated Numerics*, pages 221–227, 1995.

[156] D. Ratz. On extended interval arithmetic and inclusion isotonicity. Preprint, U. Karlsruhe, Germany, 1996.

[157] D. Ratz. *Automatic Slope Computation and its Application in Nonsmooth Global Optimization.* Shaker Verlag, 1999.

[158] D. Ratz and T. Csender. On the selection of subdivision directions in interval branch and bound methods for global optimization. *Journal of Global Optimization*, 7:183–207, 1995.

[159] M.J. Rossman, R.J. Twery, and F.D. Stone. A solution to the traveling salesman problem by combinatorial programming. Chicago, 1996.

[160] C. Roucairol. A parallel branch and bound for the quadratic assignment problem. *Discrete Applied Mathematics*, 18:211–225, 1987.

[161] S.M. Rump. Algorithm for verified inclusions – theory and practice. In R.E. Moore, editor, *Reliability in Computing*, pages 109–126. Academic Press, 1988.

[162] H.S. Ryoo and N.V. Sahinidis. Global optimization of nonconvex NLPs and MINLPs with applications in process design. *Computers and Chemical Engineering*, 15(5):551–566, 1995.

[163] F. Schoen. Stochastic techniques for global optimization: A survey of recent advances. *Journal of Global Optimization*, 11(6):207–228, 1991.

[164] Ya. D. Sergeyev, P. Daponte, D. Grimaldi, and A. Molinaro. Two methods for solving optimization problems arising in electronic measurement and electrical engineering. *To appear in SIAM J. Optimization.*

[165] S.P. Shary. Algebraic approach to interval linear static identification, tolerance and control problems. *Reliable Computing*, 2(1):3–34, 1996.

[166] 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.

[167] S. Skelboe. Computation of rational interval functions. *BIT*, 14:87–95, 1974.

[168] S.L. Smith and R.B. Schnabel. Dynamic scheduling strategies for an adaptive, asynchronous parallel global optimization algorithm. Technical Report CS-US-625-92, University of Colorado, 1992.

[169] M. Snir, S.W. Otto, S. Huss-Lederman, D.W. Walker, and J. Dongarra. *MPI The Complete Reference*. MIT Press, 1996.

[170] F.J. Solis and J.B. Wets. Minimization by random search techniques. *Mathematics of Operations Research*, 6(1):19–50, 1981.

[171] R. Stateva and S. Tsvetkov. A diverse approach for the solution of the isothermal multiphase flash problem. Application to vapor-liquid-liquid systems. *The Canadian Journal of Chemical Engineering*, 72:722–734, 1994.

[172] O. Stauning. *Automatic Validation of Numerical Solutions*. PhD thesis, Dept. of Math. Modeling, Technical University Of Denmark DTU, 1997.

[173] D. Stevenson. IEEE standard for binary floating point arithmetic (IEEE/ANSI 754-1985). Technical report, Floating-Point Working Group, Microprocessor Standards Subcommittee. IEEE, 1985.

[174] T. Sunaga. Theory of interval algebra and its application to numerical analysis. *RAAG memoirs*, 3:29–46, 1958.

[175] P.G. Szabo and L.G. Casado. Packing of equal circles in a square. In *Volume of extended abstracts of the conference of PhD Students in Computer Science*, pages 95–97, Szeged, Hungary, 1998.

[176] P.G. Szabo, T. Csendes, L.G. Casado, and I. García. Equal circles packing in a square I. – Problem setting and bounds for optimal solutions. *Submitted for publication.*

[177] A. Torm and A. Zilinskas. *Global Optimization*, volume 350 of *Lectures Notes in Computer Science*. Springer-Verlag, 1989.

[178] J. Torrellas and D. Padua. The Illinois aggressive COMA multiprocessor project (I-ACOMA). In *6th Symposium on the Frontiers of Massively Parallel Computing*, 1996.

[179] H. Trienekens. *Parallel Branch and Bound Algorithms*. PhD thesis, Erasmus University, Rotterdam, 1990.

[180] 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.

[181] P.J.M van Laarhoven and E.H.L Aarts. *Simulated Annealing: Theory and Applications*. Kluwer Academic, 1987.

[182] V. Visweswaran and C.A. Floudas. New properties and computational improvement of the GOP algorithm for problems with quadratic objective functions and constraints. *Journal of Global Optimization*, 3:439–462, 1993.

[183] 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.

[184] 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.

[185] G. Walster, E. Hansen, and S. Sengupta. Test results for global optimization algorithm. *SIAM Numerical Optimization 1984*, pages 272–287, 1985.

[186] C.Z. Xu and F.C.M. Lau. *Load Balancing Parallel Computers – Theory and Practice*. Kluwer Academic, 1997.

[187] A.A. Zhigljavsky. *Theory of Global Random Search*. Kluwer Academic, 1991.

[188] Q. Zhu, M. Kam, and R. Yeager. Non-parametric identification of QAM constellations in noise. In *IEEE Conference on Acoustic, Speech and Signal Processing*, volume 4, pages 184–187, 1993.