Tesis Doctoral

BÚSQUEDAS GENÉTICAS: MÉTODOS DE OPTIMIZACIÓN GLOBAL Y OPTIMIZACIÓN COMBINATORIA

Doctorando: Juan José Domínguez Jiménez

Directora de la tesis: Dra. Inmaculada Medina Bulo

Depto. Lenguajes y Sistemas Informáticos
Universidad de Cádiz
Diciembre, 2008

A Belén por su cariño y que sin duda no hubiese hecho realidad esta tesis sin su apoyo, a nuestra hija Sandra y a mis padres.

UCA Universidad de Cádiz

CONFORMIDAD DEL DIRECTOR DE TESIS PARA LA TRAMITACIÓN DE LA TESIS DOCTORAL

Dª Inmaculada Medina Bulo, profesora del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Cádiz, siendo Directora de la Tesis titulada *"Búsquedas Genéticas: Métodos de optimización global y optimización combinatoria"*, realizada por el doctorando D. Juan José Domínguez Jiménez dentro del programa de doctorado Ingeniería en Automática y Electrónica Industrial, Ingeniería Informática y Sistemas Eléctricos perteneciente al bienio 2003/05, para proceder a los trámites conducentes a la presentación y defensa de la tesis doctoral arriba indicada, en aplicación del art. 30 de la Normativa Reguladora de Estudios de Tercer Ciclo de la Universidad de Cádiz, informa que se autoriza la tramitación de la tesis.

En Cádiz, a quince de diciembre de 2008

Fdo.: _____

Índice general

Índice de figuras

Índice de tablas

Índice de algoritmos

Capítulo 1

Introducción

En este capítulo de la tesis se describen los antecedentes y la motivación encontrada para la realización de la misma. A continuación se indican los objetivos que tratan de cubrirse con su desarrollo y una descripción de la estructura seguida en esta documentación. Finalmente se indican las publicaciones más relevantes asociadas a esta tesis.

1.1. Antecedentes y motivación

La ingeniería es una actividad profesional orientada a la resolución de problemas. Para la búsqueda de la solución se aplican los conocimientos científicos adquiridos, las habilidades desarrolladas, y sobre todo que la solución aportada sea tecnológicamente factible en ese momento.

Los tipos de problemas a los que se enfrentan los ingenieros a lo largo de la historia son esencialmente los mismos. Entre los problemas más comunes que se puede encontrar un ingeniero durante su carrera profesional están:

- Minimizar costes.
- Mejorar la productividad.
- Mejorar la eficiencia.
- Maximizar la producción.
- ...

Todos estos problemas, se engloban dentro de la categoría de **optimización de funciones**. Un problema de optimización se resuelve tomando una secuencia óptima de decisiones para maximizar (ganancias, velocidad, eficiencia, etc.) o minimizar (costes, tiempo, riesgo, error, etc.) siguiendo un determinado criterio. Desgraciadamente, en la mayoría de los problemas

de ingeniería la optimización es mucho más compleja, debido a las numerosas restricciones que se tienen que manejar en la resolución del problema. Las restricciones significan que no cualquier decisión es posible.

Los métodos clásicos de optimización existentes se caracterizan por producir buenos resultados cuando la función es *suave* y cumple unas determinadas características en la elección del punto inicial. Sin embargo, en la mayor parte de los problemas reales es complejo encontrar tales funciones de manera que sólo se pueden realizar optimizaciones locales. Fletcher [43] en un informe técnico afirma:

> We now have good methods for many classes of problem and many insights into why these methods are successful. There is a wealth of literature available. Nevertheless there are a number of open questions of interest, and new ideas continue to enrich the field.

J. Nocedal [117] se sorprende del progreso que ha experimentado en los últimos años la optimización de funciones, sobre todo con la aparición de nuevas estrategias, e incluso destaca las debilidades de algunos algoritmos clásicos.

> I was surprised to find that remarkable progress has been made in the last 15 years in the theory of unconstrained optimization, to the point that it is reasonable to say that we have a good understanding of most of the technique used in practice...We will see that the weaknesses of several classical algorithms that have fallen out of grace.

En la mayor parte de la bibliografía consultada se puede ver como a pesar de los avances logrados en la materia de optimización, existen ciertas debilidades en los métodos planteados. Por este motivo, se considera interesante el desarrollar nuevos métodos que puedan mejorar la consecución de óptimos globales, entendiéndose como tal el conseguir mejorar el resultado o bien mejorar las condiciones iniciales de partida del proceso de optimización. Esto queda reflejado por Michalewicz y Fogel [103]:

> A prerequisite to handling the real world as a multiplayer game is the ability to manipulate an arsenal of problem-solving techniques, namely, algorithms that have been developed for a variety of conditions. Unfortunately, it's almost always the case that the real world presents us with circunstances that are slightly or considerably different than are required by these methods.

Esta tesis presenta el desarrollo de nuevas estrategias de optimización aplicables a diversos problemas mediante el uso de *Algoritmos Genéticos (AG)*.

Los AGs, introducidos por Holland [65, 66] e impulsados en años sucesivos por Goldberg [55], uno de sus estudiantes, comprenden una de las cuatro áreas de la así denominada computación evolutiva, más conocida como *Evolutionary Computation*. Son, con diferencia, la técnica de computación evolutiva más empleada [79]. Gran parte de su éxito es debido a su mecanismo basado en la naturaleza genética de selección y supervivencia. David E. Goldberg [55] da la siguiente definición:

> Genetic algorithms are search algorithms based on the mechanics of natural selection and natural genetics. They combine survival of the fittest among string structures with a structured yet randomized information exchange to form a search algorithm with some of the innovative flair of human search.

Una de las dificultades observadas en los métodos tradicionales es la realización de búsquedas que convergen a la solución más cercana al punto inicial, por lo que la elección de éste es clave en la solución obtenida. En este sentido, y ante problemas en los que la naturaleza del espacio de soluciones se desconoce, los métodos globales de optimización son particularmente útiles, puesto que se desenvuelven con eficacia en espacios multidimensionales, multimodales y con múltiples discontinuidades, donde los métodos locales se muestran ineficaces [60]. Por este motivo, pensamos que la aplicación de los AGs puede jugar un papel fundamental en la optimización de funciones.

Los AGs son capaces de llegar a una solución global o próxima a ésta, en lugar de converger hacia soluciones locales como puede ocurrir con los métodos convencionales. Una de las características de los AGs es su posibilidad de explorar grandes superficies. Sin embargo, esta ventaja se puede convertir en un inconveniente debido a la posibilidad de que una pronta convergencia puede hacer que el algoritmo se centre en una solución medianamente buena, de manera que la optimización sea ineficiente.

Entendemos que la mejora de los resultados requiere diversificar al AG, permitiendo cubrir un gran espacio de soluciones, pero también intensificar la búsqueda en áreas prometedoras.

En este sentido, nuestras propuestas van encaminadas a cubrir esos dos aspectos: diversificación e intensificación. Para ello, las diferentes estrategias de búsqueda de soluciones desarrolladas en la tesis, se caracterizan por realizar sucesivas búsquedas locales, de manera que el algoritmo de optimización pueda escapar del ámbito de atracción de los óptimos locales, con objeto de conseguir llegar al óptimo global de la función.

Una forma de conseguir esos dos aspectos es la combinación de técnicas clásicas de optimización local con AGs. En la bibliografía existen numerosos ejemplos de esta hibridización de técnicas, tanto en el ámbito de variables continuas [50, 115, 163], como en el caso de variables combinatorias [14, 16, 67, 144, 147]. Estas combinaciones han recibido el nombre

genérico de *Búsquedas Locales Genéticas* [141]. Todas estas estrategias se caracterizan por emplear la técnica de optimización local como un operador más del AG. De este modo, una vez realizada la fase de reproducción, los individuos resultantes son sometidos a un proceso de optimización local antes de incorporarse a la nueva población.

Estamos convencidos de que apoyándonos en las técnicas de optimización local podemos conseguir que el AG obtenga mejores resultados. Sin embargo, nuestra propuesta no consiste en incorporar la búsqueda local como un operador más del AG, sino emplear el AG para explorar la región delimitada por la técnica de búsqueda local. Esto permitirá, por un lado, ampliar el horizonte de búsqueda de la técnica local, debido a la facilidad del AG para explorar eficientemente todo el espacio de soluciones posibles, y, por otro, enfocar la búsqueda a zonas con posibles óptimos, de acuerdo al grado de convergencia de la técnica local, por lo que se esperan mejores soluciones.

1.2. Objetivos

Esta tesis se enmarca en la aplicación de AGs a la optimización de funciones tanto en el ámbito de variables continuas como en los problemas de optimización combinatoria. En particular, se establecerán diversas estrategias basadas en sucesivas búsquedas acotadas, que permitan evolucionar al algoritmo hacia el óptimo de la función a optimizar.

Por tanto, los objetivos a conseguir en esta tesis son diseñar y validar diversas estrategias de resolución de problemas de optimización continua, por un lado, y de optimización combinatoria, por otro, mediante métodos evolutivos diversos, que toman como base los AGs.

Las aportaciones de la tesis consiste en el desarrollo de cuatro estrategias de optimización:

- En el ámbito de variables continuas:

  Búsqueda Lineal Genética trata de extender los tradicionales métodos de optimización que usan Búsqueda Lineal, permitiendo explorar la dirección de búsqueda en un intervalo mucho más amplio y que incluye incluso la rama de valores negativos. Dicha Búsqueda Lineal Extendida se realiza mediante un sencillo AG unidimensional.

  Búsqueda Genética en Cajas corresponde a una estrategia de resolución de sucesivos problemas acotados, centrados en torno a óptimos locales obtenidos mediante un AG multidimensional y que usa una función de evaluación con memoria.

- En el ámbito de variables combinatorias:

Búsqueda Genética en Vecindades es una adaptación de la Búsqueda Genética en Cajas al problema combinatorio. De modo que, se desarrollarían sucesivos problemas acotados, centrándonos en torno a la búsqueda de un óptimo local dentro de la vecindad del punto inicial.

Búsqueda Genética de Mutantes permite generar de forma automática mutantes de programas originales en el ámbito de las pruebas del software, y más en concreto en la técnica de mutaciones. Este algoritmo se integra dentro de la herramienta **GAmera** que permite automatizar el proceso de pruebas de mutaciones para composiciones de servicios en WS-BPEL 2.0 mediante el empleo de un AG. Una de las características de esta propuesta es la optimización del número de mutantes a generar, de manera que no se generarán todos los posibles mutantes.

Para evaluar la calidad de las estrategias desarrolladas, no sólo se aplicarán a funciones clásicas en la optimización de funciones, sino que se emplearán en la búsqueda de soluciones en problemas actuales. Así, en el ámbito de la optimización de variables continuas las estrategias desarrolladas se aplicarán al entrenamiento de redes neuronales. En el caso de optimización combinatoria se resolverá, por un lado el problema del viajante que engloba a numerosos problemas de producción, y por otro lado, en el ámbito de las pruebas del software, y más en concreto en la prueba de mutaciones, se desarrollará una estrategia para la generación automática de mutantes.

Por último, es cierto que el comportamiento de los AGs depende de una parametrización, que puede influir en sus resultados. Sin embargo, se pretende reducir ésta, así como establecer unas estrategias de búsquedas que hagan un algoritmo robusto, con un comportamiento similar independientemente de los valores de los parámetros de entrada.

1.3. Estructura de la tesis

El contenido de esta tesis se encuentra estructurado en tres partes, cada una formada por una serie de capítulos seguidos de un conjunto de anexos con información complementaria.

1.3.1. Fundamentos teóricos

La primera parte de la tesis comprende los fundamentos teóricos de la optimización de funciones.

En el capítulo 2 empieza definiendo un problema de optimización, como la resolución de la siguiente ecuación:

$$\text{mín } f(x) \qquad x \in \Omega \subseteq \mathcal{R}^n \tag{1.1}$$

Donde $x = (x_1, ..., x_n)$ es un vector y representa variables de decisión, $f(x)$ es llamada *función objetivo* y representa o mide la calidad de las decisiones (usualmente números enteros o reales) y Ω es el conjunto de decisiones factibles o restricciones del problema.

Un punto x^* se considera un *mínimo global sin restricciones* de la función f si no es peor que el resto, es decir:

$$f(x^*) \leq f(x), \quad \forall x \in \mathcal{R}^n \tag{1.2}$$

Una vez realizada la introducción a los dos conceptos sobre los que gira la tesis, se realiza una exposición de los diferentes métodos existentes para la optimización de funciones unidimensionales y multidimensionales en el ámbito de variables continuas. En el caso de la optimización multidimensional, se realiza una clasificación general de los distintos métodos, abordándose aquellos que se utilizan a lo largo de la tesis. De este modo, nos centramos en los denominados métodos de descenso, al ser empleados en la estrategia que hemos desarrollado bajo el nombre de Búsqueda Lineal Genética.

Se finaliza el capítulo haciendo un repaso a las técnicas de optimización existentes para variables combinatorias.

El capítulo 3 realiza una introducción a los AGs, dado que constituyen la base de los diferentes métodos de optimización desarrollados en la tesis. Se empieza dando unas definiciones de los conceptos más importantes, como son individuo, población, generación, aptitud, selección de individuos, reproducción, cruce y mutación.

A continuación se describen cuáles son los operadores, técnicas de selección y sustitución más empleadas, en el empleo de los AGs, así como el operador de nicho [55], cuyo concepto se modifica y amplia en nuestra propuesta denominada Búsqueda Genética en Cajas.

Se finaliza con la justificación teórica del funcionamiento de los mismos a través del teorema de esquemas [66], característica que lo diferencia del resto de métodos estocásticos existentes.

1.3.2. Optimización global

La segunda parte de la tesis se centra en los dos algoritmos de optimización desarrollados para la optimización global en funciones de variables continuas.

Búsqueda Lineal Genética

El capítulo 4 detalla la propuesta que hemos denominado Búsqueda Lineal Genética. Este nuevo método trata de extender los tradicionales métodos de optimización que usan búsqueda lineal, en concreto los denominados *métodos de descenso*. Éstos realizan un proceso iterativo de obtención

del óptimo, según la siguiente fórmula:

$$x^{k+1} = x^k + \alpha^k d^k \tag{1.3}$$

La aplicación de un método de descenso requiere resolver dos aspectos. En primer lugar, la *dirección de búsqueda*, d^k. Los diferentes métodos de descenso existentes se corresponden con los distintos modos de calcular estas direcciones de búsquedas [11, 42].

El segundo aspecto es la determinación de la *longitud del paso*, α^k, que se suele conocer como el *problema de la búsqueda lineal* [11]. Para ello se realiza una búsqueda lineal a lo largo de la dirección de búsqueda, de forma que satisfaga:

$$\alpha^k = \min_{\alpha \geq 0} f(x^k + \alpha d^k) \tag{1.4}$$

El tamaño del paso debe asegurar que cumple la denominada *condición de descenso*:

$$f(x^{k+1}) < f(x^k), \forall k \geq 0 \tag{1.5}$$

El método que hemos desarrollado, al que hemos denominado la Búsqueda Lineal Genética, consiste en aplicar un método de descenso, donde la determinación de la longitud de paso, α^k, se realiza con un AG en lugar de una técnica clásica de búsqueda lineal. Este AG permite explorar la dirección de búsqueda en un intervalo mucho más amplio, incluso en la rama de valores negativos.

Una vez revisado los conceptos y problemas de la búsqueda lineal clásica, se describe el método desarrollado: la Búsqueda Lineal Genética. Éste se basa en la aplicación de un AG unidimensional de codificación en coma flotante.

Posteriormente se realiza un estudio de la influencia de los diversos parámetros de entrada del algoritmo. Al final del capítulo se realiza un estudio comparativo entre la Búsqueda Lineal Genética y los métodos tradicionales de búsqueda lineal.

Búsqueda Genética en Cajas

El capítulo 5 presenta un nuevo método de optimización global, al que hemos denominado la Búsqueda Genética en Cajas. Entre sus características destaca el hecho de que la solución al problema se realiza mediante una búsqueda iterativa en diferentes regiones, denominadas cajas, de manera que cada solución se convierte en el centro de una nueva caja. El objetivo que persigue es explorar grandes superficies, pero a su vez, centrarse en zonas específicas del espacio de soluciones de la función objetivo, permitiendo búsquedas más selectivas.

La Búsqueda Genética en Cajas se presenta como una estrategia de resolución de sucesivos problemas acotados, centrados en torno a óptimos locales, mediante un AG multidimensional.

La Búsqueda Genética en Cajas incorpora como novedad una memoria que permite tener en cuenta la historia más reciente en el proceso de optimización, a fin de mejorar el proceso de búsqueda.

Posteriormente se realiza un estudio de la influencia de los diversos parámetros de entrada del algoritmo, así como los experimentos desarrollados con diversas funciones clásicas multiextremas para determinar la eficacia de la Búsqueda Genética en Cajas.

Finalmente, el capítulo 6 muestra las comparativas que hemos efectuado entre las dos propuestas presentadas en los capítulos anteriores, así como otros AGs presentes en la bibliografía.

1.3.3. Optimización combinatoria

La tercera parte de la tesis aborda la optimización de funciones combinatorias. Dado que estos métodos están muy adaptados a los problemas que se resuelven, las estrategias desarrolladas están pensadas para un tipo de problema.

Búsqueda Genética en Vecindades

El capítulo 7 presenta una nueva estrategia para la resolución de problemas combinatorios. La técnica desarrollada la hemos denominado Búsqueda Genética en Vecindades.

Se define para cada solución $i \in F$, una *vecindad* como el conjunto de soluciones, $N(i) \subset F$, que son en cierto sentido *cercanas a* la solución i. El significado de la expresión *cercana a* i se entiende como que dicha solución puede ser alcanzada desde i en un único *movimiento*. Un movimiento es un operador que transforma una solución en otra mediante pequeñas modificaciones [155]. Estos movimientos dependerán del problema que se está resolviendo.

El método que hemos desarrollado, denominado Búsqueda Genética en Vecindades, consiste en la realización de un procedimiento de búsqueda local de mejora continua con AGs. Para ello, una de las primeras aportaciones es la ampliación del concepto de vecindad. Así, dada una solución, $i \in F$, definimos la *vecindad extendida de orden* L, al conjunto de soluciones, $N_L(i) \subset F$, que se alcanzan desde la solución i en L movimientos consecutivos.

La Búsqueda Genética en Vecindades es una adaptación de la Búsqueda Genética en Cajas al problema combinatorio. De este modo, en lugar de realizar búsquedas en cajas, se realizan búsquedas en vecindades extendidas. Para ello, se generan vecindades extendidas de orden L donde el AG

busca una solución óptima. La mejor solución encontrada se convertirá en el centro de la próxima vecindad, permitiendo de este modo avanzar por el espacio de soluciones, pero centrándonos en pequeños espacios con objeto de aprovechar la potencia del AG. La generación de vecindades sucesivas permitirá llegar al óptimo de la función.

Para comprobar la eficacia de la Búsqueda Genética en Vecindades, ésta ha sido aplicada al problema del viajante simétrico [84]. De este modo, en el capítulo se muestra cómo codificar los individuos con un movimiento válido para dicho problema.

Posteriormente se muestra el estudio que hemos realizado para comprobar la influencia de los distintos parámetros de configuración en el resultado obtenido, así como los experimentos llevados a cabo con diferentes funciones de la biblioteca TSPLIB [133], para determinar el rendimiento de la Búsqueda Genética en Vecindades.

Búsqueda Genética de Mutantes

El capítulo 8 explica un uso novedoso de los AGs a la prueba de mutaciones. Desde que Bottaci [13] emplease los AGs en la prueba del software, han ido apareciendo numerosas aplicaciones de éstos [94, 101, 127]. Sin embargo, su uso ha estado limitado a la generación de casos de prueba.

En los últimos años, los procesos de negocios basados en composiciones de servicios en WS-BPEL [118] están rápidamente incrementándose, por lo que es importante prestar especial atención a la prueba del software en este contexto. En concreto, características del lenguaje WS-BPEL, como manejadores de fallo, concurrencia de eventos, etc, hacen que muchas de las técnicas y herramientas existentes no se puedan utilizar directamente con este lenguaje de composición de servicios [17]. Así, Palacios y col. [124] realizan una revisión de las técnicas y herramientas de pruebas para procesos de negocio BPEL, concluyendo

> La amplia mayoría de las investigaciones referenciadas no se ha publicado en revistas de reconocido prestigio, lo que indica que las pruebas de procesos BPEL es un campo en el que aún deben dedicarse importantes esfuerzos para obtener un mayor impacto, al menos desde un punto de vista académico.

La prueba de mutaciones es una técnica de prueba del software de caja blanca basada en errores [26, 58], que consiste en introducir fallos simples en el programa original, aplicando para ello operadores de mutación. Los programas resultantes reciben el nombre de *mutantes*. Cada operador de mutación se corresponde con una categoría de error típico que puede cometer el programador.

Una de las principales dificultades de aplicar la prueba de mutaciones es la existencia de *mutantes equivalentes*. Éstos presentan el mismo compor-

tamiento que el programa original, es decir, la salida del mutante y del programa original es siempre la misma.

Otro de los principales inconvenientes de la prueba de mutaciones es el alto coste computacional que implica. Esto es debido a que disponemos normalmente de un número grande de operadores de mutación que generan un elevado número de mutantes, cada uno de los cuales debe ejecutarse frente al conjunto de casos de prueba hasta que muera.

Para poder aplicar la prueba de mutaciones a las composiciones de servicios WS-BPEL es necesario disponer de una herramienta capaz de generar los mutantes. Dado que no existe ninguna herramienta de generación de mutantes para WS-BPEL, nuestra investigación se ha centrado en el desarrollo de ésta, a la que hemos denominado *GAmera*, cuya finalidad es la generación automática de mutantes de composiciones de servicios web en WS-BPEL.

El sistema de generación automático de mutantes para composiciones de servicios WS-BPEL que se ha desarrollado en el transcurso de esta tesis, al que se ha denominado *GAmera*, está formado por tres componentes: el analizador, el generador de mutantes y el sistema de ejecución, el cuál ejecuta y también evalúa a los mutantes. El analizador recibe como entrada la definición del proceso original WS-BPEL y genera la información necesaria para que el generador de mutantes pueda generar a los diferentes mutantes. Durante la generación, el sistema ejecutará a los mutantes frente a un conjunto de casos de prueba y determinará los mutantes potencialmente equivalentes.

El *generador de mutantes* está formado por dos elementos. El primero, denominado Búsqueda Genética de Mutantes, es un AG en el que cada individuo representa a un mutante, encargándose de generar de forma automática un conjunto de mutantes. El segundo elemento es el conversor, que transforma un individuo del AG a un mutante.

Una de las características del generador es que sólo genera un subconjunto de todos los posibles mutantes existentes para el programa original.

El generador de mutantes tiene un doble objetivo. Por un lado, generar los mutantes para composiciones de servicios WS-BPEL y, por otro, detectar los mutantes potencialmente equivalentes.

Para comprobar la eficacia de GAmera, ésta ha sido aplicada a diversas composiciones WS-BPEL. Los resultados que hemos obtenido se muestran al final del capítulo 8.

1.3.4. Conclusiones

Finalmente, el capítulo 9 recoge un resumen de las aportaciones desarrolladas en esta tesis, junto con las conclusiones obtenidas. Se termina el capítulo con una propuesta de futuras líneas de investigación relacionadas con la aplicación de los AGs a la optimización de funciones.

1.3.5. Anexos

El documento de la tesis incorpora una serie de apéndices. El apéndice A recoge las funciones de optimización continua empleadas en las pruebas realizadas a los métodos propuestos, indicándose el óptimo global así como el valor de la función en dicho punto. El apéndice B contiene una revisión del resto de métodos de optimización citados en el capítulo 2 y que no fueron detallados dado que no se empleaban en las estrategias desarrolladas.

1.4. Publicaciones

Las propuestas desarrolladas en esta tesis junto a sus resultados han sido publicados en distintos foros, con objeto de poder evaluar y contrastar tanto la calidad de los algoritmos desarrollados como las conclusiones obtenidas. Entre las publicaciones destacamos las siguientes:

- Lozano, S.; Domínguez, J. J.; Guerrero, F.; Onieva L. & Larrañeta, J.

  Training Feedforward Neural Network Using a Genetic Line Search.

  Intelligent Engineering Systems Through Artificial Neural Networks, Vol. 7, pág. 119–124.

  ASME Press, New York. 1997.
  ISBN: 0–7918–0064–4.

- Lozano, S.; Domínguez, J. J.; Guerrero, F. & Larrañeta, J.

  Genetic Line Search.

  Actas del EUROXV-INFORMS XXXIV. Joint International Meeting, pág. 45. Barcelona, España. 1997.

- Domínguez, J. J.; Lozano, S.; Canca, D. & Cortés, P.

  Búsqueda Lineal Genética.

  Actas del XXIII Congreso Nacional de Estadística e Investigación Operativa, pág. 43.123–43.124. Valencia, España. 1997.
  CS–94–1997.

- Domínguez, J. J.; Lozano, S.; Guerrero, F. & Dobado, D.

  Búsqueda Genética en Cajas.

  Actas del XXIV Congreso Nacional de Estadística e Investigación Operativa, pág. 487–488. Murcia, España. 1998.

- Domínguez, J. J.; Lozano, S.; Guerrero, F. & Canca, D.

  Optimización sin restricciones usando búsqueda genética en cajas.

Actas de las III Jornadas Científicas Andaluzas en Tecnologías de la Información: Innovaciones en Informática, Electrónica y Automática, pág. 30–37. Cádiz, España. 1998.
CA–345/1998, ISBN: 84–89867–11–9.

- Lozano, S.; Domínguez Jiménez, J. J.; Guerrero, F.; Onieva L. & Larrañeta J.

  Unconstrained Optimization Using Genetic Box Search.

  Advances in Soft Computing. Engineering Design and Manufacturing. pág. 379–389.

  Springer-Verlag, Londres. 1999.
  ISBN: 1–85233–062–7.

- Lozano, S.; Domínguez, J. J.; Guerrero, F. & Smith, K.

  A new optimization method: Genetic Line Search.

  Intelligent Systems Design and Applications (ISDA 2001). San Francisco, Estados Unidos, 2001.

  Computational Science – ICCS 2001, Lecture notes in computer science, vol. 2074, pág. 318–326, Springer Verlag, 2001.
  ISBN: 3–540–42233–1.

- Domínguez, J. J.; Lozano, S.; Calle, M.

  Genetic Neighborhood Search.

  2nd International Conference on Computational Science
  ICCS 2002. Amsterdam, Holanda.

  Computational Science – ICCS 2002, Lecture notes in computer science, vol. 2331, pág. 544–553, Springer Verlag, 2002.
  ISBN: 3–540–43594–8.

- Domínguez Jiménez, J. J.; Lozano Segura, S.; Calle Suárez, M.

  Búsqueda genética en vecindades: Aplicación al problema del viajante simétrico.

  Actas del primer Congreso Español de Algoritmos Evolutivos y Bioinspirados — AEB'02, pág. 89–95. Mérida, España. 2002.
  BA–25–2002, ISBN: 84–607–3913–9.

- Domínguez, J.J.; Lozano, S.; Calle, M.; Smith, K.

  A new method for combinatorial optimization: genetic neighborhood search.

  Neural Network World, International Journal on Non-Standard Computing and Artificial Intelligence, vol. 12, nº 6, pág. 533–548, 2002.
  ISSN 1210-0552

- Domínguez Jiménez, J.J.; Estero Botaro, A.; Medina Bulo, I.

  Una arquitectura para la prueba de mutaciones de composiciones de servicios web basada en algoritmos genéticos

  Actas de Talleres de Ingeniería del Software y Bases de Datos. Taller celebrado en el marco de las XIII Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2008) Vol. 2, no. 4, pág. 27–32 , 2008 ISSN 1988-3455.

- Domínguez Jiménez, J.J.; Estero Botaro, A.; Medina Bulo, I.

  A framework for mutant genetic generation for WS-BPEL

  Proceedings of the 35th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2009), Špindlerův Mlýn (República Checa), 2009

- Domínguez Jiménez, J.J.; Estero Botaro, A.; Medina Bulo, I.

  Mutant Generation for Web Services Compositions with Genetic Algorithms

  Proceedings of IASK International Conference E-Activity and Leading Technologies (E-ALT08), Madrid (España), 2008

- Domínguez Jiménez, J.J.; Estero Botaro, A.; Medina Bulo, I.; García Domínguez, A.

  GAmera: An Automatic Mutant Generation System for WS-BPEL Compositions

  The 18th International World Wide Web Conference (WWW09), Madrid (España), 2009 (Pendiente de aceptación)

Parte I

Fundamentos Teóricos

Capítulo 2

Optimización de funciones

En este capítulo se realizará una revisión de las técnicas de optimización existentes. Para ello se empezará con la definición del problema de optimización y las condiciones que debe cumplir un óptimo. A continuación se detallarán las técnicas para la optimización local, tanto en funciones unidimensionales como multidimensionales. Estas técnicas sólo producen buenos resultados cuando la función es suave. Posteriormente se describirán las técnicas más conocidas para la optimización global, empleadas en funciones que se caracterizan por poseer numerosos óptimos locales, lo que dificulta las estrategias de búsquedas de mínimos. Se finalizará haciendo un resumen de las distintas estrategias más empleadas en la resolución de problemas de optimización combinatorios.

2.1. Introducción

De forma genérica, puede definirse la optimización como aquella ciencia encargada de determinar las mejores soluciones a problemas matemáticos que a menudo modelan una realidad física. Los problemas complejos de optimización multidimensionales pueden encontrarse en ingeniería, economía, geofísica y prácticamente en todos los campos de la ciencia. El objetivo que se persigue al resolver un problema de optimización es encontrar una solución óptima con un coste computacional razonable. En este aspecto, la optimización numérica ha adquirido mucha atención entre la comunidad científica durante las últimas décadas, y quizás lo más confuso para el diseñador reside en decidir qué algoritmo de optimización se ajusta mejor a las características del problema bajo análisis.

Un problema de optimización intenta dar respuesta a un tipo general de problemas de la forma:

$$\text{mín } f(x) \qquad x \in \Omega \subseteq \mathcal{R}^n \tag{2.1}$$

Donde $x = (x_1, ..., x_n)$ es un vector y representa variables de decisión, $f(x)$ es llamada *función objetivo* y representa o mide la calidad de las decisiones (usualmente números enteros o reales) y Ω es el conjunto de decisiones factibles o restricciones del problema.

Algunas veces es posible expresar el conjunto de restricciones Ω como solución de un sistema de igualdades o desigualdades.

$$\begin{array}{rcl} g(x_1, ..., x_n) & \leq & 0 \\ h(x_1, ..., x_n) & = & 0 \end{array} \tag{2.2}$$

Dentro de esa definición también se incluyen los problemas de maximización de funciones, puesto que:

$$\text{máx } f(x) = -\text{mín}(-f(x)) \tag{2.3}$$

Un punto x^* se dice que es un *mínimo local sin restricciones* de la función f si no es peor que ningún vecino, es decir, si existe un $\epsilon > 0$, tal que:

$$f(x^*) \leq f(x), \quad \forall x \quad \|x - x^*\|_2 < \epsilon \tag{2.4}$$

Un punto x^* se considera un *mínimo global sin restricciones* de la función f si no es peor que el resto, es decir:

$$f(x^*) \leq f(x), \quad \forall x \in \mathcal{R}^n \tag{2.5}$$

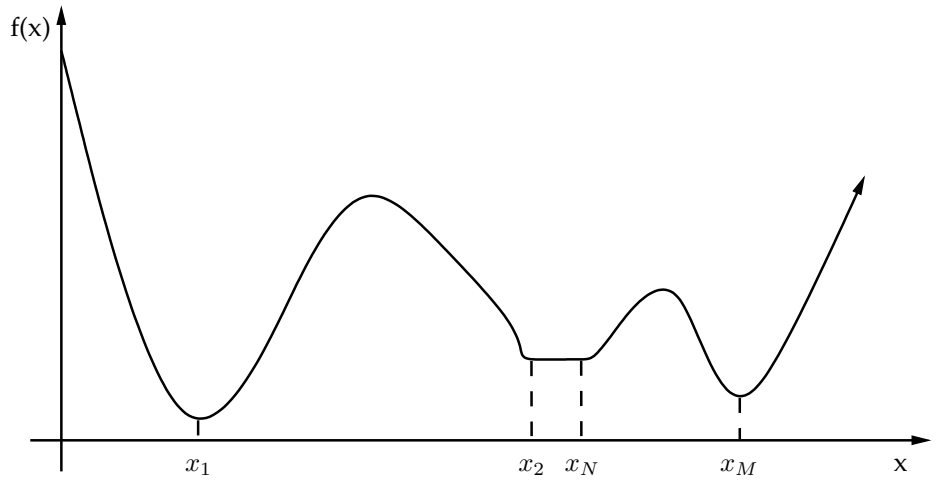
El concepto de mínimo local o global se entiende como *estricto*, en el caso de que las correspondientes inigualdades, (2.4) y (2.5), sean estrictas. La figura 2.1 muestra gráficamente estas definiciones.

De forma similar, podemos definir los *máximos* locales y globales. Pero dada la relación existente entre un problema de maximización y uno de minimización (2.3), un punto x^* es un máximo local o global de una función f, si x^* es un mínimo local o global de la función $-f$.

2.2. Condiciones de optimalidad

Las condiciones necesarias [11, 42] para que un punto x^* se considere un mínimo local de una función f son las siguientes:

1. Ser un *punto estacionario*, es decir, $\nabla f(x^*) = 0$. Esta condición nos asegura que la función f no tenga inclinación para ninguna dirección s desde x^*, es decir:
$$s^T \nabla f(x^*) = 0$$



x_1 = **Mínimo global estricto**
$x_2, x_3, ..., x_N$ = **Mínimos locales**
x_M = **Mínimo local estricto**

Figura 2.1: Mínimo global y mínimos locales

2. La matriz $\nabla^2 f(x^*)$, es decir, el Hessiano, sea semidefinida positiva. Esto nos permite asegurar que la curvatura en el punto x^* para cualquier dirección s es no-negativa, es decir:

$$s^T \nabla^2 f(x^*) s \geq 0$$

Las condiciones suficientes [11, 42] para que un punto x^* se considere un mínimo local estricto de una función f son las siguientes:

1. Ser un punto estacionario.

2. El Hessiano sea definido positivo.

Estas condiciones son fundamentales para el desarrollo de los métodos de optimización sin restricciones.

2.3. Optimización unidimensional

Las técnicas empleadas en funciones de una dimensión se basan en un proceso iterativo con objeto de reducir el intervalo donde se encuentra el mínimo. Para ello, la función objetivo debe ser *unimodal* en dicho intervalo, es decir, $f(x)$ es unimodal en el intervalo $[a, b]$, si existe un único $x^* \in [a, b]$, tal que:

$$\forall m, n \in [a, b], \text{ donde } m < n \begin{cases} m < x^* \Longrightarrow f(m) > f(n) \\ n > x^* \Longrightarrow f(m) < f(n) \end{cases} \tag{2.6}$$

De este modo, si una función f es unimodal en un intervalo $[a, b]$, según la condición (2.6), el mínimo se encontrará en el intervalo $[m, b]$ ó $[a, n]$, dependiendo de los valores de $f(m)$ y $f(n)$. La demostración de (2.6) se puede encontrar en [8].

Los diferentes métodos de optimización unidimensionales emplean este resultado, de forma que se va subdividiendo continuamente el intervalo $[a, b]$ hasta encontrar un *intervalo de incertidumbre* lo suficientemente pequeño donde se encuentra el mínimo. La diferencia entre los distintos métodos es la forma de seleccionar los puntos interiores del intervalo, es decir, m y n. La figura 2.2 muestra los métodos más comunes que se suelen emplear para la optimización de funciones unidimensionales. Solamente los métodos de *Fibonacci*, *sección áurea* y *ajuste cuadrático* son no derivativos; el resto hacen uso de, al menos, la primera derivada, y en el caso del método de *Newton* emplea también la segunda derivada de la función. Aunque los métodos de interpolación polinomial son los más potentes, suelen imponer unas condiciones de suavidad a las funciones a optimizar que en los problemas reales no suelen cumplirse. Este es el motivo por el que nuestro estudio se ha centrado principalmente en los métodos de búsqueda dicotómica, Fibonacci y sección áurea. No obstante, en el apéndice B puede encontrarse amplia información sobre el resto de los métodos de optimización.

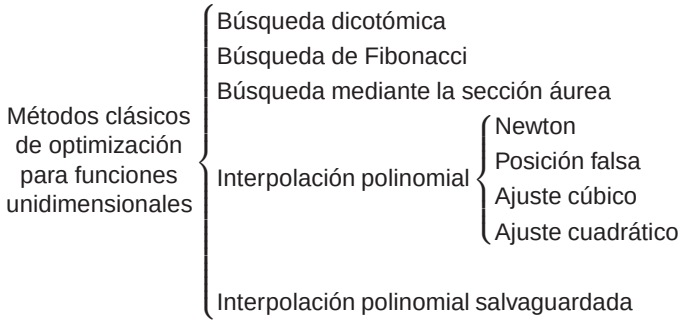


Figura 2.2: Métodos de optimización local para funciones unidimensionales

2.3.1. Búsqueda dicotómica

Supongamos que queremos minimizar una función f en un intervalo $[a, b]$, siendo f unimodal. En este método, se escogen dos puntos m y n

situados a una distancia $\epsilon > 0$ de forma simétrica del punto central del intervalo $(a+b)/2$. Con estos valores y apoyándonos en el resultado dado por (2.6) construimos un nuevo intervalo. Este proceso se va repitiendo continuamente hasta que el nuevo intervalo construido alcance el valor de incertidumbre deseado, I, es decir, la longitud del intervalo que se obtiene tras realizar k iteraciones, $[a^k, b^k]$, es menor que I. El algoritmo 2.1 refleja una posible implementación de la búsqueda dicotómica.

Algoritmo 2.1 Búsqueda dicotómica

Dicotómica : $a \times b \times \epsilon \times I \longrightarrow a \times b$
$a^k \longleftarrow a$
$b^k \longleftarrow b$
Mientras $((b^k - a^k) \geq I)$
  $m \longleftarrow \dfrac{a^k + b^k}{2} - \epsilon$
  $n \longleftarrow \dfrac{a^k + b^k}{2} + \epsilon$
  Si $(f(m) < f(n))$
    $b^k \longleftarrow n$
  Si no
    $a^k \longleftarrow m$
Devolver $\left[a^k, b^k\right]$

2.3.2. Búsqueda de Fibonacci

Este método se basa en los números de Fibonacci, definidos por la siguiente regla:

$$\begin{aligned} \forall k > 1,\ F^k &= F^{k-1} + F^{k-2} \\ F^0 &= F^1 = 1 \end{aligned} \tag{2.7}$$

Siendo $[a, b]$ el intervalo donde la función f es unimodal, y N el número máximo de muestreos o evaluaciones de la función que deseamos realizar, en función de los números de Fibonacci, se escogen dos puntos simétricamente distantes de los extremos, λ^k y μ^k, respectivamente, según la siguiente relación:

$$\begin{aligned} \forall k = 1, \ldots, N-1, \quad \lambda^k &= a^k + \frac{F^{N-k-1}}{F^{N-k+1}}(b^k - a^k) \\ \forall k = 1, \ldots, N-1, \quad \mu^k &= a^k + \frac{F^{N-k}}{F^{N-k+1}}(b^k - a^k) \end{aligned} \tag{2.8}$$

Según la relación (2.6) y la definición de λ^k y μ^k (2.8), suponiendo que $f(\lambda^k) > f(\mu^k)$ se obtiene que la longitud del nuevo intervalo de búsqueda es:

$$\begin{aligned} b^{k+1} - a^{k+1} &= b^k - \lambda^k \\ &= b^k - a^k - \frac{F^{N-k-1}}{F^{N-k+1}}(b^k - a^k) \\ &= \frac{F^{N-k}}{F^{N-k+1}}(b^k - a^k) \end{aligned} \tag{2.9}$$

De esta forma, la longitud del intervalo de búsqueda, según (2.9), tras $k = N - 1$ iteraciones vendrá dado por:

$$\begin{aligned} b^N - a^N &= \frac{F^1}{F^2}(b^{N-1} - a^{N-1}) \\ &= \frac{F^1}{F^2}\frac{F^2}{F^3}(b^{N-2} - a^{N-2}) \\ &= \ldots = \frac{F^1}{F^N}(b^1 - a^1) = \frac{b-a}{F^N} \end{aligned} \tag{2.10}$$

Por tanto, en la búsqueda de Fibonacci el intervalo de incertidumbre deseado se obtiene especificando previamente el número máximo de evaluaciones que se quiere realizar (N).

En el proceso de optimización, si consideramos que $f(\lambda^k) > f(\mu^k)$, entonces el siguiente intervalo vendrá definido por $[\lambda^k, b]$, por lo que el nuevo punto a muestrear vendrá dado por:

$$\begin{aligned} \lambda^{k+1} &= a^{k+1} + \frac{F^{N-k-2}}{F^{N-k}}(b^{k+1} - a^{k+1}) \\ &= \lambda^k + \frac{F^{N-k-2}}{F^{N-k}}(b^k - \lambda^k) \qquad \text{(sustituyendo por (2.8))} \\ &= a^k + \frac{F^{N-k-1}}{F^{N-k+1}}(b^k - a^k) + \frac{F^{N-k-2}}{F^{N-k}}\left(1 - \frac{F^{N-k-1}}{F^{N-k+1}}\right)(b^k - a^k) \\ &= a^k + \left(\frac{F^{N-k-1} + F^{N-k-2}}{F^{N-k+1}}\right)(b^k - a^k) \\ &= a^k + \frac{F^{N-k}}{F^{N-k+1}}(b^k - a^k) = \mu^k \end{aligned} \tag{2.11}$$

Como podemos ver sólo se necesita realizar una única evaluación de la función en la iteración $k + 1$. Esto hace que la búsqueda de Fibonacci sea más eficiente que la búsqueda dicotómica, en el sentido de que cada iteración sólo requiere una evaluación de la función, salvo en la primera donde emplea dos, mientras que la búsqueda dicotómica en cada iteración se realizan dos evaluaciones de la función.

Algoritmo 2.2 Búsqueda de Fibonacci

Fibonacci : $a \times b \times \epsilon \times I \longrightarrow a \times b$
$N \longleftarrow 1$
Mientras $(F^N \leq (b-a)/I)$
    $N \longleftarrow N+1$
$\lambda^k \longleftarrow a + (F^{N-2}/F^N)(b-a)$
$\mu^k \longleftarrow a + (F^{N-1}/F^N)(b-a)$
$k \longleftarrow 1$
Mientras $(k \neq N-2)$
    Si $(f(\lambda^k) > f(\mu^k))$
        $a \longleftarrow \lambda^k$
        $\lambda^k \longleftarrow \mu^k$
        $\mu^k \longleftarrow a + (F^{N-k-1}/F^{N-k})(b-a)$
    Si no
        $b \longleftarrow \mu^k$
        $\mu^k \longleftarrow \lambda^k$
        $\lambda^k \longleftarrow a + (F^{N-k-2}/F^{N-k})(b-a)$
    $k \longleftarrow k+1$
$\mu^k \longleftarrow \lambda^k + \epsilon$
Si $(f(\lambda^k) > f(\mu^k))$
    $a \longleftarrow \lambda^k$
Si no
    $b \longleftarrow \mu^k$
Devolver $[a, b]$

El algoritmo 2.2 muestra una posible implementación de la búsqueda de Fibonacci, donde además de proporcionar el intervalo de incertidumbre, I, se proporciona un parámetro adicional ya que que en la última iteración los dos puntos, λ^k y μ^k, son idénticos, por lo que se requiere una pequeña constante, ϵ, para poder diferenciarlos.

2.3.3. Búsqueda mediante la sección áurea

El inconveniente de la búsqueda de Fibonacci es el tener que expresar la exactitud deseada en función del número de mediciones a realizar (2.10).

Si hacemos que el número de mediciones tienda a infinito observamos que la relación existente entre los números de Fibonacci es:

$$\lim_{N\to\infty} \frac{F^{N-1}}{F^N} = \frac{1}{\tau} \approx 0,618 \tag{2.12}$$

donde τ es una de las raíces de la ecuación $\tau^2 + \tau - 1 = 0$, denominada *razón áurea*, De este modo, siendo $[a, b]$ el intervalo donde la función f es unimodal, se escogen dos puntos tales que:

$$\begin{aligned} \lambda^k &= a^k + (1 - \tau)(b^k - a^k) \\ \mu^k &= a^k + \tau(b^k - a^k) \end{aligned} \tag{2.13}$$

Estos puntos, al igual que con Fibonacci, verifican que en cada iteración sólo necesitan evaluar la función una única vez. El algoritmo 2.3 muestra una posible implementación de este método.

Algoritmo 2.3 Búsqueda mediante la sección áurea

Razón-áurea : $a \times b \times I \longrightarrow a \times b$
$\tau \longleftarrow 0,618$
$a^k \longleftarrow a$
$b^k \longleftarrow b$
$k \longleftarrow 1$
Mientras $((b^k - a^k) \geq I)$
  $\lambda^k \longleftarrow a^k + \tau(b^k - a^k)$
  $\mu^k \longleftarrow a^k + (1 - \tau)(b^k - a^k)$
  $k \longleftarrow k + 1$
  Si $(f(\lambda^k) < f(\mu^k))$
    $b^k \longleftarrow \mu^k$
    $\mu^k \longleftarrow \lambda^k$
    $\lambda^k \longleftarrow a^k + (1 - \tau)(b^k - a^k)$
  Si no
    $a^k \longleftarrow \lambda^k$
    $\lambda^k \longleftarrow \mu^k$
    $\mu^k \longleftarrow a^k + \tau(b^k - a^k)$
Devolver $\left[a^k, b^k\right]$

2.4. Optimización multidimensional

Consideremos ahora el problema de optimización de una función $f(x)$, con $x \in \mathcal{R}^n$. Al contrario que en la optimización unidimensional, no es posible describir un esquema genérico de cómo se realiza el proceso de optimización, ya que las diferentes técnicas existentes dependen del tipo de función al que nos enfrentamos.

Existe una bibliografía muy extensa que versa sobre los diferentes algoritmos de optimización convencionales existentes para afrontar la optimización de funciones. Sin embargo, puede establecerse una clasificación preliminar de los métodos de optimización en dos grandes bloques, según los requisitos necesarios que se exige a la función a optimizar. Así, podemos hablar de *métodos derivativos*, que hacen uso de $\nabla f(x)$, y en algunos casos de $\nabla^2 f(x)$, y *no derivativos*, que emplean únicamente los valores de la función f. La figura 2.3 muestra una clasificación con los métodos más conocidos.

Reseñar que los métodos no derivativos que aparecen en dicha clasificación son puros, en el sentido de que no hacen uso de la información de ninguna de las derivadas de la información, y tampoco hacen aproximaciones a las mismas. Sólo emplean el valor de la función que se quiere optimizar. Sin embargo, es posible hacer que cualquier método derivativo se convierta en uno no derivativo, si se realizan aproximaciones por diferencias finitas a las derivadas.

Según Horst y Tuy [68], es posible diferenciar entre técnicas que permiten localizar mínimos locales y las que pueden localizar los mínimos globales. Así, las *técnicas estándar de programación no lineal* se caracterizan porque pueden localizar mínimos locales. Estas técnicas se dirigen hacia la solución más próxima siguiendo una dirección de búsqueda y sin capacidad para discernir entre solución local y global. Pero cuando nos enfrentamos a un problema de *optimización global multiextrema*, es decir, cuando la función que tenemos que minimizar se caracteriza por poseer diferentes mínimos locales, las técnicas clásicas de optimización no nos sirven, ya que no son capaces de identificar cuando un mínimo es local o global. Por este motivo, aparecen los *métodos de resolución global*.

Las técnicas de programación no lineal se pueden clasificar en las que hacen uso de las derivadas, como los *métodos de descenso* o las que no emplean información alguna de las derivadas, como los métodos de *descenso coordenado*.

2.4.1. Métodos de descenso

Los métodos de descenso son los métodos más antiguos y conocidos de optimización de funciones multidimensionales. Son derivativos, haciendo uso de la primera o incluso de la segunda derivada de la función f que se quiere minimizar. El nombre de método de descenso tiene origen en que éstos imponen en cada iteración generada una *condición de descenso* [52]:

$$f(x^{k+1}) < f(x^k), \forall k \geq 0 \tag{2.14}$$

El algoritmo 2.4 muestra un esquema general de estos métodos.

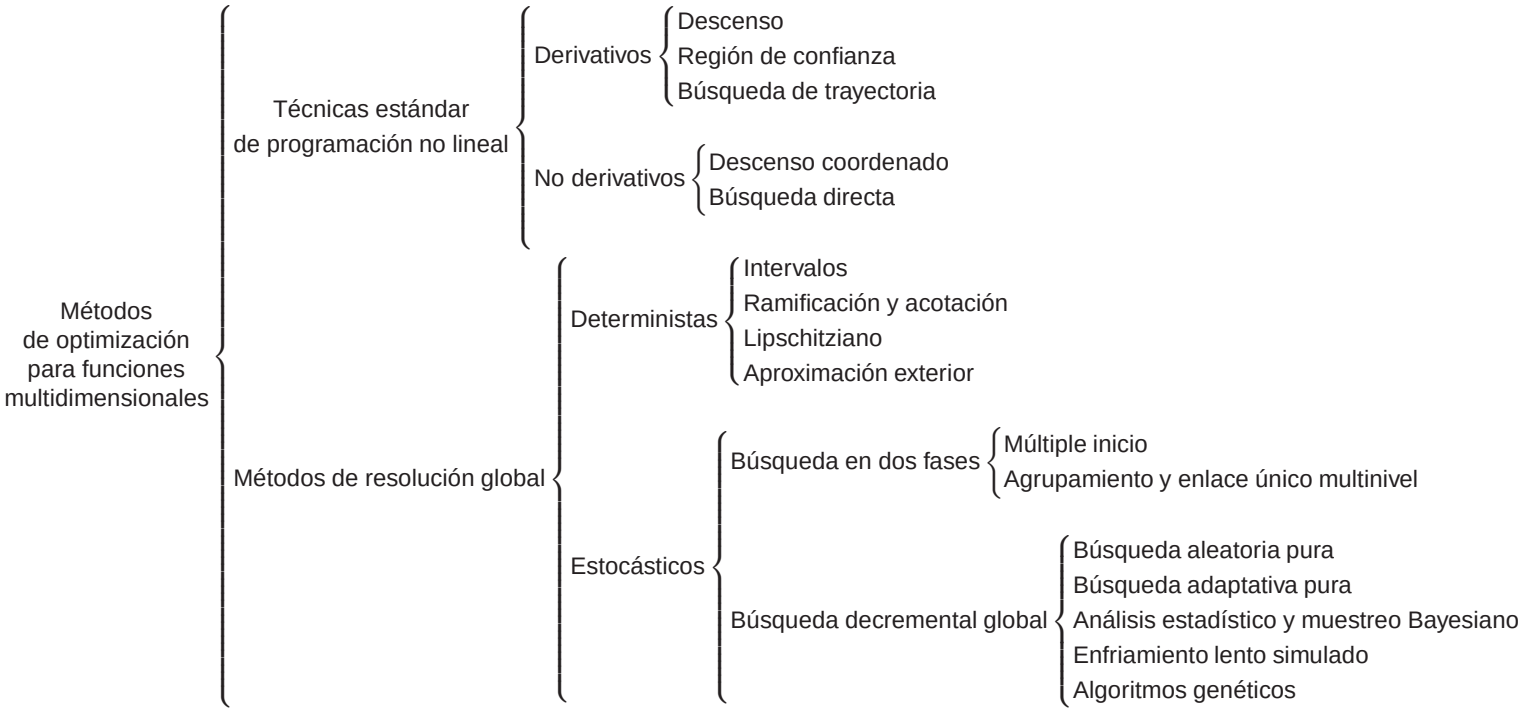


Figura 2.3: Clasificación de los métodos de optimización multidimensionales

Algoritmo 2.4 Esquema de un método de descenso

Descenso : $x \times \epsilon \longrightarrow x^*$
$k \longleftarrow 0$
$x^k \longleftarrow x$
$d^k \longleftarrow$ direccion_descenso(x^k)
$\alpha^k \longleftarrow$ longitud_paso(x^k, d^k)
$x^{k+1} \longleftarrow x^k + \alpha^k d^k$
Mientras $\left(\frac{\|x^{k+1} - x^k\|}{\|x^{k+1}\|} \leq \epsilon\right)$
  $x^k \longleftarrow x^{k+1}$
  $d^k \longleftarrow$ direccion_descenso(x^k)
  $\alpha^k \longleftarrow$ longitud_paso(x^k, d^k)
  $x^{k+1} \longleftarrow x^k + \alpha^k d^k$
Devolver x^k

Podemos ver que en cada iteración se realizan los siguientes pasos:

1. Se realiza una *prueba de convergencia* para comprobar si la solución disponible, x^k, satisface las condiciones de convergencia deseadas y, en caso afirmativo, dar por finalizada la ejecución del algoritmo.

2. Obtener un vector, d^k, denominado *dirección de búsqueda*.

3. Calcular la *longitud del paso*, α^k, para la que se asegura la condición de descenso, es decir, $f(x^k + \alpha^k d^k) < f(x^k)$.

4. Establecer el nuevo punto, según la fórmula:

$$x^{k+1} = x^k + \alpha^k d^k \tag{2.15}$$

Como podemos ver, la elección de un método de descenso requiere resolver dos cuestiones: calcular la longitud del paso, α^k, que se suele conocer como el *problema de la búsqueda lineal*[1], y determinar la dirección de búsqueda, d^k. Para cumplir la condición de descenso, esta dirección de búsqueda debe ser una *dirección de descenso* para garantizar una reducción de la función f para un paso $\alpha > 0$, es decir:

$$\nabla f(x^k)^T d^k < 0 \tag{2.16}$$

[1]Esto origina que en la bibliografía algunos autores denominen a los métodos de descenso como *métodos de búsqueda lineal*.

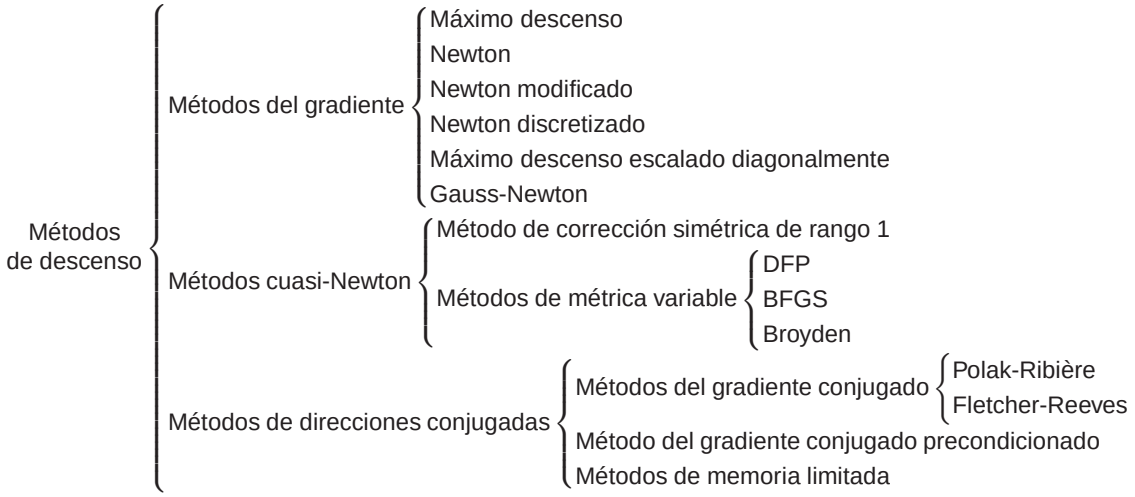


Figura 2.4: Clasificación de los métodos de descenso

Los diferentes métodos de descenso existentes se corresponden con los distintos modos de calcular estas direcciones de búsquedas. La figura 2.4 muestra una clasificación con los métodos de descenso más conocidos.

Los *métodos del gradiente* son los que utilizan la información de la primera derivada (*máximo descenso*) y, en algunos casos, de la segunda derivada (*Newton*) de la función f para determinar la nueva dirección.

Los *métodos cuasi-Newton* intentan conseguir una rápida convergencia, pero sin necesidad de tener que recurrir a evaluar el Hessiano de la función. Para ello, intentan realizar aproximaciones al mismo.

Finalmente, los *métodos de direcciones conjugadas* intentan ser eficientes y fiables sin necesidad de tener que almacenar una matriz como en el caso de los métodos cuasi-Newton. Esto les hace válidos sobre todo en problemas donde el número de dimensiones es elevado.

2.4.1.1. La búsqueda lineal

Para la determinación de la longitud del paso α^k se realiza una búsqueda lineal a lo largo de la dirección de descenso, de forma que satisfaga:

$$\alpha^k = \min_{\alpha \geq 0} f(x^k + \alpha d^k) \tag{2.17}$$

Existen dos categorías de búsquedas lineales (figura 2.5): las exactas y las inexactas. La *búsqueda lineal exacta* se implementa utilizando un algoritmo de optimización unidimensional (apartado 2.3). En algunos casos, el problema se puede transformar en uno de *búsqueda lineal limitada* [11], de forma que el parámetro α esté limitado en un intervalo, es decir:

$$\alpha^k = \min_{\alpha \in [0,s]} f(x^k + \alpha d^k) \tag{2.18}$$

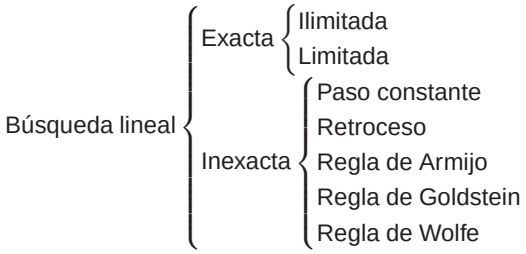


Figura 2.5: Clasificación de los métodos de búsqueda lineal

Dado que no es posible la implementación de esta búsqueda exacta en un número finito de pasos, conviene sacrificar precisión en favor de obtener un paso válido en unas pocas iteraciones, es decir, se realiza una *búsqueda*

lineal inexacta. Para asegurar la convergencia se requiere que la longitud del paso produzca un «descenso suficiente» en la función f, es decir:

$$f(x^k + \alpha d^k) < f(x^k) \tag{2.19}$$

Existen diversos criterios que aseguran esta condición, como se muestra en la figura 2.5. De todos ellos, el método más empleado por su facilidad de implementación y garantía de convergencia es la *regla de Armijo*. Ésta consiste en establecer una regla para la que se asegura la condición de descenso. Para ello, dado unos escalares $s, \beta \in (0,1)$ y $\sigma \in (0,1)$, se establece la longitud de paso en una iteración como $\alpha^k = \beta^m s$, siendo m el primer entero no negativo para el que se cumple la relación:

$$f(x^k) - f(x^k + \beta^m s d^k) \geq -\sigma \beta^m s \nabla f(x^k)^T d^k \tag{2.20}$$

El algoritmo 2.5 muestra una implementación de esta regla. Normalmente, se toma como parámetros $\sigma \in [10^{-5}, 10^{-1}]$ y $\beta \in [\frac{1}{2}, \frac{1}{10}]$ dependiendo de la confianza del valor inicial s [11]. El paso inicial s suele tener valor 1.

Algoritmo 2.5 Regla de Armijo

Regla-armijo : $s \times \beta \times \sigma \times x^k \times d^k \longrightarrow \alpha$
$m \longleftarrow 0$
$\alpha \longleftarrow \beta^m s$
Mientras $(f(x^k) - f(x^k + \alpha d^k) < -\sigma \alpha \nabla f(x^k)^T d^k)$
    $m \longleftarrow m + 1$
    $\alpha \longleftarrow \beta^m s$
$\alpha \longleftarrow \beta^{m-1} s$
Devolver α

2.4.1.2. Los métodos del gradiente

Los métodos del gradiente se pueden especificar de la forma:

$$x^{k+1} = x^k - \alpha^k D^k \nabla f(x^k) \tag{2.21}$$

donde D^k es una matriz simétrica definida positiva y la dirección de descenso es $d^k = -D^k \nabla f(x^k)$. Los diferentes métodos del gradiente se diferencian en la elección de la matriz D^k.

El método de máximo descenso

Es el método más simple y el de menor convergencia. En este método, la matriz D^k se toma como la matriz identidad, es decir:

$$D^k = I, \qquad \forall k \geq 0 \tag{2.22}$$

La lenta convergencia del método (véase [8] para un análisis más exhaustivo) viene motivada en el efecto zigzagueante de las direcciones generadas, al ser éstas siempre ortogonales a la dirección que conduce al mínimo. La figura 2.6 muestra el rastro de las direcciones generadas en el proceso de optimización empleando el método de máximo descenso partiendo desde el punto inicial x^0.

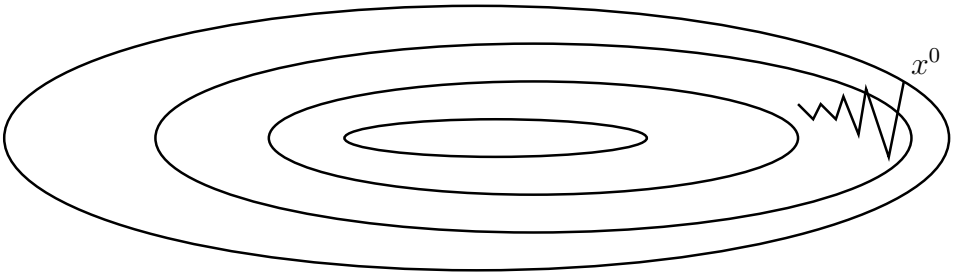


Figura 2.6: Efecto zigzagueante en el método de descenso

El método de Newton

La idea que subyace en este método es la de minimizar en cada iteración la aproximación cuadrática de la función f alrededor del punto actual x^k, en lugar de una aproximación lineal como en el máximo descenso. En este caso, la matriz D^k viene dada por el inverso del Hessiano:

$$D^k = (\nabla^2 f(x^k))^{-1}, \qquad \forall k \geq 0 \tag{2.23}$$

donde $\nabla^2 f(x^*)$ es definida positiva en el mínimo x^*.

Este método presenta una convergencia rápida al óptimo (véase [8]), y no presenta el efecto zigzagueante del máximo descenso, por lo que muchos métodos intentan emularlo. Sin embargo, presenta numerosos inconvenientes:

- El inverso del Hessiano puede no existir.
- La dirección puede que no asegure la condición de descenso, es decir:

$$f(x^{k+1}) > f(x^k)$$

- El método suele generar direcciones que tiende tanto a un máximo como a un mínimo local.

Por estas razones es necesario realizar una modificación del método. Una de las más sencillas consiste en aplicar una técnica de salvaguarda [11] de forma que cuando la dirección de Newton no esté disponible o no sea de descenso, se utiliza la dirección de máximo descenso.

2.4.1.3. Los métodos cuasi-Newton

Los métodos cuasi-Newton siguen el esquema (2.21) de los métodos del gradiente. La diferencia es que presentan una rápida convergencia, similar a los métodos de Newton, pero no necesitan realizar la segunda derivada de la función. La idea principal de estos métodos es que en dos iteraciones consecutivas, x^k y x^{k+1}, junto con los valores correspondientes del gradiente, $\nabla f(x^k)$ y $\nabla f(x^{k+1})$, es posible obtener una aproximación al Hessiano, de forma que:

$$\gamma^k \approx \nabla^2 f(x^{k+1})\delta^k \tag{2.24}$$

donde:

$$\begin{aligned} \delta^k &= x^{k+1} - x^k \\ \gamma^k &= \nabla f(x^{k+1}) - \nabla f(x^k) \end{aligned} \tag{2.25}$$

En base a las relaciones (2.25), los métodos cuasi-Newton escogen la matriz D^{k+1} a partir de la matriz D^k, así como de los vectores δ^k y γ^k, con objeto de satisfacer la llamada *ecuación de la secante* o *condición cuasi-Newton*:

$$\gamma^k = D^{k+1}\delta^k \tag{2.26}$$

Los diferentes métodos cuasi-Newton se diferencian en el modo de obtener dicha matriz D^{k+1}. A continuación veremos los más significativos.

Métodos de corrección simétrica de rango 1

Una actualización de la matriz D^{k+1} que cumple la ecuación de la secante (2.26) es la fórmula simétrica de rango 1 (SR1), sugerida independientemente por diversos autores como Broyden (1967), Davidon (1968), Fiacco y McCormick (1968), Murtagh y Sargent (1969) y Wolfe (1968) (véase [42]). Esta actualización es:

$$D^{k+1} = D^k + \frac{(\gamma^k - D^k\delta^k)(\gamma^k - D^k\delta^k)^T}{(\gamma^k - D^k\delta^k)^T\delta^k} \tag{2.27}$$

Métodos de métrica variable

Otra actualización consiste en realizarla con una matriz simétrica de rango 2 compuesta de $\nabla f(x^{k+1})$ y $\nabla f(x^k)$. En este caso:

$$D^{k+1} = D^k + \frac{\delta^k \delta^{k^T}}{\delta^{k^T}\gamma^k} - \frac{D^k\gamma^k\gamma^{k^T}D^k}{\gamma^{k^T}D^k\gamma^k} + \xi^k\tau^k v^k v^{k^T}, \qquad 0 \leq \xi^k \leq 1, \forall k \tag{2.28}$$

donde D^0 es una matriz definida positiva arbitraria (generalmente $D^0 = I$) y:

$$v^k = \frac{\delta^k}{{\delta^k}^T \gamma^k} - \frac{D^k \gamma^k}{\tau^k}$$
$$\tau^k = {\gamma^k}^T D^k \gamma^k$$

Los escalares ξ^k son los que parametrizan a los diferentes métodos. Por tanto, (2.28) representa a un conjunto de métodos, que se conoce como la *familia Broyden*. Dentro de ésta, se encuentran:

Método DFP Esta fórmula fue sugerida como parte de un método de Davidon (1959) y presentada por Fletcher y Powell (1963), de ahí su nombre. Surge haciendo $\xi^k = 1$, de forma que:

$$D^{k+1} = D^k + \frac{\delta^k {\delta^k}^T}{{\delta^k}^T \gamma^k} - \frac{D^k \gamma^k {\gamma^k}^T D^k}{{\gamma^k}^T D^k \gamma^k} + \tau^k v^k {v^k}^T \quad (2.29)$$

Método BFGS Se debe a Broyden (1970), Fletcher (1970), Goldfarb (1970) y Shanno (1970), de ahí su nombre. En este caso, $\xi^k = 0$, por lo que:

$$D^{k+1} = D^k + \frac{\delta^k {\delta^k}^T}{{\delta^k}^T \gamma^k} - \frac{D^k \gamma^k {\gamma^k}^T D^k}{{\gamma^k}^T D^k \gamma^k} \quad (2.30)$$

Si en cada iteración escogemos un ξ^k distinto, nos encontramos frente a un *método Broyden*, que se puede definir como un método cuasi-Newton, donde en cada iteración emplea un miembro de la familia de Broyden.

2.4.2. Métodos de descenso coordenado

Se fundamentan en la realización de un proceso iterativo similar al de los métodos de descenso, que se representaban por la ecuación (2.15). Sin embargo, en este caso, las direcciones de búsquedas que se van a emplear son los ejes coordenados. De este modo, el método busca a lo largo de las direcciones $d_1, d_2, \ldots d_N$, donde d_i es un vector de ceros salvo en la posición i-ésima que posee un 1. De este modo, el proceso de optimización en una iteración se puede representar como:

$$\min_{x_i} f(x_1, x_2, \ldots, x_N) \quad (2.31)$$

Así, en la dirección d_i sólo se permiten cambios en la variable x_i, permaneciendo las demás inalteradas. En [8] se muestra la convergencia de estos métodos.

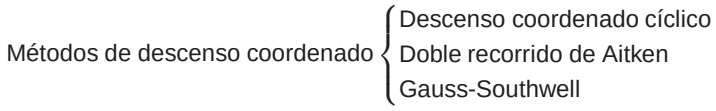


Figura 2.7: Clasificación de los métodos de descenso

La forma de seleccionar las distintas direcciones da lugar a los distintos métodos. La figura 2.7 muestra una clasificación de éstos.

El método de *descenso coordenado cíclico* realiza una minimización de la función f cíclicamente con respecto a las variables coordenadas. Así, se cambia en primer lugar x_1, luego x_2, y así sucesivamente hasta x_N. Entonces se repite el proceso comenzando de nuevo con x_1. El algoritmo 2.6 muestra una posible implementación de éste.

El método de *doble recorrido de Aitken* consiste en buscar en el orden $x_1, x_2, \ldots, x_N$, y después regresar en el orden $x_N, x_{N-1}, \ldots x_1$. Este método junto con el anterior se caracterizan por no necesitar de información sobre la derivada de la función.

Finalmente, si se dispone del gradiente de la función f, entonces podemos realizar una selección de las direcciones a partir de éste. El método *Gauss-Southwell* selecciona en cada etapa aquella coordenada que posea el mayor valor absoluto en el respectivo componente del vector gradiente.

Una ventaja importante de los métodos de descenso coordenado es su disponibilidad para la realización de una computación paralela. Supongamos que existen una serie de coordenadas $x_{i_1}, x_{i_2}, \ldots x_{i_m}$, de forma que la función $f(x)$ puede ser reescrita como $\sum_{r=1}^{m} f_{i_r}(x)$, donde $f_{i_r}(x)$ no depende de las coordenadas x_{i_s}, para todo $s \neq r$. En este caso es posible realizar las iteraciones de descenso coordenado en paralelo de forma independiente [11].

Algoritmo 2.6 Descenso coordenado cíclico

Coordenado-Cíclico : $x \times \epsilon \longrightarrow x^*$
$k \longleftarrow 1$
$x^k \longleftarrow x$
$j \longleftarrow 1$
$(d_1, d_2, \ldots d_N) \longleftarrow$ direcciones_coordenadas()
Mientras $(j \leq N)$
  $\alpha \longleftarrow \text{mín}\left\{f(x^k + \alpha d_j)\right\}$
  $x_j^{k+1} \longleftarrow x_j^k + \alpha d_j$
  $j \longleftarrow j + 1$
Mientras $\left(\frac{\|x^{k+1} - x^k\|}{\|x^{k+1}\|} \leq \epsilon\right)$
  $j \longleftarrow 1$
  $k \longleftarrow k + 1$
  Mientras $(j \leq N)$
    $\alpha \longleftarrow \text{mín}\left\{f(x^k + \alpha d_j)\right\}$
    $x_j^{k+1} \longleftarrow x_j^k + \alpha d_j$
    $j \longleftarrow j + 1$
Devolver x^k

2.4.3. Métodos de resolución global

La figura 2.3 muestra también una clasificación de los métodos de resolución global [93], en función de si incorporan algún elemento estocástico o no. Los métodos *deterministas* realizan una serie de suposiciones en la función a minimizar, por lo que están diseñados para problemas especiales con ciertas estructuras matemáticas. Por otro lado, los métodos *estocásticos* se caracterizan por evaluar la función objetivo en una serie de puntos aleatorios, y manipular dicho muestreo. Estas técnicas son más robustas que las anteriores, y permiten adaptarse a cualquier tipo de problema. Una diferencia entre los dos métodos es el resultado que proporcionan; mientras que los deterministas obtienen un conjunto de regiones donde se encuentran todos los óptimos globales, los estocásticos encuentran un óptimo global el cuál puede variar en el caso de que la función a optimizar disponga de varios.

Dentro de los métodos estocásticos podemos encontrarnos con los métodos de *búsqueda en dos fases* y los de *búsqueda decremental global*. Los primeros se caracterizan por generar inicialmente un conjunto de soluciones válidas y, posteriormente, a cada uno de ellos le aplica una técnica están-

dar de programación no lineal, con objeto de detectar los distintos óptimos locales. El mejor que se encuentra es la solución tomada como el óptimo global. Estos métodos pueden conducir a una pérdida de información elevada y, por consiguiente, la reducción de la probabilidad de convergencia.

Por otro lado, los métodos de búsqueda decremental global generan una secuencia de soluciones siguiendo una distribución probabilística previamente especificada. Estos métodos permiten mejorar la función objetivo de forma eficiente, evitando ser atrapado en un óptimo local y alcanzando el óptimo global. Durante las dos últimas décadas han aparecido nuevos métodos heurísticos, entre los cuales destacan el *enfriamiento lento simulado* [100], más conocido en la literatura como *simulated annealing*, los AGs y más recientemente la optimización con enjambre de partículas, han alcanzado una gran popularidad entre la comunidad científica por su gran flexibilidad y capacidad para acometer la resolución de problemas complejos de naturaleza muy diversa.

Dado que la tesis desarrolla una serie de estrategias de optimización que se encuadran dentro de los métodos de resolución global, haciendo uso de AGs, éstos serán objeto de estudio detallado en el siguiente capítulo.

2.5. Optimización combinatoria

La *optimización combinatoria* se puede definir como la rama de la optimización encargada de tratar con los problemas en los que las variables de decisión son discretas. En este tipo de problemas, encontrar el óptimo consiste en buscar una solución entre un conjunto finito o infinito de alternativas o soluciones factibles, que recibe el nombre de *solución global* [1]. El conjunto de soluciones del problema viene dado por un conjunto de variables de decisión, cuyos valores tienen ciertos rangos, y una solución se representa por el valor asignado a dichas variables.

De forma matemática, un problema de optimización combinatoria se puede definir como [128]:

$$\text{mín } f(T), \quad T \in F \tag{2.32}$$

donde F es el conjunto de soluciones posibles para el problema, y la función $f(T)$ nos mide la calidad de la solución aportada, de forma que:

$$f : T \longrightarrow \mathcal{R}$$

Dentro de esa definición también se incluyen los problemas de maximización de funciones, puesto que:

$$\text{máx } f(T) = -\text{mín}(-f(T)) \tag{2.33}$$

Aunque la definición de un problema combinatorio es sencilla, puede ser muy compleja, debido a que muchos de los problemas que nos encontramos en este tipo de optimización son *NP-duros*[2]. Éstos son los que pueden ser resueltos por un algoritmo no determinista en tiempo polinómico, por lo que ha tenido bastante éxito el desarrollo de algoritmos que buscan aproximaciones al óptimo con un tiempo razonable. Se distinguen dos tipos [1]:

Métodos constructivos Se caracterizan porque van construyendo la solución completa en una serie de pasos. Para ello, asignan los valores a las distintas variables de decisión una a una, tomándose en cada paso la mejor decisión posible. Las técnicas más conocidas que podemos encontrar en estos métodos son los *algoritmos voraces* [103].

Métodos de búsqueda local Éstos, a diferencia de los anteriores, toman como punto de partida una solución completa al problema, y realizan un análisis de las soluciones más cercanas a la misma. Si encuentra una mejor solución, ésta se toma como solución, y se realiza un estudio ahora de sus soluciones más próximas. Este proceso se repite hasta que no se consiga mejorar la solución encontrada.

En esta tesis se desarrollan dos estrategias que se encuadran dentro de los métodos de búsqueda local, donde la exploración de las soluciones cercanas se realiza con un AG. Por este motivo, a continuación pasamos a detallar las características de los métodos de búsqueda local.

2.5.1. Métodos de búsqueda local

Un proceso de búsqueda local puede ser visto como el procedimiento de minimizar la función de coste f en un número de pasos sucesivos, en los cuales la solución actual i se reemplaza por la solución j, tal que:

$$f(j) < f(i), j \in N(i) \tag{2.34}$$

La resolución de un problema de optimización mediante una búsqueda local implica la definición de la *vecindad*. Una vecindad se define como una función de correspondencia entre soluciones, de tal modo que:

$$N : F \longrightarrow 2^F \tag{2.35}$$

[2]En [49] podemos encontrar una descripción completa sobre la teoría de complejidad. Los aspectos introductorios se pueden consultar en el capítulo 2 de [128] y el capítulo 5 de [159], donde se proporciona una visión amplia y profunda del mismo.

Esta función define para cada solución $i \in F$, un conjunto $N(i) \subset F$ de soluciones, denominada la vecindad de i, que son en cierto sentido *cercanos a* la solución i. El significado de la expresión *cercano a* i se entiende como que dicha solución puede ser alcanzada desde i en un único *movimiento*. El término de único movimiento puede ser considerado como una vecindad de primer orden, mientras que si realizamos más movimientos consecutivos nos permite extender la vecindad, de modo que el orden de la vecindad se corresponda al número de movimientos consecutivos. Un movimiento es un operador que transforma una solución en otra mediante pequeñas modificaciones [155]. La caracterización de los movimientos que se pueden definir dependen del problema a resolver.

Los métodos de búsquedas locales se caracterizan por generar una secuencia de óptimos locales respecto a la vecindad, de manera que se dirige el proceso de optimización hasta la consecución del óptimo global. Una solución i es un óptimo local respecto a la vecindad $N(i)$, si:

$$f(i) \leq f(x), \forall x \in N(i) \tag{2.36}$$

Existen diversos modos de realizar la búsqueda local en una vecindad [1]:

Búsqueda local de mejora continua donde se sustituye la solución actual con aquella solución que mejora la función a optimizar después de explorar la vecindad completa.

Búsqueda local de primera mejora que acepta la primera solución encontrada en la vecindad que mejore la solución actual.

Capítulo 3

Algoritmos genéticos

En este capítulo se realiza una introducción a los AGs, técnica que se tomará como punto de partida para desarrollar los algoritmos de optimización desarrollados en esta tesis. En los siguientes apartados se describen las características, operadores y un pseudocódigo del AG básico. Finalmente se detalla el teorema de esquemas que justifica la convergencia de los mismos.

3.1. Introducción

Los AGs, introducidos por Holland [65, 66] e impulsados en años sucesivos por Goldberg [55], uno de sus estudiantes, comprenden una de las cuatro áreas de la denominada Computación Evolutiva (*Evolutionary Computation*). Junto a los AGs, la programación evolutiva, las estrategias de evolución y la programación genética, conforman las cuatro áreas de la computación evolutiva. Cada una de ellas siguen diferentes metodologías estocásticas para imitar la evolución biológica a nivel computacional. Centrándonos en los AGs, éstos se definen como métodos estocásticos de optimización global basados en los principios de la selección y evolución natural [55].

De acuerdo con la teoría de la evolución de Darwin, los AGs mimetizan la evolución generacional de las especies, promoviendo la supervivencia de los mejores individuos. Básicamente, una población de individuos o soluciones potenciales al problema bajo análisis oportunamente codificadas, se hace evolucionar hacia una solución óptima en base a la presión que ejercen los operadores de selección, cruce y mutación, utilizando una función de aptitud, coste o *fitness* para medir la calidad de las soluciones y proceder iterativamente al reemplazo generacional.

Una definición que resume las características de los AGs es la siguiente [28]:

> Los AGs son métodos estocásticos de búsqueda ciega de soluciones cuasi-óptimas. En ellos se mantiene una población que representa a un conjunto de posibles soluciones la cual es sometida a ciertas transformaciones con las que se trata de obtener nuevos candidatos y a un proceso de selección sesgado en favor de los mejores candidatos.

Los conceptos que pueden resumir a los AGs son la *supervivencia* de los mejores y el *carácter hereditario* de las características. Esto significa que, por un lado, los AGs favorecen a los mejores individuos, y por otro lado, generan nuevos individuos a través de la recombinación y la mutación de la información de los individuos existentes. Los AGs no son aproximaciones secuenciales que consideran una única solución cada vez, sino que trabajan con una población de soluciones y procesan toda la información que ésta contiene de forma paralela.

En cuanto a las restantes características, cabe destacar que los AGs son métodos de búsqueda:

- **ciega**, al no disponer de ningún conocimiento específico del problema, de manera que la búsqueda se basa exclusivamente en los valores de la función objetivo;
- **codificada**, puesto que no trabajan directamente sobre el dominio del problema, sino sobre representaciones de sus elementos;
- **múltiple**, al buscar simultáneamente entre un conjunto de candidatos;
- **estocástica**, referida tanto a la fase de selección como a la de transformación de individuos.

Todas las características enumeradas se introducen de manera deliberada para proporcionar más robustez a la búsqueda, es decir, para darle más eficiencia sin perder la generalidad y viceversa [55].

A diferencia de otros métodos estocásticos más recientes como la optimización con enjambre de partículas [80], el desarrollo conceptual de los AGs tiene, desde sus orígenes, un respaldo matemático que lo avala, basado principalmente en el denominado *teorema del esquema* [66], el cual conjuga aptitud, cruce y mutación para establecer como afectan a la supervivencia y propagación de las soluciones aspectos tales como el esquema de representación de los datos.

Con las bases del método bien definidas, pueden intuirse una serie de ventajas de los AGs frente a los métodos clásicos de optimización local [55]. En primer lugar, el resultado de los métodos clásicos dependen del punto inicial, mientras que los AGs son altamente independientes de las condiciones iniciales.

En general, las técnicas globales de optimización se desenvuelven con eficacia en espacios multidimensionales, multimodales y con múltiples discontinuidades, donde los métodos locales se muestran ineficaces [60]. En este sentido y ante problemas en los que la naturaleza del espacio de soluciones se desconoce, los métodos globales son particularmente útiles. Los AGs son capaces de llegar a una solución global o próxima a ésta, en lugar de converger hacia soluciones locales como puede ocurrir con los métodos convencionales.

Otros aspectos, tales como el hecho de realizar la búsqueda utilizando un conjunto de puntos sobre el espacio de soluciones en lugar de un único punto, el utilizar una función de aptitud para dirigir la búsqueda en lugar de derivadas, o que los nuevos puntos a explorar se determinen de acuerdo a reglas de decisión estocásticas en lugar de deterministas, reafirman la superioridad de los AGs sobre las técnicas de optimización local. Como contrapartida, la naturaleza estocástica inherente a los AGs, con una exploración mucho más exhaustiva del espacio de soluciones, depara una convergencia mucho más lenta que la de los métodos locales.

Durante la última década, el uso de los AGs se ha extendido a múltiples campos de la ciencia, haciendo que el radio de acción de los AGs sea muy amplio. Así, destacamos a modo de ejemplo, la aplicación al entrenamiento de redes neuronales [129], la optimización de los nodos de red en comunicaciones móviles [78], la detección multiusuario en sistemas CDMA [4], en microelectrónica aplicados a la síntesis de circuitos integrados VLSI [35], la planificación de redes de distribución de energía eléctrica [20]. Dentro del campo de la electromagnética podemos resaltar, entre otras aplicaciones, el diseño de antenas de hilo, diseño de arrays, aplicaciones en radar, diseño de antenas impresas, filtros de microondas y diseño de absorbentes, con una extensa bibliografía al respecto, recopilada de forma magistral por Rahmat-Samii y otros en [77] y [132]. En [13] se empiezan a usar dentro del campo de pruebas del software, recopilándose en [94] las distintas aplicaciones que han tenido desde entonces en este campo de la Ingeniería del Software.

El éxito de los AGs radica, más que en el exotismo o en la difusión que de éste se ha hecho para ganar nuevos adeptos, en su propio potencial y facilidad de implementación. En [28] se recogen los principales motivos del éxito de los AGs:

- Las ideas fundamentales del enfoque evolutivo están recogidas de una manera natural en dicha técnica.

- Son flexibles y adaptables a una gran cantidad de problemas diferentes pertenecientes a distintas áreas y permiten ser combinados con otras técnicas no necesariamente pertenecientes a la computación evolutiva (hibridación).

- Dentro de las técnicas de computación evolutiva los AGs son los que poseen una mayor base teórica.
- Poseen una gran versatilidad pues son los que necesitan menos conocimiento específico del problema para su resolución.
- Es posible implantarlos en ordenadores con capacidades medias obteniendo resultados muy aceptables.

En definitiva, la naturaleza de los AGs y su capacidad para manejar los problemas de la optimización hacen que las previsiones acerca de la perdurabilidad del método a largo plazo vertidas por Goldberg [54], sean hoy una realidad. De hecho se han convertido en la técnica metaheurística, con diferencia, más empleada [79], existiendo multitud de estudios previos referidos a su empleo, parametrización e implantación, así como herramientas informáticas genéricas adaptables a diferentes problemas [57, 142, 157].

3.2. AG básico

El esquema general de un AG aparece en el algoritmo 3.1 [105].

Algoritmo 3.1 Algoritmo Genético Básico

Algoritmo-Genético : $T_P \times p_c \times p_m \times G \times N \longrightarrow \bar{x}$
$t \longleftarrow 0$
$P_t \longleftarrow$ Crear_Poblacion_Inicial(T_P)
Evaluar_Aptitud()
Mientras $(t < G)$
  $t \longleftarrow t + 1$
  Mientras $(|P_t| < N)$
    $Progenitores \longleftarrow$ Seleccion_Individuos_Reproduccion(P_{t-1})
    $Descendientes \longleftarrow$ Realizar_Cruce($p_c, Progenitores$)
    $Descendientes\_mutados \longleftarrow$ Realizar_Mutacion($p_m, Descendientes$)
    Insertar_Descendientes($P_t, Descendientes\_mutados$)
  Mientras $(|P_t| < T_P)$
    Superviviente $\longleftarrow$ Seleccionar_Individuo()
    Insertar_Individuo(P_t, Superviviente)
  Evaluar_Aptitud()
$ind \longleftarrow$ Mejor_Individuo()
Devolver ind

Como puede verse, una población P_t, que consta de T_P miembros se somete a un proceso de *selección* para determinar los *progenitores* que son

los que efectivamente se van a reproducir. Sirviéndose de los *operadores genéticos*, los progenitores son sometidos a ciertas transformaciones de *cruce* y *mutación* en la fase de *reproducción* en virtud de las cuales se generan N nuevos individuos que constituyen la $Descendencia$. Para formar la nueva población P_{t+1}, se completa con una selección de *supervivientes* de la población anterior. De este modo, el principio de selección natural se realiza en sus dos vertientes: selección de progenitores (*selección*) y selección de supervivientes para la próxima generación (*sustitución*). Esta doble selección de soluciones en progenitores y supervivientes es la principal innovación de los AGs en el dominio de los métodos de búsqueda.

3.3. Elementos del AG

Para implementar un AG es necesario definir los siguientes elementos:

Codificación Se debe especificar el procedimiento con el que se hace corresponder cada punto del dominio del problema con cada individuo, es decir, cómo representamos una solución dentro del AG.

Función de aptitud Se debe concretar la función de aptitud que utilizará el AG para medir la calidad de los individuos generados.

Inicialización Se refiere a cómo se debe construir la población inicial con la que se arranca el bucle básico del AG.

Selección La selección de los progenitores debe dirigir el proceso de búsqueda en favor de los miembros más aptos, pero existen muchas maneras de llevar esto a cabo.

Operadores genéticos Se llaman así a los operadores con los que se lleva a cabo la reproducción. Todo AG hace uso de al menos dos operadores genéticos, el cruce y la mutación; no obstante ellos no son los únicos posibles y además admiten variaciones.

Sustitución Los criterios con que se seleccionan los progenitores no necesariamente han de ser los mismos con los que se seleccionan los supervivientes, de ahí la necesidad de especificarlos por separado.

Parada Se deben concretar las condiciones con las que se considera que el AG ha dado con una solución aceptable o, en su defecto, ha fracasado en la búsqueda y no tiene sentido continuar.

Otros parámetros de funcionamiento Un AG necesita que se le proporcionen ciertos parámetros de funcionamiento, tales como el tamaño de la población, las probabilidades de aplicación de los operadores genéticos, la precisión de la codificación, las tolerancias de la convergencia, etc.

3.3.1. Codificación

Cada individuo o posible solución del problema se puede representar como un conjunto de parámetros, que denominaremos *genes*, los cuales agrupados forman una lista de valores, a menudo referida como *cromosoma*.

En términos biológicos, el conjunto de parámetros que representa un cromosoma particular se denomina *fenotipo*. El fenotipo contiene la información requerida para construir un organismo, el cual se refiere como *genotipo*. Los mismos términos se utilizan en el campo de los AGs. La adaptación al problema de un individuo depende de la evaluación del genotipo. Esta última puede inferirse a partir del fenotipo, es decir puede ser computada a partir del cromosoma, usando la función de evaluación.

El esquema de codificación es altamente dependiente de las características del problema que se trate. Según sea el esquema empleado podemos encontrarnos con los siguientes tipos de algoritmos:

Binarios Los individuos codifican la información mediante una representación binaria. Se suele emplear en problemas que tienen variables de decisión.

Punto flotante La información se representa mediante números reales. Es utilizado en problemas de variables continuas [160].

Permutación En éstos, la información de los individuos se representa mediante una secuencia ordenada de valores. Se emplea en problemas de secuenciación y similares [44].

No obstante, las características del problema a resolver puede necesitar emplear otros esquemas diferentes a los anteriores, más adaptados a la naturaleza del problema.

3.3.2. Función de aptitud

La función de aptitud debe ser diseñada para cada problema de manera específica. Dado un cromosoma particular, la función de aptitud le asigna un número real, que se supone refleja el nivel de adaptación al problema del individuo representado por el cromosoma.

Idealmente nos interesaría construir funciones de aptitud con *ciertas regularidades*, es decir, funciones objetivo que verifiquen que para dos individuos que se encuentren cercanos en el espacio de búsqueda, sus respectivos valores en la función de aptitud sean similares.

La regla general para construir una buena función de aptitud es que ésta debe reflejar el valor del individuo de una manera real, pero en muchos problemas de optimización combinatoria, donde existen gran cantidad de restricciones, buena parte de los puntos del espacio de búsqueda representan individuos no válidos. En este caso, se han propuesto varias soluciones.

La primera sería la que podríamos denominar *absolutista*, en la que aquellos individuos que no verifican las restricciones, no son considerados como tales, y se siguen efectuando cruces y mutaciones hasta obtener individuos válidos, o bien a dichos individuos se les asigna una función de aptitud igual a cero.

Otra posibilidad consiste en reconstruir aquellos individuos que no verifican las restricciones. Dicha reconstrucción suele llevarse a cabo por medio de un nuevo operador que se denomina *reparador*.

Otro enfoque está basado en la penalización de la función de aptitud. La idea general consiste en dividir la función de aptitud del individuo por una cantidad (la penalización) que guarda relación con las restricciones que dicho individuo viola. Dicha cantidad puede simplemente tener en cuenta el número de restricciones violadas o bien el coste esperado de reconstrucción, es decir, el coste asociado a la conversión de dicho individuo en otro que no viole ninguna restricción.

3.3.3. Población inicial

Habitualmente la población inicial se escoge generando individuos al azar, pudiendo contener cada gen uno de los posibles valores del alfabeto con probabilidad uniforme.

3.3.4. Selección de individuos

Durante la fase reproductiva se seleccionan los individuos de la población para cruzarse y producir descendientes, que constituirán la siguiente generación de individuos. No existe un único método de selección de individuos, ni una única clasificación. Una posible clasificación de procedimientos de selección es la siguiente:

Métodos de selección dinámicos en los cuales las probabilidades de selección varían de generación en generación.

Métodos de selección estáticos en los cuales dichas probabilidades permanecen constantes.

Si se asegura que todos los individuos tienen asignada una probabilidad de selección distinta de cero el método de selección se denomina *preservativo*. En caso contrario se acostumbra a denominarlo *extintivo*.

También se pueden clasificar según el grado de intervención del azar en el proceso:

Selección directa Se toma un subconjunto de individuos de la población siguiendo un criterio fijo, del estilo de *los k mejores*, *los k peores*, etc.

Selección aleatoria simple o equiprobable Se asignan a todos los elementos de la población base las mismas probabilidades de formar parte de la muestra, realizándose una selección aleatoria.

Selección estocástica Se asignan probabilidades de selección a los elementos de la población base en función de su aptitud. Según dicho esquema, los individuos bien adaptados se escogerán probablemente varias veces por generación, mientras que los pobremente adaptados al problema, no se escogerán más que de vez en cuando. Existen muchos mecanismos de muestreo estocástico, a continuación detallamos los más habituales.

3.3.4.1. Métodos de selección estocástica

La función de selección de padres más utilizada es la *función de selección proporcional* a la función de aptitud o *selección por ruleta* [55], en la cual cada individuo tiene una probabilidad de ser seleccionado como padre que es proporcional al valor de su función aptitud. Este método se engloba dentro de los métodos de selección dinámicos.

Denotando por p_j^{prop} la probabilidad de que el individuo I_j sea seleccionado como padre, se tiene que:

$$p_j^{prop} = \frac{g(I_j)}{\sum_{j=1}^{TamPob} g(I_j)} \tag{3.1}$$

donde $g(I_j)$ es el valor de la función de aptitud para el individuo I_j.

Para seleccionar un individuo, una vez determinada las probabilidades individuales de cada individuo se realizan los siguientes pasos:

1. Se calculan las *puntuaciones acumuladas* así:

$$\begin{aligned} q_0 &:= 0 \\ q_i &:= p_1^{prop} + \cdots + p_i^{prop} \qquad (\forall i = 1, \ldots, n) \end{aligned}$$

2. Se genera un número aleatorio simple $r \in [0, 1)$

3. Se elige el individuo I_i que verifique $q_{i-1} < r < q_i$

Uno de los problemas que presenta esta selección es la rápida convergencia proveniente de los *superindividuos*, es decir, individuos que poseen una aptitud muy superior al resto de la población. Para solucionarlo se puede utilizar la *selección proporcional al rango del individuo*, con lo cual se produce una repartición más uniforme de la probabilidad de selección. Si denotamos por $rango(g(I_j))$ el rango de la función aptitud del individuo I_j

cuando los individuos de la población han sido ordenados de menor a mayor (es decir, el peor individuo tiene rango 1, mientras que el individuo con mejor función objetivo tiene rango $TamPob$), y sea p_j^{rango} la probabilidad de que el individuo I_j sea seleccionado como padre cuando la selección se efectúa proporcionalmente al rango del individuo, se tiene que

$$p_j^{rango} = \frac{rango(g(I_j))}{TamPob(TamPob+1)/2} \tag{3.2}$$

En el *modelo de selección elitista* se fuerza a que el mejor individuo de la población, sea seleccionado como padre. Es un ejemplo de método estático.

La *selección por torneo*, constituye un procedimiento de selección de padres muy extendido. Consiste en escoger al azar un número de individuos de la población, el *tamaño del torneo*, seleccionar el mejor individuo de este grupo, y repetir el proceso hasta que el número de individuos seleccionados coincida con el tamaño de la población. Habitualmente el tamaño del torneo es 2.

3.3.5. Operadores genéticos

En la fase de reproducción, una vez seleccionados los progenitores, sus cromosomas se combinan, utilizando los operadores de cruce y mutación, con objeto de formar la nueva población.

3.3.5.1. El operador de cruce

Este operador genera dos individuos nuevos, denominados *hijos*, a partir de dos individuos seleccionados previamente, denominados *padres*. El operador de cruce tiene como objetivo que los hijos hereden parte de la información almacenada en cada uno de los dos padres. Habitualmente el operador de cruce no se aplica a todos los pares de individuos que han sido seleccionados para emparejarse, sino que se aplica de manera aleatoria, normalmente con una probabilidad comprendida entre 0.5 y 1.0, denominada *probabilidad de cruce*. En el caso en que el operador de cruce no se aplique, la descendencia se obtiene simplemente duplicando los padres.

El diseño de este operador está asociado al esquema de codificación empleado en el AG. Uno de los operadores de cruce más utilizados es el cruce de un punto. Éste consiste en cortar las listas de cromosomas de los dos padres en una posición escogida al azar, para producir dos sublistas iniciales y dos sublistas finales. Después se intercambian las sublistas finales, produciéndose dos nuevos cromosomas completos (véase la figura 3.1).

Se han investigado otros operadores de cruce, habitualmente teniendo en cuenta más de un punto de cruce. De Jong [24] investigó el comportamiento del operador de cruce basado en múltiples puntos, concluyendo que el cruce basado en dos puntos, representaba una mejora mientras que

añadir más puntos de cruce no beneficiaba el comportamiento del algoritmo.

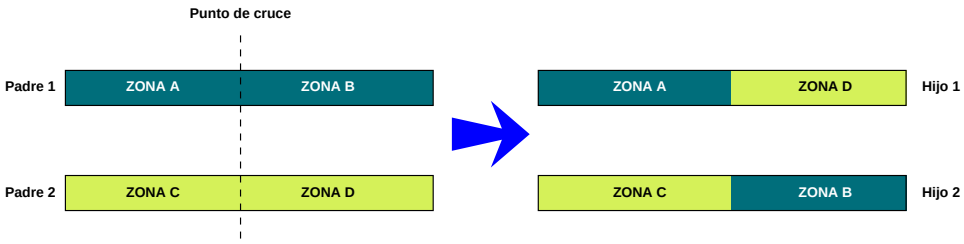


Figura 3.1: Operador de cruce basado en un punto

Existen en la bibliografía numerosas alternativas para implementar el operador de cruce cuando se emplea codificación de variables reales, como el cruce discreto, el cruce continuo, el cruce convexo o aritmético y el cruce heurístico [36, 61, 69].

Para codificaciones en permutación disponemos de cruces como el cruce basado en una correspondencia parcial (PMX) [56], el cruce basado en ciclos (CX) [122], cruce basado en el orden (OX1 y OX2) [23, 145], cruce basado en la posición (POS) [145], cruce basado en la combinación de arcos (ER) [158], etc.

3.3.5.2. El operador de mutación

La mutación se considera un operador básico, que proporciona un pequeño elemento de aleatoriedad en la vecindad (entorno) de los individuos de la población. Si bien se admite que el operador de cruce es el responsable de efectuar la búsqueda a lo largo del espacio de posibles soluciones, también parece desprenderse de los experimentos efectuados por varios investigadores que el operador de mutación va ganando en importancia a medida que la población de individuos va convergiendo [22].

El operador de mutación se aplica a cada hijo de manera individual, y consiste en la alteración aleatoria (normalmente con probabilidad pequeña) de cada gen componente del cromosoma. La figura 3.2 muestra la mutación del quinto gen del cromosoma para un individuo con codificación binaria

Descendiente	1	0	1	0	**1**	1	0	1	0	0
Descendiente mutado	1	0	1	0	**0**	1	0	1	0	0

Figura 3.2: Mutación binaria

Al igual que el operador de cruce, el diseño de este operador también está asociado al esquema de codificación empleado en el AG. En este sentido, el operador más empleado consiste en cambiar el valor de un gen por otro del rango de valores permitidos para ese gen.

Si bien puede en principio pensarse que el operador de cruce es más importante que el operador de mutación, ya que proporciona una exploración rápida del espacio de búsqueda, este último asegura que ningún punto del espacio de búsqueda tenga probabilidad cero de ser examinado, y es de capital importancia para asegurar la convergencia de los AGs.

3.3.6. Criterios de parada

El criterio habitual empleado para la finalización de la ejecución de un AG es el número máximo de generaciones. Sin embargo, es posible combinarlo con el grado de convergencia del algoritmo. Si el AG ha sido correctamente implementado, la población evolucionará a lo largo de las generaciones sucesivas de tal manera que la adaptación media extendida a todos los individuos de la población, así como la adaptación del mejor individuo se irán incrementando hacia el óptimo global.

En este sentido, es muy útil la definición de convergencia introducida por De Jong en su tesis doctoral [24]. El concepto de convergencia está relacionado con la progresión hacia la uniformidad: un gen ha convergido cuando al menos el 95 % de los individuos de la población comparten el mismo valor para dicho gen. Se dice que la población converge cuando todos los genes han convergido. Se puede generalizar dicha definición al caso en que al menos un β % de los individuos de la población hayan convergido.

3.3.7. Sustitución de individuos

Una vez obtenidos los individuos resultantes de las operaciones de cruce y mutación, es necesario determinar la estrategia de selección para determinar qué individuos de la antigua población van a salir. Existen varias alternativas:

Sustitución inmediata Los descendientes sustituyen a sus progenitores.

Sustitución con factor de llenado Los descendientes sustituyen a aquellos miembros de la población que más se le parecen según el fenotipo.

Sustitución por inserción Los descendientes sustituyen a otros miembros de la población según un determinado criterio basado en la aptitud, normalmente se sustituyen a los peores individuos.

Sustitución por inclusión Se selecciona la población entre todos los descendientes y los progenitores.

3.3.8. Renovación de la población

Según el modo de renovación de la población en cada generación, los AGs pueden clasificarse en:

Generacional En cada generación se reemplaza la población entera con otra nueva resultante de la realización de operaciones de cruces y mutaciones con la población anterior.

Estado permanente También conocidos como *incrementales*, sustituyen en cada generación solo unos pocos (uno o dos) miembros de la población [146].

Paralelos La población tiene una estructura espacial, y tanto la reproducción como la selección se realiza de forma localmente distribuida. El AG sigue el denominado *modelo de islas* [143]. La idea básica consiste en dividir la población total en varias subpoblaciones en cada una de las cuales se lleva a cabo un AG. Cada cierto número de generaciones, se efectúa un intercambio de información entre las subpoblaciones, proceso que se denomina *emigración*. La introducción de la emigración hace que los modelos de islas sean capaces de explotar las diferencias entre las diversas subpoblaciones, obteniéndose de esta manera una fuente de diversidad genética. Cada subpoblación es una *isla*, definiéndose un procedimiento por medio del cual se mueve el material genético de una isla a otra. La determinación de la tasa de migración, es un asunto de capital importancia, ya que de ella puede depender la convergencia prematura de la búsqueda. Se pueden distinguir diferentes modelos de islas en función de la comunicación entre las subpoblaciones, como la comunicación en estrella, en red o en anillo. En [18] podemos encontrar una revisión bibliográfica sobre las técnicas más representativas de los AGs paralelos.

3.4. Clasificación de AGs

No existe un único tipo de AG, sino familias enteras que difieren principalmente en el esquema de codificación empleado y en el modo en que renuevan la población en cada generación. Según estos criterios, la figura 3.3 muestra una posible clasificación de los AGs más comúnmente utilizados.

Nótese que esta clasificación no es completa, puesto que el esquema es dependiente del problema a resolver. Así, por ejemplo, podemos tener algoritmos cuyos individuos son variables enteras, o bien matrices.

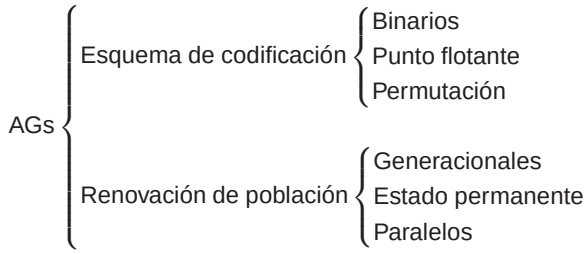


Figura 3.3: Clasificación de los AGs

3.5. Nichos

Además de los operadores genéticos de cruce y mutación, existen otros operadores más especializados que permiten resolver problemas que se presentan en los AGs. Estos operadores no se usan en todos los AGs, sino que deben determinarse, según la naturaleza del problema a resolver, si es necesario disponer de dichos operadores. Uno de ellos son los denominados *nichos*.

Los operadores de nichos [55] están dirigidos a mantener la diversidad genética de la población, de forma que cromosomas similares sustituyan sólo a cromosomas similares, y son especialmente útiles en problemas con muchas soluciones. Así, por ejemplo, si consideramos la figura 3.4, podemos ver que al ejecutar un AG básico, los individuos tienden a agruparse en la zona de la función con un mayor valor, sin conseguir una exploración de toda la región. En este caso, sólo obtendríamos una única solución al problema.

Los nichos tienen como objetivo el mantener la población lo más dispersa posible, con objeto de poder localizar los distintos óptimos locales, y con ello el óptimo global del problema.

Para ello, las aptitudes de los individuos se modifican, de manera que se penalizan a aquellos grupos de individuos que se encuentran cerca de la misma solución o son muy similares entre sí.

Para llevar a cabo esta técnica es necesario determinar el *tamaño del nicho*. Éste no es mas que la distancia mínima de separación de dos individuos para que no interfieran entre sí. En este sentido, es necesario especificar una *función de compartición*, que indica cómo de similar es un cromosoma respecto al resto de la población. De este modo, la aptitud de un individuo tendrá que tener en cuenta el valor de esta función de compartición, de forma que se facilita la diversidad genética y la aparición de individuos diferentes.

De este modo, si aplicamos la técnica de nichos a la misma función de la figura 3.4, podemos ver en la figura 3.5 que ahora los distintos indivi-

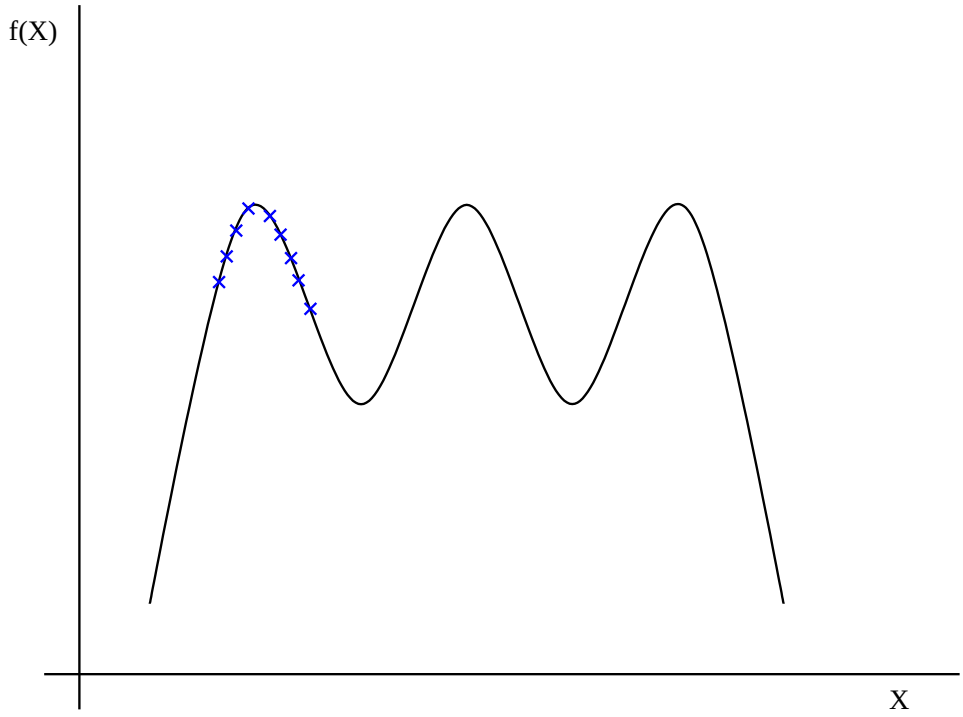


Figura 3.4: Ejecución de un AG básico sin nichos

duos de la población se reparte a lo largo de todo el espacio de búsqueda localizando los distintos óptimos que presenta la función.

3.6. Teorema de esquemas

Los apartados anteriores han mostrado el funcionamiento de los AGs. Sin embargo, éstos tienen un fundamento teórico que los diferencia del resto de técnicas heurísticas que son los *esquemas* [66].

3.6.1. Definiciones

Un *esquema* es un patrón de similitud que se construye introduciendo el signo "$*$" de *indiferencia* en el alfabeto de alelos. De este modo un esquema representa a todos los individuos que encajan en su representación.

Por ejemplo, el esquema `(*1*1100100)` representa a estas cuatro cadenas binarias:

```
0101100100    0111100100    1101100100    1111100100
```

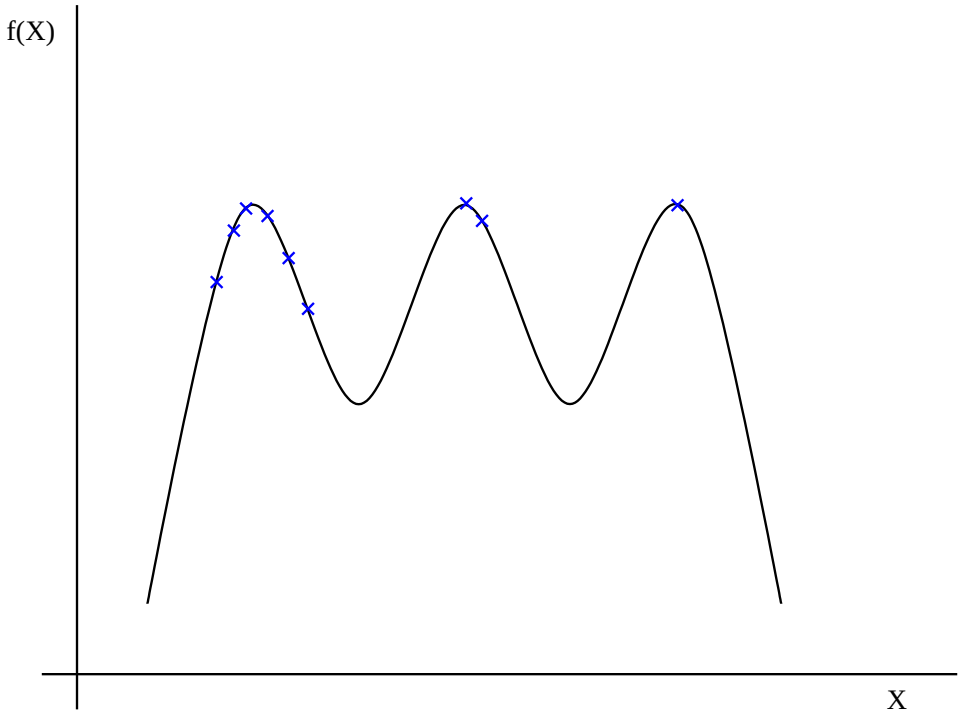


Figura 3.5: Ejecución de un AG básico con nichos

De este modo, el esquema `(1001110001)` sólo representa a una cadena y `(**********)` representa a todas las cadenas de longitud 10.

Es inmediato verificar las siguientes propiedades:

1. Si un esquema contiene k símbolos de indiferencia entonces representa a 2^k cadenas binarias.
2. Una cadena binaria de longitud L encaja en 2^L esquemas distintos.
3. Una población de $TamPop$ cadenas binarias de longitud L contiene entre 2^L y $TamPop \cdot 2^L$ esquemas distintos.

Dado un esquema S se definen:

- **Orden de un esquema**, $o(S)$, es el número de posiciones especificadas, es decir, con `0` ó `1`, que contiene dicho esquema.
- **Longitud característica de un esquema**, $\delta(S)$, es la distancia en dígitos entre las posiciones fijas extremas.

Así, por ejemplo, dado el esquema $S = (****00**1*)$, su orden es $o(S) = 3$ y tiene una longitud característica $\delta(S) = 9 - 5 = 4$.

Dado que un esquema S representa a $2^{L-o(S)}$ cadenas, cuanto mayor sea el orden del esquema a menos cadenas representará. Por eso se dice que el orden de un esquema da una medida de su especificidad. En el presente desarrollo el orden servirá para calcular la probabilidad de supervivencia de un esquema frente a las mutaciones. Por otra parte, la longitud característica da una medida de la compacidad de la información contenida en el esquema. Nótese que un esquema con una sola posición especificada tiene una longitud característica de cero. Aquí se usará ese concepto para calcular la probabilidad de supervivencia de un esquema frente a los cruces.

Dada una población de cadenas binarias $P[t]$ y un esquema S se definen también:

- **Presencia de S en P[t]**, $\xi(S, P[t])$, es el número de cadenas de $P[t]$ que encajan en el esquema S.
- **Aptitud del esquema S en P[t]**, $AptEsquema(S, P[t])$, es el promedio de las aptitudes de todas las cadenas de la población que encajan en el esquema S en el instante t. Es decir, si se numeran como $v_1, \ldots, v_p$ a las cadenas de $P[t]$ que encajan en S, siendo $p = \xi(S, P[t])$, resulta que:

$$AptEsquema(S, P[t]) = \frac{1}{p} \sum_{i=1}^{p} Aptitud(v_i)$$

- **Aptitud media de la población en el instante t**, $AptMedia(P[t])$, es el promedio de las aptitudes de todas las cadenas de la población en el instante t, o lo que es equivalente, la aptitud del esquema `(*...*)` en $P[t]$:

$$\begin{aligned} AptMedia(P[t]) &= AptEsquema((*...*), P[t]) \\ &= \frac{1}{TamPop} \sum_{v_i} \in P[t] Aptitud(v_i) \end{aligned}$$

- **Aptitud relativa de S en P[t]**, $AptRel(S, P[t])$, es el cociente entre la aptitud del esquema en la población y la aptitud media de la población en cierto instante, es decir,

$$AptRel(S, P[t]) = \frac{AptEsquema(S, P[t])}{AptMedia(P[t])}$$

3.6.2. La ecuación de crecimiento de los esquemas

La evolución de un esquema dentro de una población se puede demostrar mediante la denominada *ecuación de crecimiento reproductivo de un esquema en una población*.

Sea $P[t]$ una población de $TamPop$, lo suficientemente grande, con individuos de tamaño L, que evoluciona a través de un AG clásico. La presencia media de un esquema S en la población $P[t]$, representado por $\overline{\xi}(S, P[t+1])$, evoluciona según la siguiente fórmula:

$$\overline{\xi}(S, P[t+1]) = \overline{\xi}(S, P[t])\kappa_g\kappa_s \tag{3.3}$$

donde κ_g es el *factor de crecimiento* de S en la población y κ_s es el *factor de supervivencia*. El factor de crecimiento mide la tendencia del esquema a aumentar su presencia en la población intermedia y el factor de supervivencia mide la probabilidad de que pase a la siguiente generación. El elemento que hace aumentar la presencia de los esquemas en la población intermedia es la selección; de hecho es muy sencillo ver que, si se usa una selección por torneo, la presencia media de un esquema en la población intermedia $Q[t]$ evoluciona según:

$$\begin{aligned}\overline{\xi}(S, Q[t]) &= \overline{\xi}(S, P[t])AptRelativa(S, P[t]) \\ &= \overline{\xi}(S, P[t])\frac{AptEsquema(S, P[t])}{AptMedia(P[t])}\end{aligned}$$

de manera que:

$$\kappa_g = AptRelativa(S, P[t])$$

Ahora bien, existe la posibilidad de que los esquemas sean destruidos por los operadores genéticos antes de pasar a la próxima generación; es esta posibilidad la que cuantifica el factor de supervivencia κ_s. Tal factor es difícil de calcular exactamente, pero se puede acotar fácilmente mediante:

$$\kappa_s \geq \left(1 - \frac{p_c\ \delta(S)}{L-1}\right)(1-p_m)^{o(S)}$$

donde p_c y p_m representan las probabilidades de cruce y mutación, respectivamente.

El primer factor mide la probabilidad de que S sobreviva a un cruce, o más exactamente, la probabilidad de que el cruce no ocurra sobre su longitud característica. El segundo factor mide la probabilidad de que no se vea afectado por una mutación. Todo esto presupone que se está usando la sustitución inmediata.

En definitiva, siendo p_c y p_m las probabilidades de cruce y mutación, respectivamente, del AG simple, la presencia media de cierto esquema S en la población $P[t]$ evoluciona según:

$$\overline{\xi}(S, P[t+1]) \geq \overline{\xi}(S, t)AptRel(S, P[t])\left(1 - \frac{p_c\ \delta(S)}{L-1}\right)(1-p_m)^{o(S)}$$

Como habitualmente $p_m \ll 1$ se puede hacer

$$(1-p_m)^{o(S)} \simeq 1 - p_m\ o(S)$$

de modo que,

$$\kappa_s \gtrsim \left(1 - \frac{p_c\ \delta(S)}{L-1} - p_m\ o(S)\right)$$

lo que proporciona la siguiente formulación aproximada de (3.3):

$$\overline{\xi}(S, P[t+1]) \geq \overline{\xi}(S, P[t]) AptRelativa(S, P[t]) \left(1 - \frac{p_c\ \delta(S)}{L-1} - p_m\ o(S)\right)$$

3.6.3. El teorema fundamental de los AGs

Basándose en los resultados de la sección anterior cabe enunciar el *Teorema fundamental de los AGs*: en las sucesivas generaciones de un AG simple, la presencia de un esquema S en la población $P[t]$ evoluciona estadísticamente de modo exponencial según la ecuación (3.3).

De la ecuación (3.3) se extraen inmediatamente las siguientes consecuencias prácticas:

- La presencia de un esquema en una población evoluciona estadísticamente en progresión geométrica, progresión cuyo factor está determinado por la aptitud relativa del esquema, su longitud característica y su orden. Es fundamental darse cuenta de que los esquemas con una aptitud *por encima de la media*, que se denominarán *esquemas aventajados*, incrementan exponencialmente su presencia en sucesivas generaciones ($\kappa_g > 1$), mientras que los que tienen la aptitud por debajo de la media decrementan exponencialmente su presencia en la población ($\kappa_g < 1$).

- La tendencia de los esquemas aventajados a incrementar su presencia en sucesivas generaciones se acentúa cuando el esquema es corto y de bajo orden, pues entonces $\kappa_s \simeq 1$.

Desde esta perspectiva, se puede explicar la introducción de los tres operadores del AG de la siguiente manera:

1. *Selección:* La selección se encarga de incrementar geométricamente la presencia de los esquemas aventajados y reducir la presencia de los retrasados. No obstante, la selección no introduce nuevos esquemas.

2. *Cruce:* El cruce permite el intercambio estructurado de información útil (esquemas cortos, de bajo orden y aventajados) entre individuos. Así visto, el cruce es un operador esencial para el eficaz funcionamiento de un AG.

3. *Mutación:* La mutación introduce variedad en el juego de esquemas de la población y proporciona un mecanismo de seguridad frente a posibles pérdidas de información valiosa. Visto de esta manera, la

mutación es un operador secundario; de ahí que se aplique con bastante menor frecuencia que el cruce.

A la vista de todos estos resultados prácticos parece bastante razonable formular la siguiente hipótesis, que no ha sido completamente probada, la *Hipótesis de los Bloques Constructivos*: Los AGs exploran el espacio de búsqueda a través de la yuxtaposición de esquemas aventajados, cortos y de bajo orden. A tales esquemas se les llama *bloques constructivos*. De este modo, podemos considerar que los AGs procesan esquemas y, en particular, procesan bloques constructivos. Durante este procesamiento, los AGs dan oportunidades de proliferación exponencialmente crecientes a los esquemas más aptos, y exponencialmente decrecientes a los menos aptos.

Parte II

Optimización global

Capítulo 4

Búsqueda lineal genética

En este capítulo presentamos una nueva estrategia de resolución de problemas de optimización global, a la que hemos denominado Búsqueda Lineal Genética, desarrollada en el curso de esta tesis. Este nuevo método trata de extender los tradicionales métodos de optimización que usan búsqueda lineal, permitiendo explorar la dirección de búsqueda en un intervalo mucho más amplio, incluso en la rama de valores negativos. Se comenzará el capítulo revisando los conceptos y problemas de la búsqueda lineal, para continuar describiendo el método que hemos desarrollado: la Búsqueda Lineal Genética. Posteriormente se muestran los resultados obtenidos con la Búsqueda Lineal Genética frente a los métodos tradicionales de búsqueda lineal.

4.1. Introducción

Los métodos más antiguos y conocidos de optimización de funciones multidimensionales son los métodos de descenso [52] (véase 2.4.1 para una explicación más detallada). Estos métodos realizan un proceso iterativo de obtención del óptimo, según la siguiente fórmula:

$$x^{k+1} = x^k + \alpha^k d^k \tag{4.1}$$

La elección de un método de descenso requiere resolver dos cuestiones. Por un lado, determinar la dirección de búsqueda, d^k. Los diferentes métodos de descenso existentes se corresponden con los distintos modos de calcular estas direcciones de búsquedas [11, 42].

La segunda cuestión a resolver es la determinación de la longitud del paso, α^k, que se suele conocer como el problema de la búsqueda lineal [11]. Para ello se realiza una búsqueda a lo largo de la dirección de descenso, de forma que satisfaga:

$$\alpha^k = \min_{\alpha \geq 0} f(x^k + \alpha d^k) \tag{4.2}$$

La resolución de la búsqueda lineal mediante las estrategias clásicas (figura 2.5) se satisface obteniendo un paso que satisfaga la denominada condición de descenso (2.14). Para ello, se realiza una búsqueda en un espacio unimodal, para lo que se escoge un pequeño intervalo. De este modo, a partir del punto x^k realizan una búsqueda local en la dirección dada.

Los métodos de descenso suelen caracterizarse por ser técnicas de optimización rápidas, pero frecuentemente suelen quedar atrapadas en óptimos locales, salvo que el punto inicial se encuentre lo suficientemente cercano al óptimo global [10]. Esta dificultad es la que ha incentivado la proliferación de técnicas de optimización global, dado que son muchos los problemas existentes en la ingeniería que requieren de estas técnicas.

En los últimos años, varios algoritmos estocásticos y deterministas para la resolución de problemas de optimización global han sido propuestos [68, 148]. Los resultados computacionales han permitido comprobar que los métodos estocásticos son mejores que los métodos deterministas [135]. Uno de los métodos estocásticos más empleados son los AGs, diseñados por Holland [66]. Aunque es una técnica que no puede garantizar que siempre produce una solución óptima global, sí se considera como una técnica válida para resolver este tipo de problemas. Sin embargo, uno de los principales obstáculos en aplicar los AGs a problemas complejos ha sido su alto coste computacional además de su lenta convergencia en comparación con las técnicas clásicas de búsqueda local.

Una estrategia que intenta mejorar la convergencia de los AGs es el desarrollo de técnicas híbridas que combinan los AGs con los métodos de optimización local. El método más habitual de combinar ambas estrategias es introducir la técnica de optimización local como un operador más del AG [50, 115, 163]. De este modo, cada individuo de la población, antes de incorporarse a una población, sufre un proceso de optimización local. Podemos decir, que la característica de estos mecanismos híbridos es la aplicación en cadena de una técnica de optimización local con un AG.

Nuestra propuesta consiste en combinar una estrategia local, como es el máximo descenso con una estrategia global, como son los AGs. Sin embargo, a diferencia de los trabajos previos de métodos híbridos, el AG se va a integrar dentro del método de optimización local, de manera que se adquiere la convergencia de la técnica local y la facilidad del AG para explorar eficientemente todo el espacio de soluciones posibles.

En esta tesis se presenta una nueva estrategia de optimización, a la que hemos denominado la Búsqueda Lineal Genética (BLG). Ésta trata de extender la búsqueda lineal que se realiza en el método de descenso, permitiendo explorar la dirección de búsqueda en un intervalo mucho más amplio, incluyendo incluso la rama de valores negativos. Esta *búsqueda local extendida*

la realizaremos mediante un sencillo AG unidimensional.

La figura 4.1 ilustra la diferencia entre la búsqueda local tradicional realizada por los métodos tradicionales, y la búsqueda local extendida empleada en la BLG. En ella podemos descubrir cuál es el perfil que adquiere la función en la dirección de descenso obtenida en un punto x^k. Como se puede observar, la ampliación del límite de búsqueda permite descubrir nuevos óptimos que se encontraban más alejados del punto actual. Incluso la posibilidad de explorar el eje negativo permite determinar la existencia de nuevos óptimos más lejanos, que podrían incluso ser mejores que los disponibles en la dirección natural de descenso.

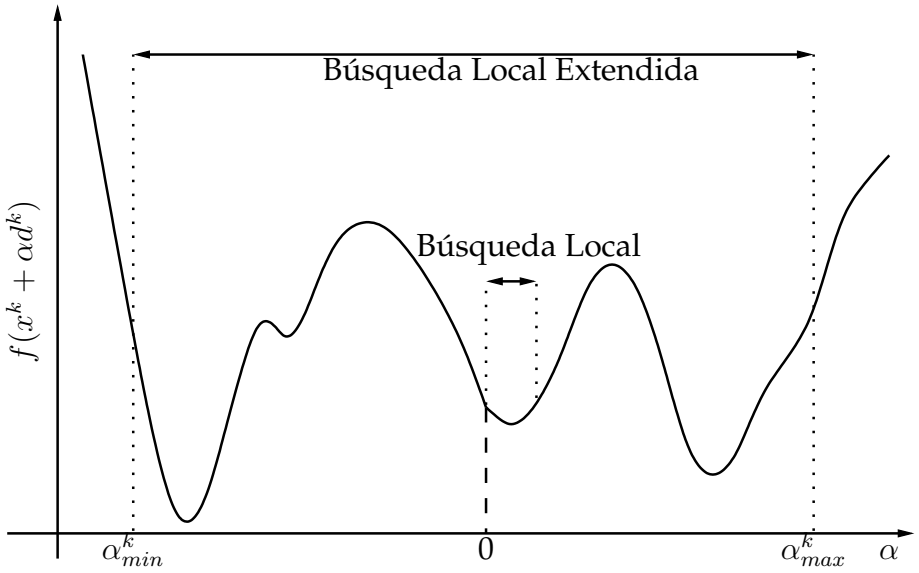


Figura 4.1: Búsqueda lineal local y extendida

La estructura del capítulo es la siguiente: el apartado 4.2 detalla los aspectos de implementación de la BLG que se ha desarrollado. El apartado 4.3.1 muestra el estudio realizado sobre la influencia de los distintos parámetros de la BLG en los resultados obtenidos. Los apartados 4.3 y 4.4 detallan las pruebas que hemos efectuado a la BLG con un conjunto de funciones multiextremas, recogidas en el anexo A, y su aplicación a las redes neuronales, respectivamente. Finalmente, el apartado 4.5 recoge las conclusiones extraídas de los resultados obtenidos en los experimentos realizados con la BLG.

4.2. La BLG

La BLG se realiza mediante un AG unidimensional, cuyas características principales se pueden resumir en las siguientes:

- Los individuos son escalares, codificados como números reales.
- Implementación de un AG de estado permanente, generándose en cada iteración un nuevo individuo.
- Sólo se emplean dos operadores, uno para cruce, y otro para mutación.
- La función de aptitud de los individuos es la misma función que se desea optimizar.
- Con objeto de evitar una rápida convergencia del algoritmo se emplean nichos.

4.2.1. Codificación de individuos

Los individuos en el AG representan el paso α^k que se tendría que dar en la k-ésima búsqueda lineal realizada. De este modo, la codificación de los mismos será de números reales de coma flotante [74].

4.2.2. Extensión de la Búsqueda Local

Asociada a la codificación de los individuos existen dos parámetros importantes del algoritmo, que van a determinar la extensión de la búsqueda lineal a realizar. Éstos son la escala del problema, S, y el tamaño de paso máximo α^M.

El tamaño de paso máximo nos permite imponer una condición del máximo valor que puede darse al paso α^k en la k-ésima búsqueda lineal realizada.

La escala del problema nos determina la magnitud de las variables de decisión, indicándonos si es del orden de la unidad o del orden de miles o millones. Ésta nos permite también fijar el tamaño de paso máximo a dar en cada iteración.

Aunque la BLG se ha presentado como una técnica de optimización de funciones sin restricciones, es posible utilizarla cuando las variables presentan unas cotas inferiores y superiores. La existencia de éstas afectan únicamente a la extensión de la búsqueda local a realizar, es decir, al rango de valores que van a poder tomar cada individuo en las distintas generaciones.

Una característica de los métodos tradicionales de búsqueda lineal es la realización de una exploración de valores positivos, dado que la dirección

de búsqueda seleccionada permite decrementar la función objetivo localmente en dicha dirección. Sin embargo, la BLG no se va a restringir a los vecinos del punto actual, x^k, sino que amplia el rango de búsqueda, tanto en el eje positivo como en el negativo. Aunque pequeños valores negativos incrementen el valor de la función objetivo, puede ser que existan óptimos globales en la lejanía del eje negativo.

Por lo tanto, cada individuo n-ésimo del AG en la k búsqueda lineal, se encuentra acotado de forma que:

$$\alpha^k_{min} \leq \alpha^n \leq \alpha^k_{max} \tag{4.3}$$

donde:

$$\begin{aligned} \alpha^k_{max} &\leq \text{mín} \left\{ \min_{i:d^k_i>0} \left\{ \frac{u_i - x^k_i}{d^k_i} \right\}, \min_{i:d^k_i<0} \left\{ \frac{l_i - x^k_i}{d^k_i} \right\}, \frac{S}{\|d^k\|}, \alpha^M \right\} \\ \alpha^k_{min} &\geq \text{máx} \left\{ \max_{i:d^k_i<0} \left\{ \frac{u_i - x^k_i}{d^k_i} \right\}, \max_{i:d^k_i>0} \left\{ \frac{l_i - x^k_i}{d^k_i} \right\}, -\frac{S}{\|d^k\|}, -\alpha^M \right\} \end{aligned} \tag{4.4}$$

siendo x^k el punto actual, d^k la dirección de búsqueda, l_i y u_i los límites inferiores y superiores, respectivamente, de la coordenada i del punto actual x^k, x^k_i y d^k_i la coordenada i del punto x^k y la dirección d^k, respectivamente.

4.2.3. Función de aptitud

Con objeto de determinar la bondad de la solución representada por cada individuo de la población, se define una función de aptitud. Dado que los AGs tratan de maximizar el valor de la función de aptitud de los distintos individuos a lo largo de las distintas generaciones, ésta vendrá dada según el problema de optimización en cuestión:

- En el caso de tratarse de un problema de maximización, la función objetivo será la propia función de aptitud.

$$F(\alpha^n) = f(x^k + \alpha^n d^k) \tag{4.5}$$

- En el caso de tratarse de un problema de minimización, la función de aptitud se define como:

$$F(\alpha^n) = -f(x^k + \alpha^n d^k) \tag{4.6}$$

Dado que el AG requiere que los individuos presenten siempre un valor positivo en la función de aptitud, es necesario realizar una translación de esos valores, con objeto de asegurar dicha condición. Ésta se representa por la siguiente ecuación:

$$F(\alpha^n) = F(\alpha^n) + |\min_j \left\{ F(\alpha^j) \right\}| + 1 \qquad \text{si} \quad \min_j \left\{ F(\alpha^j) \right\} < 0 \tag{4.7}$$

4.2.4. Operadores

Se emplearán únicamente dos operadores, uno para el cruce y otro para la mutación. La operación de cruce tendrá una probabilidad de realizarse $p_c \in [0, 1]$, mientras que la de mutación tendrá una probabilidad, de acuerdo a la siguiente fórmula:

$$p_m = 1 - p_c \tag{4.8}$$

Esta relación entre ambas probabilidades es debido a que en cada generación sólo se introducirá un único individuo, al implementarse el AG como uno de estado permanente.

4.2.4.1. Operador de cruce

El operador de cruce que se emplea es la *combinación lineal convexa*, también denominado *cruce aritmético* [102], de dos individuos seleccionados previamente. De esta forma, si p_1 y p_2 son los dos individuos seleccionados, denominados *padres*, los dos *hijos*, h_1 y h_2, que resultan son:

$$\begin{aligned} h_1 &= \beta p_1 + (1 - \beta) p_2 \\ h_2 &= (1 - \beta) p_1 + \beta p_2 \end{aligned} \tag{4.9}$$

donde $\beta \in [0, 1]$ es escogido de forma aleatoria.

Este tipo de operación, asegura que si los dos individuos padres seleccionados se encuentran dentro del rango $[\alpha^k_{min}, \alpha^k_{max}]$ donde realizar la búsqueda local extendida, los dos nuevos hijos generados también se encuentran dentro del mismo intervalo.

4.2.4.2. Operador de mutación

Se emplea una mutación consistente en generar una perturbación gaussiana aleatoria alrededor del individuo seleccionado. Así, si p es el individuo seleccionado, el individuo mutado resultante, m, será:

$$m = p + N(0, \theta) \tag{4.10}$$

La magnitud de la perturbación que se realiza se controla a través de la desviación estándar θ, cuyo valor se mantiene constante en todo el proceso de optimización. Esta magnitud está relacionada con la probabilidad de mutación y el intervalo de búsqueda, de forma que:

$$\theta = (\alpha^k_{max} - \alpha^k_{min})(1 - p_m) \tag{4.11}$$

De este modo, si las mutaciones son muy frecuentes, el tamaño de la magnitud de éstas decrece, y viceversa.

Cuando una mutación tiene lugar, el nuevo individuo puede situarse fuera del intervalo de la búsqueda local extendida $[\alpha^k_{min}, \alpha^k_{max}]$, por lo que tras la operación es necesario realizar una reparación del individuo consistente en establecer su genotipo al valor más cercano a dicho intervalo. De forma matemática, si m es el individuo resultante tras la mutación, el individuo que se genera al realizar la translación, m_e, es:

$$\begin{aligned} m_e &= \alpha^k_{min} \qquad \text{si} \quad m < \alpha^k_{min} \\ m_e &= \alpha^k_{max} \qquad \text{si} \quad m > \alpha^k_{max} \\ m_e &= m \qquad \text{si} \quad \alpha^k_{min} \leq m \leq \alpha^k_{max} \end{aligned} \tag{4.12}$$

4.2.4.3. Selección de individuos

En la realización de las operaciones de cruce y mutación es necesario realizar una selección de individuos. Para ello se realiza una selección con probabilidad uniforme sin tener en cuenta el valor de la función de aptitud en cada individuo.

4.2.5. Generaciones

El AG empleado será uno de estado permanente, donde en cada generación se produce un único individuo, ralentizando la convergencia rápida del algoritmo, con objeto de explorar todo el rango de valores posibles.

En cada generación se realiza una única operación, un cruce o una mutación. Cuando se realiza una mutación se requiere un único individuo para producir otro. En el caso de la operación de cruce, se necesitan dos padres y genera dos hijos, de los cuales, el que presenta un mejor valor de la función de aptitud se selecciona. De este modo, la población se renueva en un ratio de un individuo por generación.

4.2.5.1. Eliminación de individuos

Para la introducción del nuevo individuo generado en la población, es necesario eliminar uno previamente. Esta selección es independiente de los individuos empleados en la reproducción. De este modo, la selección para eliminar un individuo se tiene en cuenta la aptitud del individuo, empleándose una distribución probabilística geométrica.

4.2.6. Nichos

En el proceso de optimización, cuando un AG falla encontrando el óptimo global, el problema es frecuentemente atribuido a una convergencia

prematura hacia un óptimo local. Con objeto de incentivar la diversificación y evitar esa convergencia prematura, se emplea un esquema de división para facilitar la formación de nichos [55]. Esto modifica la aptitud de cada individuo, penalizando la acumulación de individuos cercanos. Cuando varios individuos son muy similares entre sí, mutuamente se reducen su aptitud.

En esta técnica existe un parámetro, denominado el *tamaño del nicho*, σ, que establece la distancia mínima de separación de dos individuos para que no interfieran entre sí. En lugar de tener que establecer este parámetro, el usuario especifica el *número de soluciones por nicho*. De este modo, el número de nichos que se esperan en el intervalo de búsqueda se obtiene según la relación siguiente:

$$NumNichos = \frac{TamPob}{NumSolNicho} \tag{4.13}$$

A continuación, el tamaño del nicho, σ se puede establecer en base a la longitud del intervalo de búsqueda y el número de nichos:

$$\sigma = \frac{\alpha^k_{max} - \alpha^k_{min}}{NumNichos} \tag{4.14}$$

La penalización de la aptitud de los individuos con nichos consiste en determinar cuál es el grado de compartición del individuo x^k con el resto de la población, de forma que:

$$F_N(x^k) = \frac{F(x^k)}{m^k} \tag{4.15}$$

donde $F(x^k)$ es el valor de la función de aptitud del individuo x^k, y m^k es el contador de nicho, que define la cantidad de solapamiento (o compartición) del individuo x^k con el resto de la población. Este contador de nicho, se calcula en base a la suma de la *función de compartición*, F_C, de todos los miembros de la población con dicho individuo:

$$m^k = \sum_{j=1}^{TamPob} F_C(d(x^k, x^j)) \tag{4.16}$$

donde $d(x^i, x^j)$ representa la distancia entre el individuo x^i y x^j en la población

La función de compartición determina si la distancia entre los dos individuos se encuentra dentro del radio establecido por el tamaño del nicho, σ, devolviendo un valor entre 0 y 1, que se incrementa según el grado de similitud entre los dos individuos. En otro caso, devuelve un 0. De este modo, cada miembro de la población se considera centro de un nicho, y el

valor de su aptitud es reducido por todos los individuos cuya distancia a él sea menor que el tamaño del nicho, σ. De forma matemática:

$$F_C(d(x^i, x^j)) = \begin{cases} 1 - \left(\frac{d(x^i,x^j)}{\sigma}\right)^{\xi} & \text{si} \quad d(x^i, x^j) < \sigma \\ 0 & \text{en otro caso} \end{cases} \tag{4.17}$$

donde ξ determina el comportamiento de la función de compartición. Normalmente, suele establecerse a 1, y la función resultante se denomina la *función de compartición triangular* [104]. El algoritmo empleado en la BLG $\xi = 1$.

En relación a la distancia de los individuos, $d(x^i, x^j)$, se toma una métrica basada en el genotipo de los individuos, de forma que:

$$d(x^i, x^j) = |x^i - x^j| \tag{4.18}$$

Algoritmo 4.1 BLG

BLG : $T_P \times p_c \times N_N \times G \times \alpha^M \times x \times \epsilon \longrightarrow x^*$
$k \longleftarrow 0$
$x^k \longleftarrow x$
$d^k \longleftarrow$ direccion_búsqueda(x^k)
$\alpha^k \longleftarrow$ AG-BLG$(T_P, p_c, N_N, G, \alpha^M, x^k, d^k)$
$x^{k+1} \longleftarrow x^k + \alpha^k d^k$
$k \longleftarrow k + 1$
Mientras $\left(\frac{|f(x^{k+1}) - f(x^k)|}{|f(x^{k+1})|} \leq \epsilon\right)$
  $x^k \longleftarrow x^{k+1}$
  $d^k \longleftarrow$ direccion_búsqueda(x^k)
  $\alpha^k \longleftarrow$ AG-BLG$(T_P, p_c, N_N, G, \alpha^M, x^k, d^k)$
  $x^{k+1} \longleftarrow x^k + \alpha^k d^k$
  $k \longleftarrow k + 1$
Devolver x^k

4.2.7. Proceso de optimización

El algoritmo 4.1 presenta un pseudocódigo de la BLG. El AG que realiza la búsqueda lineal extendida se muestra en el algoritmo 4.2. Los parámetros de entrada que necesita para su funcionamiento son los siguientes:

- Tamaño de la población: T_P.
- Probabilidad de cruce: p_c.

- Número de soluciones por nicho: N_N.
- Número de generaciones: G.
- Paso máximo: α^M.
- Punto inicial donde realizar la búsqueda: x.
- Aproximación al óptimo: ϵ.

Algoritmo 4.2 Algoritmo genético de la BLG

AG-BLG : $T_P \times p_c \times N_N \times G \times \alpha^M \times x^k \times d^k \longrightarrow \alpha$
$t \longleftarrow 0$
$P_t \longleftarrow$ Crear_Poblacion_Inicial(T_P, α^M)
Evaluar_Aptitud$()$
Aplicar_Nichos(N_N)
Mientras $(t < G)$
  $t \longleftarrow t + 1$
  $P_t \longleftarrow P_{t-1}$
  Eliminar_Individuo(P_t)
  Si $(op =$ CRUCE$)$ **entonces**
    $[padre_1, padre_2] \longleftarrow$ Seleccionar_Padres(P_{t-1})
    $[hijo_1, hijo_2] \longleftarrow$ Cruzar$(padre_1, padre_2)$
    Evaluar_Aptitud$(hijo_1, hijo_2)$
    Si $(F(hijo_1) < F(hijo_2))$
      Introducir_Individuo$(P_t, hijo_2)$
    Si no
      Introducir_Individuo$(P_t, hijo_1)$
  Si no
    $ind \longleftarrow$ Seleccionar_Individuo(P_{t-1})
    $mutante \longleftarrow$ Mutar(ind)
    Introducir_Individuo$(P_t, mutante)$
  Evaluar_Aptitud$()$
  Aplicar_Nichos(N_N)
Evaluar_Aptitud$()$
$\alpha \longleftarrow$ Mejor_Individuo$()$
Devolver α

Con objeto de acelerar el proceso de optimización global, también se introduce un parámetro más que es el número de generaciones consecutivas sin mejorar. De este modo, cuando se realicen dichas generaciones y el mejor individuo de la población no haya variado, el algoritmo finalizará. Este

parámetro nos sirve para detectar de una forma rápida cuando el algoritmo ha alcanzado el óptimo en la búsqueda local realizada.

Como se puede apreciar en el algoritmo 4.1, la BLG permite emplearse en numerosos métodos de optimización que requieran el cálculo de una dirección de optimización, tales como los métodos de descenso. Sin embargo, su versatilidad permite integrarse incluso en técnicas no derivativas como los métodos de descenso coordenado, lo que permite ampliar el rango de funciones donde es posible aplicar dicha estrategia.

4.3. Resultados con la BLG

Las pruebas efectuadas con la BLG han sido realizadas en dos fases. En primer lugar, se ha determinado cómo influyen los distintos parámetros de entrada del AG en los resultados obtenidos. Una vez fijados los parámetros, se ha procedido a realizar pruebas computacionales con diversas funciones de la bibliografía.

4.3.1. Influencia de los parámetros de entrada

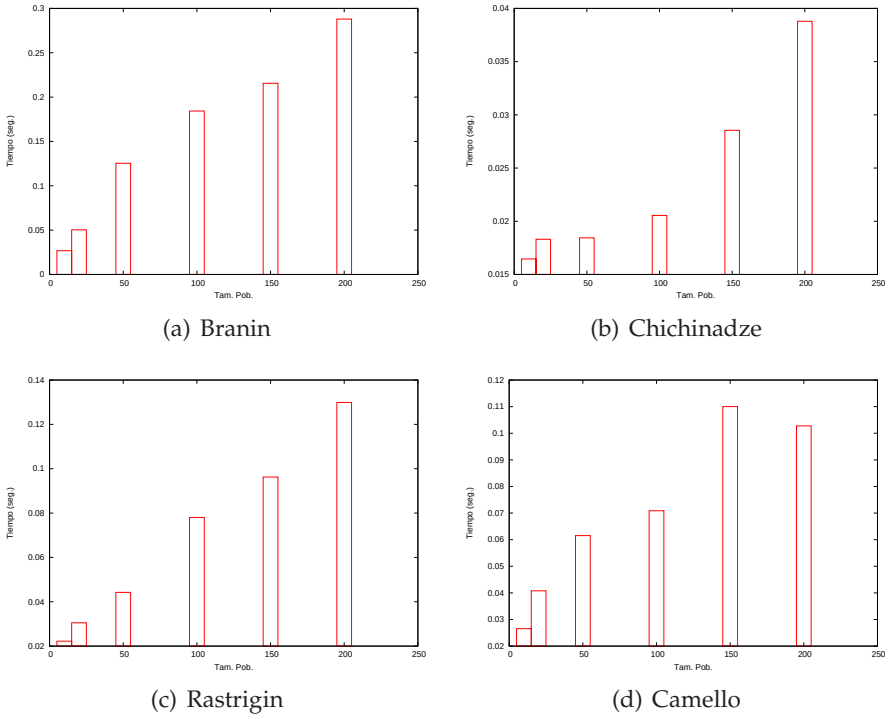
Dado la existencia de un conjunto de parámetros de entrada de la BLG, es necesario comprender la influencia de éstos en el funcionamiento del algoritmo, con objeto de determinar los valores más adecuados.

4.3.1.1. Tamaño de la población

Un parámetro importante en la evolución de la BLG es precisamente el tamaño de la población. En las pruebas computacionales realizadas sobre las funciones del apéndice A, se ha observado como a medida que aumenta el tamaño de la población el tiempo empleado en la optimización del algoritmo aumenta. Esto se puede observar en la figura 4.2, donde se muestra el tiempo medio empleado en la optimización de tres funciones empezando en 100 puntos escogidos de forma aleatoria. En las simulaciones se ha establecido un número de generaciones de 250, y una probabilidad de cruce de 0.9. Este resultado lógico es debido a que la realización de una generación es más lenta dado que tiene que evaluar un mayor número de individuos.

Sin embargo, el tamaño de la población incide de forma directa sobre el éxito en la búsqueda del óptimo del problema planteado. Esto es debido a que un mayor tamaño de la población permite generar un mayor número de puntos aleatorios en las búsquedas lineales realizadas por el AG, lo que permite aumentar la posibilidad de encontrar el óptimo. Esto se puede observar en la tabla 4.1, donde para distintos tamaños de la población, el porcentaje de éxito logrado, es decir, el número de veces que el algoritmo llega al óptimo, aumenta a medida que el tamaño es más grande. Se puede

observar, que a partir de cierto tamaño, el porcentaje de éxito es del 100 %[1]. En las pruebas computacionales de dicha tabla se han realizado 500 generaciones para todos los tamaños, manteniéndose una probabilidad de cruce de 0.5.



(a) Branin (b) Chichinadze

(c) Rastrigin (d) Camello

Figura 4.2: Relación entre el tamaño de la población y el tiempo de ejecución

Función	Tamaño Población				
	10	20	50	100	200
Branin	36 %	60 %	70 %	72 %	66 %
Chichinadze	88 %	96 %	100 %	98 %	100 %
Joroba de Camello	80 %	96 %	96 %	100 %	100 %
Rastrigin	76 %	80 %	80 %	86 %	76 %

Tabla 4.1: Relación entre el tamaño de la población y el porcentaje de éxito

En la tabla 4.1 podemos destacar el hecho de que a veces un tamaño más grande en la población no asegura una mejor solución. Así, en el caso de la función de Branin, con una población de 200 individuos el porcentaje

[1]Se ha tomado como éxito cuando el error cometido es inferior a 0.001.

de aciertos disminuye en un 6 % respecto a los aciertos que se obtienen con una población de 100 individuos. Similar comportamiento se observa en la función de Rastrigin.

4.3.1.2. Número de generaciones

Otro parámetro que determina el comportamiento del funcionamiento del algoritmo es el número de generaciones. La BLG emplea un algoritmo de estado permanente (apartado 4.2.5), lo que implica que en cada generación se va a producir un único individuo nuevo, manteniéndose el resto de la población constante. Este hecho, motiva que el número de generaciones esté relacionada con el tamaño de la población.

De este modo, si empleamos un tamaño de población de 50 individuos, el realizar 200 generaciones significa que se van a introducir 200 nuevos individuos sobre la población inicial, lo que supone un incremento de la población en un 400 %, es decir, se generan cuatro poblaciones completas. Sin embargo, si el tamaño de la población inicial es de 500 individuos, el realizar 200 generaciones supone un incremento del 40 %, por lo que ni la mitad de la población inicial cambia.

Func.	Número de generaciones							
	50		100		250		500	
	Tiempo	Éxito	Tiempo	Éxito	Tiempo	Éxito	Tiempo	Éxito
BR	0.00378	6 %	0.01241	26 %	0.01913	24 %	0.03682	28 %
CH	0.01810	84 %	0.01838	98 %	0.01851	99 %	0.02063	100 %
JC	0.01063	84 %	0.01637	88 %	0.04929	98 %	0.04651	96 %
RA	0.06734	74 %	0.12911	86 %	0.31442	90 %	0.26253	76 %

Tabla 4.2: Relación entre el número de generaciones y el porcentaje de éxito

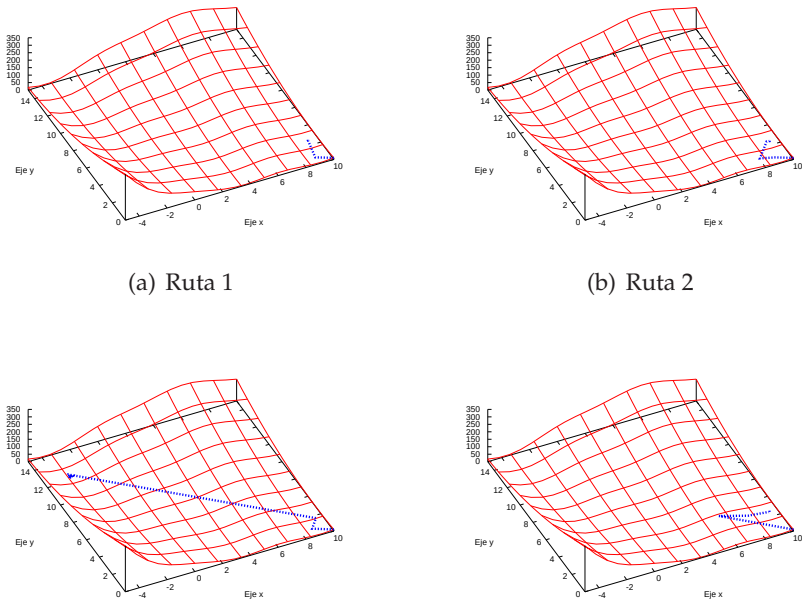
El número de generaciones permite al algoritmo el poder realizar una exploración más óptima del intervalo de búsqueda lineal, ya que en cada punto puede generar una mayor cantidad de individuos. Esto puede permitir a la BLG obtener mejores soluciones. Sin embargo, es obvio que la realización de un mayor número de generaciones requiere un mayor tiempo en el proceso de optimización. La tabla 4.2 muestra este resultado para las funciones de Branin (BR), Chichinadze (CH), Joroba de Camello (JC) y Rastrigin (RA). En ella se recoge, para cien puntos iniciales distribuidos aleatoriamente por la región de búsqueda, el tiempo medio empleado así como el porcentaje de aciertos[2] logrado para diversas funciones. Se ha empleado el método de descenso coordenado, utilizándose la BLG con un tamaño de población de 200 individuos y una probabilidad de cruce de 0.5.

[2]Se ha tomado como éxito cuando el error cometido es inferior a 0.001.

Podemos observar que los mejores resultados se obtienen cuando el número de generaciones permite renovar al menos el 50 % de la población.

4.3.1.3. Aleatoriedad

Una de las características de la BLG es precisamente su carácter aleatorio. Aunque se ajusten los parámetros del algoritmo hemos de tener en cuenta que para un mismo punto de partida y un juego de parámetros idénticos es posible conseguir distintas ejecuciones.



(a) Ruta 1 (b) Ruta 2

(c) Ruta 3 (d) Ruta 4

Figura 4.3: Rutas realizadas en la optimización de la función de Branin

Así, la figura 4.3 muestra cuatro ejecuciones que se han obtenido en la optimización de la función de Branin, partiendo del punto (10, 0). Los parámetros que se han empleado son los siguientes:

- Método de máximo descenso.
- Tamaño de la población: 100 individuos.
- Número de generaciones: 200.
- Probabilidad de cruce: 0.6.
- Número de soluciones por nicho: 0.2.

4.3.1.4. Número de soluciones por nicho

El número de soluciones por nicho permite incluir en la BLG una penalización de los individuos debido a su cercanía. Esto permite evitar la deriva genética y una convergencia prematura del algoritmo. Con objeto de favorecer la dispersión de los individuos de la población a lo largo de todo el intervalo de búsqueda lineal, se realizaba una penalización de su aptitud en base a la cercanía de éstos.

Las figuras 4.4, 4.5 y 4.6 muestran el efecto que produce en la BLG la introducción de estos nichos. En ellas se representa para la primera iteración del proceso de optimización de la función de Rastrigin la evolución que sufre el AG. Se muestra los individuos que se tienen en la población inicial, así como la población existente tras 25, 50 y 75 generaciones respectivamente, para diferentes tamaño de nichos. Dado que el tamaño de la población empleada es de 50 individuos, podemos considerar esta última como una BLG sin nichos.

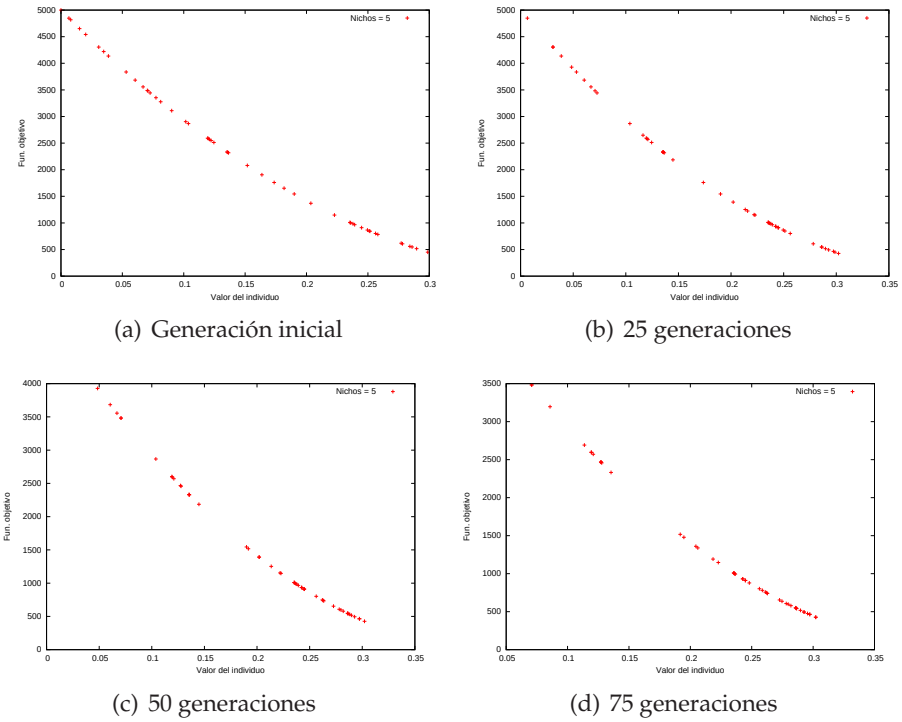
Se puede apreciar, como inicialmente en los tres casos la población está repartida por todo el espacio de búsqueda. Sin embargo, a medida que evoluciona en el caso de no aplicar nichos, los individuos sufren la deriva genética que provoca la rápida convergencia del algoritmo. Por otro lado, la introducción de nichos, permite mantener individuos a lo largo de la extensión de búsqueda, haciéndose más grande la presencia en caso de tener un menor número de nichos.

4.3.1.5. Tamaño de paso

Un aspecto importante en la BLG es la extensión de la búsqueda lineal. Mientras los métodos clásicos sólo realizan una exploración en un intervalo pequeño según la dirección de descenso, la BLG permite extenderlo, incluso en la dirección del eje negativo. El tamaño de paso es el parámetro del algoritmo que permite determinar la longitud de la misma. La ecuación (4.4) reflejaba las cotas máximas y mínimas que presentaban los distintos individuos.

Dicha relación permite especificar un intervalo de búsqueda adaptativo al punto de solución actual donde se encontraba. Esto es debido a que cuando se encuentra lejos del óptimo el tamaño del intervalo de búsqueda es bastante pequeño. Sin embargo, a medida que el algoritmo se iba acercando al óptimo (gradiente cercano a cero) se incrementaba la longitud del intervalo. De este modo, cuando estamos lejos de una solución la BLG se conforma con dar pasos pequeños que mejoren su situación. Sin embargo, cuando detecta la cercanía de un óptimo amplía su horizonte de búsqueda con objeto de determinar si se encuentra en un óptimo local, lo cuál le permitiría escapar de él, o en uno global, que le dejaría en él.

Este comportamiento se puede observar en la figura 4.7, donde se mues-



(a) Generación inicial (b) 25 generaciones

(c) 50 generaciones (d) 75 generaciones

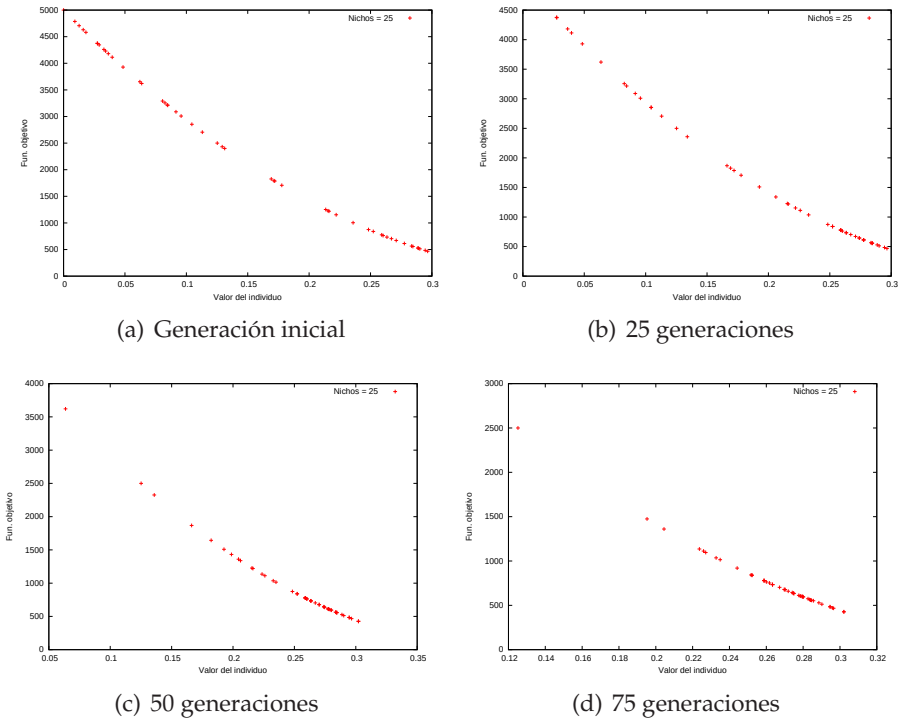
Figura 4.4: Resultados obtenidos con un tamaño de nicho = 5

tra en cada iteración el tamaño del intervalo donde realizar la BLG para distintos parámetros de tamaño del paso máximo. Podemos ver que conforme nos acercamos al óptimo de la función (en este caso la de Rastrigin) la longitud del tamaño del intervalo aumenta. Por contra, si nos encontramos alejados del óptimo el tamaño suele ser pequeño.

4.3.2. Eficacia de la búsqueda lineal extendida

En este apartado se procederá a la comparación entre la BLG y los métodos tradicionales que se emplean para la realización de una búsqueda lineal. Para ello se han realizado diversas pruebas computacionales con las funciones que aparecen en el apéndice A.

Para las pruebas computacionales realizadas hemos utilizado dos métodos derivativos, el método de máximo descenso (apartado 2.4.1.2) y el método BFGS (apartado 2.4.1.3), así como un método no derivativo como es el descenso coordenado cíclico (apartado 2.4.2). En todos ellos las búsquedas lineales han sido realizadas con la BLG y con técnicas tradicionales. En concreto, se ha utilizado un método de búsqueda inexacta como la regla de Armijo (2.4.1.1) y otro de búsqueda exacta, como la búsqueda de la



(a) Generación inicial (b) 25 generaciones

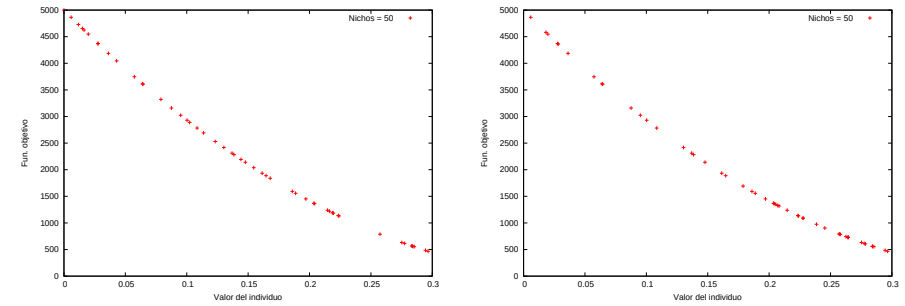
(c) 50 generaciones (d) 75 generaciones

Figura 4.5: Resultados obtenidos con un tamaño de nicho = 25

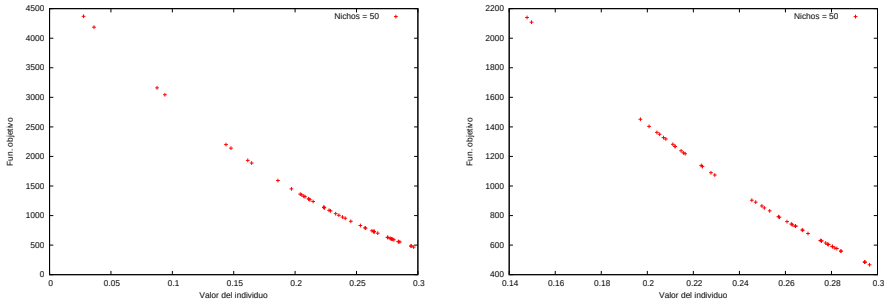
sección áurea (apartado 2.3.3).

Una diferencia que se encuentra con los métodos tradicionales y la BLG es precisamente el proceso de búsqueda en sí. Así, para un mismo punto, mientras que los métodos tradicionales tardan poco tiempo en encontrar una solución, la BLG emplea más tiempo, pero a su vez, el avance que se da en la función objetivo es mucho mayor.

Las figuras 4.8 y 4.9 muestran precisamente esta diferencia, ilustrándose el proceso de optimización de la función de Rastrigin en la primera iteración desde el punto inicial $[-50, -50]$. En el caso de los métodos tradicionales se muestra el mejor valor de la función objetivo que hasta ese momento han encontrado, y para la BLG el valor del mejor individuo encontrado. Las ejecuciones que se muestran son con el máximo descenso, utilizándose la regla de Armijo y la sección áurea, así como cinco ejecuciones de la BLG, ya que dado su comportamiento aleatorio, no siempre se obtienen los mismos resultados.



(a) Generación inicial (b) 25 generaciones



(c) 50 generaciones (d) 75 generaciones

Figura 4.6: Resultados obtenidos con un tamaño de nicho = 50

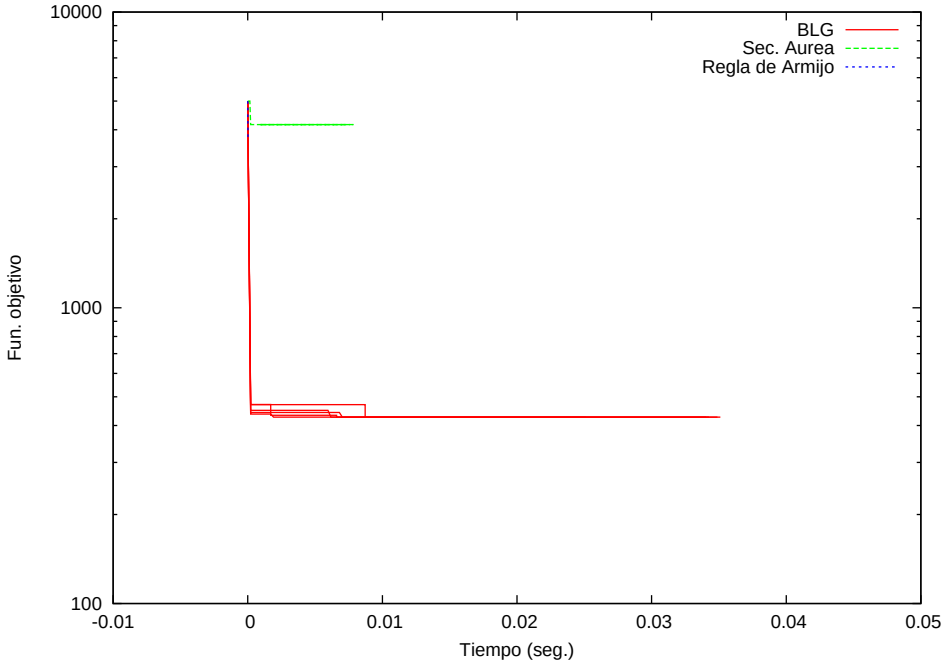


Figura 4.8: Evolución en el tiempo de la primera búsqueda lineal

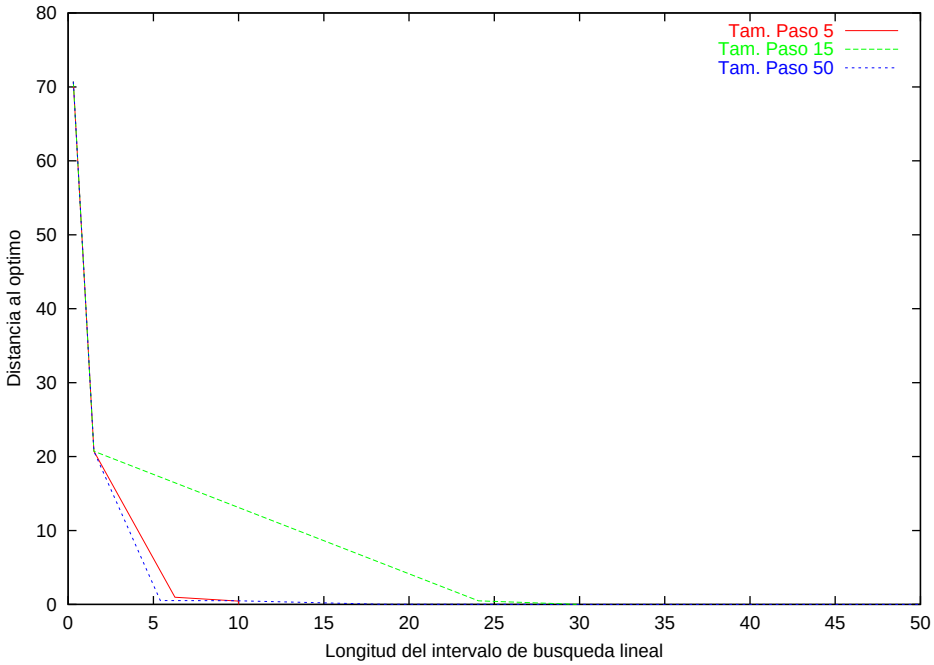


Figura 4.7: Variación de la longitud del intervalo de búsqueda lineal

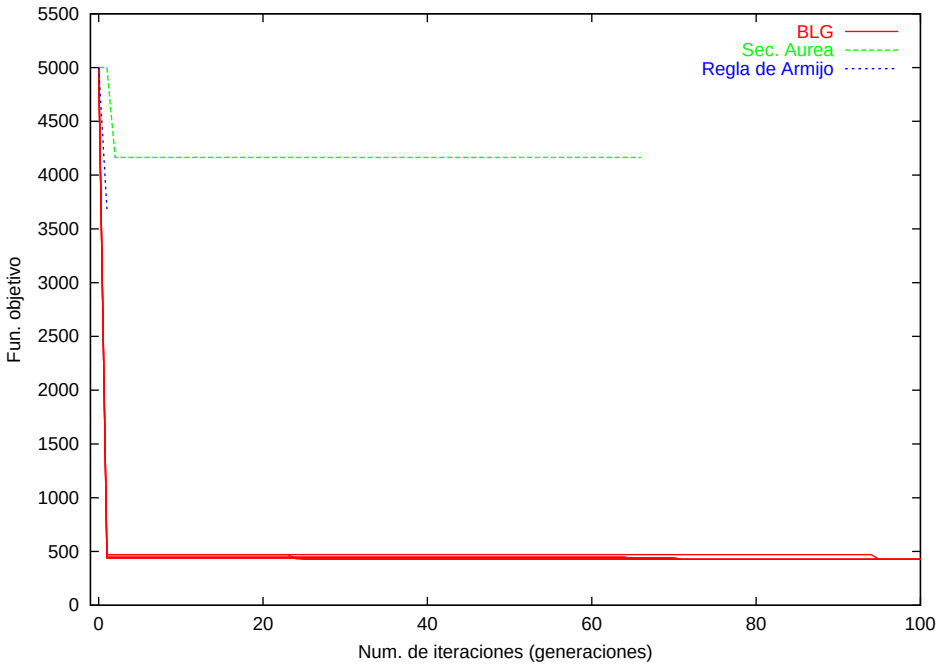
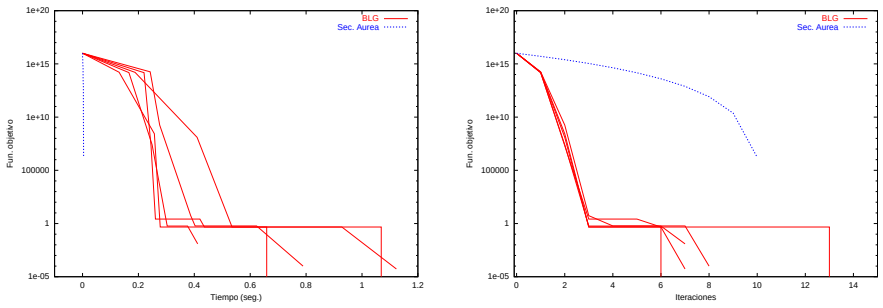


Figura 4.9: Evolución de la primera búsqueda lineal frente al número de iteraciones

En la figura 4.8 podemos ver como los métodos tradicionales emplean poco tiempo en la realización de la primera iteración, pero la disminución de la función objetivo, en comparación con la BLG es menor. Esto es debido principalmente a que la región de exploración de la BLG es más amplia, extendiéndose incluso al rango de valores negativos.

Por otro lado, la figura 4.9 ilustra el número de iteraciones que tienen que hacer para explorar el intervalo. En la BLG una iteración corresponde con una generación, mientras que en los métodos tradicionales son las iteraciones para determinar el paso óptimo α. Podemos apreciar como el número de iteraciones que debe realizar el AG es mucho mayor que en el caso de los métodos tradicionales. Destacar el hecho de que la regla de Armijo con la primera iteración encuentra un paso α que asegura la condición de descenso. Por contra, la búsqueda mediante la sección áurea necesita realizar un mayor número de iteraciones para localizar el óptimo en dicha región.

Sin embargo, analizando el proceso de optimización completo, se puede observar cómo los métodos tradicionales no sirven cuando se emplean funciones multiextrema, debido a la existencia de numerosos óptimos locales. Sin embargo, la BLG, dado que realiza una exploración más profunda del intervalo de búsqueda lineal, consigue escapar de esos óptimos locales y llegar a la solución de la función.



(a) Evolución frente al tiempo

(b) Evolución frente al número de iteraciones

Figura 4.10: BFGS con BLG y Sección áurea

Esto se puede observar en la figura 4.10 donde se muestra la evolución de la función de Beale frente al tiempo y frente al número de iteraciones tanto la BLG como para la sección áurea. Escogiendo como punto inicial el $[-100, -100]$ se han realizado diversas ejecuciones con la BLG, dado su comportamiento aleatorio. Los parámetros que se han empleado son: tamaño de población de 50 individuos, 750 generaciones, y una probabilidad de cruce de 0.4. Estas búsquedas se comparan con la sección áurea. En ambas

estrategias se ha escogido como dirección de descenso el método BFGS.

En la figura 4.10 se puede comprobar como la BLG emplea un mayor tiempo que los métodos tradicionales. Sin embargo, este incremento de tiempo supone conseguir llegar al óptimo.

Una diferencia notable entre la BLG y los métodos tradicionales es el número de iteraciones que realizan, siendo menor en la BLG que en métodos clásicos como la sección áurea o la regla de Armijo. En el caso de la función de Beale (figura 4.10), podemos ver que la mayoría de las ocasiones no pasa de 8 iteraciones.

Este número menor de iteraciones se puede comprobar de forma generalizada en el comportamiento de la BLG. Por ejemplo, la tabla 4.3 muestra para la función de Griewank y diez puntos distribuidos de forma aleatoria en el espacio de direcciones, el número de iteraciones que se realizan y el valor de la función objetivo final conseguido empleándose el método de la sección áurea y la BLG dentro de BFGS. Podemos ver como la BLG realiza siempre menos iteraciones, y se consigue una mejora de la función objetivo mucho mejor que si se emplease el método de la sección áurea. Resultados similares se obtienen en la comparación de la regla de Armijo y la BLG.

$f(x^0)$	Sección áurea			BLG		
	Iter.	$f(x^k)$	$f(x^0)-f(x^k)$	Iter.	$f(x^k)$	$f(x^0)-f(x^k)$
43.1683	31	1.727	41.4413	8	0.003286	43.165
47.1283	31	1.48769	45.6406	6	0.000290	47.128
44.7311	32	1.14603	43.5851	11	0.000009	44.7311
46.4387	32	1.94654	44.4922	7	0.000442	46.4383
59.368	32	5.87041	53.4976	12	0.000000	59.368
57.4296	33	4.91727	52.5123	8	0.006674	57.4229
69.8163	33	9.53073	60.2856	9	0.001926	69.8144
47.9501	36	0.654316	47.2958	7	0.000442	47.9497
67.5013	36	4.31506	63.1862	10	0.007438	67.4939
78.6493	37	7.80712	70.8422	11	0.018661	78.6306

Tabla 4.3: Comparación del número de iteraciones

Las figuras 4.11 y 4.12 permiten comparar con más detenimiento el número de iteraciones y el tiempo que se emplea en la realización de la optimización con el método de máximo descenso, empleándose la BLG, la sección áurea y la regla de Armijo. Se muestra, para cincuenta puntos iniciales escogidos aleatoriamente por la región de búsqueda, la diferencia entre el valor de la función del óptimo y el que se consigue con el proceso de optimización frente al tiempo empleado y el número de generaciones realizadas. En este caso la optimización es de la función de Chichinadze, y se emplea una BLG con 200 individuos, 100 generaciones y una probabilidad de cruce de 0.9. Se puede apreciar como la BLG requiere un menor número de iteraciones que los métodos tradicionales. Sin embargo, el tiempo que se requiere

es superior. No obstante, el tiempo necesario medio se sitúa en torno a 0.1 segundo de ejecución.

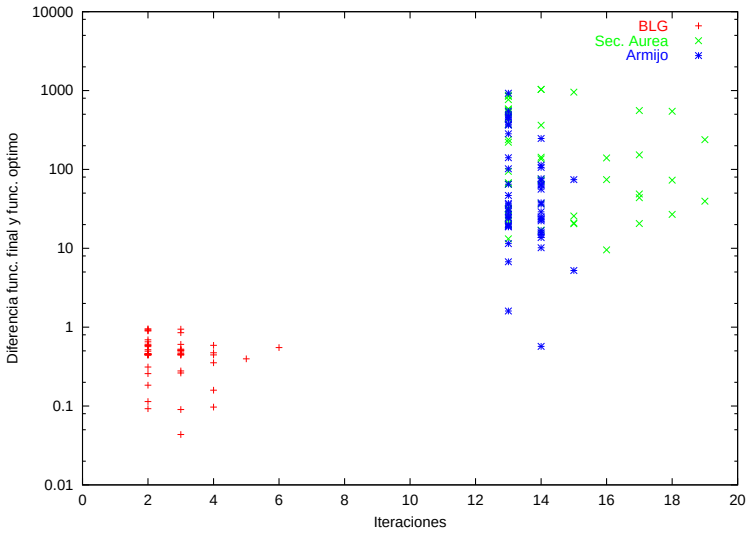


Figura 4.11: Evolución del máximo descenso con BLG, Sección áurea y Armijo frente al nº de iteraciones

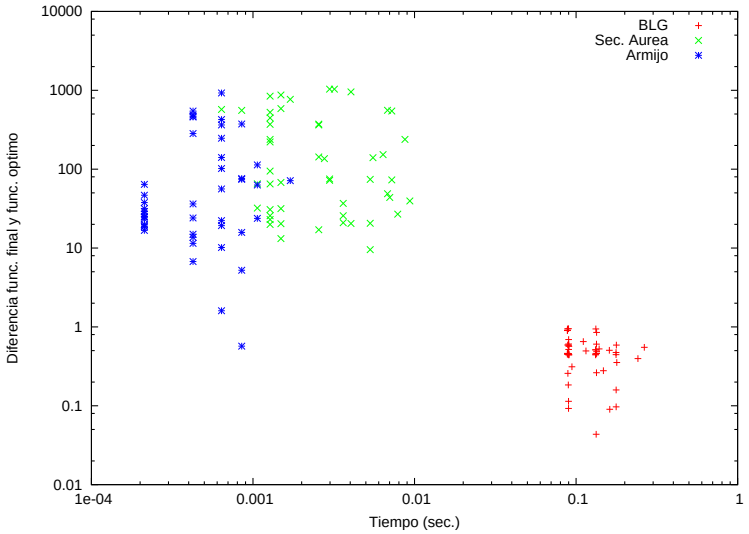
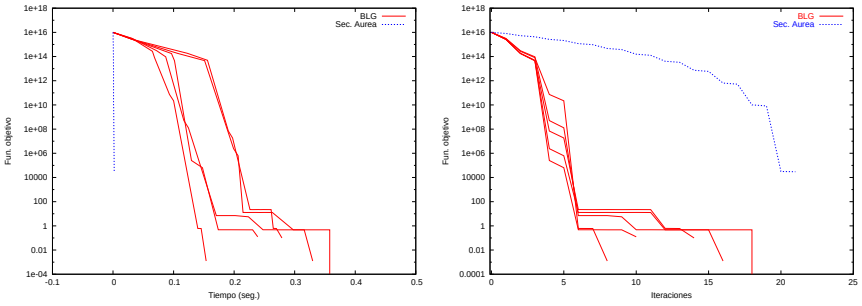


Figura 4.12: Evolución del máximo descenso con BLG, Sección áurea y Armijo frente al tiempo

4.3.2.1. Elección de la dirección de descenso

En las pruebas computacionales realizadas se ha podido comprobar que la BLG funciona perfectamente con cualquier método que determine la dirección de descenso. La figura 4.13 muestra para la función de Beale, la evolución que sufre frente al tiempo y al número de iteraciones, tanto con la BLG y la sección áurea, empleándose en ambos casos un método de descenso coordenado.



(a) Evolución frente al tiempo (b) Evolución frente al número de iteraciones

Figura 4.13: Descenso coordenado con BLG y Sección áurea

Si se compara con la figura 4.10 se puede observar cómo la BLG consigue resultados similares en ambos casos. Estas pruebas computacionales permiten determinar que la BLG es un método robusto en cuanto a la selección de la dirección de descenso.

4.3.2.2. Elección del punto inicial

Una característica que presenta la BLG frente a los métodos tradicionales es que la elección del punto inicial donde comenzar el proceso de optimización no es un punto conflictivo en la convergencia del método. La figura 4.14 muestra, para la función de Griewank de 2 variables, cómo en el caso de los métodos tradicionales es necesario realizar una buena elección del punto inicial, al contrario que en la BLG, donde no es tan crítico dicha elección.

4.3.3. Resumen de resultados

La tabla 4.4 muestra, para las funciones de Branin (BR), Chichinadze (CH), Joroba de Camello (JC) y Rastrigin (RA), el porcentaje de acierto[3]

[3]Se ha tomado como éxito cuando el error cometido es inferior a 0.00001.

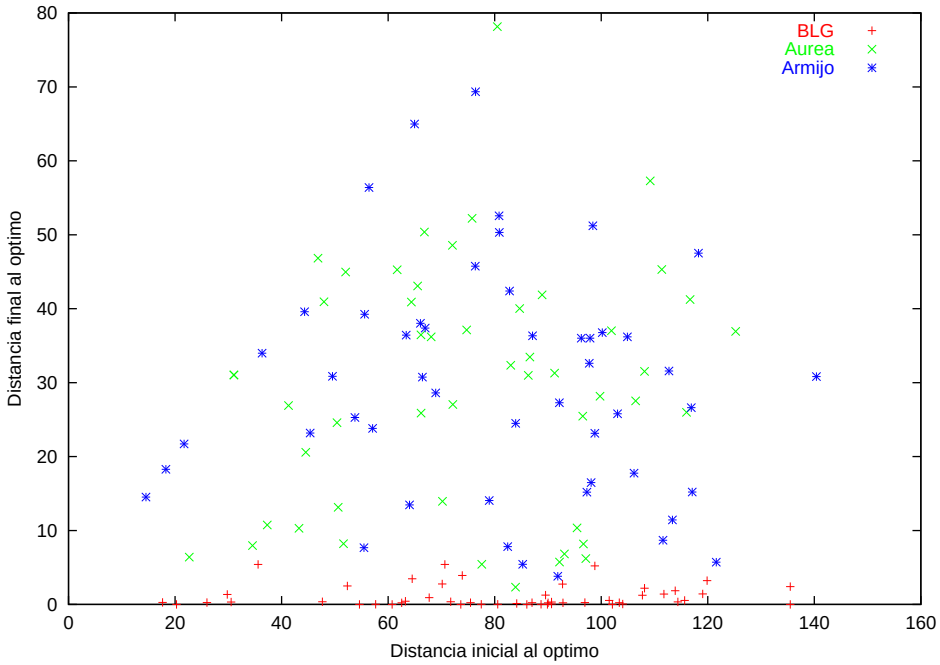


Figura 4.14: Relación entre distancia inicial y distancia final al óptimo

obtenido para cien ejecuciones desde otros tantos puntos iniciales escogidos aleatoriamente, el número de iteraciones medias realizadas, así como el tiempo medio de las ejecuciones. Para cada una de ellas, se indica los parámetros empleados en la BLG.

Función	Tam. Pob.	Gen.	Prob. Cruce	Iter.	Tiempo	Aciertos
BR	500	1000	0.4	5.8	0.1132	100 %
RA	250	750	0.4	5.8	0.1941	100 %
JC	100	500	0.5	9.7	0.1813	100 %
CH	200	500	0.5	11.6	0.0779	100 %

Tabla 4.4: Resumen de resultados de la BLG

A la vista de las pruebas computacionales realizadas podemos ver que la BLG apoyada de un método de descenso (derivativo o no) permite realizar la optimización de funciones multiextrema, es decir, funciones que poseen muchos mínimos locales de manera eficiente.

4.4. Aplicación a las redes neuronales

Hasta el momento, el objeto principal de la presente memoria ha consistido en estudiar las posibilidades que ofrece la BLG desarrollada en la resolución de funciones de optimización de la bibliografía. Esto es fácilmente justificable en cuanto a que muchos de los problemas clásicos de la ingeniería admiten fácilmente un planteamiento como problemas de búsqueda, por lo que es del máximo interés disponer de técnicas lo más generales posible para resolverlos.

Sin embargo, existen problemas que no pueden plantearse como una búsqueda de un óptimo, sino que requieren realizar una clasificación de elementos de acuerdo a una serie de propiedades. Los enfoques más utilizados para resolver los problemas así planteados son los basados en redes neuronales.

Las redes neuronales [62, 63, 86] son estructuras adaptativas de procesamiento de información inspiradas en la estructura cerebral, donde el procesamiento se lleva a cabo mediante la interconexión de elementos de proceso muy sencillos a los que se denominan *neuronas*. Esta arquitectura da lugar a estructuras altamente paralelizables donde el flujo de información no sigue un camino secuencial, sino que se distribuye a través de las conexiones de los elementos de proceso donde la información es tratada.

Una de las arquitecturas más populares es la red multicapa con propagación hacia atrás. El entrenamiento de estas redes consiste en minimizar el error cuadrático medio de la función de los pesos que interconectan las distintas neuronas. Este problema puede ser considerado como la resolución de un proceso de optimización sin restricciones. La regla de entrenamiento original para este tipo de redes es la denominada *regla delta generalizada* (4.19) que utiliza el gradiente negativo de la función de coste como dirección de descenso, d^t y un paso de tamaño fijo, α, denominado *tasa de aprendizaje*. De este modo, el nuevo conjunto de pesos, W^{t+1}, vendrá dado por la siguiente fórmula:

$$W^{t+1} = W^t + \alpha d^t \tag{4.19}$$

Existen otras propuestas que no emplean el gradiente puro como dirección de descenso. Así, Rumelhart y col. [137] emplea el término momento para suavizar las direcciones de descenso consecutivas. También se han empleado métodos cuasi-Newton o de gradientes conjugados [7, 27, 76, 123]. Sin embargo, el cálculo de la tasa de aprendizaje se ha mantenido fijo en las distintas alternativas.

Los AGs han sido empleados para el entrenamiento de las redes neuronales multicapas con realimentación. La solución habitual considera una población donde cada individuo codifica todos los pesos de las conexiones, es decir, representa una solución completa [107]. También han sido

empleados los AGs para determinar la configuración óptima de capas y neuronas [138], así como el espacio de representación interna de la misma [59].

Nuestra propuesta consiste en aplicar la BLG en el proceso de entrenamiento de la red neuronal. Es decir, realizar el cálculo de la tasa de aprendizaje en cada iteración, con objeto de mejorar el proceso de aprendizaje. Matemáticamente, el problema a resolver se puede plantear como:

$$\alpha = \text{mín}\{W^t + \alpha d^t\} \tag{4.20}$$

Para comprobar la efectividad del entrenamiento de una red neuronal con la BLG se utiliza un problema de clasificación. Éste consiste en clasificar muestras de aceite de oliva de acuerdo al contenido en cincuenta ácidos grasos obtenidos mediante un análisis químico de las muestras. Se establecen cinco clases basadas en el porcentaje de los mayores cuatro ácidos grasos presente en el mismo. De estos cuatros ácidos, dos son saturados (Palmítico y Esteárico) y los otros dos son no saturados (Oléico y Linoléico). Disponemos de 184 muestras etiquetas[4], distribuidas según la tabla 4.5. Dado que disponemos de muchas muestras únicamente en una de las clases, se realiza una selección similar en cuanto al número de muestras a emplear en la fase de entrenamiento. Podemos observar que las muestras de Palmítico y Linoléico sólo poseen un único patrón para la fase de prueba de la red.

Clase de aceite	Entrenamiento	Prueba	Total
Normal (N)	10	132	142
Esteárico (E)	10	14	24
Oléico (O)	4	1	5
Palmítico (P)	7	1	8
Linoléico (L)	4	1	5

Tabla 4.5: Datos de entrenamiento y prueba de tipos de aceite

Para la resolución de este problema de clasificación se emplea una red con una única capa oculta que posee tres neuronas. La figura 4.15 muestra el esquema de la misma. La elección de esta configuración fue realizada de forma experimental, resolviendo el problema con diferentes configuraciones hasta encontrar el menor número de neuronas en la capa oculta que permitía aprender el conjunto de datos de entrenamiento.

La red neuronal fue entrenada usando la regla delta generalizada, empleando el método del gradiente conjugado como dirección de descenso y

[4]Muestras cedidas por un laboratorio de aceites de la provincia de Sevilla, cuyo nombre desea mantener en el anonimato.

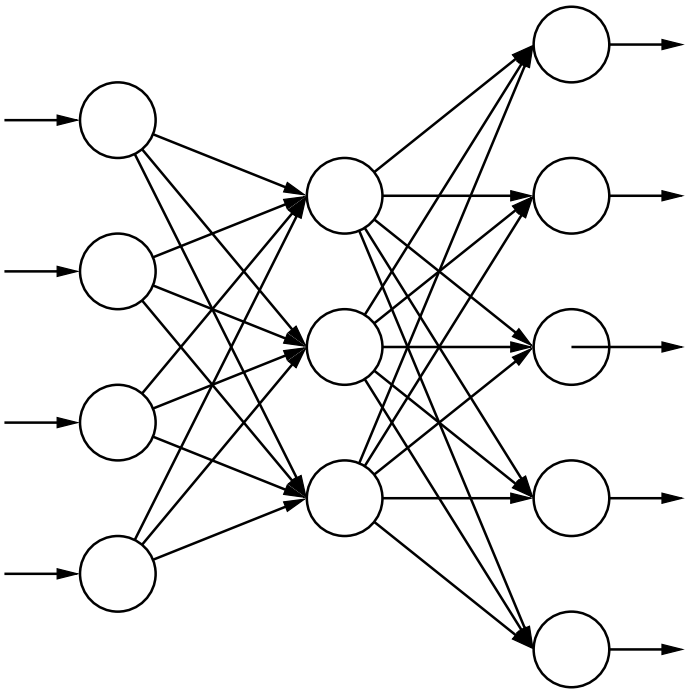


Figura 4.15: Arquitectura de la red neuronal

la BLG para determinar el valor óptimo del paso, α, en cada iteración del proceso de aprendizaje. Se configuró la BLG con un tamaño máximo de paso de 25, una población de 20, número de generaciones de 200, tamaño de nicho de 3 y una probabilidad de cruce de 0.7.

Para compararlo se realizó un entrenamiento siguiendo la misma dirección de descenso, pero aplicando una búsqueda lineal de sección áurea. La tabla 4.6 muestra la matriz de confusión obtenida con los datos de entrenamiento y los datos de prueba. Podemos observar que aunque se consiguen altos porcentajes para los datos de prueba, la clase Linoléica no consigue clasificarla. Esto es debido posiblemente a que se dispone de un conjunto de datos escaso en dicho patrón. Sin embargo, a pesar del alto porcentaje de éxito logrado con los datos de prueba para la clase Esteárica, con los datos reservado para la fase de prueba sólo se consigue un único acierto, es decir, un 0,071 % de tasa de éxito.

La tabla 4.7 muestra la matriz de confusión obtenida durante la fase de entrenamiento y la fase de prueba con la BLG. Podemos ver que la clase Oléico no tiene ninguna confusión, tanto en los datos de entrenamiento como en los datos de prueba. Igualmente, las clases Palmítico y Linoléico, aunque con el entrenamiento no se alcanza el 100 %, el único dato disponible para la fase de pruebas es catalogado correctamente. En el resto de

clases se consigue un alto porcentaje de equivalencias, salvo en la clase Esteárico donde se consigue un porcentaje inferior al 50 %, aún así muy superior al obtenido con la otra red neuronal.

DATOS DE ENTRENAMIENTO							DATOS DE PRUEBA					
Clase	**N**	**E**	**O**	**P**	**L**	**Éxito**	**N**	**E**	**O**	**P**	**L**	**Éxito**
N	**9**	0	0	1	0	0.9	**101**	4	0	27	0	0.765
E	3	**7**	0	0	0	0.7	9	**1**	0	4	0	0.071
O	0	0	**4**	0	0	1.0	0	0	**1**	0	0	**1.0**
P	0	0	0	**7**	1	1.0	0	0	0	**1**	0	**1.0**
L	0	0	0	4	**0**	0.0	0	0	0	1	**0**	0.0

Tabla 4.6: Matriz de confusión para un paso fijo

DATOS DE ENTRENAMIENTO							DATOS DE PRUEBA					
Clase	**N**	**E**	**O**	**P**	**L**	**Éxito**	**N**	**E**	**O**	**P**	**L**	**Éxito**
N	**9**	0	0	1	0	0.9	**106**	3	0	23	0	0.8
E	3	**6**	0	1	0	0.6	6	**6**	0	2	0	0.43
O	0	0	**4**	0	0	1.0	0	0	**1**	0	0	**1.0**
P	0	0	0	**6**	1	0.86	0	0	0	**1**	0	**1.0**
L	0	0	0	2	**2**	0.5	0	0	0	0	**1**	**1.0**

Tabla 4.7: Matriz de confusión para la BLG

4.5. Conclusiones

Las pruebas computacionales realizadas con otros algoritmos de búsqueda lineal en el apartado anterior han permitido extraer las siguientes conclusiones sobre la BLG desarrollada dentro de esta tesis:

- Es un método robusto que consigue llegar al óptimo con independencia del punto inicial escogido.

- Presenta pocos requisitos de almacenamiento, dado que el AG sobre el que se desarrolla es unidimensional, lo que permite adaptarse rápidamente a problemas con un gran número de variables, sin que los requisitos de memoria supongan un problema en su utilización. Un ejemplo de su adaptación lo podemos comprobar en el aprendizaje de las redes neuronales.

- Realización de pocas iteraciones en el proceso de optimización, ya que el realizar una búsqueda local extendida permite generar pasos más grandes, avanzado más rápidamente en cada iteración hacia el óptimo.
- Independencia del método empleado para determinar la dirección de descenso, debido a que se explora una dirección en un gran rango de valores, permitiendo descubrir rápidamente otros óptimos más lejanos.
- Posibilidad de aplicarlo a cualquier tipo de función. En el caso de necesitar funciones que no sean derivables, se utilizaría dentro de un método de descenso coordenado.
- Requiere la realización de pocas iteraciones para llegar al óptimo, y emplea poco tiempo en el proceso completo.

Capítulo 5

Búsqueda genética en cajas

En este capítulo se presenta una nueva estrategia de resolución de problemas de optimización global denominada Búsqueda Genética en Cajas, desarrollada en el curso de esta tesis. Este nuevo método es una estrategia de resolución de sucesivos problemas acotados, centrados en torno a óptimos locales, mediante un AG multidimensional. Una vez descrita la Búsqueda Genética en Cajas, se realizará un estudio sobre la influencia de los distintos parámetros de entrada del algoritmo. Finalmente, se describirán los experimentos realizados para determinar la eficacia de la Búsqueda Genética en Cajas.

5.1. Introducción

Holland [65, 66] fue uno de los primeros que empleó los AGs para la optimización de funciones con variables continuas. Sin embargo, existen pocos trabajos en la bibliografía que tratan con la aplicación de los AGs a la optimización global de funciones con variables continuas [6, 9, 24, 41, 55, 102, 113]. En esos trabajos, se han realizado pruebas a funciones multiextremas clásicas. Sin embargo, a pesar de que los resultados obtenidos por Berthiau y Siarry [9] eran esperanzadores, no eran del todo satisfactorios, debido principalmente a la simplicidad del AG diseñado y los operadores empleados.

Una de las características de los AGs es su posibilidad de explorar grandes superficies. Sin embargo, esta ventaja se puede convertir en un inconveniente debido a la posibilidad de que una pronta convergencia puede hacer que el algoritmo se centre en una solución medianamente buena. Aunque es posible aplicar técnicas en el algoritmo para evitar esta deriva genética, penalizando individuos cercanos, pero en optimizaciones donde el alcance de las variables es muy grande puede ser ineficiente.

Entendemos que la mejora de los resultados requiere diversificar al AG,

permitiendo cubrir un gran espacio de soluciones [112, 114], pero también intensificar la búsqueda en áreas prometedoras.

Nuestra propuesta, denominada Búsqueda Genética en Cajas (BGC), permite explorar grandes superficies, pero a su vez, centrarse en zonas específicas del espacio de la función objetivo, permitiendo búsquedas más selectivas. Se puede definir como una estrategia de resolución de sucesivos problemas acotados, centrados en torno a óptimos locales obtenidos mediante un AG multidimensional.

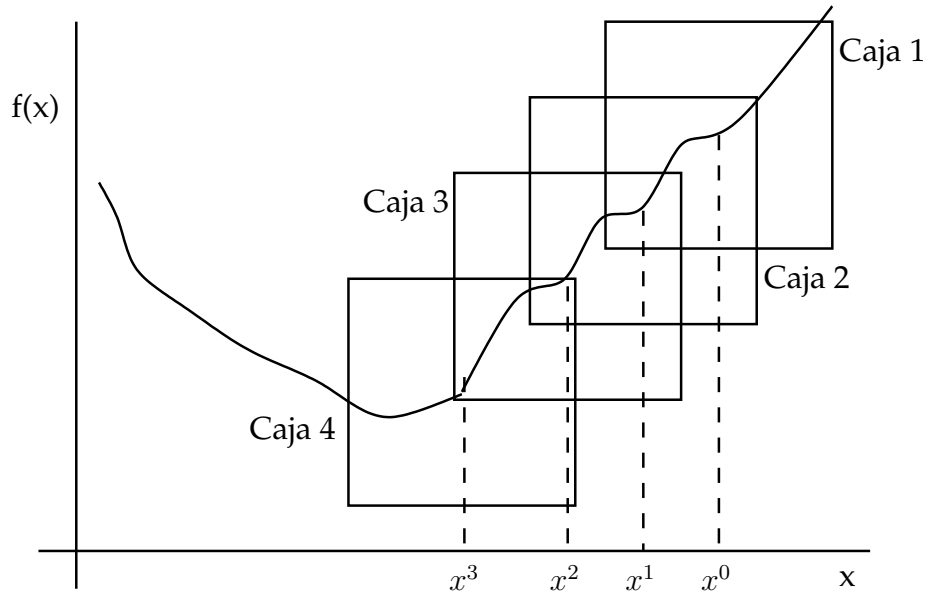


Figura 5.1: BGC

Para ello, se generan cajas n-dimensionales, donde el AG busca una solución óptima. La mejor solución encontrada se convertirá en el centro de la próxima caja, permitiendo de este modo avanzar por la superficie, pero centrándonos en pequeños espacios con objeto de aprovechar la potencia del AG. La generación de cajas sucesivas permitirá llegar al óptimo de la función. La figura 5.1 permite ver este proceso, donde cada óptimo encontrado en cada caja se va convirtiendo en el centro de la siguiente caja.

Con objeto de evitar caer en un óptimo local donde el AG no pueda escapar, el volumen de la caja se irá aumentando sucesivamente mientras no se consiga mejorar. De este modo, se podrán tener cajas lo suficientemente grandes como para poder salir del valle donde se encuentra el óptimo local, y conseguir alcanzar uno mejor.

Para no tener una rápida convergencia a un óptimo local, se dota al AG de una memoria, de forma que vaya recordando los distintos óptimos locales que ha encontrado en el proceso de optimización. Esto le permite

realizar una penalización de los individuos cercanos a ellos, permitiendo la diversificación de la población del AG, con objeto de llegar al óptimo global, y no converger a un óptimo local.

El concepto de memoria introducido al AG es una ampliación de la formación de nichos [55], que se emplea para incentivar la diversificación y evitar la convergencia prematura. Sin embargo, la diferencia existente entre los nichos y el operador de memoria que se ha diseñado es aumentar la diversificación tanto en los individuos que conforman la población actual, como también presentar la diversificación en relación a las mejores soluciones que se han ido generando en las iteraciones anteriores.

La estructura del capítulo es la siguiente: en el apartado 5.2 se detalla las características del método que hemos desarrollado bajo el nombre de BGC. El apartado 5.3 recoge los resultados obtenidos la BGC que hemos obtenido al aplicarla a un conjunto de funciones multiextremas, recogidas en el anexo A. Finalmente, el apartado 5.4 recoge las conclusiones extraídas de los resultados obtenidos con la resolución de la optimización de funciones multiextrema con la BGC.

5.2. La BGC

Para explorar cada una de las cajas se empleará un AG multidimensional, cuyas características principales se pueden resumir en las siguientes:

- Los individuos son vectores de la misma dimensión que la función a optimizar, cuyos componentes están codificados como números reales normalizados en el intervalo $[0, 1]$.
- Implementación de un AG generacional, donde en cada generación se conservará siempre un porcentaje de individuos de la población anterior.
- Sólo se emplean dos operadores, uno para cruce, y otro para mutación, que dará lugar a los nuevos individuos que aparecerán en cada generación.
- La función de aptitud de los individuos es la misma función que se desea optimizar.
- El tamaño de la población depende del volumen de la caja, que indica la extensión de la región a explorar.
- Con objeto de evitar una rápida convergencia del algoritmo se emplea una memoria de los últimos óptimos locales obtenidos.

5.2.1. Volumen de la caja

Asociada a la codificación de los individuos existen dos parámetros importantes del algoritmo, que van a determinar la extensión de la búsqueda genética a realizar. Éste es el volumen inicial de la caja, S_0, que será un valor suministrado por el usuario. De este modo, los límites de la caja vendrán dados por:

$$\begin{aligned} a_i^k &= x_i^k - \frac{S_0^{1/N}}{2} \\ b_i^k &= x_i^k + \frac{S_0^{1/N}}{2} \end{aligned} \tag{5.1}$$

donde a_i^k y b_i^k son los componentes i-ésimos de los límites de la caja en la generación k-ésima, N es la dimensión del problema a optimizar, x_i^k la componente i-ésima del centro de la caja k-ésima y S_0 es el volumen inicial de la caja.

Aunque la BGC se ha presentado como una técnica de optimización de funciones sin restricciones, es posible utilizarla cuando las variables presentan unas cotas inferiores y superiores. La existencia de éstas afectan únicamente a la caja donde se realizará la búsqueda genética. De este modo, en el establecimiento de los límites de la misma tendrá que tenerse en cuenta estas condiciones, de forma que si l_i y u_i son los límites inferiores y superiores del componente i-ésimo de la región estudiada, la caja deberá truncarse apropiadamente. Así, los nuevos límites de la caja vendrán dados por:

$$\begin{aligned} a_i &= \text{máx}\left\{ l_i, x_i - \frac{S^{1/N}}{2} \right\} \\ b_i &= \text{mín}\left\{ u_i, x_i + \frac{S^{1/N}}{2} \right\} \end{aligned} \tag{5.2}$$

Dado que el volumen de la caja S controla la región que tiene que ser optimizada, el tamaño de la población se realiza proporcional a ese parámetro. Así, a volúmenes grandes de cajas le deben corresponder tamaños de poblaciones grandes. De este modo, si S es el volumen de la caja a generar, el tamaño de la población vendrá dado por:

$$TamPob = \frac{S}{\sigma} \tag{5.3}$$

donde σ será un parámetro de entrada del algoritmo.

Igualmente, el número de generaciones debe ser proporcional al tamaño de la población, con objeto de que el AG disponga de suficiente tiempo

para poder explorar la caja y realizar una convergencia a un buen óptimo. De este modo, el número de generaciones G se puede expresar como:

$$G = \theta \cdot TamPob \tag{5.4}$$

donde θ será un parámetro de entrada del algoritmo.

5.2.1.1. Aumento del volumen de la caja

Existe la posibilidad que el volumen de caja seleccionado sea pequeño y no permita mejorar la solución inicial, o bien, centrar la búsqueda en un óptimo local. Para evitar estas convergencias a óptimos locales, si la mejor solución encontrada en la búsqueda dentro de la caja no es mejor que la solución actual, la caja dobla su volumen y se explora de nuevo. El número de veces que la caja puede aumentar su volumen viene dado por un parámetro adicional, A, suministrado por el usuario.

De este modo, el volumen máximo de la caja que se podrá explorar será una función del volumen de la caja original y el número de veces que ésta puede aumentar. De forma matemática:

$$S_{max} = S \cdot 2^A \tag{5.5}$$

5.2.2. Codificación de individuos

Los individuos en el AG representan distintos puntos dentro del espacio de búsqueda. Éstos están codificados mediante vectores de coma flotante normalizados dentro del intervalo $[0, 1]$. El individuo $[0{,}5, 0{,}5, \ldots, 0{,}5]$ representa al centro de la caja que se está explorando en ese momento. De este modo, si x^k es el vector con las coordenadas del centro de la caja, y los vectores a^k y b^k son los límites de la misma (5.2), un individuo en la población k-ésima, ind^k, se corresponde a la solución sol^k con valores:

$$sol_i^k = \begin{cases} x_i^k + 2(ind_i^k - 0{,}5)(b_i^k - x_i^k) & \text{si} \quad ind_i^k \geq 0{,}5 \\ x_i^k - 2(0{,}5 - ind_i^k)(x_i^k - a_i^k) & \text{si} \quad ind_i^k < 0{,}5 \end{cases} \tag{5.6}$$

donde ind_i^k es la componente i-ésima del vector que codifica el individuo ind^k, x_i^k es la componente i-ésima del centro de la caja x^k, a_i^k y b_i^k los componentes i-ésimos de los vectores de límites de la caja, respectivamente.

5.2.3. Función de aptitud

Para determinar la bondad de la solución representada por cada individuo de la población, la función de aptitud empleada vendrá dada según el

problema de optimización en cuestión. Dado que los AGs tratan de maximizar el valor de la función de aptitud de los distintos individuos a lo largo de las distintas generaciones:

- En el caso de tratarse de un problema de maximización, la función objetivo será la propia función de aptitud.

$$F(x) = f(x) \tag{5.7}$$

- En el caso de tratarse de un problema de minimización, la función de aptitud se define como la negación de la función objetivo:

$$F(x) = -f(x) \tag{5.8}$$

Dado que el AG requiere que los individuos presenten siempre un valor positivo en la función de aptitud, es necesario realizar una translación de esos valores, con objeto de asegurar dicha condición. Ésta se representa por la siguiente ecuación:

$$F(x^n) = F(x^n) + |\min_j \{F(x^j)\}| + 1 \qquad \text{si} \quad \min_j \{F(x^j)\} < 0 \tag{5.9}$$

5.2.4. Memoria en la BGC

La BGC puede presentar problemas de estancamiento en valles profundos, donde incluso el aumento del volumen de las cajas no consigue mejorar la solución. Una forma de evitar este estancamiento es penalizar la función aptitud. Para ello, el algoritmo recordará cuáles han sido sus últimos pasos, con el fin de penalizar a los individuos que vuelvan a zonas anteriores.

De este modo, hemos dotado de una memoria al AG con objeto de recordar los últimos L centros de las cajas donde se han realizado las búsquedas genéticas. Esta memoria es una característica novedosa en los AGs, puesto que en el proceso de deriva genética del funcionamiento de los AGs sólo se tiene en cuenta la información de la población actual. Con la dotación de una memoria, permitiremos que nuestro AG pueda acceder a la información de los mejores individuos en poblaciones anteriores.

La incorporación de la memoria al AG es una técnica adaptada de la formación de nichos [55], que se emplea para incentivar la diversificación y evitar la convergencia prematura. La diferencia es que en lugar de penalizar la aptitud de cada individuo por la acumulación de individuos cercanos, ésta se realiza en base a la cercanía con los óptimos locales que se encuentran almacenados en la memoria.

De este modo, a cada individuo lo penalizaríamos en base a la distancia de cada uno de los datos almacenados en la memoria. De este modo, para un individuo de la generación k-ésima, ind, su función de aptitud sería:

$$F'(ind) = F(ind) + \beta \sum_{t=1}^{L} \text{mín} \left\{ d(ind, M^{L-t}), S^{1/n} \right\} \tag{5.10}$$

donde M^j representa el óptimo alcanzado hace j BGC. En relación a la distancia del individuo con los óptimos almacenados en memoria, es decir, el componente $d(ind, M^{L-t})$, se toma una métrica basada en el genotipo de los individuos, de forma que:

$$d(ind, M^{L-t}) = \|ind - M^{L-t}\| \tag{5.11}$$

Se puede ver que esa memoria sólo afectará cuando los óptimos se encuentran dentro de la caja que estamos explorando en cada momento.

El problema es determinar el valor de la constante β. Si es un valor pequeño no se evitarían los ciclos, y por lo tanto, no se conseguiría salir de los valles. Si es grande desvirtuaría la función original. Por otro lado, cuando nos encontremos en lo más profundo del valle y lejos de los individuos almacenados en la memoria, se cumple que:

$$\beta \sum_{t=1}^{L} \text{mín} \left\{ d(ind, M^{L-t}), S^{1/n} \right\} \longrightarrow \beta L S^{1/n} \tag{5.12}$$

Para evitar que la penalización de la memoria sea muy elevada y desvirtuar la función aptitud, nuestro parámetro β deberá ser de la forma:

$$\beta = \frac{D}{LS^{1/n}} \tag{5.13}$$

Nos quedaría establecer el valor de la constante D en la ecuación anterior. Para ello, nos fijamos en los valores de la función en los distintos puntos que tenemos almacenado en la memoria. Durante el proceso de optimización, los valores de la función objetivo tenderán a disminuir. Conforme nos acercamos a un óptimo global, esas diferencias se irán atenuando, de forma que las diferencias con el último elemento introducido en la memoria serán muy pequeñas. Sin embargo, cuando estamos lejos del óptimo, las diferencias existentes entre ellos serán más acentuadas.

De esta forma, cuando las diferencias entre los valores de la función del conjunto de centros de caja almacenado en la memoria es muy cercano a 0, podemos asegurar que estamos en un buen óptimo, es decir:

$$\sum_{t=1}^{L} \left\{ F(M^{L-t+1}) - F(M^{L-T}) \right\} \longrightarrow 0 \tag{5.14}$$

Por tanto, haciendo:

$$\beta = \frac{1}{LS^{1/n}} \sum_{t=1}^{L} \left\{ F(M^{L-t+1}) - F(M^{L-T}) \right\} \tag{5.15}$$

se consigue un valor adaptativo. Es más, el valor obtenido permite evitar penalizaciones grandes, debido a $\frac{1}{LS^{1/n}}$, y consigue reducirse cuando se encuentra cerca de un óptimo, gracias al segundo componente de β, el término $\sum_{t=1}^{L} \{F(M^{L-t+1}) - F(M^{L-T})\}$.

5.2.5. Operadores

Se emplearán únicamente dos operadores, uno para el cruce y otro para la mutación. La operación de cruce tendrá una probabilidad de realizarse $p_c \in [0, 1]$, mientras que la de mutación tendrá una probabilidad inversa a la de cruce, es decir, $p_m = 1 - p_c$.

5.2.5.1. Operador de cruce

El operador de cruce que se emplea es la *combinación lineal convexa*, también denominado *cruce aritmético* [102], de dos individuos seleccionados previamente. De esta forma, si p_1 y p_2 son los dos individuos seleccionados, denominados *padres*, los dos *hijos*, h_1 y h_2, que resultan son:

$$\begin{aligned} h_{1i} &= \alpha p_{1i} + (1-\alpha) p_{2i} \\ h_{2i} &= (1-\alpha) p_{1i} + \alpha p_{2i} \end{aligned} \tag{5.16}$$

donde $\alpha \in [0, 1]$ es escogido de forma aleatoria, p_{Ni} y h_{Ni} son las componentes i-ésimas del padre e hijo N, respectivamente.

Este tipo de operación, asegura que si cada componente i-ésima de los dos individuos padres seleccionados se encuentran normalizados en el intervalo $[0, 1]$, los dos nuevos hijos generados también se encuentran dentro del mismo intervalo.

5.2.5.2. Operador de mutación

Se emplea una mutación consistente en generar una perturbación gaussiana aleatoria alrededor del individuo seleccionado. Así, si ind es el individuo seleccionado, la mutación resultante, m, será:

$$m_i = ind_i + N(0, \delta) \tag{5.17}$$

donde m_i e ind_i son la componente i-ésima del mutante y el individuo seleccionado, respectivamente.

La magnitud de la perturbación que se realiza se controla a través de la desviación estándar δ, cuyo valor se mantiene constante en todo el proceso de optimización. Esta magnitud está relacionada con la probabilidad de mutación, de forma que:

$$\delta = (1 - p_m) \tag{5.18}$$

De este modo, si las mutaciones son muy frecuentes, el tamaño de la magnitud de éstas decrece, y viceversa.

Cuando una mutación tiene lugar, el nuevo individuo puede situarse fuera del intervalo de normalización $[0, 1]$, por lo que tras la operación es necesario realizar una reparación del individuo consistente en establecer su genotipo al valor más cercano a dicho intervalo. De forma matemática, si m es el individuo resultante tras la mutación, los componentes del individuo que se genera al realizar la translación, m_{ti} serán:

$$\begin{aligned} m_{ti} &= 0 \quad \text{si} \quad m_i < 0 \\ m_{ti} &= 1 \quad \text{si} \quad m_i > 1 \\ m_{ti} &= m_i \quad \text{si} \quad 0 \leq m_i \leq 1 \end{aligned} \tag{5.19}$$

5.2.5.3. Selección de individuos

En la realización de las operaciones de cruce y mutación es necesario realizar una selección de individuos. Para ello se realiza una selección con probabilidad proporcional a su aptitud, empleándose en concreto el método de la ruleta [55].

5.2.6. Generaciones

El AG empleado será uno de tipo generacional donde cada generación se crea en tres pasos. El primero consiste en hacer pasar al mejor individuo de la población a la siguiente. Es decir, se emplea *elitismo*. Dado que la población inicial de cada búsqueda en una caja incluye el centro de la misma, y puesto que se emplea elitismo, nos encontramos que la mejor solución que se puede encontrar en las sucesivas generaciones de la BGC no es peor que el centro de la misma, que es la mejor solución encontrada en la búsqueda genética anterior. Esto conlleva que se produce una mejora monótona en el proceso de optimización.

En segundo lugar, un porcentaje de individuos de la población previa se seleccionan y se copian a la nueva población. Este porcentaje es un parámetro introducido por el usuario. La selección de individuos se hace mediante una probabilidad proporcional a su aptitud (apartado 5.2.5.3).

Finalmente, el resto de individuos necesarios para completar la nueva población es generada a partir de cruces y mutaciones aplicadas a los individuos de la población previa. Así, para el cruce son seleccionados dos individuos padres, introduciéndose los dos hijos resultantes. En el caso del operador de mutación se selecciona un individuo y su mutación se introduce en la nueva población. La selección de los individuos implicados en

las operaciones se realiza de forma proporcional a su aptitud, tal como se comentó en el apartado 5.2.5.3.

Algoritmo 5.1 BGC

BGC : $T_P \times p_c \times p_r \times G \times S \times A \times x^0 \times L \times \epsilon \longrightarrow x^*$
$k \longleftarrow 0$
$x^k \longleftarrow x^0$
$a \longleftarrow 0$
Crear_Memoria(L)
$x^{k+1} \longleftarrow$ AG-BGC$(T_P, p_c, p_r, G, S, A, x^k)$
Actualizar_Memoria(x^{k+1})
$optimo \longleftarrow$ LOCAL
Mientras $(optimo =$ LOCAL$)$
  Si $\left(\frac{|f(x^{k+1}) - f(x^k)|}{|f(x^{k+1})|} \leq \epsilon\right)$
    $k \longleftarrow k + 1$
    $a \longleftarrow 0$
    $Caja \longleftarrow S$
    $x^{k+1} \longleftarrow$ AG-BGC$(T_P, p_c, p_r, G, Caja, x^k)$
    Actualizar_Memoria(x^{k+1})
  Si no Si $(a < A)$ **entonces**
    $a \longleftarrow a + 1$
    $Caja \longleftarrow 2Caja$
    $x^{k+1} \longleftarrow$ AG-BGC$(T_P, p_c, p_r, G, Caja, x^k)$
  Si no
    $optimo \longleftarrow$ GLOBAL
Devolver x^k

5.2.7. Proceso de optimización

El algoritmo 5.1 presenta un pseudocódigo de la BGC. El AG que realiza la búsqueda en las distintas cajas se muestra en el algoritmo 5.2. Los parámetros de entrada que necesita para su funcionamiento son los siguientes:

- Tamaño de la población: T_p, que especificará el valor σ de (5.3).
- Probabilidad de cruce: p_c.
- Número de individuos de la población previa: p_r.
- Número de generaciones: G, que especificará el valor θ de (5.4).

- Volumen de la caja: S.
- Número de aumentos de la caja: A.
- Tamaño de la memoria: L.
- Punto central de la primera caja donde realizar la búsqueda: x^0.

Algoritmo 5.2 Algoritmo genético de la BGC

```
AG-BGC : T_P × p_c × p_r × G × S × x^k ⟶ x
t ⟵ 0
P_t ⟵ Crear_Poblacion_Inicial(T_P, S, x^k)
Evaluar_Aptitud()
Aplicar_Memoria()
Mientras (t < G)
    t ⟵ t + 1
    ind ⟵ Mejor_Individuo(P_{t-1})
    Introducir_Individuo(P_t, ind)
    Mientras (|P_t| < p_r)
        ind ⟵ Seleccionar_Individuo(P_{t-1})
        Introducir_Individuo(P_t, ind)
    Mientras (|P_t| < T_P)
        op ⟵ Escoger_Operacion(p_m, p_c)
        Si (op = CRUCE) entonces
            [padre_1, padre_2] ⟵ Seleccionar_Padres(P_{t-1})
            [hijo_1, hijo_2] ⟵ Cruzar(padre_1, padre_2)
            Introducir_Individuo(P_t, hijo_1, hijo_2)
        Si no
            ind ⟵ Seleccionar_Individuo(P_{t-1})
            mutante ⟵ Mutar(ind)
            Introducir_Individuo(P_t, mutante)
    Evaluar_Aptitud()
    Aplicar_Memoria()
Evaluar_Aptitud()
ind ⟵ Mejor_Individuo()
Devolver ind
```

Podemos ver que la BGC realiza un proceso de mejora continuado en la solución, donde la mejor solución encontrada en la caja mediante el AG se convierte en el centro de una nueva caja. El AG que explora una caja realiza una generación en tres pasos. En el primero, se efectúa el elitismo

seleccionando al mejor individuo de la generación previa, lo que garantiza la mejora continua de la solución. El segundo paso consiste en copiar una serie de individuos de la población anterior a la nueva. Finalmente, se realizan las operaciones de cruce y mutación para obtener nuevos individuos.

Con objeto de acelerar el proceso de optimización global, también se puede introducir un parámetro más que es el número de generaciones consecutivas sin mejorar. De este modo, cuando se realicen dichas generaciones y el mejor individuo de la población no haya variado, el algoritmo finalizará. Este parámetro nos sirve para detectar de una forma rápida cuando el algoritmo ha alcanzado el óptimo en la búsqueda local realizada.

5.3. Resultados con la BGC

Las pruebas efectuadas con la BGC han sido realizadas en dos fases. En primer lugar, se ha determinado cómo influyen los distintos parámetros de entrada del AG en los resultados obtenidos. Una vez fijados los parámetros, se ha procedido a realizar pruebas computacionales con diversas funciones de la bibliografía.

5.3.1. Influencia de los parámetros de entrada

Dado la existencia de un conjunto de parámetros de entrada de la BGC, es necesario comprender la influencia de éstos en el funcionamiento del algoritmo, con objeto de determinar los valores más adecuados.

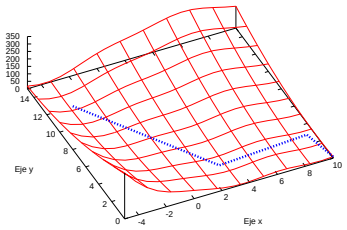
5.3.1.1. Aleatoriedad

Una de las características de la BGC es su carácter aleatorio. Esto hace que a pesar de ajustar los parámetros del algoritmo, para un mismo punto de partida y un juego de parámetros idénticos es posible conseguir distintas ejecuciones.

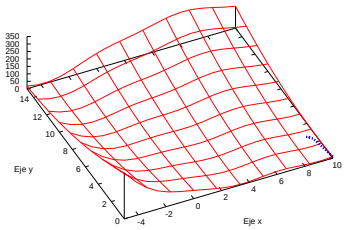
Así, la figura 5.2 muestra tres ejecuciones que se han obtenido en la optimización de la función de Branin (A.1), partiendo del punto (10, 0). Las rutas que se originan consisten en la unión de los sucesivos centros de caja por los que ha ido evolucionando el algoritmo. Los parámetros que se han empleado son los siguientes:

- Volumen inicial de la caja: 100.
- Tamaño de la población: 50 individuos.
- Número de generaciones: 250.
- Probabilidad de cruce: 0.5.
- Porcentaje de individuos repetidores: 20 %.

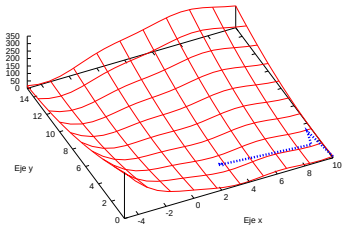
- Número máximo de aumentos del volumen de la caja: 4.



(a) Ruta 1



(b) Ruta 2



(c) Ruta 3

Figura 5.2: Rutas realizadas en la optimización de la función de Branin

En este caso, conviene resaltar que el carácter aleatorio de la BGC permite encontrar diversos óptimos globales. En nuestro caso, la función de Branin posee tres óptimos (puntos $(-3{,}14159, 12{,}275000)$, $(9{,}42478, 2{,}27500)$ y $(3{,}14159, 2{,}27500)$), alcanzándose los tres en otras tantas ejecuciones partiendo del mismo punto inicial.

5.3.1.2. Volumen de la caja

Un parámetro importante del algoritmo es el volumen de la caja, ya que éste determina la región de la zona a explorar en cada iteración. Si es pequeño puede quedar atrapado en óptimos locales. Por otro lado, otros parámetros del algoritmo, como son el tamaño de la población y el número de generaciones, dependen del volumen de la caja, según las ecuaciones (5.3) y (5.4).

Las figuras 5.3 y 5.4 muestran la evolución que sufre el valor de la función de Griewank (A.3) partiendo desde el punto $(100, 100)$ en relación al número de iteraciones y al tiempo empleado. Para ello, se han utilizado

diversos volúmenes de cajas. Se muestran los resultados obtenidos para 10, 50 y 100. Podemos ver que para un volumen de caja pequeño (10) las ejecuciones son más rápidas (del orden de 0.02 segundos), mientras que para cajas de mayor volumen (100), las ejecuciones suelen estar sobre los 30 segundos. Por contra, las ejecuciones con volúmenes de caja más grande, consiguen mejores aproximaciones al óptimo (cercano al 0.001), mientras que las BGC con un volumen pequeño pueden no alcanzar la precisión de 0.01.

Los parámetros de entrada de la BGC se ha mantenido igual en ambos casos; así, se ha utilizado una probabilidad de cruce de 0.6, un número máximo de 10 aumentos, y en cada generación se ha mantenido el 30 % de individuos de la anterior. Del mismo modo, la relación existente entre el tamaño de la población, el número de generaciones y el volumen de la caja se ha mantenido igual en ambos casos, de forma que:

$$TamPob = \frac{S}{2}$$
$$G = 5 \cdot TamPob$$

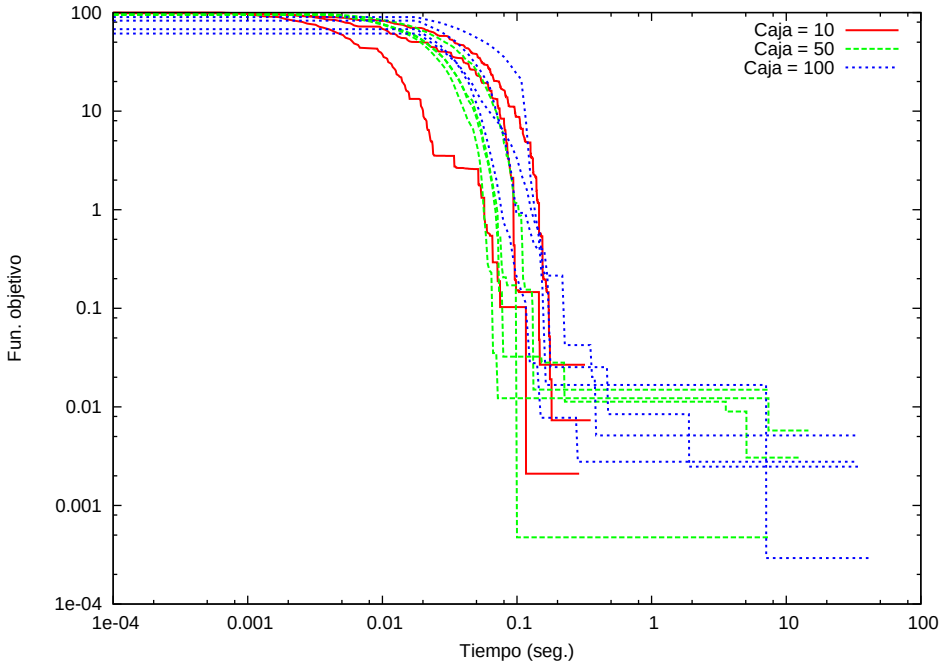


Figura 5.3: Evolución de la BGC frente al tiempo con diversos volúmenes de cajas

En la figura 5.4 se puede observar que un mayor volumen de la caja

requiere un menor número de iteraciones. Esto se debe a que la ampliación de la extensión de búsqueda en cada iteración permite avanzar más rápidamente hacia el óptimo, por lo que se requiere menos búsquedas en cajas. Sin embargo, al ser una superficie de búsqueda mayor requiere una mayor población para poder realizar una exploración con éxito de la misma, lo que implica un mayor tiempo (véase figura 5.3). Así, un volumen de caja de 10 requiere una media inferior a 0.02 segundos, mientras que con el volumen de 100 suele situarse el tiempo medio en 30 segundos.

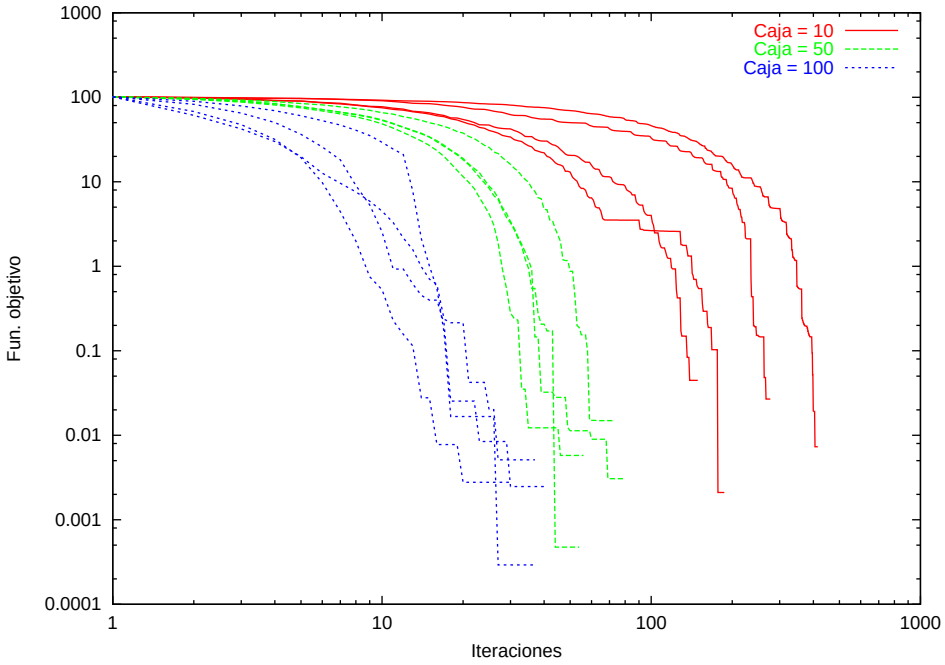


Figura 5.4: Evolución de la BGC frente al número de iteraciones con diversos volúmenes de cajas

Por otro lado, el volumen de la caja juega un papel importante en la calidad de la solución obtenida. Si el volumen de la caja que se utiliza es pequeño, existe una mayor probabilidad de caer en un óptimo local y no alcanzar el óptimo global. Esto se puede apreciar en la figura 5.4, donde podemos ver cómo con una caja de pequeño volumen (10), las soluciones suelen tener uno o dos dígitos de precisión, mientras que cuando mayor es el volumen de la caja permite alcanzar mejores resultados (entre tres y cuatro dígitos de precisión).

5.3.1.3. Tamaño de la población

Si el volumen de la caja nos permite indicar el tamaño de la región a explorar en cada iteración, otro parámetro importante es precisamente el tamaño de la población. Aunque está relacionado con el volumen de la caja, según (5.3), podemos ver que si el tamaño de la población no es lo suficientemente válido para poder explorar con garantías la caja en cada iteración, no se consiguen buenos resultados.

Esto se puede observar en la figura 5.5. En ella se representa, para cincuenta puntos escogidos aleatoriamente dentro del espacio de búsqueda, la distancia del punto inicial respecto al óptimo frente a la distancia de la solución obtenida por la BGC respecto al óptimo para distintos tamaños de la población en el proceso de optimización de la función de Griewank de 2 variables. Podemos observar que con independencia de la distancia del punto inicial siempre estamos en la cercanía del óptimo. Así, para un tamaño de población de 50, podemos ver cómo los puntos iniciales oscilan entre una distancia de 20 a 140, mientras que la solución proporcionada por la BGC se encuentra a una distancia entre 0.1 y 0.01 del óptimo de la función.

En las pruebas computacionales de la figura se han empleado los siguientes parámetros:

- Volumen de caja: 10.
- Número de generaciones: cinco veces superior al tamaño de la población.
- Probabilidad de cruce: 0.6
- Porcentaje de individuos que se mantiene en cada generación: 30 %.
- Número de aumentos: 6.

La figura 5.5 permite ver cómo con un tamaño de población del doble del volumen de la caja (20) se consiguen mejores resultados, mientras que una población pequeña (el 50 % del volumen, es decir, 5) las soluciones alcanzadas son peores, quedándose en una ocasión bastante alejado del óptimo.

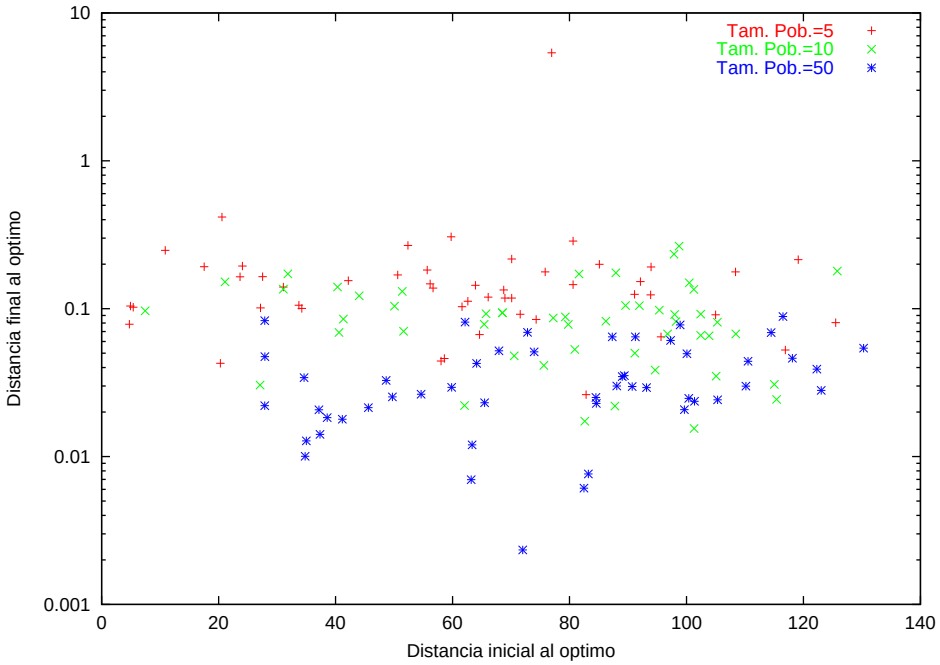


Figura 5.5: Relación entre el tamaño de la población y el porcentaje de éxito

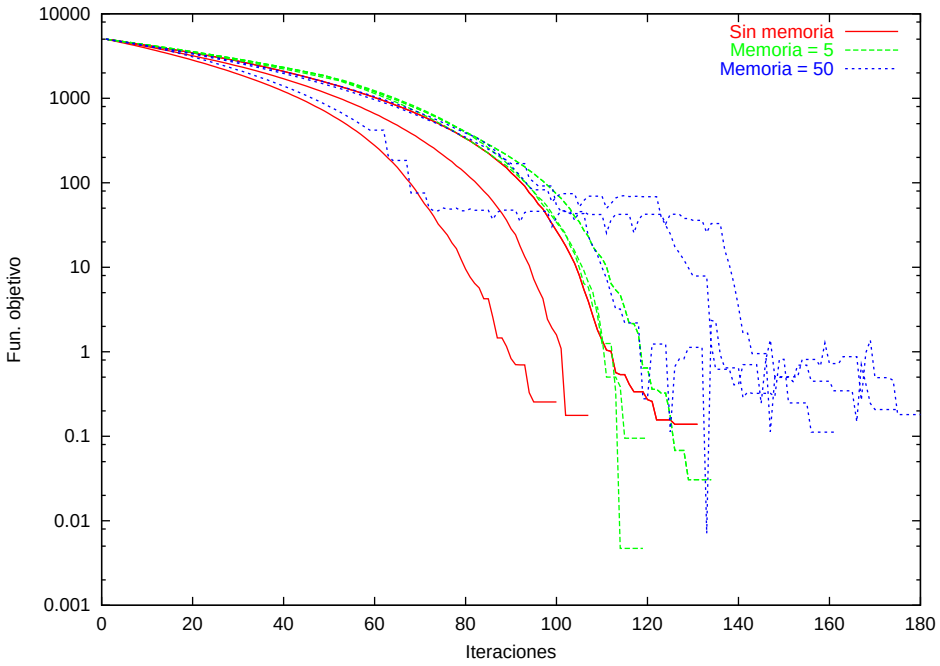


Figura 5.6: Influencia de la memoria en el nº de iteraciones

5.3.1.4. Memoria

Un aspecto importante en el funcionamiento de la BGC es precisamente la memoria. Este parámetro permite determinar cuántos centros de cajas de iteraciones anteriores se van a conservar con objeto de penalizar la distancia de los nuevos individuos a ellos. Este parámetro va a permitir escapar de óptimos locales sobre todo cuando el volumen de la caja no es lo suficientemente grande.

La figuras 5.6 y 5.7 permiten ver la evolución que sufre la función de Rastrigin (A.5) para distintos tamaño de memoria. Se muestra para cada uno de los tamaños de memoria tres ejecuciones desde el punto inicial $(50, 50)$. En ellas se aprecia cómo cuando disponemos de una memoria de 5 elementos los resultados que se obtienen son mejores, debido a que ésta permite escapar de óptimos locales. Sin embargo, destaca el hecho de que si la memoria que dispone el algoritmo es muy grande, en lugar de beneficiar al algoritmo se consigue un empeoramiento en su rendimiento.

Los parámetros empleados en esas pruebas computacionales han sido un volumen de caja de 2 con 4 aumentos posibles. El AG realiza 100 generaciones con 20 individuos, una probabilidad de cruce de 0.5 y el 15 % de los individuos se conservan entre las generaciones.

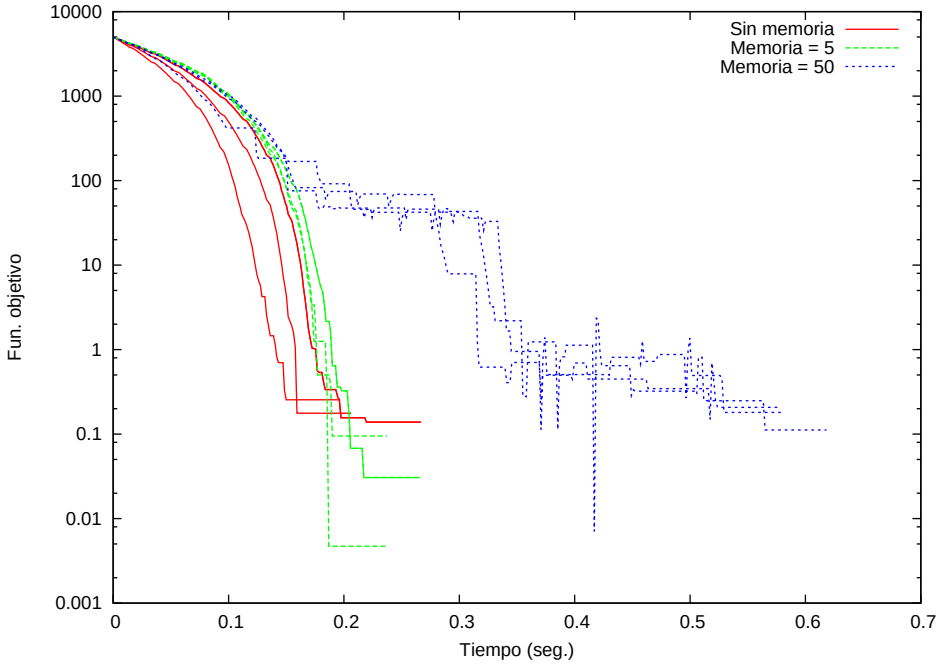


Figura 5.7: Influencia de la memoria en el tiempo

Un aspecto a destacar de la memoria de la BGC es su carácter adapta-

tivo, según (5.10) y (5.15). La figura 5.8 permite ver la evolución que sufre el parámetro β en cada iteración, mostrándose su valor frente a la distancia al óptimo en distintos tamaño de memoria. Se puede observar cómo inicialmente al estar la memoria vacía, el valor que posee dicho parámetro es pequeño, y conforme se va llenando la memoria empieza a aumentar, aportando una mayor influencia. Una vez que está la memoria llena, empieza a disminuir de valor, y conforme se acerca al óptimo su valor es cada vez más pequeño, con objeto de no desvirtuar la función aptitud con la penalización de la memoria. Destaca el hecho de que con una memoria muy grande (valor de 50) existen unas oscilaciones grandes de este parámetro en las cercanías al óptimo. Esto explica que un gran tamaño de la memoria desvirtue la función aptitud, provocando un empeoramiento del proceso de optimización, tal como se muestra en la figura 5.6.

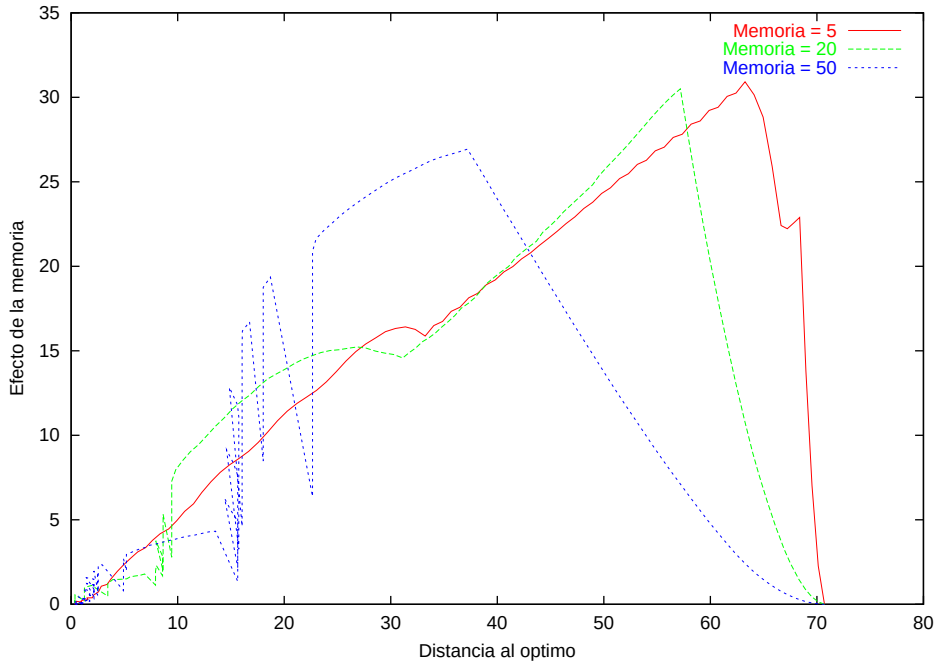


Figura 5.8: Evolución del parámetro β de influencia de la memoria según la distancia al óptimo

5.3.1.5. Elección del punto inicial

Una característica que presenta la BGC es que la elección del punto inicial no es un punto conflictivo en la convergencia del método. La figura 5.9 muestra, en el caso de la función de Beale (A.7), como el valor de la función objetivo final está muy cercano al óptimo (vale 0), independientemente de

la distancia inicial al mismo.

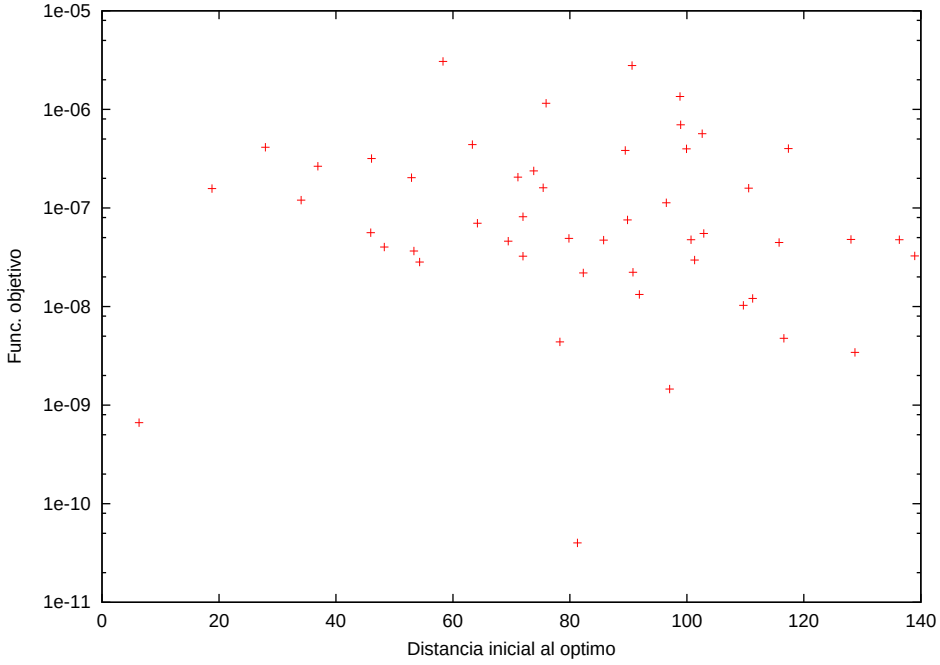


Figura 5.9: Relación entre distancia inicial y el valor de la función objetivo

5.3.2. Eficacia de la BGC

Las tablas 5.1, 5.2, 5.3 y 5.4 muestran para diversas funciones mostradas en el apéndice A el porcentaje de acierto[1] obtenido para cien ejecuciones desde otros tantos puntos iniciales escogidos aleatoriamente, el número de iteraciones medias realizadas, así como el tiempo medio de las ejecuciones. Para cada una de ellas, se indica los parámetros principales empleados en la BGC.

Podemos observar cómo se consigue un alto porcentaje en la consecución del óptimo global de la función, siendo en muchos casos el 100 % de éxito. Del mismo modo, a medida que disminuimos la probabilidad de cruce, y por tanto aumentamos la probabilidad de mutación, la convergencia del algoritmo es más rápida, pero suelen obtenerse peores resultados.

A la vista de las pruebas computacionales realizadas podemos ver que la BGC es un método eficaz para la realización de optimización en funciones multiextrema, es decir, funciones que poseen muchos mínimos locales.

[1]Se ha tomado como éxito cuando el error cometido es inferior a 0.001.

Caja	Tam. Pob.	Nº Gen.	Memoria	Iter.	Tiempo	Aciertos
$p_c = 0{,}7$						
10	50	250	10	45	1.50187	100 %
10	20	100	5	54	1.49568	100 %
5	50	250	10	31	0.71362	98 %
5	20	100	5	32	0.73722	99 %
$p_c = 0{,}5$						
10	50	250	10	24	1.09031	98 %
10	20	100	5	26	1.05304	99 %
5	50	250	10	22	0.42578	97 %
5	20	100	5	15	0.42289	95 %

Tabla 5.1: Resumen de resultados de la BGC para la función de Branin

Caja	Tam. Pob.	Nº Gen.	Memoria	Iter.	Tiempo	Aciertos
$p_c = 0{,}7$						
5	50	250	5	117	0.73300	100 %
5	50	250	10	127	0.83925	100 %
2	20	100	5	67	0.43185	100 %
2	20	100	10	75	0.51902	100 %
$p_c = 0{,}5$						
5	50	250	5	115	0.70108	99 %
5	50	250	10	114	0.87542	99 %
2	20	100	5	53	0.43610	99 %
2	20	100	10	64	0.51817	100 %

Tabla 5.2: Resumen de resultados de la BGC para la función de Rastrigin

5.4. Conclusiones

Conviene hacer algunas consideraciones sobre el método:

- El proceso elitista del algoritmo, así como que el centro de las sucesivas cajas sea el óptimo de la búsqueda anterior, produce una mejora monótona en el proceso de optimización.
- Si el volumen de la caja es lo suficientemente grande para contener múltiples valles, el método es capaz de saltar a través de los mismos, y llegar al óptimo global.
- El método se detendrá si para una búsqueda genética en una caja no encuentra una mejor solución que el propio centro de la caja. Esto

Caja	Tam. Pob.	Nº Gen.	Memoria	Iter.	Tiempo	Aciertos
$p_c = 0{,}7$						
2	20	100	10	31	0.65580	100 %
2	20	100	5	26	0.59680	100 %
10	50	250	10	45	1.13914	100 %
10	50	250	5	39	1.02668	98 %
$p_c = 0{,}5$						
2	20	100	10	33	0.65855	100 %
2	20	100	5	28	0.62485	99 %
10	50	250	10	39	1.06408	99 %
10	50	250	5	43	1.05234	100 %

Tabla 5.3: Resumen de resultados de la BGC para la función de Joroba de Camello

Caja	Tam. Pob.	Nº Gen.	Memoria	Iter.	Tiempo	Aciertos
$p_c = 0{,}7$						
5	30	150	10	32	1.37552	100 %
5	30	150	5	28	1.26395	98 %
15	20	250	10	52	2.04754	100 %
15	20	250	5	46	1.58200	99 %
$p_c = 0{,}5$						
5	30	150	10	33	1.27883	100 %
5	30	150	5	27	1.14780	100 %
15	20	250	10	51	1,98456	100 %
15	20	250	5	47	1.66945	99 %

Tabla 5.4: Resumen de resultados de la BGC para la función de Chichinadze

sucederá cuando se haya encontrado el óptimo global, pero también cuando el volumen de la caja no es lo suficientemente grande. Para diferenciar ambos casos, el volumen de la caja se aumenta el doble y se reinicia de nuevo la búsqueda. Si después de un cierto número de pasos duplicando el volumen de la caja, el método continúa sin avanzar, se presupone que se ha convergido al óptimo global. Por el contrario, si el centro de la caja era un óptimo local y se puede escapar, el método continúa en una nueva caja y se restaura el volumen de la caja a su valor original.

- El volumen máximo de la caja que se podrá explorar será una función

del volumen de la caja original y el número de veces que la caja puede aumentar (5.5).

Las pruebas computacionales realizadas han permitido obtener las siguientes características:

- Es un método robusto al conseguir llegar al óptimo con independencia del punto inicial escogido.
- Realización de pocas iteraciones en el proceso de optimización, aunque éstas dependen del volumen de la caja.
- Distinción entre óptimos locales y globales, encontrándose los primeros localizados como los centros de las sucesivas cajas de búsquedas, y el global es el último punto localizado en el proceso de optimización.
- Posibilidad de aplicarlo a cualquier tipo de función, ya que lo único que necesita para poder evaluar a los individuos del AG es la propia función, sin necesidad de emplear derivadas.

Capítulo 6

Estudios comparativos de la BLG y la BGC

En este capítulo se recogen diversas pruebas computacionales efectuadas con los métodos descritos en los capítulos previos. Para ello se utilizará un conjunto de funciones de la bibliografía empleadas en la validación de métodos de resolución global.

6.1. Introducción

En los capítulos 4 y 5 se ha llevado a cabo un análisis en profundidad acerca de los dos métodos elaborados como trabajo de la tesis, la BLG y la BGC, respectivamente. Sin embargo, el estudio se ha hecho de forma individualizada, aplicando los métodos a diferentes funciones y extrayendo las conclusiones oportunas en cada caso, sin establecer realmente una comparativa entre ellos que permita decidir acerca del método más idóneo para una aplicación concreta.

Con este objetivo, este capítulo muestra el estudio comparativo del rendimiento de ambos métodos con otras dos técnicas de resolución global estocásticas basadas en AG. La primera será el AG continuo presentado por [19], y la segunda un AG generacional puro que explore por completo toda la región a optimizar. En el apartado 6.2 se describen brevemente estos métodos. La tabla 6.1 recoge los cuatro métodos a emplear, con su correspondiente abreviatura.

Para la realización de esta comparación se emplearán las distintas funciones que aparecen en el apéndice A. La tabla 6.2 muestra las características de éstas y las abreviaturas empleadas.

Método	Siglas
BLG	BLG
BGC	BGC
AG Continuo	AGC
AG Generacional	AGG

Tabla 6.1: Métodos empleados en la comparación

Función	Dimensiones	Siglas
Branin	2	BR
Chichinadze	2	CH
Griewank	2	GR
Joroba de Camello	2	JC
Rastrigin	2	RA
Levy nº 5	2	LE
Beale	2	BE
Rosenbrock	10	RO^{10}
	50	RO^{50}
	100	RO^{100}
Zakharov	10	ZA^{10}
	50	ZA^{50}
	100	ZA^{100}

Tabla 6.2: Funciones empleadas en la comparación

6.2. Descripción de métodos

Para la comparación se utilizarán los métodos expuestos en la tabla 6.1. Dado que la BLG y la BGC han sido descritas en los capítulos 4 y 5, respectivamente, pasaremos a describir de forma breve los otros dos métodos empleados para su comparación. Destacar que para las pruebas computacionales se han tenido que desarrollar e implementar todos estos algoritmos empleados en la comparación.

6.2.1. AG Continuo

Este algoritmo, propuesto Chelouah y Siarry [19], consiste en un AG donde los individuos representan distintos puntos dentro del espacio de búsqueda. Éstos están codificados mediante vectores de coma flotante.

La característica principal de este método es la capacidad de adaptar al AG conceptos de la búsqueda tabú como son la diversificación y la intensificación.

Para ello, el algoritmo genera una población inicial lo más dispersa posible en la región de búsqueda. Es lo que se denomina la fase de diversificación. En este paso se proporciona como parámetro de entrada una distancia crítica, ϵ, de forma que si un individuo se encuentra cercano a otro (distancia inferior a ϵ) se descarta. De esta forma, la generación de una población inicial requiere de un mayor tiempo de computación, con objeto de que los individuos de la misma estén lo más repartido posible por todo el espacio de búsqueda.

Una vez que se obtiene la población inicial se realiza un AG generacional clásico con elitismo, de forma que siempre el mejor individuo de una nueva generación pasa a la siguiente.

Tras realizar el número máximo de generaciones se procede a la fase de intensificación. Ésta consiste en coger al mejor individuo y tomarlo como centro para una caja que se crea a su alrededor. Establecida ésta, se procede de nuevo a reiniciar el algoritmo por su fase de diversificación, pero el problema a optimizar sólo se resuelve en la región de la caja que se ha creado. Este proceso iterativo finaliza cuando la caja que se crea alrededor del mejor individuo de un AG es lo suficientemente pequeña.

Cada vez que se va generando una caja el tamaño de la población va disminuyendo. Esto se controla indicándose como parámetros de entrada el tamaño de la población inicial y final. Del mismo modo, hay que establecer el grado de reducción de la caja en cada iteración.

6.2.2. AG Generacional

Un método empleado para la comparación de las dos estrategias propuestas ha sido la utilización de un AG clásico de tipo generacional. Éste emplea una codificación de los individuos consistente en un vector de la misma dimensión que la función a optimizar, cuyos componentes están codificados como números reales dentro del intervalo a explorar. Se realiza un cruce aritmético igual al utilizado en la BGC (apartado 5.2.5.1), y una mutación consistente en alterar el valor de un individuo en cada una de sus componentes de forma similar a la BGC (apartado 5.2.5.2).

La característica de este algoritmo es que sus individuos pueden estar en toda la región a optimizar, y el número máximo de generaciones a realizar determinará la finalización del método.

El criterio de finalización del algoritmo es el número máximo de generaciones, o bien cuando lleve un número de generaciones consecutivas sin mejorar. Ambos son parámetros de entrada del algoritmo.

6.3. Comparativas

En primer lugar procederemos a comparar los distintos métodos en relación al número medio de iteraciones realizadas, tiempo medio empleado, así como el porcentaje de éxito alcanzado. En este caso, se considerará un éxito cuando se cumpla la siguiente inigualdad [19]:

$$|f(x_{OBT}) - f(x_{CON})| < \epsilon_{rel} f(x_{CON}) + \epsilon_{abs} \tag{6.1}$$

donde $\epsilon_{rel} = 10^{-4}$, $\epsilon_{abs} = 10^{-6}$, x_{OBT} es el óptimo encontrado en la aplicación del método de optimización, y x_{CON} es el óptimo conocido.

Los parámetros empleados en la BLG, la BGC y el AGG se muestran en las tablas 6.3, 6.4 y 6.5, respectivamente. En el caso del AGC se emplean los valores recomendados en [19], es decir, un tamaño de población inicial de 30, un tamaño de población final de 10, una probabilidad de cruce de 0.8 y una de mutación de 0.85, y unos parámetros de reducción de 2, para la creación de la caja a explorar, y 5, para la población.

Func.	Dirección	TamPob	Gen.	Paso	p_c	Nicho
BR	Máximo descenso	30	300	5	0.6	4
CH	Descenso coordenado	100	500	5	0.5	10
GR	BFGS	100	500	10	0.8	8
JC	BFGS	100	500	2	0.75	5
RA	BFGS	250	750	5	0.4	12
LE	Descenso coordenado	50	200	5	0.5	6
BE	Descenso coordenado	100	500	10	0.5	5
RO^{10}	BFGS	50	200	5	0.7	6
RO^{50}	Máximo descenso	75	300	5	0.6	3
RO^{100}	BFGS	100	400	5	0.5	2
ZA^{10}	Descenso coordenado	50	200	5	0.7	6
ZA^{50}	Descenso coordenado	50	200	5	0.5	8
ZA^{100}	Descenso coordenado	50	200	5	0.6	2

Tabla 6.3: Parámetros empleados en la BLG

La tabla 6.6 muestra para cada método y función empleada los resultados obtenidos para cien ejecuciones aleatorias, iniciando el algoritmo en un punto escogido de forma aleatoria por la región a optimizar. En el caso de la BLG se han realizado pruebas con diversos métodos de obtención de la dirección de búsqueda, como el descenso coordenado, el máximo descenso y el BFGS. De todas ellas, se muestra la que mejores resultados ha obtenido.

De la tabla 6.6 podemos apreciar cómo la ejecución de un AGG sin ninguna estrategia específica para realizar la optimización no consigue buenos

Func.	Tam. Pob.	Gen.	Caja	p_c	Rep.	Mem.
BR	50	250	2	0.7	0.15	5
CH	30	150	5	0.8	0.2	10
GR	20	150	2	0.7	0.1	5
JC	20	100	1	0.5	0.05	10
RA	20	100	2	0.65	0.1	10
LE	40	250	3	0.4	0.2	6
BE	50	250	5	0.5	0.35	8
RO^{10}	20	150	1	0.8	0.2	5
RO^{50}	25	200	1	0.5	0.25	8
RO^{100}	20	300	1	0.6	0.3	6
ZA^{10}	20	150	1	0.6	0.3	6
ZA^{50}	30	250	2	0.5	0.25	5
ZA^{100}	40	350	2	0.5	0.3	8

Tabla 6.4: Parámetros empleados en la BGC

resultados. Esto es debido sobre todo a que el AG pierde parte de sus ventajas cuanto más grande es la superficie a explorar, ya que la deriva genética existente evita que se pueda realizar un estudio más exhaustivo de la región a optimizar.

Por otro lado, la BLG y la BGC obtienen unos resultados de exactitud del 100 % en las funciones de dos dimensiones, salvo en la función de Levy, que se queda en torno al 95 %. Resultados similares se pueden observar en el AGC, pero ligeramente inferiores.

Es posible observar que en el caso de funciones de muchas dimensiones, como son la de Rosenbrock y la de Zakharov, se obtienen resultados superiores al 50 % en los métodos propuestos. Sin embargo, destacar que en el caso de esta última, el porcentaje de acierto ronda el 90 %.

Si se observa el número medio de iteraciones, es posible apreciar cómo la BLG realiza un menor número de iteraciones. Esto se debe a que explora en una dirección pudiendo avanzar rápidamente hacia el óptimo. Por contra, la BGC al realizar una optimización dentro de una caja e ir avanzando de caja en caja, su avance va ligado al tamaño de ésta. En el caso de las funciones de Rosenbrock y de Zakharov, el elevado número de iteraciones es debido precisamente a que el volumen de caja empleado es pequeño. Por este motivo, el número de iteraciones que necesita es mayor. Si observamos el AGC, su número de iteraciones es un valor intermedio entre la BLG y la BGC. Sin embargo, el tiempo necesario en el AGC tiende a ser ligeramente superior, debido a la creación de una población inicial con ciertas características (fase de diversificación).

Un dato que destaca de las pruebas computacionales realizadas es que

Func.	Tam. Pob.	Gen.	Gen. sin mejorar	p_c
BR	3000	10000	2000	0.6
CH	5000	10000	3000	0.4
GR	4000	10000	2000	0.5
JC	3000	10000	2000	0.6
RA	3000	10000	2000	0.8
LE	2500	10000	2000	0.2
BE	1000	10000	2000	0.3
RO^{10}	3000	15000	2000	0.7
RO^{50}	5000	15000	2000	0.5
RO^{100}	8000	15000	2000	0.3
ZA^{10}	8000	15000	2000	0.9
ZA^{50}	7500	15000	2000	0.75
ZA^{100}	6500	15000	2000	0.6

Tabla 6.5: Parámetros empleados en el AGG

la BLG requiere un tiempo siempre inferior a la BGC. Es más, el tiempo medio en la realización de una iteración es siempre menor en la BLG y la BGC. Esto es debido, principalmente, a la propia estructura del AG. Mientras que en la BLG es un simple algoritmo unidimensional, lo que permite que las operaciones del propio AG sean muy rápidas, mientras que en la BGC el AG emplea individuos que son vectores de coma flotante, por lo que al aumentar la dimensión de la función a optimizar, el tamaño de éstos aumenta. Esto implica que el algoritmo genético requiera un mayor tiempo en la realización de una generación.

Las figuras 6.1 y 6.2 representan el valor final de la función a optimizar frente al número de iteraciones y al tiempo necesario para alcanzar dicho valor, en cincuenta ejecuciones aleatorias, mediante el empleo de la BLG, la BGC y el AGC en el proceso de optimización de la función de Griewank. No se muestra el AGG, dado que su porcentaje de éxito es muy inferior a los otros métodos, que se encuentran más igualados. Podemos observar que la BLG requiere de menos iteraciones y menos tiempo que los otros métodos, que se encuentran más igualados. Sin embargo, destaca que aunque el porcentaje de éxito alcanzado, según (6.1), es el mismo, se puede observar cómo la BGC obtiene mejores aproximaciones al óptimo, dado que su nube de punto está más cercana al 0 que la del AGC. Del mismo modo, la BLG consigue alcanzar en tres ocasiones un mejor óptimo que la BGC y el AGC.

Función	BLG			BGC			AG			AGC		
	Éxitos	Iter.	Tiempo	Éxitos	Iter.	Tiempo	Éxitos	Iter.	Tiempo	Éxitos	Iter.	Tiempo
BR	100 %	12.48	0.14686	100 %	24.8	0.97731	55 %	5343.5	30.7976	100 %	23.3	1.13459
CH	100 %	4.26	0.38351	100 %	37.96	0.51743	50 %	3001.23	6.49276	99.9 %	47.96	0.72912
GR	100 %	15.22	0.51889	100 %	153.7	0.96628	57 %	6522.3	12.8746	100 %	125.32	1.06645
JC	100 %	6.86	0.11865	100 %	34.28	0.21390	47.23 %	4342.89	8.93553	100 %	30.76	0.63766
RA	100 %	10.46	0.82176	100 %	80.16	0.29123	29.3 %	8234.2	18.1017	100 %	32.45	0.84440
LE	95 %	15.96	0.89785	94 %	50.14	1.39140	12.9 %	4322.92	10.3689	87 %	35.66	0.96482
BE	100 %	7.62	0.73982	100 %	90.18	3.31502	65.2 %	7563.5	16.1687	100 %	46.87	1.31834
RO^{10}	64 %	29.04	3.39715	76 %	96.56	9.51174	15 %	8135.8	29.3794	60 %	69.3	7.38648
RO^{50}	60 %	30.28	3.82737	62 %	332.02	13.2433	5 %	10235.6	49.3139	57 %	156.8	12.1479
RO^{100}	53 %	35.62	5.95503	57 %	241.02	14.0925	4 %	9593.46	60.2495	55 %	128.87	14.2944
ZA^{10}	95 %	19.32	2.82185	93 %	119.52	9.62510	15 %	4573	29.45946	90 %	102.45	7.86683
ZA^{50}	90 %	25.42	2.49968	89 %	377.24	28.8508	13 %	5498.56	34.6446	85 %	206.74	43.2932
ZA^{100}	85 %	23.83	4.67155	89 %	459.03	32.6040	10 %	9473.61	45.5827	85 %	284.23	47.8589

Tabla 6.6: Resultados para 13 funciones y diversos métodos de resolución global

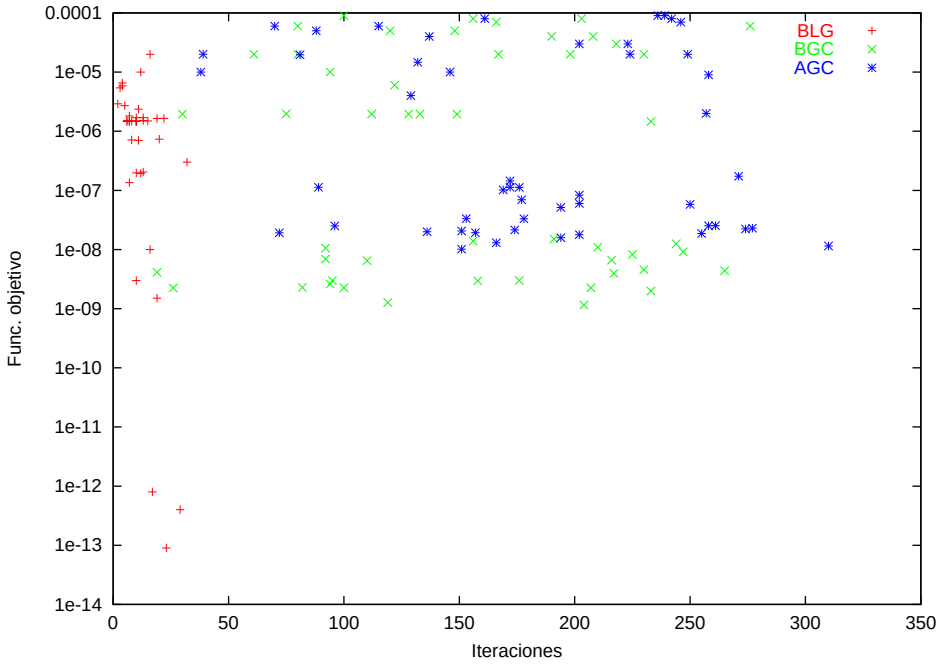


Figura 6.1: Números de iteraciones empleadas en la optimización de la función de Griewank

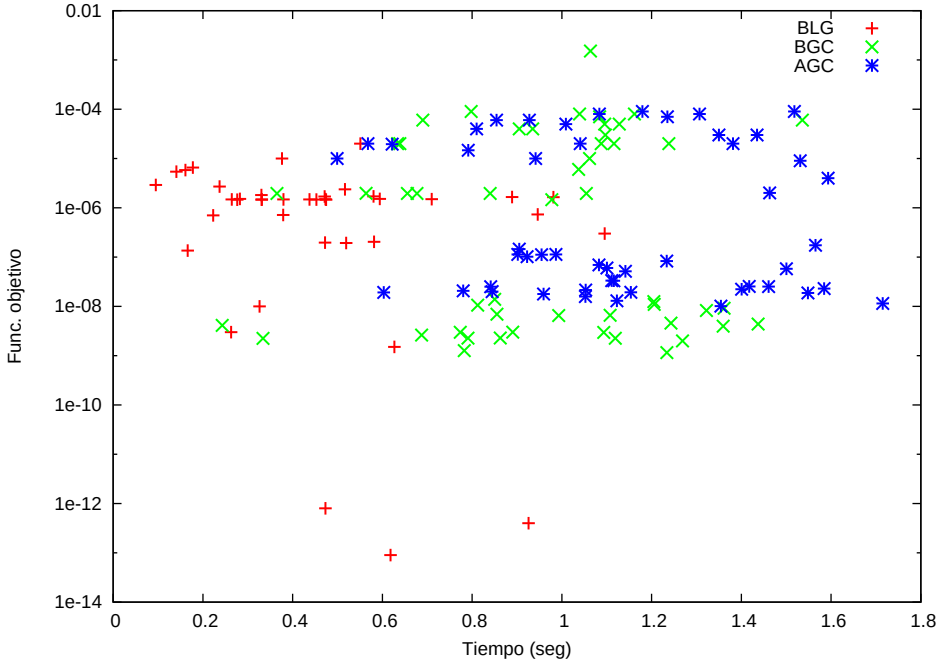


Figura 6.2: Tiempos empleados en la optimización de la función de Griewank

La tabla 6.7 muestra para cada método y función empleada el número de evaluaciones de la función y de la derivada para cien ejecuciones aleatorias, iniciando el algoritmo en un punto escogido de forma aleatoria por la región a optimizar.

Se puede apreciar cómo los métodos no derivativos, tales como la BGC, el AG y el AGC, no necesitan hacer uso de la derivada de la función. Sin embargo, la BLG requiere del uso de la derivada. En algunos casos, como en la función de Chichinadze, la de Levy, la de Beale o la de Zakharov, el hecho de que el número de evaluaciones de la derivada sea 0, es debido a que se está empleando un método de descenso coordenado. En el resto de las ocasiones se ha utilizado o bien el método de máximo descenso o el BFGS, indistintamente.

Destacar de los resultados mostrados que la BGC requiere como norma general un mayor número de evaluaciones de la función que la BLG. Esto es debido a que necesita realizar una exploración de sucesivas cajas por la región, lo que motiva que en cada una de ellas realice un número elevado de evaluaciones de la función. Este número se ve incrementado de forma considerable con la dimensión del problema. Así, de realizar 9022.32 evaluaciones en la función de Branin de 2 dimensiones, se pasa a realizar 397424.1 en el caso de la función de Zakharov de 100 dimensiones. Del mismo modo, la BLG destaca por su bajo número de evaluaciones de la función.

6.4. Conclusiones

A la vista de las pruebas computacionales realizadas se pueden extraer las siguientes conclusiones:

- La BLG y la BGC se presentan como dos métodos robustos para la resolución de problemas de optimización global.
- Son estrategias que permiten adaptarse, no sólo a funciones multiextremas, sino a funciones de muchas dimensiones.
- Posibilidad de aplicarlo a funciones no derivables. Mientras que la BGC sólo requiere del valor de la función, la BLG se puede enmarcar dentro de un método de descenso coordenado.
- Los tiempos de computación necesarios en ambos casos son escasos, y no sufren un incremento exponencial con el número de dimensiones de la función.
- Destacar que la BLG es precisamente el método que requiere menos tiempo en su ejecución y un menor número de iteraciones, consiguiendo resultados de gran exactitud.

Función	BLG		BGC		AG		AGC	
	Función	Derivada	Función	Derivada	Función	Derivada	Función	Derivada
BR	7925.68	24.96	9022.32	0	8457.06	0	8731.1	0
CH	8231.6	0	7784.68	0	8460.04	0	7034.9	0
GR	14792.6	30.44	16070.3	0	8459.82	0	12460	0
JC	2750.86	13.72	5012.42	0	8460.74	0	4236.12	0
RA	8530.82	20.92	6455.16	0	8461.46	0	8231.11	0
LE	21668.7	0	12426.4	0	8458.86	0	15232.6	0
BE	38026.7	0	28378.7	0	8460.42	0	30343.1	0
RO^{10}	22433.1	190.4	93719.2	0	72901.64	0	45562.23	0
RO^{50}	35666.4	1514	165617	0	72903	0	94637	0
RO^{100}	47345.6	1924	242321	0	73838	0	134936	0
ZA^{10}	39529.7	0	96237.7	0	92901.4	0	85671.2	0
ZA^{50}	197704	0	204954	0	92903	0	193462.4	0
ZA^{100}	208493	0	397424.1	0	99371.9	0	302373.1	0

Tabla 6.7: Número de evaluaciones de la función y derivada en cada método empleado

- Ambos métodos son independientes de la elección del punto inicial.
- Tanto la BLG como la BGC hacen uso de pocas iteraciones para alcanzar el óptimo.

Parte III

Optimización combinatoria

Capítulo 7

Búsqueda genética en vecindades

En este capítulo se propone una estrategia de optimización combinatoria, a la que hemos denominado Búsqueda Genética en Vecindades. Mostraremos que es una estrategia de resolución de sucesivos problemas acotados, centrados en torno a óptimos locales, mediante un AG. La característica principal del algoritmo desarrollado es que los individuos no codifican una solución, sino que construyen una solución a partir del óptimo local anterior. Una vez descrita la Búsqueda Genética en Vecindades, se realizará un estudio sobre la influencia de los distintos parámetros de entrada del algoritmo. Finalmente, se describirán los experimentos realizados para determinar la eficacia del método propuesto.

7.1. Introducción

Las resoluciones de problemas combinatorios, dada las restricciones que se suelen imponer a las variables, suelen presentarse como problemas NP-duros. En este sentido, suelen emplearse estrategias de optimización que buscan aproximaciones a la solución.

Entre estos métodos se encuentran los denominados de búsqueda local (véase el apartado 2.5), que se caracterizan por realizar un proceso iterativo de mejora de la solución mediante pequeñas búsquedas locales. Para ello, se define para cada solución $i \in F$, una *vecindad* como el conjunto de soluciones, $N(i) \subset F$, que son en cierto sentido *cercanas a* la solución i. El significado de la expresión *cercana a* i se entiende como que dicha solución puede ser alcanzada desde i en un único *movimiento*. Un movimiento es un operador que transforma una solución en otra mediante pequeñas modificaciones [155]. La caracterización de los movimientos que se pueden definir

dependen del problema a resolver.

Dentro de los métodos de búsqueda local se encuentran los denominados de *mejora continua* [1]. Éstos sustituyen la solución actual con aquella solución que mejora la función a optimizar después de explorar la vecindad completa.

La estrategia que se propone en este capítulo es un mecanismo híbrido, que utiliza el concepto de búsqueda local de mejora continua y hace uso de un AG para la exploración de las vecindades.

Dado que en este tipo de estrategia los movimientos que se pueden realizar dependen del problema a resolver, es necesario definir previamente el tipo de problema que se pretende resolver para determinar la validez del método desarrollado.

El problema del viajante es un problema clásico en la optimización combinatoria [84], y más en concreto en los métodos de búsqueda local. Aunque su definición es sencilla, es un problema NP-duro. Sin embargo, debido a su aplicabilidad y fácil comprensión del problema, el problema del viajante se ha convertido en un referente para la aportación de nuevas ideas que pretenden resolver problemas de optimización combinatoria [1].

Para la resolución del problema del viajante han sido propuesto varias combinaciones de AG con métodos de búsqueda local [14, 16, 67, 144, 147]. Todos ellos se caracterizan por introducir dentro del propio AG un nuevo operador que es la búsqueda local, el cuál es el elemento diferenciador de dichos trabajos. De este modo, cada vez que se realizan las operaciones de reproducción (cruce y mutación) a los individuos resultantes se les aplica un método de búsqueda local para optimizar la solución que representa cada uno. Esta combinación ha recibido el nombre de Búsqueda Local Genética (*Genetic Local Search*) [99].

Dada la efectividad de estas técnicas [151], algunos investigadores se han centrado en mejorar la Búsqueda Local Genética. Así, se han propuesto nuevos operadores de cruce y mutación específicos [45, 46], o bien, procedimientos para mejorar la calidad de la población inicial [84].

Hu y Xie [71] adaptan la técnica de nichos de los AGs al método de la búsqueda local que se emplea en la Búsqueda Local Genética. De este modo, se busca mejorar la eficiencia del procedimiento de búsqueda local sin necesidad de explorar la vecindad completa.

Nuestra propuesta, a la que hemos denominado Búsqueda Genética en Vecindades (BGV), consiste en la realización de un procedimiento de búsqueda local de mejora continua con AGs. Para ello, una de las primeras aportaciones es la ampliación del concepto de vecindad.

Así, dada una solución, $i \in F$, definimos la *vecindad extendida de orden* L, $N_L(i) \subset F$, al conjunto de soluciones que se alcanzan desde la solución i en L movimientos consecutivos.

La BGV se caracteriza por realizar una exploración en vecindades extendidas. Para ello, se generan vecindades extendidas de orden L donde el

AG busca una solución óptima. La mejor solución encontrada se convertirá en el centro de la próxima vecindad, permitiendo de este modo avanzar por el espacio de soluciones, pero centrándonos en pequeños espacios con objeto de aprovechar la potencia del AG. La generación de vecindades sucesivas permitirá llegar al óptimo de la función.

Con objeto de evitar caer en un óptimo local donde el AG no pueda escapar, el orden de la vecindad se irá aumentando sucesivamente mientras no se consiga mejorar. De este modo, se podrá tener vecindades lo suficientemente grandes para poder salir del valle donde se encuentra el óptimo local, y conseguir alcanzar uno mejor. Este procedimiento de aumento de la vecindad es análogo al aumento de caja empleado en la BGC.

La estructura del capítulo es la siguiente: en el apartado 7.2 se detalla las características del método que hemos desarrollado bajo el nombre de BGC. El apartado 7.3 recoge cómo se adapta la BGC al problema del viajante simétrico. El apartado 7.4 muestra los resultados obtenidos con la BGV a diversos ejemplos de la biblioteca TSPLIB [133]. Finalmente, el apartado 7.5 recoge las conclusiones extraídas de los resultados obtenidos con la BGC.

7.2. Búsqueda Genética en Vecindades

El algoritmo BGV es una adaptación de la BGC (capítulo 5) al problema combinatorio. El esquema que se sigue es el siguiente: dado un punto inicial, x^0, perteneciente al conjunto de soluciones, se determina una vecindad extendida, $N_L(x^0)$, alrededor de dicho punto de orden L. El AG propuesto realizará una exploración de dicha vecindad, determinándose un nuevo individuo x^1. Éste se convertirá en el centro de una nueva vecindad que será explorada por el AG, y así sucesivamente. En este sentido, el orden de la vecindad, L, será uno de los parámetros importantes de entrada del algoritmo, y equivale al tamaño de la caja en la BGC, dado que determina la región de búsqueda del algoritmo.

Las características principales del AG que se encargará de explorar cada una de las vecindades, se pueden resumir en las siguientes:

- Los individuos serán codificados como una sucesión de L movimientos a partir del individuo central de la vecindad.
- Implementación de un AG de estado permanente, donde en cada generación se realizará una única operación.
- Se emplearán dos operadores de cruce y tres de mutación, que darán lugar a un único individuo que se introducirá en cada generación.
- La función de aptitud de los individuos es la misma función que se desea optimizar.

- El tamaño de la población depende del orden de la vecindad, que indica la extensión de la región a explorar.

7.2.1. Codificación de los individuos

Los individuos se codifican como una sucesión de L movimientos desde el punto central de la vecindad, al igual que una cadena de expulsión [53]. Aunque la realización de un único movimiento puede conducir a una solución peor que el centro de la vecindad, este movimiento puede producir una buena cadena de movimientos, la cuál permite obtener una buena solución.

El tipo de movimientos factibles son dependientes del problema específico que se esté resolviendo. En el apartado 7.3 veremos cómo codificar los individuos para resolver el problema del viajante simétrico.

7.2.2. Tamaño de la vecindad extendida

Asociada a la codificación del individuo se encuentra un parámetro importante del algoritmo, el orden de la vecindad, L, que nos va a determinar el tamaño de la región a explorar en cada iteración. Este parámetro será uno de los introducidos por el usuario para iniciar el proceso de optimización. Dado que determina la amplitud de la región a optimizar con el AG, otros parámetros del algoritmo dependerán del orden de la vecindad L. Así, tanto el tamaño de la población como el número de generaciones del AG serán proporcionales a L, de acuerdo a la siguiente fórmula:

$$\begin{aligned} TamPob &= L/\sigma \\ G &= TamPob \cdot \theta \end{aligned} \tag{7.1}$$

donde σ y θ serán parámetros de entrada del algoritmo.

7.2.2.1. Incremento del orden de la vecindad

Existe la posibilidad de que el orden de la vecindad seleccionada sea pequeña y no permita mejorar la solución inicial. Para evitar estas convergencias a óptimos locales, si la mejor solución encontrada en la búsqueda dentro de la vecindad no es mejor que la solución actual, la vecindad incrementa su orden en un tamaño igual al valor original. Una vez incrementada la vecindad, se vuelve a explorar de nuevo.

El número de veces que la vecindad puede aumentar su orden viene dado por un parámetro adicional, A, suministrado por el usuario. De este modo, el orden máximo de una vecindad que se puede explorar será una

función del orden original y el número de veces que ésta puede aumentar. De forma matemática:

$$L_{max} = LA \tag{7.2}$$

Este incremento del orden de la vecindad inicial tiene como objetivo escapar de los óptimos locales, de manera que el proceso de optimización consiga localizar un óptimo global.

7.2.3. Función de aptitud

La evaluación de un individuo consiste en calcular el valor de la función aptitud para cada uno de los movimientos indicados en su cadena. La mejor solución en la trayectoria del individuo será el valor de la aptitud del individuo. Así, si $Aptitud_{i,j}$ es el valor de la función aptitud para el individuo i después del movimiento j-ésimo, la aptitud del individuo es:

$$Aptitud_i = \text{máx}\{Aptitud_{i,j}, \forall j = 1, \ldots, L\} \tag{7.3}$$

Nótese que la aptitud del individuo no es el valor alcanzado en el último movimiento. De este modo, el individuo no representa a una única solución, sino que está representando L posibles soluciones, de manera que la población del AG implícitamente está representando a $TamPob \cdot L$ soluciones.

7.2.4. Operadores

La BGV implementa un AG de estado permanente. De este modo, en cada iteración sólo se realiza una única operación, o un cruce o una mutación. De este modo, la probabilidad del cruce, p_c, y la probabilidad de mutación, p_m, son complementarias, es decir:

$$p_m + p_c = 1 \tag{7.4}$$

Se han diseñado tres operadores de mutación y dos operadores de cruce. Cada uno de los operadores tendrá una probabilidad, cuya suma será la probabilidad del tipo de operador.

7.2.4.1. Operadores de mutación

El AG emplea tres operadores de mutación:

Mutación por Intercambio consiste en intercambiar la posición de un par de movimientos seleccionados aleatoriamente. La figura 7.1 muestra un ejemplo de este operador, donde se intercambian los movimientos situados en la posición 2 y 4, respectivamente.

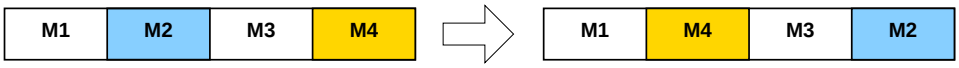


Figura 7.1: Mutación por intercambio

Mutación por Modificación consiste en alterar el valor de algunos movimientos del individuo. El número máximo de movimientos a modificar es el orden de la vecindad a explorar L. Cada movimiento tiene asociada una probabilidad, p_{mm}, para modificar el valor del mismo.

Mutación dirigida por Aptitud consiste en aplicar el operador Mutación por Modificación, pero únicamente a aquellos movimientos que se encuentren situados tras el movimiento que define la aptitud del individuo. La principal característica de este operador es que el individuo resultante es igual o mejor que el individuo seleccionado para realizar la mutación.

Cada uno de los operadores de mutación tiene una probabilidad asociada, p_{mi} es la de la mutación por intercambio, p_{mm} es la de la mutación por modificación y p_{mda} es la de la mutación dirigida por aptitud, de manera que la suma de estas probabilidades es la probabilidad de mutación del algoritmo, es decir:

$$p_m = p_{mi} + p_{mm} + p_{mda} \tag{7.5}$$

7.2.4.2. Operadores de cruce

Los dos operadores de cruce diseñados para la BGV son los siguientes:

Cruce Uniforme consiste en generar un nuevo individuo cuyos movimientos son seleccionados aleatoriamente de los dos padres. Podemos decir que es un cruce multipunto aplicado a todos los movimientos de los individuos a cruzar.

Cruce dirigido por Aptitud consiste en cruzar únicamente aquellos movimientos situados tras el movimiento que define la aptitud del individuo. De los dos hijos que se genera, sólo el mejor se introduce en la población. La principal característica de este operador es que el nuevo individuo resultante es siempre igual o mejor que el mejor de los dos padres. La figura 7.2 muestra este operador, donde la zona marcada en azul indica los movimientos que definen la aptitud en los dos padres. Podemos ver cómo se intercambian el resto de movimientos hasta el máximo posible, en este caso N movimientos.

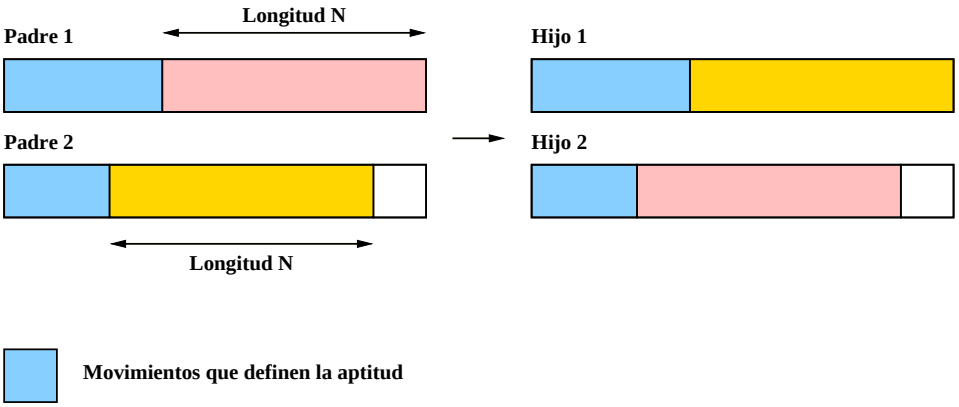


Figura 7.2: Cruce dirigido por aptitud

Cada operador de cruce tiene una probabilidad asociada, p_{cu} y p_{cda}, respectivamente, de modo que la suma de dichas probabilidad es la probabilidad de cruce, es decir:

$$p_c = p_{cu} + p_{cda} \tag{7.6}$$

7.2.4.3. Selección de individuos

Tanto los operadores de mutación como los operadores de cruce vistos en el apartado anterior necesitan la selección de los individuos participantes en las operaciones, así como el individuo que va a salir de la población. Para ambos operadores, cruce y mutación, la selección del individuo a sustituir se realizará mediante una sustitución basada en aptitud, de manera que aquellos individuos con peor aptitud tendrán una mayor probabilidad de selección que los individuos con mejor aptitud.

En el caso de la mutación, el individuo necesario para la aplicación del operador se selecciona aleatoriamente, es decir, la selección no está basada en la aptitud. Sin embargo, para el cruce los dos progenitores se seleccionarán proporcionalmente a la aptitud, empleando el método de la ruleta [55]. De este modo, los individuos con peor aptitud tendrán menos posibilidades de ser seleccionados que los individuos con mejor aptitud.

7.2.5. Proceso de optimización

La BGV se enmarca dentro de un proceso iterativo de búsquedas locales de mejora continua, que puede ser descrito con el algoritmo 7.1. El AG encargado de explorar una vecindad puede ser descrito con el pseudocódigo 7.2. Los parámetros de entrada necesarios para su funcionamiento son:

- Orden de la vecindad: L

- Parámetros de población y generaciones: θ y σ, que permitirá especificar tanto el tamaño de la población como el número de generaciones, según (7.1).

- Probabilidad de los operadores de cruce: p_{cu} y p_{cda}.

- Probabilidad de los operadores de mutación: p_{mi}, p_{mm} y p_{mda}.

- Número de aumentos de la vecindad: A.

- Porcentaje de generaciones sin mejorar: p_g.

- Punto central de la primera vecindad donde realizar la búsqueda: x^0.

Algoritmo 7.1 BGV

Optimización : $L \times A \times \theta \times \sigma \times p_{cu} \times p_{cda} \times p_{mi} \times p_{mm} \times p_{mda} \times p_g \times x^0 \times \epsilon \longrightarrow x^*$
$k \longleftarrow 0$
$a \longleftarrow 0$
$x^k \longleftarrow x^0$
$Vecindad \longleftarrow L$
$x^{k+1} \longleftarrow$ AG-BGV$(Vecindad, \theta, \sigma, p_{cu}, p_{cda}, p_{mi}, p_{mm}, p_{mda}, p_g, x^k)$
$optimo \longleftarrow$ LOCAL
Mientras $(optimo =$ LOCAL$)$
  Si $\left(\frac{|f(x^{k+1}) - f(x^k)|}{|f(x^{k+1})|} \leq \epsilon\right)$
    $k \longleftarrow k + 1$
    $a \longleftarrow 0$
    $Vecindad \longleftarrow L$
    $x^{k+1} \longleftarrow$ AG-BGV$(Vecindad, \theta, \sigma, p_{cu}, p_{cda}, p_{mi}, p_{mm}, p_{mda}, p_g, x^k)$
  Si no Si $(a < A)$ **entonces**
    $a \longleftarrow a + 1$
    $Vecindad \longleftarrow Vecindad + L$
    $x^{k+1} \longleftarrow$ AG-BGV$(Vecindad, \theta, \sigma, p_{cu}, p_{cda}, p_{mi}, p_{mm}, p_{mda}, p_g, x^k)$
  Si no
    $optimo \longleftarrow$ GLOBAL
Devolver x^k

Algoritmo 7.2 AG de la BGV

AG-BGV : $L \times \theta \times \sigma \times p_{cu} \times p_{cda} \times p_{mi} \times p_{mm} \times p_{mda} \times p_g \times x^0 \longrightarrow x$
$t \longleftarrow 0$
$P_t \longleftarrow$ Crear_Poblacion_Inicial$(L, \sigma, \bar{x}^0)$
Evaluar_Aptitud()
$G \longrightarrow L/\sigma \times \theta$
$GB \longrightarrow G \times p_g$
$continuar \longrightarrow 0$
Mientras $(t < G \quad \&\& \quad continuar < GB)$
  $t \longleftarrow t + 1$
  $mejor \longleftarrow$ Mejor_Individuo(P_{t-1})
  $op \longleftarrow$ Escoger_Operacion$(p_{mm}, p_{mi}, p_{mda}, p_{cu}, p_{cda})$
  $[padre_1, padre_2] \longleftarrow$ Seleccionar_Padres(P_{t-1})
  $hijo \longleftarrow$ Realizar_Operador$(op, padre_1, padre_2)$
  $ind \longleftarrow$ Seleccionar_Individuo(P_{t-1})
  Introducir_Individuo$(P_t, hijo, ind)$
  Si (Aptitud$(hijo) >$ Aptitud$(mejor)$ **entonces**
    $mejor \longleftarrow hijo$
    $continuar \longleftarrow 0$
  Si no
    $continuar \longleftarrow continuar + 1$
  Evaluar_Aptitud()
Devolver $mejor$

El criterio de finalización del algoritmo es el número máximo de generaciones, G, o bien si se alcanza un número consecutivo de generaciones, GB, sin obtener una mejor solución que el centro de la vecindad. Este parámetro es un dato de entrada del algoritmo especificado como porcentaje del número de generaciones totales del algoritmo, p_g, de tal modo que:

$$GB = p_g G \tag{7.7}$$

7.3. Aplicación al problema del viajante simétrico

En este apartado detallaremos como aplicar la BGV al problema del viajante simétrico. Para ello, describiremos brevemente el problema a resolver para posteriormente pasar a explicar los aspectos del AG, en concreto, la codificación de los individuos y la evaluación de su aptitud.

7.3.1. El problema del viajante simétrico

Existen numerosas variantes del problema del viajante. Una de ellas es el denominado *problema del viajante simétrico*. Aunque su definición es sencilla, es un problema PLS-completo[1] Este problema tiene muchas aplicaciones, desde la fabricación de circuitos VLSI [12] hasta la cristalografía de rayos X [83].

El problema del viajante simétrico se puede enunciar como sigue:

> Un vendedor, partiendo de la ciudad donde reside, quiere encontrar el camino más corto posible por el conjunto de ciudades de sus clientes, así como volver a su ciudad de residencia.

Más formalmente, el problema se define por N ciudades y una matriz simétrica D, de dimensiones $N \times N$, donde $d_{ij} = d_{ji}$ son las distancias entre el par de ciudades i y j, $\forall i, j = 1, \ldots, N$. La característica del problema del viajante simétrico, es que la distancia entre cualquier par de ciudades es independiente de la dirección del viaje. El objetivo del problema es encontrar la ruta de longitud mínima que visite cada ciudad exactamente una única vez [1]. Es decir, el objetivo es encontrar la permutación cíclica μ en las N ciudades, tal que minimice el coste del viaje:

$$F(\mu) = \sum_{i=1}^{N-1} d(c_{\mu(i)}, c_{\mu(i+1)}) + d(c_{\mu(N)}, c_{\mu(1)}) \tag{7.8}$$

donde $c_{\mu(i)}$ es la ciudad que se visita en el orden i-ésimo según la permutación μ.

7.3.2. El movimiento 2-intercambio

En el apartado 7.2.1 se indicó que los individuos de la BGV se codifican como una sucesión de L movimientos.

En el problema del viajante simétrico, el movimiento que se emplea es el denominado *2-intercambio*. Éste consiste en seleccionar aleatoriamente dos ciudades, P y Q, de tal modo que los arcos $(P, \Pi(P))$ y $(Q, \Pi(Q))$ son eliminados, y los nuevos arcos (P, Q) y $(\Pi(P), \Pi(Q))$ son introducidos en la ruta. La notación $\Pi(P)$ indica la siguiente ciudad a P en la solución actual. La figura 7.3 muestra gráficamente este movimiento.

La codificación de este movimiento consiste en un vector de dos componentes, que representan los dos arcos a eliminar. Cada arco se codifica con un vector de dos elementos: la ciudad origen y la dirección del arco. Aunque para el problema del viajante simétrico la dirección del movimiento no es relevante, en la codificación de los individuos, se considerará esta

[1] Problema de Búsqueda Local de Tiempo Polinómico(véase capítulo 2 de [1]).

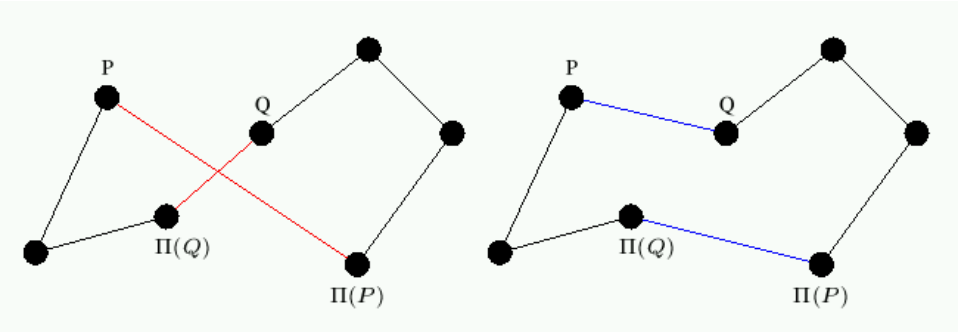


Figura 7.3: El movimiento 2-intercambio

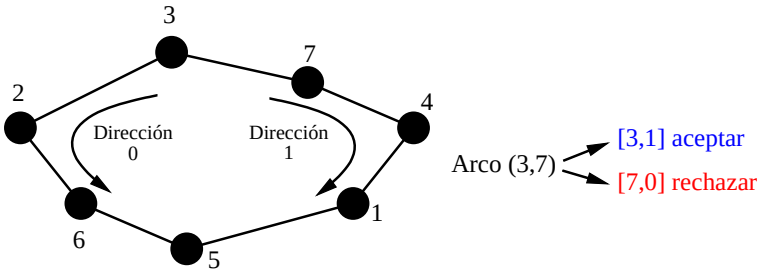


Figura 7.4: Codificación de un arco en el movimiento de la BGV

dirección. Así, si la dirección del arco tiene la misma dirección que el viaje, se codifica con un 1, y con un 0 en caso contrario. La figura 7.4 ilustra una posible solución, donde la dirección seleccionada es en sentido del reloj. En este ejemplo, el arco entre las ciudades 3 y 7 se codifica $[3, 1]$, mientras que el arco entre las ciudades 3 y 2 se codifica como $[3, 0]$.

El problema de usar esta representación es que cualquier arco posee dos códigos. La solución al mismo consiste en usar únicamente la codificación que lleva asociada la ciudad con una numeración menor. En el ejemplo de la figura 7.4, el arco entre las ciudades 3 y 7, con codificaciones $[3, 1]$ y $[7, 0]$, únicamente se representa por $[3, 1]$.

7.3.3. Aptitud de los individuos

Cada movimiento de un individuo tiene un coste asociado, debido a los dos arcos que se eliminan, disminuyendo el valor de la función aptitud, y los dos nuevos arcos que se introducen en la ruta, aumentando el valor de la función aptitud. Así, el individuo i tiene un incremento en la función

aptitud debido al movimiento j:

$$\Delta_{ij} = d_{p,q} + d_{\Pi(p),\Pi(q)} - d_{p,\Pi(p)} - d_{q,\Pi(q)} \tag{7.9}$$

En la generación k-ésima de la BGV, donde el centro de la vecindad explorada viene representado por x^k, el valor de aptitud del individuo i después del movimiento j-ésimo es,

$$Aptitud_{i,j} = F(x^k) + \sum_{h=1}^{j-1} \Delta_{ih} + \Delta_{ij} \tag{7.10}$$

El valor de la aptitud del individuo será el mejor valor que se obtenga en los L movimientos que lo componen. Matemáticamente:

$$Aptitud_i = \text{máx}\{Fitness_{i,j}, j = 1, \ldots, L\} \tag{7.11}$$

7.4. Resultados con la BGV

Las pruebas efectuadas con la BGV han sido realizadas en dos fases. En primer lugar, se ha determinado cómo influyen los distintos parámetros de entrada del AG en los resultados obtenidos. Una vez fijados los parámetros, se ha procedido a realizar pruebas computacionales con diversos problemas de la TSPLIB.

7.4.1. Influencia de los parámetros de entrada

Dado que la BGV depende de un conjunto de parámetros de entrada, es necesario comprender la influencia de éstos en el funcionamiento del algoritmo, con objeto de determinar los valores más adecuados.

Para medir el rendimiento de la BGV haremos uso de los ejemplos mostrados en la tabla 7.1, donde se muestra el número de ciudades del problema, así como el valor óptimo de dicho problema. Todos son instancias del problema del viajante simétrico tomadas de la biblioteca TSPLIB [133]. Esta biblioteca constituye un repositorio accesible a través de Internet, con objeto de poder medir el rendimiento de las diferentes propuestas existentes para resolver el problema. Muchas de las propuestas presentadas en dicha biblioteca son aplicaciones reales del problema del viajante simétrico.

Para medir la calidad de la solución obtenida por la BGV, ésta se midió como el porcentaje de error cometido sobre la solución óptima. Matemáticamente:

$$\frac{\text{Coste\_solución} - \text{Coste\_Óptimo}}{\text{Coste\_Óptimo}} 100 \tag{7.12}$$

Problema	Número de ciudades	Óptimo
EIL51	51	426
ST70	70	675
KROA200	200	29368

Tabla 7.1: Instancias de problemas de la TSPLIB para comprobar los parámetros de la BGV

7.4.1.1. Tamaño de la población

Un parámetro importante del funcionamiento de la BGV es el tamaño de la población. Aunque está relacionado con el orden de la vecindad a explorar (7.1), es necesario establecer un tamaño mínimo para poder explorar con garantías las vecindades.

Para comprobar el efecto de la población en el rendimiento, en el problema ST70 se han realizado 100 ejecuciones de la BGV, empleando probabilidades uniforme para cada operador, así como un orden de vecindad de $L = 15$, con tamaño de población 15, 25, 75 y 150. La figura 7.5 muestra los resultados obtenidos en cuanto a iteraciones y tiempo de CPU frente a la calidad de la solución obtenida en dichas ejecuciones.

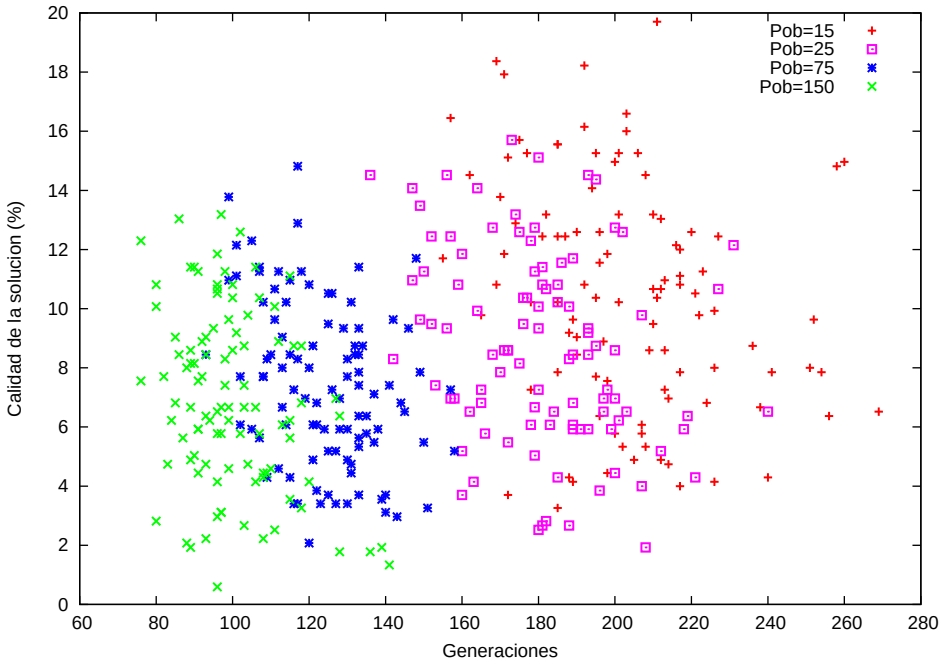
Nótese que un tamaño de población pequeño requiere más iteraciones que una gran población. Obviamente, una gran población requiere de más tiempo de computación, debido a que necesita evaluar más individuos, y más generaciones (véase (7.1)). Sin embargo, un incremento del tiempo de computación no siempre se traduce en obtener mejores soluciones, por lo que debe mantenerse un compromiso entre intensificación y diversificación para que la búsqueda sea efectiva.

7.4.1.2. Orden de la vecindad

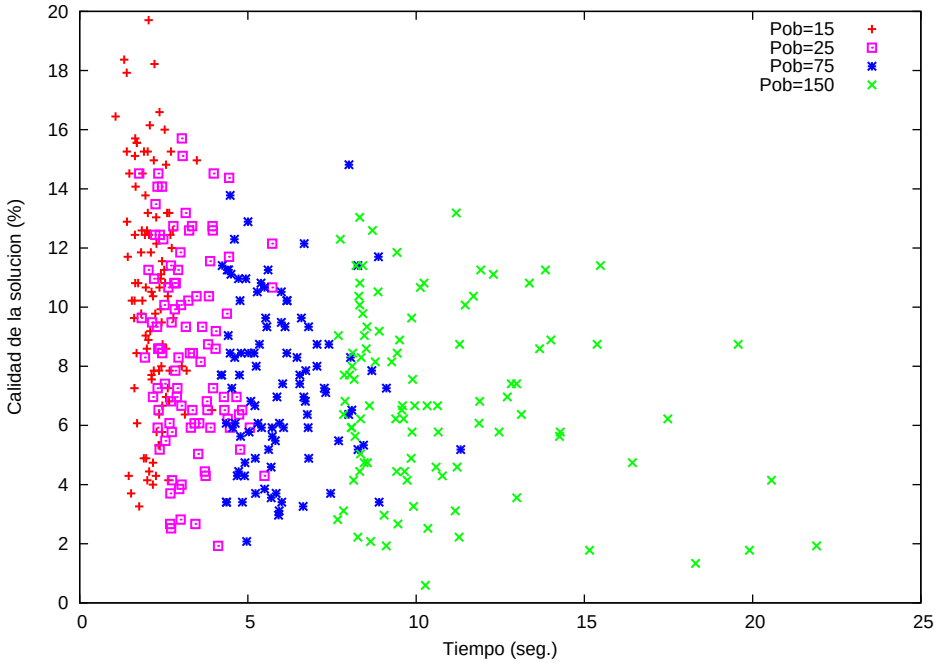
El orden de la vecindad extendida, L, es el principal parámetro de la BGV, puesto que define el tamaño de la zona a explorar en cada búsqueda local. Además, otros parámetros del algoritmo, como son el tamaño de la población y el número de generaciones, dependen directamente del mismo (7.1).

La figura 7.6 muestra para los problemas EIL51 y ST70, los resultados obtenidos por la BGV para 50 ejecuciones, usando una probabilidad uniforme para cada operador, un tamaño de población del 20 % del número de ciudades, así como órdenes de la vecindad de 5, 10 y 20, respectivamente.

Nótese que una vecindad de mayor orden parece conducir a mejores soluciones que las de menor orden. Esto es debido a que la BGV, al tener un mayor radio de exploración, consigue realizar una búsqueda más efectiva, evitando caer en óptimos locales.



a) Iteraciones frente a la calidad de la solución



b) Tiempo de CPU frente a la calidad de la solución

Figura 7.5: Resultados en el problema ST70 para 100 ejecuciones con diferentes tamaños de población

Sin embargo, cuando empleamos vecindades de menor orden, los tiempos de ejecución son menores. Por tanto, es necesario establecer un compromiso entre tiempo de ejecución y calidad de la solución obtenida.

En los experimentos realizados el tamaño óptimo del orden de la vecindad se sitúa entre 10 y 15. Dado que este orden inicial puede ser incrementado cuando no se mejora (apartado 7.2.2.1), los experimentos nos demuestran que los valores óptimos para dicho parámetro se encuentran en el rango de 6 a 10.

7.4.1.3. Elección del punto inicial

Otro parámetro de la BGV desarrollada es el punto inicial o, en el caso del problema del viajante simétrico, la ruta inicial. Para comprobar cómo depende de este parámetro se han realizado diversos experimentos. Así, para los problemas EIL51 y ST70 se han efectuado 100 ejecuciones empezando en puntos iniciales generados aleatoriamente. La figura 7.7 muestra cuál es la calidad de la solución obtenida con la BGV en los problemas EIL51 y ST70 frente a la distancia inicial existente entre el punto de partida y el óptimo.

La figura 7.8 permite analizar para los mismos experimentos anteriores, la relación entre el tiempo de CPU empleado en la ejecución frente a la distancia inicial existente entre el punto de partida y el óptimo.

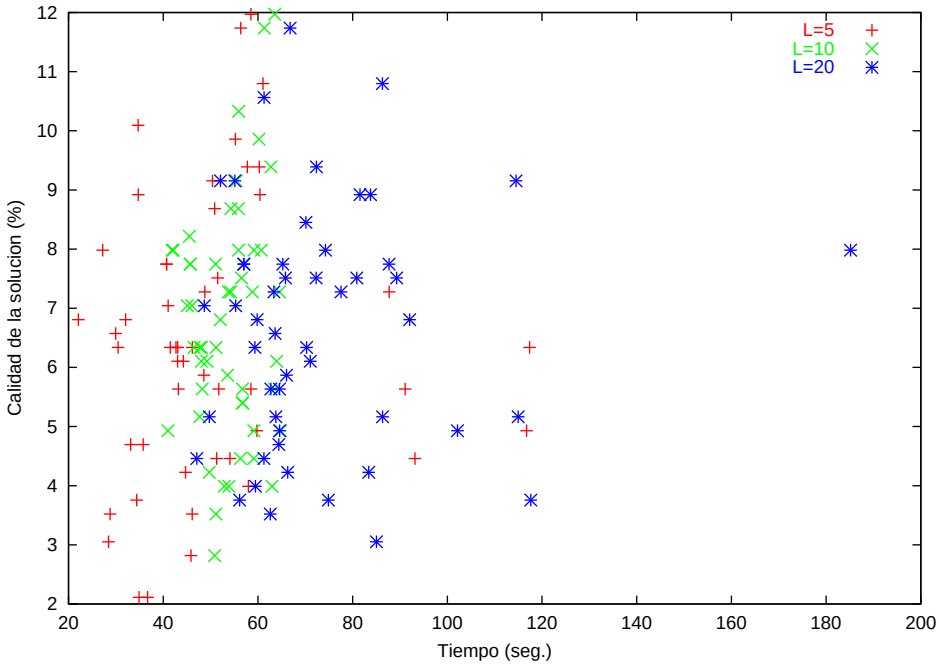
En estos experimentos, se utilizó un tamaño de población del 20 % del número de ciudades, es decir, 10 y 15 respectivamente. El orden de la vecindad se fijó a 10, y se dispusieron probabilidades uniformes para los diferentes operadores.

Nótese que, independientemente del punto inicial, la BGV consigue una buena aproximación al óptimo. De forma similar, el tiempo de CPU requerido en las ejecuciones no depende del punto inicial seleccionado.

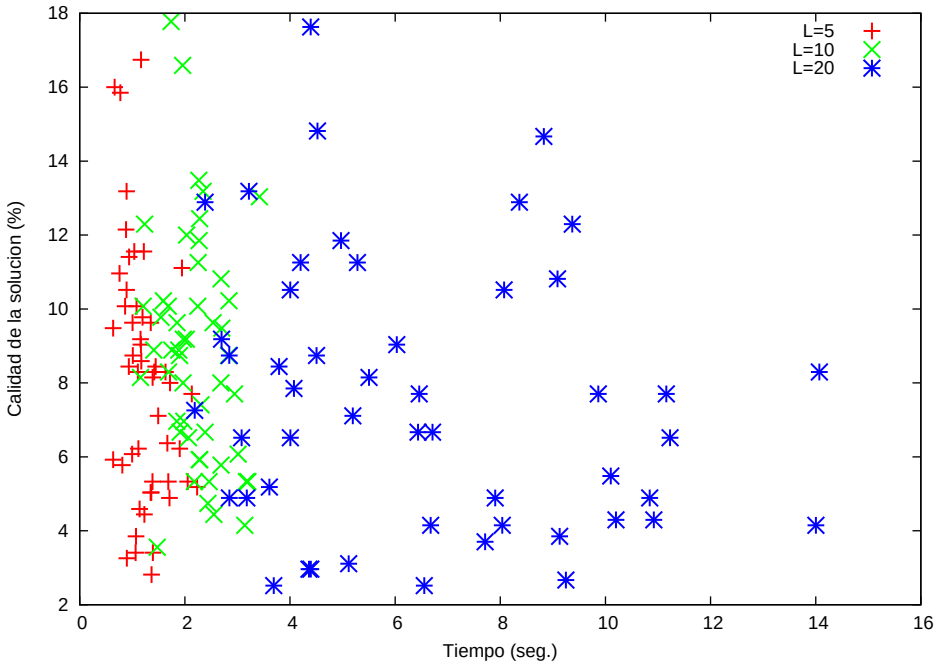
7.4.1.4. Operadores

Una de las características de la BGV es la existencia de cinco operadores, tres de mutación y dos de cruce. Cada uno de esos operadores lleva asociado una probabilidad para su ejecución. A pesar de ese numeroso conjunto de parámetros, las pruebas efectuadas demuestran que la BGV es muy estable en relación a dichas probabilidades.

La tabla 7.2 muestra los resultados obtenidos en el problema EIL51 para diferentes probabilidades. Hemos realizado 100 ejecuciones con diferentes semillas aleatorias e idéntico punto inicial. En concreto, se ha tomado como punto inicial la denominada ruta canónica, es decir, la ruta $1, 2, 3, \ldots, 51$. De forma análoga la tabla 7.3 muestra los resultados obtenidos para el problema KROA200.

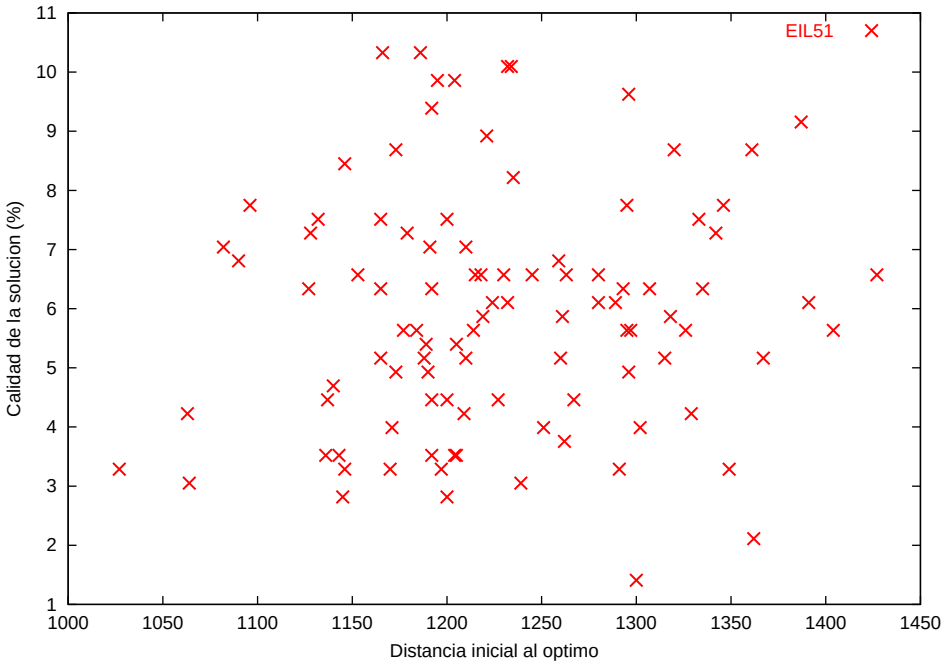


a) Problema EIL51

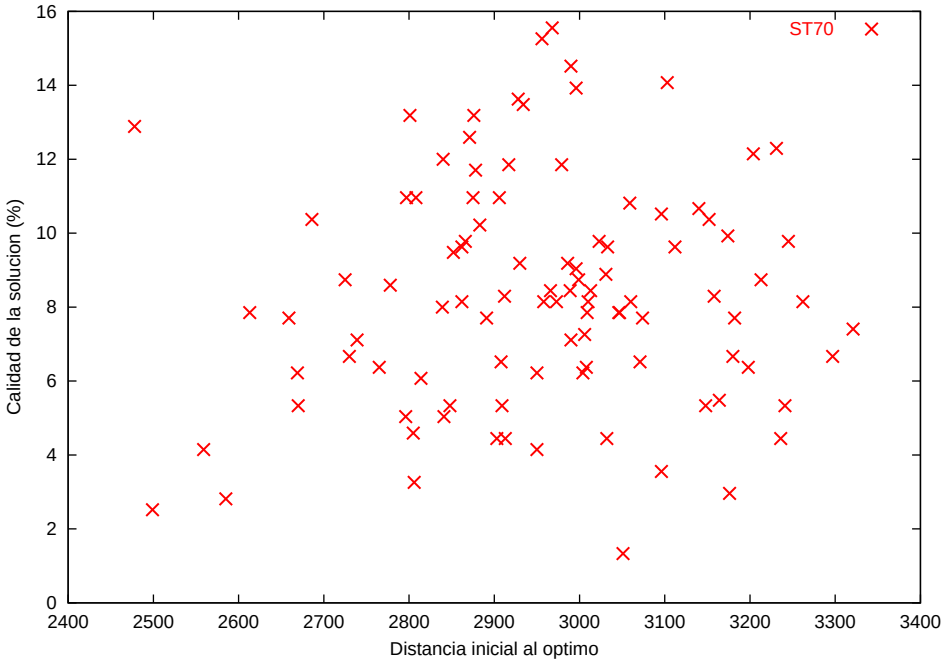


b) Problema ST70

Figura 7.6: Tiempo de CPU frente a la calidad de la solución obtenida para diferentes órdenes de vecindad

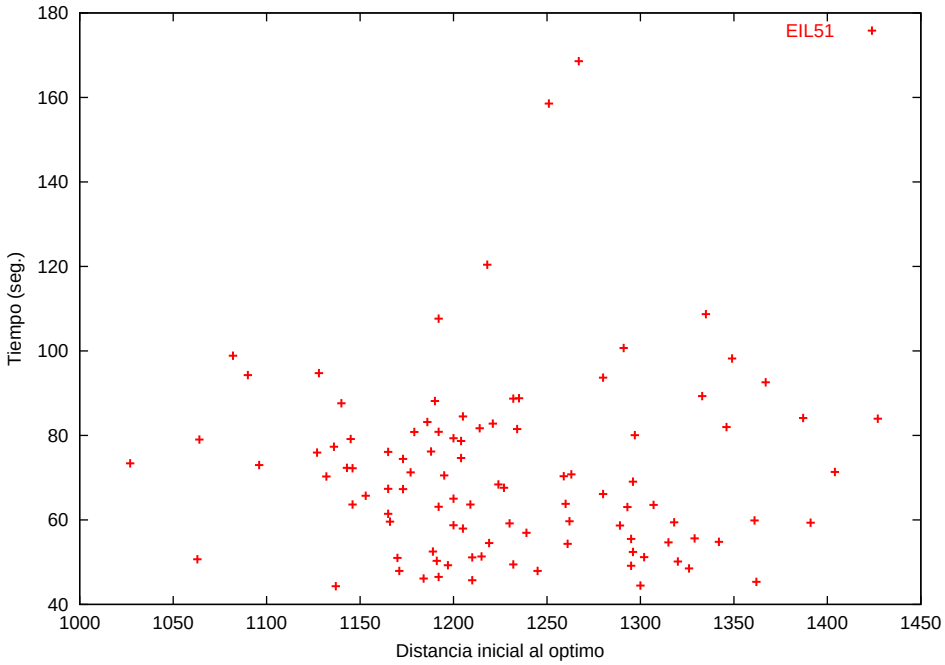


a) Problema EIL51

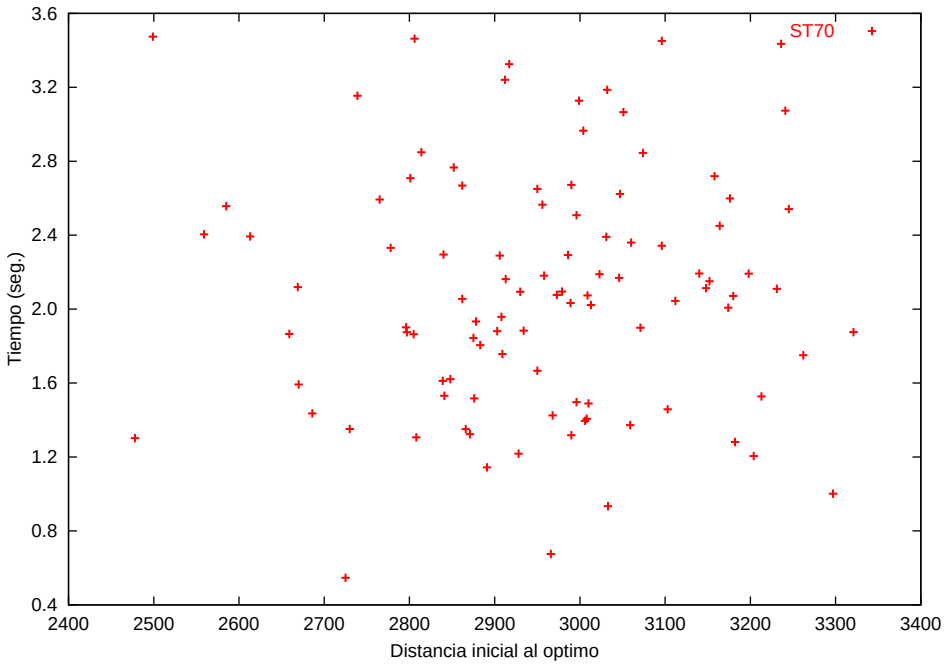


b) Problema ST70

Figura 7.7: Distancia inicial al óptimo frente a la calidad de la solución para 100 puntos iniciales aleatorios



a) Problema EIL51



b) Problema ST70

Figura 7.8: Distancia inicial al óptimo frente al tiempo de CPU para 100 puntos iniciales aleatorios

Tam. Pob.	Probabilidades					Error Medio (%)	Desviación Estándar (%)
	p_{cda}	p_{cu}	p_{mi}	p_{mm}	p_{mda}		
100	0.64	0.16	0.04	0.04	0.12	0.00	0.09
	0.50	0.20	0.04	0.04	0.12	0.05	0.12
	0.35	0.35	0.04	0.04	0.12	0.60	0.43
	0.20	0.50	0.04	0.04	0.12	0.10	0.11
	0.16	0.64	0.04	0.04	0.12	0.98	1.45
	0.64	0.16	0.12	0.04	0.04	0.11	1.01
	0.64	0.16	0.04	0.12	0.04	0.25	2.19
					Media	0.30	0.77

Tabla 7.2: Resultados para el problema EIL51 con diferentes probabilidades

Podemos ver que aunque las probabilidades son diferentes, los errores medios obtenidos para las diferentes ejecuciones son muy similares. En concreto, la media del error medio en el problema EIL51 es de 0.30, y en el problema KROA200 es de 1.54. Además, la desviación estándar que se consigue en ambos problemas son pequeñas. En el caso del problema EIL51 es de 0.77, mientras que en el problema KROA200 es de 2.17.

Tam. Pob.	Probabilidades					Error Medio (%)	Desviación Estándar (%)
	p_{cda}	p_{cu}	p_{mi}	p_{mm}	p_{mda}		
300	0.50	0.10	0.10	0.10	0.20	0.01	0.02
	0.40	0.20	0.10	0.10	0.20	0.99	0.25
	0.30	0.30	0.10	0.10	0.20	3.45	0.86
	0.20	0.40	0.10	0.10	0.20	2.83	2.69
	0.10	0.50	0.10	0.10	0.20	1.14	2.78
	0.50	0.10	0.20	0.10	0.10	1.57	5.35
	0.50	0.10	0.10	0.20	0.10	0.81	3.25
					Media	1.54	2.17

Tabla 7.3: Resultados para el problema KROA200 con diferentes probabilidades

7.4.2. Eficacia de la BGV

Para comprobar el rendimiento de la BGV se han escogido una serie de problemas de la TSPLIB, en concreto, EIL51, ST70, KROA200, PCB442, ATT532, RAT783 y FL1577. La tabla 7.4 muestra, para cada uno de ellos, el número de ciudades, así como el valor óptimo de dichos problemas.

Problema	Número de ciudades	Óptimo
EIL51	51	426
ST70	70	675
KROA200	200	29368
PCB442	442	50778
D657	657	48912
RAT783	783	8806
FL1400	1400	20127
FNL4461	4461	182566

Tabla 7.4: Instancias de los problemas de la TSPLIB empleados

La tabla 7.5 muestra los parámetros empleados en los diferentes problemas, tales como tamaño de la población, orden de la vecindad, probabilidades de los operadores de cruce y mutación, número máximo de generaciones, así como el número máximo de generaciones sin mejorar el centro de la vecindad.

La tabla 7.6 muestra los resultados obtenidos en la resolución de los problemas que aparecen en la tabla 7.4 con los parámetros de la tabla 7.5 para 100 ejecuciones. Podemos observar que para los dos problemas más pequeños, la BGV encuentra la solución óptima. Para problemas con un gran número de ciudades, la BGV obtiene buenas soluciones, con un error medio inferior al 10 %, salvo para el problema FNL4461, que se considera dentro de la TSPLIB como un gran problema al disponer de 4461 ciudades.

Una característica importante del algoritmo desarrollado es que la desviación estándar que se obtiene en las ejecuciones suele ser pequeña, lo que da muestra de la estabilidad y consistencia del algoritmo.

7.5. Conclusiones

El algoritmo denominado BGV presenta una nueva estrategia de optimización para los problemas combinatorios. Consiste en realizar sucesivas búsquedas locales explorando las vecindades mediante un AG.

Una de las características del algoritmo desarrollado es que explora vecindades extendidas, considerando la mejor solución obtenida por el AG como el centro de una nueva vecindad extendida.

La originalidad del método radica tanto en la exploración de vecindades de gran orden mediante un AG, así como la evaluación de la aptitud de los individuos, que consiste en la mejor solución que se encuentra a lo largo de la trayectoria que representa con sus múltiples movimientos.

Tanto la codificación de los individuos, como los propios operadores de mutación y cruce dependen del problema a resolver. En concreto, se han

Problema	Tamaño Población	Orden Vecindad	Probabilidades					Máximo Generaciones	Generaciones sin mejora
			p_{cda}	p_{uc}	p_{mi}	p_{mm}	p_{mda}		
EIL51	100	5	0.64	0.16	0.04	0.04	0.12	250	100
ST70	150	10	0.64	0.16	0.04	0.04	0.12	250	100
KROA200	300	10	0.50	0.10	0.10	0.10	0.20	1000	200
PCB442	500	15	0.50	0.10	0.10	0.10	0.20	2000	300
D657	500	15	0.50	0.10	0.10	0.10	0.20	2000	300
RAT783	500	15	0.45	0.15	0.15	0.15	0.10	3000	400
FL1400	750	15	0.45	0.15	0.15	0.15	0.10	3000	400
FNL4461	1500	15	0.55	0.15	0.10	0.10	0.10	5000	500

Tabla 7.5: Parámetros empleados por la BGV en diferentes problemas de la TSPLIB

Problema	Núm. Itera.	Tpo. CPU (seg.)	Error medio (%)	Desv. estándar (%)	Óptimo	Mejor	Peor
EIL51	125	1.34042	0.00	0.09	426	426	430
ST70	189	1.74468	0.05	0.18	675	675	681
KROA200	846	26.6382	0.01	0.02	29368	29369	29385
PCB442	927	85.2127	0.52	0.35	50778	50926	51350
D657	1020	190.255	5.16	3.83	48912	49201	53721
RAT783	1192	196.510	6.62	8.16	8806	8899	14327
FL1400	2953	528.595	7.83	3.16	20127	21111	27589
FNL4461	6890	2971.12	25.30	6.42	182566	225829	251429

Tabla 7.6: Resultados computaciones con diversos problemas de la TSPLIB

diseñado cinco operadores, dos para cruce, y tres para mutación, que dará lugar a los nuevos individuos que aparecerán en cada generación.

Las pruebas computacionales se han efectuado resolviendo el problema del viajante simétrico. Los resultados nos demuestran que la BGV es un método robusto respecto al punto inicial, obteniendo soluciones de calidad en un tiempo razonable de CPU.

Además, presenta un alto grado de estabilidad frente a los diferentes parámetros que configuran su comportamiento, como lo demuestran las pruebas realizadas.

Es capaz de diferenciar entre óptimos locales y globales, encontrándose los primeros localizados como los centros de las sucesivas vecindades de búsquedas, y el global es el último punto localizado en el proceso de optimización.

Capítulo 8

Búsqueda genética de mutantes

En los últimos años, los procesos de negocios basados en composiciones de servicios en WS-BPEL están rápidamente incrementándose, por lo que es importante prestar especial atención a la prueba del software en este contexto. Una de las técnicas clásicas de prueba de caja blanca es la prueba de mutaciones. Ésta ha sido aplicada con éxito a programas escritos en otros lenguajes de programación. En este capítulo se presenta un AG para la generación de mutantes de composiciones de servicios web en WS-BPEL. A diferencia de otros generadores, este generador minimiza el número de mutantes generados sin pérdida de información. Este generador se integra dentro de la herramienta GAmera, que permite detectar los mutantes potencialmente equivalentes, así como permite medir la calidad de los casos de prueba.

8.1. Introducción

La fase de pruebas es una de las más costosas del ciclo de vida del software. En sentido estricto, deben realizarse pruebas de todos los elementos generados durante la construcción de un producto, lo que incluye especificaciones de requisitos, casos de uso, diagramas de diversos tipos y, por supuesto, el código fuente y el resto de productos que forman parte de la aplicación (por ejemplo las bases de datos). Obviamente, se aplican diferentes técnicas de prueba a cada tipo de producto software.

Desde que en 1972 Dijkstra [29] indicase que las pruebas del software servían para mostrar la presencia de errores, pero nunca su ausencia, se ha producido un constante desarrollo de las técnicas de prueba de programas. Sin embargo, ni en el ciclo de vida software del estándar ISO/IEC 12207 [21] se define un proceso de Pruebas como tal, sino que aconseja, durante la ejecución de los procesos principales o de la organización, utilizar

los procesos de soporte. Myers y col. [116] indican que aproximadamente el 50 % del tiempo de desarrollo de un proyecto se dedica a la corrección de errores, lo que demuestra la importancia que tiene la realización de pruebas en el desarrollo del software actual.

La evolución del software hacia las arquitecturas orientadas a servicios ha conducido a la definición de un lenguaje que facilite la composición de servicios web. Un *servicio web* (*web services*, WS), según el World Wide Web Consortium, se define como una aplicación software identificada por un URI cuyas interfaces se pueden definir, describir y descubrir mediante documentos XML. Los WS permiten la interoperación de sistemas distribuidos heterogéneos con independencia de las plataformas hardware y software empleadas [39]. De esta forma, permiten un desarrollo rápido de aplicaciones, caracterizado tanto por un coste bajo, como por la facilidad de componer aplicaciones distribuidas [126]. Estas características de flexibilidad e integración han permitido que más de la mitad de las grandes empresas desarrolladoras de software hayan empezado a desarrollar aplicaciones con una arquitectura orientada a servicios y más del 70 % de las empresas norteamericanas hagan uso de aplicaciones SOA [64].

En este sentido, el estándar OASIS WS-BPEL 2.0 [118] se ha convertido en la referencia a nivel industrial. Éste permite especificar la lógica de la composición de los servicios (envío de mensajes, sincronización, iteración, tratamiento de transacciones erróneas, etc.) independientemente de su implementación.

El impacto económico de las composiciones de servicios WS-BPEL está creciendo rápidamente [73]. Esta evolución del software hacia composiciones de WS requiere introducir cambios en la metodología de pruebas realizadas tradicionalmente, puesto que muchas de las técnicas y herramientas existentes no se pueden utilizar directamente con los lenguajes de composición de WS [17].

Palacios y col. [124] realizan una revisión de las técnicas y herramientas de pruebas para procesos de negocio WS-BPEL, enumerando distintos enfoques para generar casos de prueba, propuestas de uso de la monitorización como complemento a las pruebas, herramientas de soporte y diferentes métodos para verificar propiedades de los procesos de negocio.

Las técnicas de caja blanca y, más en concreto, la prueba de mutaciones (*mutation testing*) pueden jugar un papel importante en la definición de estrategias de prueba para las composiciones de WS. En este sentido, es necesario el desarrollo de herramientas que, de forma automática, generen mutantes para las composiciones de WS, con objeto de poder automatizar el proceso de pruebas.

La aplicación de AGs a las pruebas del software se remontan a Xanthakis y col. [162]. Desde entonces, han aparecido numerosas aplicaciones [5, 13, 94, 98, 101, 127], aunque la mayoría de ellas se han limitado a la generación de casos de prueba. Por tanto, a la vista de la revisión bibliográfica

realizada, el algoritmo que se desarrolla en esta tesis se convertiría en la primera aplicación de los AGs a la generación de mutantes. Sólo Adamopoulos y col. [3] emplean AGs en prueba de mutaciones. Sin embargo, no generan mutantes, sino que éstos se generan previamente con la herramienta Mothra [81]. Posteriormente, el AG se encarga de seleccionar el conjunto óptimo de mutantes para el conjunto de casos de prueba inicial.

Este capítulo presenta un uso novedoso de los AGs a la generación de mutantes. El generador se integrará en la herramienta GAmera, un sistema automático de generación de mutantes para composiciones WS-BPEL, diseñada dentro de la tesis y que en el apartado 8.4 se expondrá sus características. Este generador será capaz de detectar los mutantes potencialmente equivalentes y, de este modo, mejorar la calidad de los casos de prueba.

Normalmente, los sistemas de generación de mutantes simplemente generan todos los posibles mutantes. En lugar de esto, GAmera conducirá el proceso de generación de mutantes mediante el AG, optimizando el número de mutantes generados de acuerdo a una función de aptitud.

La estructura del capítulo es la siguiente. En el apartado 8.2 se realiza una revisión de la técnica de prueba de mutaciones. El apartado 8.3 da una visión general del lenguaje WS-BPEL para el que se generan los mutantes con la herramienta GAmera que hemos diseñado. En el apartado 8.4 se exponen los aspectos más técnicos relacionados con el diseño de GAmera. Finalmente, los apartados 8.5 y 8.6 muestran los resultados obtenidos en las ejecuciones que hemos realizado con composiciones de servicios WS-BPEL y las conclusiones obtenidas, respectivamente.

8.2. Prueba de mutaciones

La prueba de mutaciones es una técnica de prueba del software de caja blanca basada en errores [26, 58], que consiste en introducir fallos simples en el programa original, aplicando para ello *operadores de mutación*. Los programas resultantes reciben el nombre de *mutantes*. Cada operador de mutación se corresponde con una categoría de error típico que puede cometer el programador. Así, si un programa presenta en su código la instrucción `compra > 5000` y disponemos de operadores de mutación sobre los operadores relacionales, que consisten en cambiar un operador por otro, el mutante resultante podría tener como instrucción, por ejemplo, `compra < 5000`. Si un caso de prueba es capaz de distinguir al programa original del mutante, es decir, la salida del mutante y la del programa original son diferentes, se dice que mata al mutante. Si por el contrario ningún caso de prueba es capaz de diferenciar al mutante del programa original, es decir, la salida del mutante y del programa original es la misma, se habla de un mutante vivo para el conjunto de casos de prueba empleado.

La prueba de mutaciones puede ser empleada con dos objetivos. Por un

lado, nos permite determinar la calidad de un conjunto de casos de prueba. Esto recibe el nombre de análisis de mutaciones (*mutation analysis*). Por otro, nos permite generar nuevos casos de prueba que maten a los mutantes que permanecen vivos, de manera que se mejore la calidad del conjunto inicial de casos de prueba.

Una de las principales dificultades de aplicar la prueba de mutaciones es la existencia de *mutantes equivalentes*. Éstos presentan el mismo comportamiento que el programa original, es decir, la salida del mutante y del programa original es siempre la misma. Estos mutantes no deben confundirse con los mutantes difíciles de matar (*stubborn non-equivalent mutants*), que se deben a que el conjunto de casos de prueba no es suficiente para poder detectarlos. El problema general de determinar si un mutante es equivalente al programa original es indecidible [167].

La calidad de los conjuntos de casos de prueba se mide mediante la puntuación de mutación (*mutation score*), que indica el número de mutantes muertos frente al número de mutantes no equivalentes, de acuerdo a la siguiente fórmula:

$$TM = \frac{D}{M - E} \tag{8.1}$$

donde D es el número de mutantes muertos, M el número total de mutantes y E el número de mutantes equivalentes.

Otro de los principales inconvenientes de la prueba de mutaciones es el alto coste computacional que implica. Esto es debido a que disponemos normalmente de un número grande de operadores de mutación que generan un elevado número de mutantes, cada uno de los cuales debe ejecutarse frente al conjunto de casos de prueba hasta que muera. Offutt y col. [119] demuestran empíricamente que, bajo ciertas condiciones, para un programa de n líneas se generan del orden de n^2 mutantes. Existen diversas estrategias para reducir el elevado coste computacional de la prueba de mutaciones [121]:

- Realizar pocas mutaciones (*Do fewer*): consiste en ejecutar un pequeño número de mutantes sin incurrir en una pérdida de información. Dentro de esta estrategia nos encontramos:
  - Mutación selectiva [120], selecciona sólo los mutantes que realmente son diferentes al resto, aplicando para ello los operadores de mutación más críticos.
  - Muestreo de mutantes, donde sólo se ejecuta un subconjunto aleatorio del total de mutantes generados.
- Realizar mutaciones pequeñas (*Do smarter*): tiene como objetivo distribuir el coste computacional entre varias máquinas o bien evitar ejecuciones completas de los mutantes. Entre éstas se encuentran:

  - Mutación débil [70]: esta técnica compara el estado interno del mutante y del programa original tras ejecutar la porción mutada del programa.
  - Mutación en arquitecturas distribuidas: se caracteriza por distribuir el coste entre varias máquinas como vectores de procesadores, SIMD, MIMD, redes de ordenadores, etc.

- Realizar mutaciones rápidamente (*Do faster*): se centra en la generación y ejecución de cada mutante lo más rápidamente posible. La técnica más extendida es la mutación basada en esquemas [152], que codifica todas las mutaciones en un único programa fuente denominado *metamutante*.

Existen varios trabajos en la bibliografía que abordan el desarrollo de sistemas para la generación automática de mutantes, tales como Mothra [81] para Fortran, MuJava [92] para Java, Proteum [25] para C, y SQL-Mutation [150] para SQL. Todos estos sistemas se caracterizan por generar todos los mutantes posibles de acuerdo a los operadores de mutación empleados. Podemos constatar que no existen herramientas generadores de mutantes para composiciones de servicios WS-BPEL. En este sentido, Rice [134] enumera y explica los diez retos más importantes en la automatización del proceso de pruebas. Entre ellos, se encuentra la falta de herramientas, bien por su elevado precio o bien porque las existentes no se ajusten al propósito o entorno para el que se necesitan. Por este motivo, consideramos que el desarrollo de un generador automático de mutantes para WS-BPEL permitirá mejorar el proceso de pruebas en este tipo de lenguaje de programación.

8.3. El lenguaje WS-BPEL

WS-BPEL [118] es un lenguaje basado en XML que permite especificar el comportamiento de un proceso de negocio basado en interacciones con WS. La estructura de un proceso WS-BPEL se divide en cuatro secciones: definición de relaciones con los socios externos, que son el cliente que utiliza el proceso de negocio y los WS a los que llama el proceso, definición de las variables que emplea el proceso, definición de los distintos tipos de manejadores que puede utilizar el proceso (manejadores de fallos y de eventos), y descripción del comportamiento del proceso de negocio; esto se logra a través de las actividades que proporciona el lenguaje.

Todos los elementos definidos anteriormente son globales si se declaran dentro del proceso. También existe la posibilidad de declararlos de forma local mediante el contenedor `scope`, que permite dividir el proceso de negocio en diferentes ámbitos.

Los principales elementos constructivos de un proceso WS-BPEL son las *actividades*, que pueden ser de dos tipos: básicas y estructuradas. Las básicas son las que realizan una determinada labor (recepción de un mensaje, envío de un mensaje, ...), mientras que las estructuradas pueden contener otras actividades y definen la lógica de negocio.

A las actividades pueden asociarse un conjunto de atributos y de contenedores. Estos últimos pueden incluir diferentes elementos, que a su vez pueden tener atributos asociados.

```
<flow>  ← Actividad estructurada
 <links>  ← Contenedor
  <link name="Comprueba-Reserva-Vuelo"/> ← Elemento
 </links>
 <invoke name="CompruebaVuelo" ... > ← Actividad básica
  <sources>  ← Container
   <source linkName="Comprueba-Reserva-Vuelo"/> ← Elemento
  </sources>
 </invoke>
 <invoke name="CompruebaHotel" ... />
 <invoke name="CompruebaCoche" ... />
 <invoke name="ReservarVuelo" ← Atributo ...>
  <targets>  ← Contenedor
   <target linkName="Comprueba-Reserva-Vuelo" />
  </targets>
 </invoke>
</flow>
```

Figura 8.1: Ejemplo de WS-BPEL

Además, WS-BPEL permite realizar acciones en paralelo y de forma sincronizada. Por ejemplo, la actividad `flow` permite ejecutar un conjunto de actividades concurrentemente especificando las condiciones de sincronización entre ellas.

En la figura 8.1 podemos ver una actividad `flow` que invoca a tres WS en paralelo, `CompruebaVuelo`, `CompruebaHotel`, y `CompruebaCoche`. Existe otro WS, `ReservarVuelo`, que sólo será invocado si se completa con éxito el WS `CompruebaVuelo`. Para conseguir esta sincronización entre actividades se establece un enlace, de manera que la actividad `target` del enlace sólo se ejecuta si la actividad `source` del enlace ha sido completada.

8.3.1. Operadores de mutación

En la revisión bibliográfica realizada, únicamente Estero y otros [38] han definidos operadores de mutación para el lenguaje WS-BPEL. En concreto, definen 26 operadores de mutación. Estos operadores han sido clasificados en cuatro categorías, dependiendo del tipo de elemento sintáctico con el que se relacionan. Éstos son identificados por letras mayúsculas:

- I (*operadores de sustitución de Identificadores*),

- E (*operadores de Expresiones*),
- A (*operadores de Actividades*),
- X (*operadores de Excepciones y Eventos*).

Dentro de cada categoría se definen varios operadores de mutación que se identifican mediante tres letras mayúsculas: la primera de ellas coincide con la que identifica la categoría a la que pertenece el operador, mientras que las dos últimas identifican al operador dentro de la categoría. La tabla 8.1 muestra los nombres de cada operador, así como una descripción breve de cada uno de ellos.

OPERADOR	DESCRIPCIÓN
	MUTACIÓN DE IDENTIFICADORES
ISV	Sustituye el identificador de una variable por el de otra del mismo tipo
	MUTACIÓN DE EXPRESIONES
EAA	Sustituye un operador aritmético por otro del mismo tipo
EEU	Elimina el operador - unario de cualquier expresión
ERR	Sustituye un operador relacional por otro del mismo tipo
ELL	Sustituye un operador lógico por otro del mismo tipo
ECC	Sustituye un operador de camino por otro del mismo tipo
ECN	Modifica una constante numérica
EMD	Modifica una expresión de duración
EMF	Modifica una expresión de fecha límite
	MUTACIÓN DE ACTIVIDADES
	Relacionados con la concurrencia
ACI	Cambia el atributo `createInstance` de las actividades de recepción de mensajes a *no*
AFP	Cambia una actividad `forEach` secuencial a paralela
ASF	Cambia una actividad `sequence` por una actividad `flow`
AIS	Cambia el atributo `isolated` de un `scope` a *no*
	No concurrentes
AIE	Elimina un elemento `elseif` o el elemento `else` de una actividad `if`
AWR	Cambia una actividad `while` por una `repeatUntil` y viceversa
AJC	Elimina el atributo `joinCondition` de cualquier actividad en la que aparezca
ASI	Intercambia el orden de dos actividades consecutivas hijas de una actividad `sequence`
APM	Elimina un elemento `onMessage` de una actividad `pick`
APA	Elimina el elemento `onAlarm` de una actividad `pick` o de un manejador de eventos
	MUTACIÓN DE CONDICIONES EXCEPCIONALES Y EVENTOS
XMF	Elimina un elemento `catch` o el elemento `catchAll` de un manejador de fallos
XRF	Elimina el atributo `faultName` de una actividad `reply`
XMC	Elimina la definición de un manejador de compensación
XMT	Elimina la definición de un manejador de terminación
XTF	Cambia el fallo lanzado por una actividad `throw`
XER	Elimina una actividad `rethrow`
XEE	Elimina un elemento `onEvent` de un manejador de eventos

Tabla 8.1: Operadores de mutación para WS-BPEL 2.0

Estos operadores de mutación modelan los fallos que podrían cometerse al generar una composición WS-BPEL 2.0, habiéndose tenido en cuenta que normalmente éstas no se suelen escribir de forma directa, sino que se utilizan herramientas gráficas. Por lo que muchos de los fallos que pueden aparecer en programas escritos en otros lenguajes, debidos a errores

al teclear código, no aparecerán en WS-BPEL, como por ejemplo insertar el operador - unario delante de una expresión.

8.4. GAmera

El sistema de generación automático de mutantes para composiciones de servicios WS-BPEL que se ha desarrollado en el transcurso de esta tesis, al que se ha denominado *GAmera*[1], está formado por tres componentes principales (véase figura 8.2): el analizador, el generador de mutantes y el sistema de ejecución, el cuál ejecuta y también evalúa a los mutantes. El analizador recibe como entrada la definición del proceso original WS-BPEL y genera la información necesaria para que el generador de mutantes pueda generar a los diferentes mutantes. Durante la generación, el sistema ejecutará a los mutantes frente a un conjunto de casos de prueba y determinará los mutantes potencialmente equivalentes.

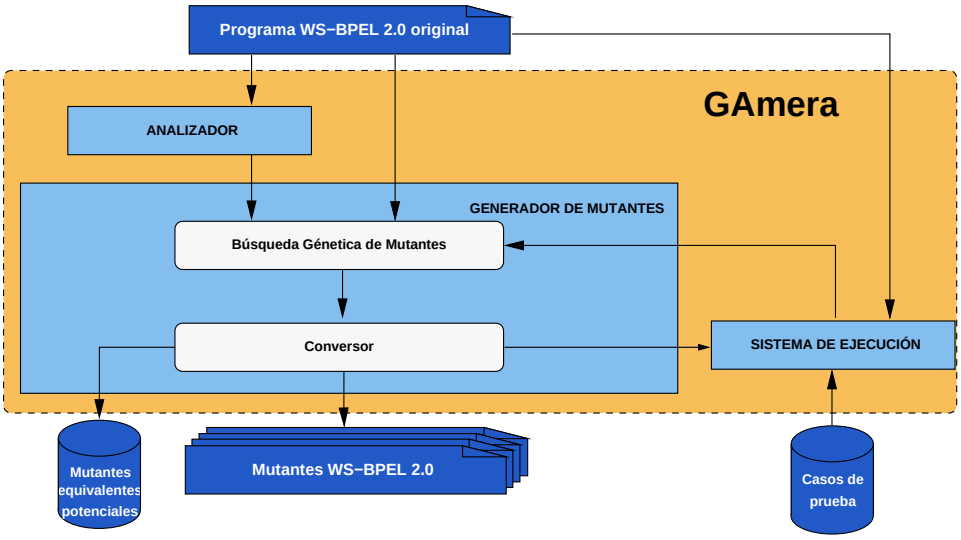


Figura 8.2: Sistema de generación automática de mutantes para WS-BPEL

8.4.1. Analizador de WS-BPEL

Antes de ejecutar el generador de mutantes, el sistema analizará el programa original WS-BPEL. En esta fase se identificarán las instrucciones o elementos del proceso original que pueden sufrir mutación de acuerdo al conjunto de operadores de mutación definidos para el lenguaje. Esta labor

[1]Esta herramienta se encuentra disponible en `http://neptuno.uca.es/~gamera`

es desempeñada por el *analizador*, el cuál recibe como entrada una definición de proceso WS-BPEL y obtiene como resultado una lista de los distintos operadores de mutación que pueden emplearse en el proceso especificado, así como el número de instrucciones donde pueden ser aplicados.

```
<process
 name="loanApprovalProcess"
 ...
 <partnerLinks>
  <partnerLink name="approver" ... />
  <partnerLink name="assessor" ... />
  <partnerLink name="customer" ... />
 </partnerLinks>
 <variables>
  <variable name="risk" ... />
  <variable name="approval" ... />
  <variable name="request" ... />
 </variables>
 <faultHandlers>
  <catch faultName="loanProcessFault" >
  <reply  faultName="unableToHandleRequest" ... />
  </catch>
 </faultHandlers>
 <sequence>
  <receive name="ReceiveRequest" ... />
  <if name="IfSmallAmount">
   <condition>
    ( $request.amount &lt;= 10000 )
   </condition>
    <sequence name="SmallAmount">
     <invoke name="AssessRiskOfSmallAmt" ... />
     <if name="IfLowRisk">
      <condition>
       ( $risk.level = 'low' )
      </condition>
      <assign name="ApproveLowRiskSmallAmtLoans">
      <copy>
       <from>true()</from>
       <to part="accept" variable="approval"/>
      </copy>
      </assign>
     <else>
      <invoke name="CheckApprover" ... />
     </else>
     </if>
    </sequence>
  <else>
   <invoke name="ApproveLargeAmt" ... />
  </else>
  </if>
  <reply name="ReportApproval" ... />
 </sequence>
</process>
```

Figura 8.3: La composición WS-BPEL *loan-approval*

Supongamos la composición de WS-BPEL consistente en la aprobación de un préstamo (*loan-approval*) que se muestra en la figura 8.3. Esta composición recibe un mensaje de un cliente que solicita una cierta cantidad de dinero. Dependiendo de la cantidad solicitada, el proceso WS-BPEL invoca

al WS asesor (*asessor*) cuando la cantidad que se solicita es menor o igual a 10000, o al WS aprobador (*approver*), en otro caso. La salida del WS asesor (*asessor*) se corresponde con el nivel de riesgo del cliente. Si el riesgo es bajo, se concede el préstamo, en caso contrario la petición se envía al WS aprobador (*approver*), el cuál toma la decisión final de aceptación del préstamo.

La salida del analizador para esa composición es la que aparece en la figura 8.4. Podemos ver que sobre dicha composición pueden actuar seis operadores de mutación: ERR, ECN, ASF, AIE, ASI y XRF. Además, nos indica que existen dos expresiones relacionales donde el operador ERR puede aplicarse, una constante numérica para el operador ECN, dos actividades `if` para el operador AIE, dos actividades `sequence` para el operador ASF, tres actividades hijas en `sequence` para el operador ASI, así como un atributo `faultname` para el operador XRF.

```
ERR 2
ECN 1
ASF 2
AIE 2
ASI 3
XRF 1
```

Figura 8.4: Salida del analizador para la composición *loan-approval*

Cada operador de mutación definido en GAmera implementa su propio analizador como parte de una hoja de estilos XSLT 2.0, que se ejecuta en el motor de código abierto SaxonB 9.1 [139]. Estos analizadores recorren la definición del proceso original WS-BPEL con objeto de determinar aquellos elementos donde pueden ser aplicados, así como toda la información extra necesaria que pudiesen requerir. Los elementos son identificados como referencias a los nodos actuales en el árbol del documento DOM XML de la definición del proceso WS-BPEL. Una vez recorrido todo el árbol, el analizador devuelve el número de elementos asociado a cada operador, con objeto de que el generador de mutantes pueda iniciar su labor.

Dado que las hojas de estilo XSLT pueden únicamente recorrer documentos XML, se ha tenido que extender Saxon para que las expresiones XPath embebidas en la definición del proceso WS-BPEL puedan ser convertidas a árboles de sintaxis abstractas XML y, así, poderlas analizar con la misma hoja de estilo. Para realizar este procesamiento, se ha implementado un analizador de gramática usando JavaCC [154].

Los operadores de mutación que afectan a las expresiones XPath, es decir, todos cuyos nombres comienzan por E, excepto EMD y EMF, almacenan en la lista de elementos donde actúan, no sólo la referencia al nodo de la definición del proceso WS-BPEL, sino también al árbol de sintaxis abstracta

XML asociada a dicha expresión, así como la referencia al nodo exacto sobre el que actúa dicho operador.

8.4.2. Generador de mutantes

El *generador de mutantes* está formado por dos elementos:

Búsqueda Genética de Mutantes un AG, en el que cada individuo representa a un mutante, capaz de generar de forma automática un conjunto de mutantes.

Conversor que transforma un individuo del AG a un mutante.

El generador de mutantes tiene un doble objetivo. Por un lado, generar los mutantes para composiciones de servicios WS-BPEL y, por otro, detectar los mutantes potencialmente equivalentes. La figura 8.5 muestra un esquema del generador de mutantes. Podemos observar cómo el generador de mutantes tiene como entrada la definición del proceso WS-BPEL original y la salida del analizador, así como los parámetros de configuración del AG. La salida que produce el generador está formada por los mutantes generados, así como los mutantes potencialmente equivalentes.

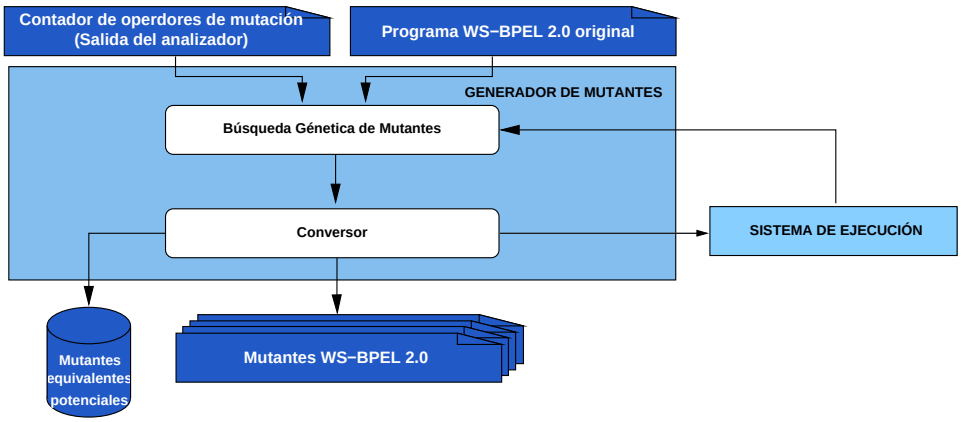


Figura 8.5: El generador de mutantes para WS-BPEL

Dado que el AG trabaja con una codificación propia del mutante, se necesita un conversor para poder generar el mutante WS-BPEL. Asimismo, para poder evaluar la aptitud de los individuos, es necesario ejecutar los mutantes. Esto se consigue transformando el individuo a mutante mediante el conversor, y, posteriormente, pasarlo al sistema de ejecución, que será descrito en el apartado 8.4.4.

8.4.3. Búsqueda genética de mutantes

Para la generación de mutantes, hemos desarrollado un AG, al que se ha denominado *Búsqueda Genética de Mutantes* (BGM). En la revisión bibliográfica realizada no hemos encontrado ningún otro AG cuyo objetivo sea la generación de mutantes, por lo que podemos considerar como la primera aplicación de los AGs en este campo. Este algoritmo ha sido implementado utilizando la biblioteca de C++ GAlib [156]. A continuación pasamos a detallar los distintos aspectos del algoritmo.

8.4.3.1. Representación de los individuos

Cada individuo codifica la mutación a realizar al programa original. La figura 8.6 muestra la representación de un individuo. Podemos ver que cada individuo tiene tres campos: uno para identificar el operador, otro para referenciar la instrucción donde se aplicará, y un tercero que contiene información para la aplicación del operador de mutación.

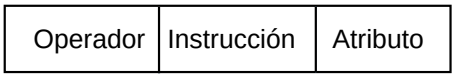


Figura 8.6: Representación de un individuo

El campo *Operador* identifica al operador de mutación que se aplica. Se codifica con un valor entero comprendido en el rango de 1 a 26, que se corresponde con el número de operadores de mutación definidos para el lenguaje WS-BPEL.

El campo *Instrucción* representa el número de instrucción del programa original donde se aplicará el operador. También se codifica con un valor entero. Con objeto de realizar una distribución uniforme entre todos los operadores independientemente del número de instrucciones asociadas a cada uno, el campo se codifica con un valor entero comprendido en el rango de 1 a I, donde $I = \mathrm{mcm}\{m_i | 1 \leq i \leq 26\}$, siendo m_i el número de instrucciones que existen en el programa original a las que se puede aplicar el operador de mutación i-ésimo.

Posteriormente, para obtener el número de instrucción donde se realiza la operación de mutación es necesario realizar una traducción. De este modo, si I_i es el valor de un individuo en el campo *Instrucción*, representará una operación de mutación en la instrucción $\lceil \frac{I_i \cdot m_i}{I} \rceil$.

Por ejemplo, consideremos que disponemos de dos operadores de mutación, con 2 y 3 instrucciones a las que se pueden aplicar, respectivamente. El valor máximo de este campo será $I = \mathrm{mcm}\{2, 3\} = 6$. De este modo, los individuos tendrán valores en dicho campo de 1 a 6. Un individuo que para el primer operador tenga un valor de 4, significa que está representando

un cambio en la instrucción $\lceil \frac{4 \cdot 2}{6} \rceil = 2$ donde se pueda aplicar. Obsérvese, que la mitad de los casos, de 1 a 3, se aplica a la primera instrucción, y la otra mitad, de 4 a 6, se aplica a la segunda instrucción.

Operador	Valor	Máx. Valor del Atributo
ISV	1	N
EAA	2	5 `+, -, *, div, mod`
EEU	3	1
ERR	4	6 `<, >, >=, <=, =, !=`
ELL	5	2 `and, or`
ECC	6	2 `/, //`
ECN	7	4 +1, -1, añadir, eliminar
EMD	8	2 0, $\frac{1}{2}$
EMF	9	2 0, $\frac{1}{2}$
ACI	10	1
AFP	11	1
ASF	12	1
AIS	13	1
AIE	14	1
AWR	15	1
AJC	16	1
ASI	17	1
APM	18	1
APA	19	1
XMF	20	1
XRF	21	1
XMC	22	1
XMT	23	1
XTF	24	N
XER	25	1
XEE	26	1

Tabla 8.2: Valores y atributos para los operadores de mutación

El campo *Atributo* representa la información necesaria para la aplicación del operador de mutación especificado en el campo *Operador*. Dado que una mutación consiste en la alteración de un elemento del programa, este campo especifica cuál será el nuevo valor que tome el elemento. Para ello, los valores que puede tomar el elemento se representarán mediante un entero. El rango de posibles valores va a depender del operador de mutación que se esté aplicando. La tabla 8.2 muestra, para cada operador, su valor (campo *Operador*) así como el valor máximo que puede tomar en el campo *Atributo*. Existen dos operadores, ISV y XTF, cuyos valores depen-

den del programa original. Sin embargo, otros como EEU, sólo tienen un único valor debido a que sólo pueden producir un único mutante. En estos casos, el campo *Atributo* no tiene significado.

Al igual que en el campo *Instrucción*, con objeto de realizar una distribución uniforme entre todos los individuos, el campo *Atributo* contiene un valor entero dentro del rango 1 a V, donde $V = \text{mcm}\{v_i | 1 \leq i \leq 26\}$, siendo v_i el número de valores que puede tomar el operador de mutación i-ésimo. Posteriormente se realiza una traducción para la obtención del desplazamiento. Si A_i es el valor de un individuo en el campo *Atributo*, representará un desplazamiento $\lceil \frac{A_i \cdot v_i}{V} \rceil$.

Por ejemplo, consideremos el programa original que aparece en la figura 8.3, donde pueden actuar los operadores de mutación ERR, ECN, ASF, AIE, ASI y XRF. El campo *Instrucción* estará en el rango 1 a 6, donde $6 = \text{mcm}\{2, 3\}$. El campo *Atributo* estará en el rango 1 a 12, donde $12 = \text{mcm}\{6, 4, 1, 1, 1, 1\}$. Así, las figuras 8.7 y 8.8 representan a los mutantes que se obtienen de los individuos (4, 1, 10) y (14, 1, 5), respectivamente.

Nótese que el individuo (4, 1, 10) cambia el operador relacional `<=` por `=`, y que el individuo (14, 1, 5) elimina el elemento `else` de la actividad `if` denominada `IfLowRisk`.

Este esquema de codificación de individuos permite que la arquitectura pueda adaptarse fácilmente a cualquier lenguaje de programación. Simplemente habría que numerar los distintos operadores de mutación disponibles y, para cada uno de ellos, sus posibles valores.

8.4.3.2. Aptitud de los individuos

Para poder evaluar la aptitud de un individuo necesitamos ejecutar el mutante al que representa frente a los casos de prueba hasta que encontremos uno que lo mate. Esta ejecución se explicará en el apartado 8.4.4. Al finalizar la ejecución de todos los mutantes, dispondremos de una matriz, denominada *matriz de ejecución*, de la siguiente forma:

$$(m_{ij})_{M \times T} = \begin{pmatrix} 0 & 0 & 1 & \dots & 0 \\ 1 & 0 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 0 \end{pmatrix} \tag{8.2}$$

donde m_{ij} valdrá 1 ó 0, dependiendo de si el mutante i-ésimo ha sido matado o no con el caso de prueba j-ésimo, M representa el número máximo de mutantes, es decir, el tamaño de la población, y T el número máximo de casos de prueba de que disponemos. En esa matriz se establecen las siguientes relaciones:

```
<process
 name="loanApprovalProcess"
 ...
 <partnerLinks>
  <partnerLink name="approver" ... />
  <partnerLink name="assessor" ... />
  <partnerLink name="customer" ... />
 </partnerLinks>
 <variables>
  <variable name="risk" ... />
  <variable name="approval" ... />
  <variable name="request" ... />
 </variables>
 <faultHandlers>
  <catch faultName="loanProcessFault" >
  <reply  faultName="unableToHandleRequest" ... />
  </catch>
 </faultHandlers>
 <sequence>
  <receive name="ReceiveRequest" ... />
  <if name="IfSmallAmount">
   <condition>
    ( $request.amount = 10000 )
   </condition>
    <sequence name="SmallAmount">
     <invoke name="AssessRiskOfSmallAmt" ... />
     <if name="IfLowRisk">
      <condition>
       ( $risk.level = 'low' )
      </condition>
      <assign name="ApproveLowRiskSmallAmtLoans">
      <copy>
       <from>true()</from>
       <to part="accept" variable="approval"/>
      </copy>
      </assign>
     <else>
      <invoke name="CheckApprover" ... />
     </else>
     </if>
    </sequence>
  <else>
   <invoke name="ApproveLargeAmt" ... />
  </else>
  </if>
  <reply name="ReportApproval" ... />
 </sequence>
</process>
```

Figura 8.7: Individuo (4, 1, 10)

```
<process
 name="loanApprovalProcess"
 ...
 <partnerLinks>
  <partnerLink name="approver" ... />
  <partnerLink name="assessor" ... />
  <partnerLink name="customer" ... />
 </partnerLinks>
 <variables>
  <variable name="risk" ... />
  <variable name="approval" ... />
  <variable name="request" ... />
 </variables>
 <faultHandlers>
  <catch faultName="loanProcessFault" >
  <reply  faultName="unableToHandleRequest" ... />
  </catch>
 </faultHandlers>
 <sequence>
  <receive name="ReceiveRequest" ... />
  <if name="IfSmallAmount">
   <condition>
    ( $request.amount &lt;= 10000 )
   </condition>
    <sequence name="SmallAmount">
     <invoke name="AssessRiskOfSmallAmt" ... />
     <if name="IfLowRisk">
      <condition>
       ( $risk.level = 'low' )
      </condition>
      <assign name="ApproveLowRiskSmallAmtLoans">
      <copy>
       <from>true()</from>
       <to part="accept" variable="approval"/>
      </copy>
      </assign>



     </if>
    </sequence>
  <else>
   <invoke name="ApproveLargeAmt" ... />
  </else>
  </if>
  <reply name="ReportApproval" ... />
 </sequence>
</process>
```

Figura 8.8: Individuo (14, 1, 5)

- Un mutante puede ser matado por un caso de prueba o seguir vivo:

$$\sum_{j=1}^{T} m_{ij} \in [0,1], \forall i \tag{8.3}$$

- Un caso de prueba puede matar a ninguno o a varios mutantes:

$$\sum_{i=1}^{M} m_{ij} \in [0,M], \forall j \tag{8.4}$$

Cuando un caso de prueba no mata a ningún mutante puede indicarnos dos cosas: el caso de prueba tiene baja calidad, o bien los mutantes generados no son adecuados para ese caso de prueba.

La función de aptitud de un individuo tendrá en cuenta si éste es muerto o no por los casos de prueba, y cuántos mutantes son también matados por el mismo caso de prueba. De este modo, la función de aptitud para el individuo I en la generación G vendrá dada por la siguiente ecuación:

$$\text{Aptitud}(I,G) = M - \sum_{j=1}^{T} \left(m_{Ij} \cdot \sum_{i=1}^{M} m_{ij} \right) \tag{8.5}$$

Con esa función de aptitud intentamos penalizar aquellos grupos de mutantes que son matados por el mismo caso de prueba, y favorecer a aquellos que son matados por un caso de prueba que sólo los reconoce a ellos. El objetivo es generar los mutantes más difíciles de matar y descartar los que son fáciles de matar dado que no aportan información al proceso de prueba.

Sin embargo, cuando un mutante no es matado por ningún caso de prueba, es decir, no es detectado (tiene una aptitud de valor M), será almacenado en una memoria del algoritmo. Así, esta memoria contiene los mutantes potencialmente equivalentes.

Una característica de la representación de individuos utilizada es la posibilidad de que existan diferentes individuos que representen al mismo mutante. Por ejemplo, los individuos (14, 1, 5) y (14, 1, 10) son el mismo mutante, debido a que los dos representan al operador AIE en la primera instrucción donde se pueda aplicar. En este caso, el campo *Atributo* no tiene significado ya que el operador AIE sólo tiene un único valor posible para el campo *Atributo*.

Cuando el AG genera un individuo que represente a un mutante que ha sido generado previamente, el mutante no será ejecutado, su aptitud se establecerá a 0 y su fila en la matriz de ejecución se establecerá también a 0, con objeto de no penalizar al resto de individuos.

8.4.3.3. Generaciones

La primera población del AG será generada aleatoriamente. El tamaño de la población M constituye un parámetro de entrada del algoritmo. Se emplea un AG generacional, donde los individuos de las siguientes generaciones provienen de:

- Individuos generados aleatoriamente. Para ello, el algoritmo dispone de un parámetro de entrada, N, que determina cuántos mutantes serán generados aleatoriamente.
- Individuos procedentes de las operaciones de cruce y mutación. En este caso, el algoritmo genera nuevos individuos mediante la realización de operaciones de cruce y mutación del AG sobre los individuos de la población anterior. La selección de los individuos participantes en las operaciones se realiza mediante la técnica de la ruleta [55]. Se realizan operaciones de cruce o mutación, ambas excluyentes, de acuerdo a las probabilidades de estas operaciones, que mantendrán la siguiente regla:

$$p_m = 1 - p_c \tag{8.6}$$

  donde p_m es la probabilidad de realizar una operación de mutación y p_c la probabilidad de cruce, siendo éste un parámetro de entrada del algoritmo.

8.4.3.4. Operadores genéticos

El AG para la generación de nuevos individuos aplica los dos operadores genéticos: el cruce y la mutación.

El operador de cruce consiste en un intercambio de campos de los individuos participantes. El punto de cruce se escoge aleatoriamente entre los posibles campos que componen los individuos. La figura 8.9 representa de forma gráfica las distintas posibilidades de realizar un cruce entre dos individuos padres para generar dos hijos, en función del punto de cruce seleccionado.

El operador de mutación consiste en cambiar el valor de un elemento del individuo. De este modo existen 3 tipos de operadores de mutación, según donde se apliquen: mutación de instrucción, mutación de operador y mutación de atributo.

Dado que las tres mutaciones cambian elementos codificados como enteros, se empleará una mutación consistente en añadir un número aleatorio al valor del individuo seleccionado. La operación se realiza en módulo L, siendo L el valor máximo que puede alcanzar el elemento sobre el que se realiza la mutación, con el objeto de que al realizar la mutación el nuevo valor resultante siempre se sitúe dentro de los valores permitidos. Así, si α es el valor seleccionado, la mutación resultante, β, será:

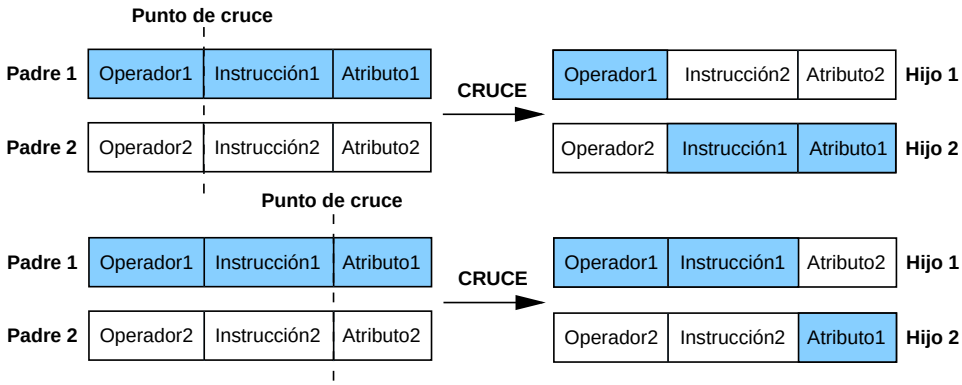


Figura 8.9: Operación de cruce

$$\beta = \alpha + random(1, L-1) \pmod{L} \tag{8.7}$$

En las operaciones de cruce y mutación es necesario seleccionar los individuos participantes. Hemos optado por realizar una selección con probabilidad uniforme sin tener en cuenta el valor de la función de aptitud en cada individuo.

8.4.3.5. Criterio de parada

El AG finalizará cuando se cumpla alguno de los dos criterios siguientes:

- Se alcanza el número máximo de generaciones, G.
- El número de mutantes generadores por la BGM alcanza el porcentaje, P, del total de mutantes posibles de la composición original.

8.4.3.6. Parámetros de configuración

La BGM puede ser descrita mediante el algoritmo 8.1. Los parámetros de entrada que necesita para su funcionamiento son los siguientes:

- Tamaño de la población, M.
- Número máximo de generaciones, G.
- Probabilidad de cruce, p_c.
- Porcentaje de nuevos individuos generados aleatoriamente entre generaciones, N.

- Porcentaje de mutantes a generar, P.
- Fichero con el resultado del analizador, $f\_analizador$.
- Fichero WS-BPEL con la composición original, $original$.

Algoritmo 8.1 BGM

BGM : $M \times G \times p_c \times N \times P \times f\_analizador \times original \longrightarrow m$
$operadores\_mutacion \longleftarrow$ leer($f\_analizador$)
$NumMutantes \longleftarrow P \cdot$ Calcular_Mutantes_Totales($operadores\_mutacion$)
$P_t \longleftarrow$ Crear_Poblacion_Inicial($M, operadores\_mutacion$)
$Matriz\_Ejecucion \longleftarrow$ Ejecutar_Mutantes($P_t, original$)
Evaluar_Aptitud($Matriz\_Ejecucion$)
$m \longleftarrow$ Contar_Mutantes(P_t)
$BD\_Equivalentes \longleftarrow$ Determinar_Equivalentes(P_t)
$t \longleftarrow 0$
Mientras $((t < G)||(m < NumMutantes))$
  $t \longleftarrow t + 1$
  Mientras $(|P_t| < NM)$
    $ind \longleftarrow$ Generar_Individuo_Nuevo()
    Introducir_Individuo(P_t, ind)
  Mientras $(|P_t| < M)$
  Si $(op =$ CRUCE$)$ **entonces**
    $[padre_1, padre_2] \longleftarrow$ Seleccionar_Padres(P_{t-1})
    $[hijo_1, hijo_2] \longleftarrow$ Cruzar($padre_1, padre_2$)
    Introducir_Individuo($P_t, hijo_1$)
    Introducir_Individuo($P_t, hijo_2$)
  Si no
    $ind \longleftarrow$ Seleccionar_Individuo(P_{t-1})
    $mutante \longleftarrow$ Mutar(ind)
    Introducir_Individuo($P_t, mutante$)
  $Matriz\_Ejecucion \longleftarrow$ Ejecutar_Mutantes($P_t, original$)
  Evaluar_Aptitud($Matriz\_Ejecucion$)
  $m \longleftarrow m +$ Contar_Mutantes(P_t)
  $BD\_Equivalentes \longleftarrow$ Determinar_Equivalentes(P_t)
Devolver m

Uno de los parámetros que también se utilizan en la BGM es el número de mutantes totales que posee la composición original WS-BPEL. A partir de la salida del analizador es posible determinar el número de mutantes totales que pueden obtenerse por operador, y con ello el número de mutantes

totales del programa. De este modo, para el operador x, si I_x es el número de instrucciones donde puede aplicarse y A_x es el número de atributos que posee dicho operador, el número total de mutantes asociado a dicho operador es $I_x A_x$. Por tanto, el número total de mutantes de la composición original WS-BPEL es:

$$NumMutantes = \sum_{i=1}^{O} I_i A_i \tag{8.8}$$

donde O es el número total de operadores de mutación aplicables a la composición WS-BPEL.

En el algoritmo 8.1 podemos ver que tras analizar el fichero con la composición original se calcula el número de mutantes totales, según la ecuación (8.8). Con este número se establece uno de los criterios de parada del algoritmo en base al número de mutantes que se deben de generar según el porcentaje P especificado.

Cada vez que se invoca al procedimiento de ejecución de los mutantes, éste devuelve la matriz de ejecución (8.2), que permite realizar la evaluación de la aptitud de los individuos.

En cada generación del algoritmo se realizan dos pasos bien diferenciados. En primer lugar se generan individuos nuevos que se introducen en la nueva población. El número de individuos vendrá dado por el porcentaje de la población, N, que debe generarse aleatoriamente, indicado como parámetro de entrada. El segundo paso consiste en completar la población mediante la realización de las operaciones de cruce y mutación. Una vez completada la población, se procede a calcular la aptitud de los individuos. Tras ello se determina cuántos mutantes nuevos han sido generados en la población y cuáles son los mutantes potencialmente equivalentes.

8.4.3.7. Conversor de individuo a mutante

Este componente recibe los tres campos que componen la codificación de un individuo del AG, y produce el fichero WS-BPEL del mutante asociado a dicha codificación. Para realizar esta conversión, se utiliza el campo *Operador* del individuo para determinar la hoja de estilos XSLT que se va a emplear. Esta hoja implementa la transformación requerida para generar el mutante WS-BPEL a partir de la composición original. Estas hojas de estilos son las mismas que intervienen en el analizador, pero se ejecuta una zona de código diferente.

Cada hoja de estilo XSLT sólo recibe como entrada los campos *Instrucción* y *Atributo*. Después de recrear la lista original de operandos de la composición original WS-BPEL, se accede al elemento n-ésimo, según especifique el campo *Instrucción*, con objeto de obtener la referencia exacta al nodo que vaya a transformarse. Esta referencia es única: no hay dos nodos que

tengan la misma referencia, incluso si tienen exactamente el mismo contenido.

Usando esta referencia y el campo *Atributo*, se realiza una segunda pasada por la definición del proceso WS-BPEL. Sin embargo, en este recorrido, en lugar de obtener los operandos, se realiza la mutación indicada. Para ello, en cada nodo que se está recorriendo se verifica si ese nodo es el que se está referenciando en la codificación del individuo. En caso afirmativo, se realiza la mutación; en caso contrario, copiamos el elemento XML y sus atributos, y se continúa con el recorrido de forma recursiva en el árbol. La figura 8.10 muestra este esquema de transformación.

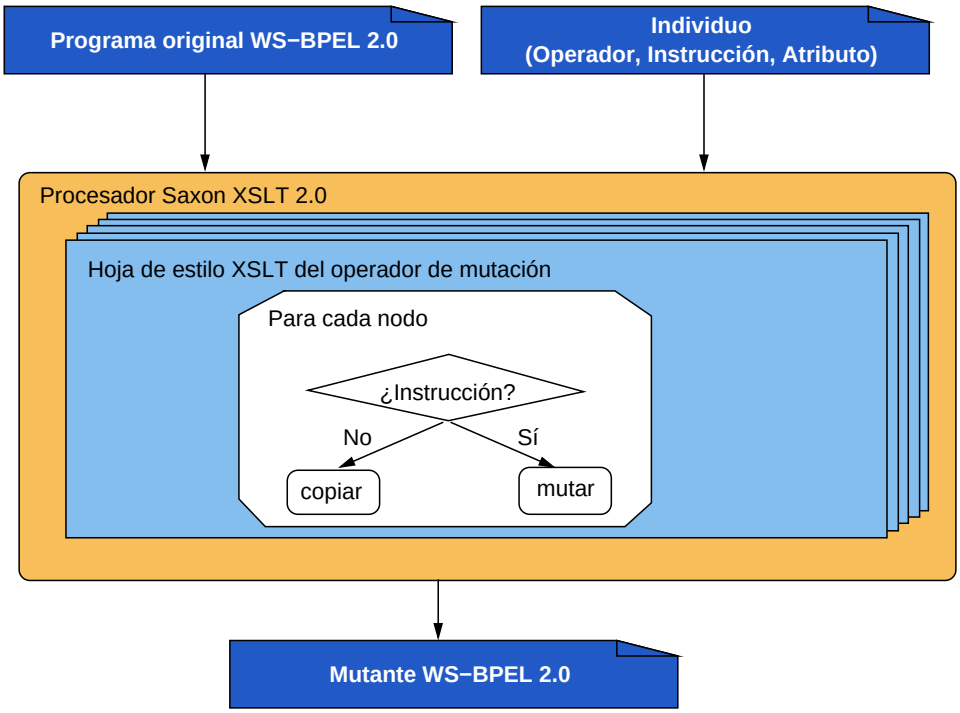


Figura 8.10: Proceso de conversión de individuo a mutante

El campo *Atributo* sólo se utiliza en el proceso de transformación en el caso de ser necesario, como se explicó en la definición de la representación del individuo del AG en el apartado 8.4.3.1. Sin embargo, se ha tenido especial cuidado en la implementación del conversor para que no produzca mutantes equivalentes sintácticamente. Así, es imposible por diseño, si se aplica el operador de mutación ERR sustituir, por ejemplo, el operador relacional $>$ por él mismo.

8.4.4. El sistema de ejecución

El *sistema de ejecución* es tanto una simplificación como una extensión del empleado en la herramienta Takuan [125]. La figura 8.11 muestra los diferentes pasos del sistema de ejecución. Podemos ver que este sistema realiza el proceso en tres pasos, a los que hemos denominado paso de empaquetamiento, paso de ejecución y paso de comparación. A continuación pasamos a detallar dichos pasos.

8.4.4.1. El paso de empaquetamiento

En este paso, se prepara al mutante para que pueda ser desplegado en el motor WS-BPEL 2.0 ActiveBPEL 4.1 [2] que será donde se ejecute. La definición del proceso no se cambia. Sólo se realiza un análisis del mismo para poder producir los ficheros específicos del motor necesarios para su despliegue. Estos ficheros son empaquetados en un único archivo, el cuál será enviado en el paso de ejecución al ActiveBPEL deployment WS.

Este paso cumple con dos objetivos: primero, se evita a los desarrolladores el problema de tener que conocer los detalles particulares de implementación de ActiveBPEL. En segundo lugar, usando algunas de las características de ActiveBPEL es posible sustituir la dirección de algunos de los WS invocados en la composición con WS modelados en el sistema como *mockups*. Esto es muy útil para la realización de pruebas, dado que no requiere que todos los WS estén disponibles, o bien porque podemos comprobar el comportamiento de la composición bajo determinadas condiciones.

8.4.4.2. El paso de ejecución

Con los ficheros específicos del motor generados en el paso previo, el mutante puede ser ejecutado. Para ello se realizan lo siguientes pasos:

1. se despliega el proceso WS-BPEL que representa el mutante,
2. se invoca al mutante con cada uno de los casos de prueba y se recoge los mensajes de respuestas,
3. y se finaliza el proceso WS-BPEL.

Estas tareas son realizadas en GAmera mediante BPELUnit [96], una biblioteca de pruebas unitarias para WS-BPEL que utiliza ficheros XML para describir los casos de prueba.

Aquellos WS externos cuyas direcciones fueron cambiadas en el paso previo provocará que se llamen a los *mockups* implementados por la biblioteca de pruebas BPELUnit. Estos *mockups* pueden introducir ciertos retrasos en la ejecución y responder con unos mensajes SOAP predefinidos (incluyendo mensajes de error). Para ello, BPELUnit se configurará de acuerdo a la especificación de casos de prueba.

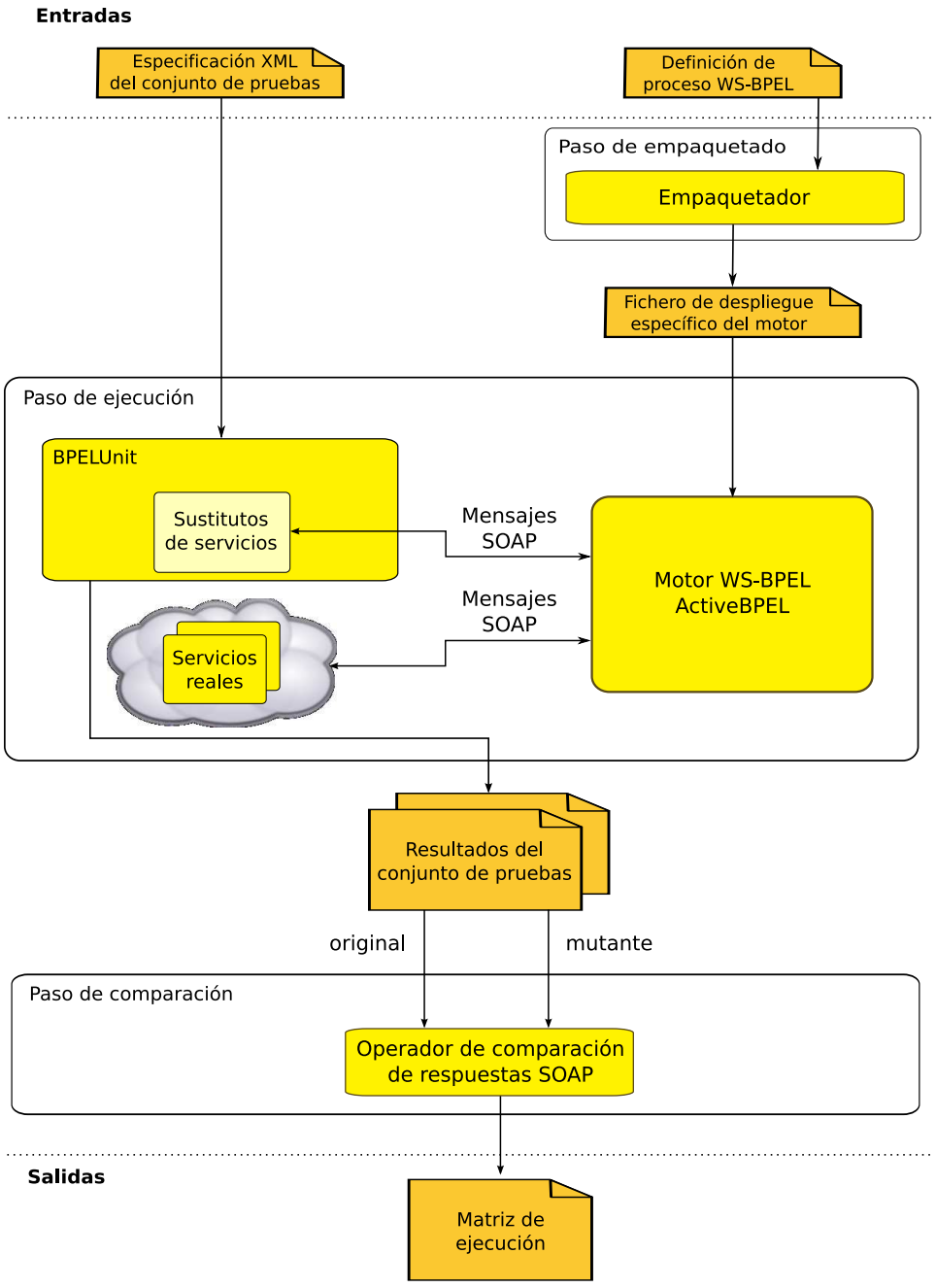


Figura 8.11: El sistema de ejecución

Con objeto de evitar la ejecución repetida del proceso original, guardaremos los resultados de ésta en un fichero con formato XML. No necesitamos guardar los resultados de cada mutante, dado que éstos, una vez que sean utilizados en el siguiente paso de comparación, serán eliminados.

8.4.4.3. El paso de comparación

En el paso previo, hemos obtenido las salidas de cada mutante producidas por cada caso de prueba. A continuación se realiza la comparación de estas salidas con las del programa original, con objeto de determinar si el mutante está muerto o sigue vivo.

Para ello se define un *operador de comparación*. Éste simplemente realiza una comparación estricta uno a uno de los mensajes de respuestas SOAP. Estos mensajes de respuestas no necesitan indicar éxito en su ejecución: si el proceso original WS-BPEL falló en la ejecución, el mutante sólo permanecerá vivo si también produce el mismo fallo. Por contra, si el proceso original WS-BPEL finalizó correctamente, el mutante también tendrá que completar su ejecución, y además deberá de producir el mismo resultado que el original para que siga vivo.

A partir de estas comparaciones, podemos producir la matriz de ejecución (8.2): para cada fila, establecemos a 1 la posición correspondiente al primer caso donde el mensaje SOAP del mutante es diferente al del proceso original. El resto de celdas se establecen a 0. Esta matriz nos permitirá calcular usando la ecuación (8.5) la aptitud de cada individuo.

8.5. Resultados con GAmera

En este apartado se procederá a la presentación de los resultados obtenidos en la aplicación de GAmera a diversas composiciones de servicios WS-BPEL.

Dado que GAmera tiene como objetivo la optimización del número de mutantes generados mediante el empleo de la BGM, las pruebas han estado centrada en la determinación del número mínimo de mutantes que se necesitan generar para una composición sin pérdida de información relevantes. Para ello se compararán los resultados obtenidos relativos al número de mutantes muertos y el número de mutantes potencialmente equivalentes generados en relación al número total de mutantes posibles existentes en la generación.

8.5.1. Composición Loan-Approval

El primer ejemplo de uso de GAmera se realizará con la composición *loan-approval* que aparece en la figura 8.3.

La salida del analizador obtenida para ese ejemplo, nos indica que sólo se pueden aplicar 6 de los 26 operadores de mutación definidos para WS-BPEL. La tabla 8.3 muestra cuáles son estos operadores, así como el número de atributos de cada uno y el número de instrucciones donde pueden aplicarse.

	ERR	ECN	ASF	AIE	ASI	XRF
Instrucción	2	1	2	2	3	1
Atributo	6	4	1	1	1	1

Tabla 8.3: Operadores de mutación para el proceso *loan-approval*

A partir de esta salida, es posible determinar el número de mutantes totales que pueden obtenerse por operador, y con ello el número de mutantes totales del programa, de acuerdo a la fórmula (8.8). En el caso del *loan-approval* es posible generarse hasta un total de 22 mutantes. Hay que tener en cuenta que para el operador ERR disponemos de 6 atributos, pero dado que el programa original incluye uno de esos valores, sólo se pueden considerar cinco atributos para la generación de mutantes.

La tabla 8.4 muestra los diferentes experimentos realizados junto a los parámetros que han ido variando, el porcentaje de mutantes generados (P) y el porcentaje de nuevos individuos generados aleatoriamente entre generaciones del AG (N). En todos los casos, se han establecido como parámetros fijos los siguientes:

- Tamaño de población: $M = 5$ individuos.
- Número de generaciones: $G = 25$ generaciones.
- Probabilidad de cruce: $p_c = 0{,}7$.

Podemos ver que para $N = 50\,\%$, generando un 80 %, 70 %, y 60 % del total de mutantes posibles, se detectan dos mutantes potencialmente equivalentes. Sin embargo, generando el 50 % de los mutantes posibles, sólo se detecta un único mutante potencialmente equivalente.

Por otro lado, para $N = 25\,\%$, se detectan los dos mutantes potencialmente equivalentes cuando generamos el 80 % y el 70 % del número total de mutantes.

Para poder comprobar la posible pérdida de información sufrida en la generación selectiva realizada por GAmera, se han generado previamente el 100 % de los mutantes posibles de esta composición. Con el mismo conjunto de casos de prueba, se han detectado dos mutantes potencialmente equivalentes, y 20 mutantes han muerto, es decir, un 90.9 % de mutantes muertos. A la vista de estos resultados, podemos ver que GAmera es capaz, no sólo de generar un subconjunto de mutantes del conjunto total, sino que puede detectar todos los mutantes potencialmente equivalentes.

P	N	Generaciones reales	Equiv.	Muertos	Muertos (%)
80 %	50 %	18	2	16	88.9 %
70 %	50 %	10	2	14	87.5 %
60 %	50 %	7	2	11	84.6 %
50 %	50 %	5	1	10	90.9 %
80 %	25 %	17	2	16	88.9 %
70 %	25 %	12	2	14	87.5 %
60 %	25 %	8	1	12	92.3 %
50 %	25 %	7	1	11	91.7 %

Tabla 8.4: Resultados de GAmera para *loan-approval*

Una vez determinada la existencia de mutantes potencialmente equivalentes, se examinaron manualmente para determinar si dichos mutantes eran equivalentes, o bien si el conjunto de casos de prueba no era suficiente para poderlos detectar. Pudimos constatar que dichos mutantes no eran equivalentes, y añadiendo un nuevo y apropiado caso de pruebas al conjunto inicial eran matados, alcanzando el 100 % de mutantes muertos. Por tanto, GAmera no sólo es capaz de generar mutantes, sino que la detección de mutantes potencialmente equivalentes permite incluso mejorar la calidad del conjunto inicial de casos de prueba.

Una característica importante de GAmera es que la mutación selectiva no se centra en un único operador, sino que la BGM, a través de la aptitud de los individuos va determinando qué operadores son los más adecuados. La tabla 8.5 muestra cuáles son los operadores de mutación que poseen los distintos mutantes generados por GAmera en las distintas ejecuciones. Podemos ver que cuando se genera un 80 % de los mutantes totales, se emplean todos los operadores de mutación de la composición WS-BPEL original. Sin embargo, para otros porcentajes, sólo los operadores XRF y ASI no se utilizan.

8.5.2. Composición MetaSearch

La composición *MetaSearch* es otro ejemplo WS-BPEL donde comprobaremos el funcionamiento de la herramienta GAmera desarrollada en esta tesis. Esta composición implementa una búsqueda múltiple en diversos motores de búsqueda de Internet. En concreto, el proceso *MetaSearch* realiza una petición a Google y MSN, combina los resultados de ambos motores, elimina los duplicados, y devuelve el resultado al cliente. Mayer [95] realiza una descripción detallada del funcionamiento de la composición.

La salida del analizador obtenida para esta composición, nos indica que es posible aplicar 11 operadores de mutación definidos para WS-BPEL. La

P	N	ERR	ECN	ASF	AIE	ASI	XRF
80	50	7	4	2	2	2	1
80	25	7	3	2	2	3	1
70	50	7	4	2	2	1	0
70	25	6	3	2	2	3	0
60	50	6	3	2	2	0	0
60	25	3	3	2	2	3	0
50	50	5	2	2	2	0	0
50	25	2	3	2	2	3	0

Tabla 8.5: Operadores de mutación usados en la generación de mutantes

tabla 8.6 muestra cuáles son estos operadores, así como el número de atributos de cada uno y el número de instrucciones donde pueden aplicarse. Podemos ver que, según la fórmula (8.8), en esta composición disponemos de 488 mutantes.

Operador	ISV	EAA	ERR	ELL	ECC	ECN	ASF	AIE	AWR	ASI	XRF
Instrucción	45	6	16	4	42	25	17	12	2	26	1
Atributo	4	4	5	1	1	4	1	1	1	1	1

Tabla 8.6: Operadores de mutación para el proceso *MetaSearch*

Para poder comprobar la calidad de los resultados obtenidos con la herramienta GAmera, inicialmente hemos obtenido el 100 % de mutantes y ejecutados éstos con el conjunto de casos de prueba inicial. Estos casos son los que vienen reflejados en [95]. De los 488 mutantes disponibles para esa composición, 332 son clasificados como mutantes muertos y 156 siguen vivos, es decir, son potencialmente equivalentes, lo que supone un 68,03 % de mutantes muertos con el conjunto de casos de prueba inicial. La tabla 8.7 muestra la distribución de mutantes vivos y muertos por cada tipo de operador empleado.

	ISV	EAA	ERR	ELL	ECC	ECN	ASF	AIE	AWR	ASI	XRF
Muertos	168	23	57	3	1	43	6	4	2	24	1
Pot. Equival.	12	1	23	1	41	57	11	8	0	2	0

Tabla 8.7: Distribución de mutantes entre los operadores de mutación para el proceso *MetaSearch*

Se han realizado diversos experimentos con diversos porcentajes hasta alcanzar un máximo del 90 % de mutantes posibles con GAmera. Los parámetros empleados en dichas ejecuciones se pueden ver en la tabla 8.8 muestra las características de éstos. Destacar el hecho de que 41 de los 42

mutantes existentes para el operador ECC son potencialmente equivalentes tras la ejecución del conjunto de casos inicial.

Ejecución	Tam. Población (M)	p_c	p_m	Nuevos Individuos (N)
E1	60	0.7	0.3	50 %
E2	60	0.7	0.3	25 %
E3	30	0.7	0.3	50 %
E4	30	0.7	0.3	25 %
E5	60	0.5	0.5	25 %
E6	60	0.5	0.5	25 %

Tabla 8.8: Parámetros de GAmera en las ejecuciones de *MetaSearch*

La tabla 8.9 muestra para cada experimento realizado, el porcentaje de mutantes generados (P), los mutantes muertos y los potencialmente equivalentes. Para poder conocer la posible pérdida de información que pudiésemos tener generando un subconjunto de mutantes, nos fijaremos en los mutantes potencialmente equivalentes. Así, destacamos aquellas ejecuciones que alcanzan el mayor número de mutantes potencialmente equivalentes en cada porcentaje.

Podemos ver que para las ejecuciones con un tamaño de población más pequeño (E3 y E4), se obtienen el mayor número de mutantes potencialmente equivalentes. Destacar que generando pocos mutantes (entre el 50 % y el 60 %) la diferencia en el número de mutantes potencialmente equivalentes con las otras ejecuciones es mayor, entre el 5 % y el 10 %.

Sin embargo, la variación del parámetro N del número de individuos nuevos a generar en cada generación no produce una alteración significativa en los resultados, destacando en este sentido la estabilidad de GAmera.

Destaca el hecho de que las ejecuciones E5 y E6, caracterizadas por poseer una probabilidad de cruce igual a la de mutación, son las que obtienen los peores resultados. En este sentido, los experimentos nos confirman que es deseable mantener una probabilidad de cruce siempre superior a la de mutación.

En esta composición, al contrario que en la composición *loan-approval*, no se consigue generar el 100 % de los mutantes potencialmente equivalentes existentes con el conjunto de casos de prueba inicial. Para poder comprobar la efectividad de los mutantes generados, vamos a analizar aquellos mutantes potencialmente equivalentes que quedan por generar, con objeto de determinar la posible pérdida de información del resultado de GAmera.

La tabla 8.10 muestra la distribución de los mutantes que se obtienen para la ejecución E3 cuando obtenemos el 90 % de los mutantes equivalentes. Esta ejecución es la que obtiene un mayor número de mutantes potencialmente equivalentes, 153, frente a los 156 existentes con el conjunto de

Ejecución	P	Equiv.	Muertos	Muertos (%)
E1	90 %	149	290	66,06 %
	80 %	140	251	64,19 %
	70 %	**126**	216	63,16 %
	60 %	106	187	63,82 %
	50 %	85	159	65,16 %
E2	90 %	152	288	65,45 %
	80 %	**144**	247	63,17 %
	70 %	**126**	216	63,15 %
	60 %	109	184	62,80 %
	50 %	87	157	64,34 %
E3	90 %	**153**	286	65,14 %
	80 %	135	256	65,47 %
	70 %	**126**	216	63,15 %
	60 %	107	186	63,48 %
	50 %	84	160	65,57 %
E4	90 %	150	289	65,83 %
	80 %	137	254	64,96 %
	70 %	124	218	63,75 %
	60 %	**113**	180	61,43 %
	50 %	**98**	146	59,83 %
E5	90 %	149	291	66,14 %
	80 %	140	251	64,19 %
	70 %	123	219	64,03 %
	60 %	106	187	63,82 %
	50 %	93	151	61,88 %
E6	90 %	148	292	66,36 %
	80 %	138	253	64,70 %
	70 %	124	218	63,74 %
	60 %	107	186	63,48 %
	50 %	82	162	66,39 %

Tabla 8.9: Resultados de GAmera en la composición *MetaSearch*

casos de prueba inicial.

	ISV	EAA	ERR	ELL	ECC	ECN	ASF	AIE	AWR	ASI	XRF
Muertos	126	23	55	3	1	41	6	4	2	24	1
Pot. Equival.	11	1	23	1	41	55	11	8	0	2	0

Tabla 8.10: Mutantes generados con GAmera para *MetaSearch* en la ejecución E3

La tabla 8.11 muestra los mutantes potencialmente equivalentes no generados en esta ejecución. De los 3 mutantes potencialmente equivalentes, uno corresponde al operador ISV, y dos al operador ECN. Destacar el hecho de la existencia de un mutante que es exactamente igual al proceso original, en concreto, el mutante $(7, 3, 3)$. Esto es debido al diseño del propio operador de mutación ECN, que dispone de dos atributos para añadir y eliminar un dígito. Nuestra implementación ha consistido en dividir y multiplicar por 10 el número, con objeto de añadir y eliminar un 0 por la parte menos significativa del número, es decir, el número 10 se puede transformar en 1 ó en 100. Este diseño hace que si la composición WS-BPEL que se está probando dispone de la constante numérica 0, las operaciones de añadir y eliminar un dígito hacen que el mutante sea el mismo proceso original. En este sentido, GAmera permite incluso determinar la eficacia y calidad de los distintos operadores de mutación diseñados para WS-BPEL.

Mutante	Descripción
(7, 2, 3)	Muere con otro caso de prueba diseñado para matar a otros mutantes potencialmente equivalentes del operador ECN
(7, 3, 3)	Igual al proceso original
(1, 38,1)	Mutante equivalente

Tabla 8.11: Mutantes no generados en la ejecución E3

Por tanto, ninguno de los mutantes potencialmente equivalentes que no se han generado producen pérdida de información. Por tanto, los experimentos nos confirman que GAmera no sólo es capaz de generar mutantes, sino que la selección realizada en la generación de mutantes no supone ninguna pérdida de información para el proceso de pruebas.

La tabla 8.12 muestra la distribución de los mutantes que se obtienen para la ejecución E1 cuando obtenemos el 90 % de los mutantes equivalentes. En esta ejecución, no se generan 7 mutantes potencialmente equivalentes; en concreto, dos del operador ISV, uno del operador ERR y cuatro del operador ECN. Analizando dichos mutantes, se constata que 2 corresponden al programa original, de manera similar al mutante $(7, 3, 3)$ de la ejecución E3, mientras que los otros eran muertos con casos de prueba di-

señados para matar otros mutantes potencialmente equivalentes. Por tanto, ninguno de los mutantes potencialmente equivalentes que no se han generado producen pérdida de información. De este modo, los experimentos nuevamente nos confirman que GAmera genera mutantes sin pérdida de información para el proceso de pruebas.

	ISV	EAA	ERR	ELL	ECC	ECN	ASF	AIE	AWR	ASI	XRF
Muertos	126	23	57	3	1	43	6	4	2	24	1
Pot. Equival.	10	1	22	1	41	53	11	8	0	2	0

Tabla 8.12: Mutantes generados con GAmera para *MetaSearch* en la ejecución E1

Una característica importante de GAmera es que la mutación selectiva no se centra en un único operador. La tabla 8.13 muestra para las ejecuciones E1 y E3, diferenciándose éstas en el tamaño de la población, la evolución en el número de mutantes generados por cada tipo de operador. En la tabla se han resaltado aquellos instantes donde se alcanza el 100 % de los mutantes de un determinado operador. Podemos ver cómo desde que obtenemos el 10 % de los mutantes generados, ya disponemos de representantes de todos los operadores de mutación que se pueden aplicar a la composición. Incluso con el 10 % generado, existen dos operadores, ELL y XRF, que han conseguido el 100 % de sus mutantes posibles, en ambas ejecuciones.

Durante las pruebas efectuadas hemos podido comprobar que en pocas generaciones la BGM que se encuentra dentro de GAmera permite obtener un elevado número de mutantes. Así, la figura 8.12 muestra para las ejecuciones de la tabla 8.8 la relación entre el número de generaciones realizada y el porcentaje de mutantes generados. Podemos ver que, las ejecuciones que requieren un mayor número de generaciones, E3 y E4, son aquéllas que poseen un tamaño más pequeño de población. Esto es debido a que necesitan producir más individuos para poder cubrir el mismo espacio de soluciones que las ejecuciones con tamaño de población mayor. Por otro lado, es de destacar cómo con pocas generaciones se consigue un alto porcentaje de mutantes generados. Así, con unas 50 generaciones se obtiene el 70 % de los mutantes posibles, salvo en la ejecución E4 que necesita de 98 generaciones para obtener el mismo número de mutantes.

8.6. Conclusiones

La aparición de los WS y las composiciones de servicios obliga a crear nuevas técnicas de prueba adaptadas a las características de éstos. En este sentido, se ha desarrollado una herramienta basada en AGs, GAmera, para la generación de mutantes de composiciones de servicios en WS-BPEL.

	10 %	20 %	30 %	40 %	50 %	60 %	70 %	80 %	90 %
Ejecución E1									
ISV	9	14	26	43	60	68	83	106	136
EAA	5	9	12	17	20	21	23	**24**	24
ERR	5	13	20	33	39	51	62	73	79
ELL	**4**	4	4	4	4	4	4	4	4
ECC	7	13	18	23	30	34	41	**42**	42
ECN	7	15	29	33	40	58	71	83	96
ASF	7	10	15	16	16	**17**	17	17	17
AIE	1	5	9	10	11	**12**	12	12	12
AWR	1	**2**	2	2	2	2	2	2	2
ASI	1	6	20	13	21	24	25	**26**	26
XRF	**1**	1	1	1	1	1	1	1	1
Ejecución E3									
ISV	11	17	28	42	56	62	76	109	137
EAA	4	9	16	21	22	**24**	24	24	24
ERR	4	17	28	37	50	59	73	73	78
ELL	**4**	4	4	4	4	4	4	4	4
ECC	7	12	14	20	26	33	41	41	**42**
ECN	5	13	19	26	33	53	81	81	96
ASF	7	8	14	15	16	**17**	17	17	17
AIE	2	6	11	**12**	12	12	12	12	12
AWR	1	**2**	2	2	2	2	2	2	2
ASI	2	3	9	15	22	25	**26**	26	26
XRF	**1**	1	1	1	1	1	1	1	1

Tabla 8.13: Evolución de los mutantes generados según el tipo de operador de mutación

La principal aportación de esta herramienta es la BGM que emplea un AG para la generación de mutantes sin necesidad de generar todos los mutantes posibles. Las propias características del AG permiten seleccionar un subconjunto de mutantes de acuerdo a una función de aptitud, reduciendo de este modo el coste computacional asociado a la prueba de mutaciones. Las pruebas realizadas, nos indican que sin necesidad de generar el 100 % de los mutantes mutantes, no se pierde información relevante para el proceso de pruebas. Así, en el caso del *loan-approval* es suficiente el 60 %, mientras que en el *MetaSearch* se requiere generar el 90 % de los mutantes.

Otra ventaja del sistema desarrollado es que permite detectar automáticamente los mutantes potencialmente equivalentes. Estos mutantes pueden ser realmente mutantes equivalentes, o simplemente indicarnos una baja calidad del conjunto inicial de casos de prueba. El estudio manual de

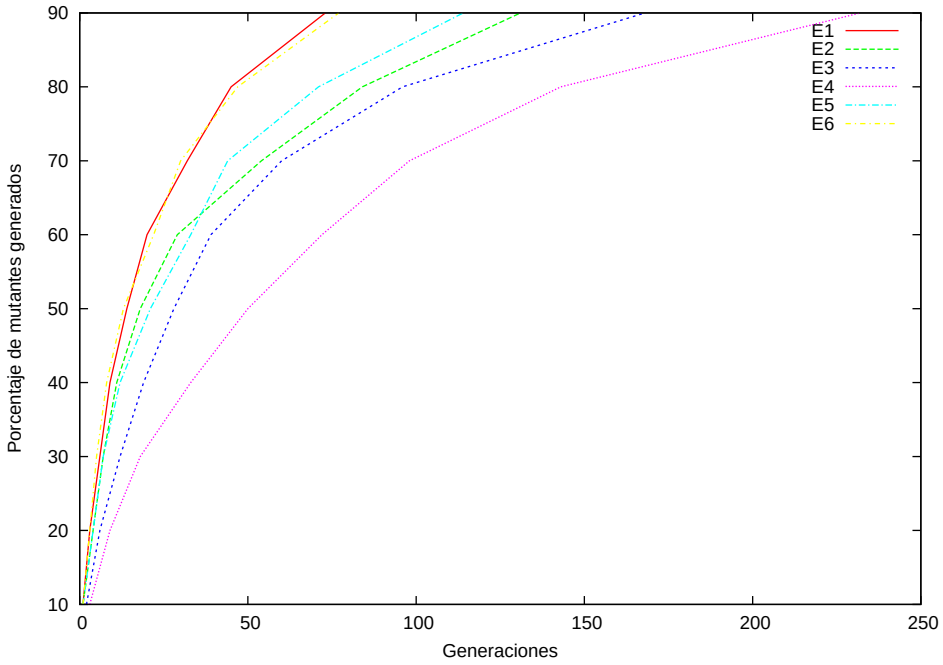


Figura 8.12: Evolución de las generaciones en GAmera

estos mutantes permitiría mejorar la calidad del conjunto inicial de casos de prueba.

En los experimentos realizados, hemos podido constatar que, en el caso del *loan-approval* se han detectado todos los mutantes potencialmente equivalentes. Éstos han permitido determinar que nuestro conjunto inicial de casos de prueba era deficitario, puesto que añadiendo un nuevo caso de pruebas, los mutantes eran muertos.

Por otro lado, en el ejemplo del *MetaSearch* se ha podido generar un porcentaje de mutantes detectando casi todos los mutantes potencialmente equivalentes y sin pérdida de información. Esto es debido a que aquellos mutantes potencialmente equivalentes no generados, eran o bien iguales a la composición original, o bien morían con un caso de prueba diseñado para matar a otro mutante potencialmente equivalente. En este sentido, GAmera no sólo ha permitido mejorar la calidad del conjunto inicial de casos de prueba detectando los mutantes potencialmente equivalente, sino que incluso ha podido determinar algunos comportamientos anómalos en los operadores de mutación.

Por tanto, GAmera no sólo es una herramienta capaz de generar mutantes, sino que la detección de los mutantes potencialmente equivalentes permite mejorar la calidad del conjunto de casos de prueba.

Además, los experimentos han podido comprobar que GAmera puede

incluso ayudar a la validación de los diferentes operadores de mutación diseñados para WS-BPEL. Así, hemos podido comprobar cómo el operador de mutación ECN genera mutantes que son iguales al programa original. Igualmente, el operador ECC genera en su mayoría mutantes potencialmente equivalentes. Los datos obtenidos reflejan la necesidad de realizar un estudio más detallado sobre el diseño de estos operadores.

Capítulo 9

Conclusiones

En este último capítulo de la tesis se describen las conclusiones obtenidas tras la realización de nuestra investigación sobre diferentes métodos de optimización y se exponen futuras líneas de trabajo relacionadas con el tema.

9.1. Conclusiones

En el capítulo 1 de esta tesis se describía como objetivo principal de la misma la aplicación de AGs a la optimización de funciones tanto en el ámbito de variables continuas como en los problemas de optimización combinatoria. En particular, se pretendía desarrollar diversas estrategias de resolución de problemas de optimización, tomando como base los AGs.

Para conseguir diseñar las nuevas estrategias se ha considerado como punto de partida la realización de sucesivas búsquedas acotadas, que permitan evolucionar hacia la consecución del óptimo de la función a optimizar.

En el trascurso de esta tesis se han desarrollado cuatro estrategias, dos en el plano de variables continuas, y otras dos en la optimización combinatoria.

9.1.1. Estrategias de optimización global

Búsqueda Lineal Genética

La primera estrategia desarrollada ha sido denominada Búsqueda Lineal Genética. Ésta toma como punto de partida los tradicionales métodos de descenso para la optimización de funciones multidimensionales. Sin embargo, incorpora como novedad la ampliación de la búsqueda lineal para calcular el tamaño del paso. Esta ampliación, no sólo consiste en una extensión en valores positivos, sino también incluso en valores negativos.

Las pruebas computacionales a las que hemos sometido la BLG, nos ha podido contrastar la robustez del método respecto al punto inicial escogido para iniciar el proceso de optimización, algo que depende en los métodos tradicionales.

La realización de la búsqueda lineal extendida ha permitido disminuir el número de iteraciones frente a las técnicas clásicas, ya que el realizar una búsqueda local extendida permite generar pasos más grandes, avanzando más rápidamente en cada iteración hacia el óptimo.

La BLG se ha apoyado en un AG unidimensional, donde los individuos codifican el paso a dar en cada iteración mediante una variable de coma flotante. Esto hace que presente pocos requisitos de almacenamiento, lo que facilita la aplicación a problemas con un gran número de variables, sin que los requisitos de memoria supongan un problema en su utilización.

Una ventaja considerable de la BLG respecto a otras técnicas clásicas, es que ha conseguido independizar la búsqueda del paso de la dirección de descenso. De este modo, la elección de una buena dirección de descenso no es un elemento crítico en la estrategia de optimización, al contrario que en las técnicas clásicas. En este sentido, la búsqueda lineal extendida que se realiza ha permitido descubrir rápidamente los óptimos globales, y escapar de los óptimos locales, a pesar de disponer de direcciones no tan óptimas. Esto hace que la BLG no imponga ninguna restricción a la función a optimizar, en cuanto a necesidad de conocer su derivada, dado que es posible enmarcar la BLG dentro de los métodos de descenso coordenado.

Se ha podido aplicar de forma efectiva, no sólo a diversas funciones clásicas en las pruebas de los métodos de optimización [89], sino que también hemos comprobado su eficacia en el entrenamiento de redes neuronales realimentadas [88].

Búsqueda Genética en Cajas

La segunda estrategia desarrollada en el ámbito de variables continuas ha sido denominada Búsqueda Genética en Cajas. Ésta trata de optimizar las búsquedas del AG, disminuyendo el campo de acción, con objeto de realizar una mejor exploración. Para ello realiza un proceso iterativo, donde las soluciones que va encontrando en cada paso, se convierte en el centro de una nueva región de búsqueda, a la que hemos denominado caja.

Una característica novedosa de este método es la incorporación de una memoria al AG. Ésta ha permitido que el AG no sólo tenga presente la diversificación dentro de los individuos que conforman la población actual, sino también presentar la diversificación en relación a las mejores soluciones que se han ido generando en las iteraciones anteriores. Esta memoria ha supuesto una ampliación del operador de nichos.

Las pruebas computacionales a las que hemos sometido la BGC, ha permitido comprobar el grado de robustez del método respecto al punto inicial escogido para iniciar el proceso de optimización. En este sentido, se ha ob-

tenido soluciones cercana al óptimo con independencia del punto inicial de partida, al contrario que los métodos tradicionales.

La incorporación de la memoria al AG ha permitido que la BGC pueda distinguir entre los óptimos locales y el óptimo global. Mientras este último es la solución final proporcionada por la BGC, los óptimos locales son todos aquellos puntos que se encuentran en la memoria del algoritmo, que coinciden con los centros de las sucesivas cajas de búsquedas exploradas durante el proceso de optimización.

Dado que la BGC únicamente utiliza el valor de la función a optimizar para determinar la aptitud de los individuos, el método que se ha desarrollado permite aplicarlo a cualquier tipo de función que se desee optimizar. Es decir, no se impone ninguna restricción a la función a optimizar, en cuanto a necesidad de conocer derivada, etc.

Se ha podido aplicar de forma efectiva a diversas funciones clásicas en las pruebas de los métodos de optimización [90]. En las pruebas efectuadas, hemos podido comprobar que los tiempos de computación son pequeños, y no sufren un incremento exponencial con el número de dimensiones de la función.

9.1.2. Estrategias de optimización combinatoria

Búsqueda Genética en Vecindades

La primera estrategia desarrollada en el ámbito de los problemas combinatorios, la hemos denominado Búsqueda Genética en Vecindades. Ésta consiste en adaptar la Búsqueda Genética en Cajas al problema combinatorio. Para ello, se sustituye el concepto de caja por vecindad. De este modo, se realizan sucesivas búsquedas locales de vecindades mediante un AG.

La BGV presenta dos características diferenciadoras en relación a los métodos de búsqueda local existentes. Por un lado, introduce el concepto de vecindad extendida, de manera que el AG realiza la exploración de dicha vecindad, considerando la mejor solución obtenida como el centro de una nueva vecindad extendida.

El segundo aspecto diferenciador de la estrategia que se propone es la evaluación de la aptitud de los individuos. Cada individuo codifica un conjunto de movimientos desde el punto central, determinado por el orden de la vecindad. La aptitud del individuo no será la solución que se representa tras la secuencia de movimientos, sino que será la mejor solución encontrada a lo largo de la trayectoria que representa.

Para la BGV se han diseñado cinco operadores, dos para cruce, y tres para mutación, que dará lugar a los nuevos individuos que aparecerán en cada generación. En este sentido, tanto el cruce dirigido por aptitud como la mutación dirigida por aptitud, son específicos para la BGV, dado que están asociados al concepto de aptitud asociado a un individuo.

Para verificar su efectividad, se han resuelto diversas instancias del problema del viajante simétrico [31]. Los resultados que hemos obtenido nos ha permitido comprobar el grado de robustez del método respecto al punto inicial escogido para iniciar el proceso de optimización.

Por otro lado, la parametrización del algoritmo no supone ningún inconveniente en la resolución de un problema de optimización. Las pruebas efectuadas nos han permitido constatar el alto grado de estabilidad frente a los diferentes parámetros que configuran su comportamiento [34].

Búsqueda Genética de Mutantes

La última propuesta ha consistido en el desarrollo de una herramienta para la generación automática de mutantes para composiciones de servicios en WS-BPEL, a la que hemos denominado GAmera. Esta aportación constituye una novedad en el campo de las pruebas de mutaciones, dado que es el primer generador de mutantes para composiciones WS-BPEL.

Una característica particular de GAmera es que la generación de mutantes se realiza mediante un AG, al que hemos denominado Búsqueda Genética de Mutantes. Este generador se caracteriza por generar un subconjunto de todos los mutantes posibles para una composición WS-BPEL. Este trabajo es también la primera vez que se emplean AG para la generación de mutantes [32].

La evaluación de la aptitud de los individuos constituye una aportación novedosa, puesto que permite buscar mutantes de un modo selectivo. En este sentido, las pruebas realizadas, nos indican que generando únicamente un subconjunto del total de mutantes, no se pierde información relevante para el proceso de pruebas [33].

Una característica del sistema desarrollado es que detecta los mutantes equivalentes potenciales, lo que nos permite comprobar si el conjunto de casos de prueba no dispone de una calidad suficiente.

Nuestra herramienta, GAmera, no sólo cumple con el objetivo inicial de generar automáticamente mutantes, sino que la detección de los mutantes equivalentes potenciales nos sirve para medir y mejorar la calidad del conjunto de casos de prueba inicial.

Por otro lado, la generación de mutantes cubre todos los operadores de mutación empleados en la composición original. Esto hace que la herramienta sirva para comprobar la validez de los operadores de mutación diseñados, así como medir la calidad de éstos.

9.2. Reflexiones sobre los AGs

A continuación haremos algunos comentarios generales sobre la experiencia que ha supuesto desarrollar el trabajo descrito en esta memoria y la

impresión que hemos obtenido a lo largo de él sobre cuáles son los puntos fuertes y débiles de los AGs.

- La simplicidad de los conceptos teóricos que subyacen sobre los AGs facilita la comprensión y entendimiento del proceso de optimización que suelen realizar. De este modo, la curva de aprendizaje inicial de los AGs, creemos que es poco pronunciada. No obstante, en nuestra experiencia consideramos que se debe dedicar al menos un año a adquirir la destreza necesaria para el diseño de nuevas estrategias de optimización basadas en AGs.

- Los AGs presentan un esquema de funcionamiento muy configurable. En este sentido, no sólo se pueden configurar las distintas operaciones de selección y reproducción a efectuar, sino también el modo en que se van generando las distintas poblaciones. Así, por ejemplo, en esta tesis se han desarrollado AGs con diferentes esquemas:

  - La BLG se implementa con un AG de estado permanente, donde cada generación únicamente produce un único individuo.
  - La BGC y la BGV se implementan con un AG generacional, donde un porcentaje de la nueva población es seleccionada de la población anterior, y el resto mediante operaciones de cruce y mutación.
  - La BGM también emplea un AG generacional, pero un porcentaje de la nueva población se crea con individuos creados aleatoriamente, y el resto mediante operaciones de cruce y mutación.

  No sólo es posible configurar el modo en que se realiza el esquema de generación de la nueva población, sino también cómo realizar las operaciones de reproducción. El uso habitual es la realización de operaciones de cruce, y sobre los individuos generados realizar la operación de mutación. Sin embargo, en esta tesis se ha optado por seguir un esquema independiente de los operadores. Así, cuando se realizan operaciones de reproducción, o bien se realiza un cruce, o bien una mutación, pero no ambas operaciones. La ventaja que se obtiene al seguir este esquema es la reducción de los parámetros de entrada, puesto que permite relacionar las probabilidades de ambos operadores.

- En cuanto a las operaciones a realizar en cada generación, aunque existe en la bibliografía un amplio abanico de operaciones de cruce y mutación para diferentes codificaciones, es posible diseñar nuevos operadores en el AG. Así, en el desarrollo de la tesis, se han diseñado los siguientes operadores:

  - La BGC incorpora un operador de memoria, donde se amplia el concepto de nicho para introducir un recordatorio de las generaciones anteriores.
  - La BGV presenta dos operadores diseñados para la codificación empleada: la mutación dirigida por aptitud, y el cruce dirigido por aptitud.

- Uno de los inconvenientes que poseen los AGs es el excesivo número de parámetros que suelen necesitar para poder configurar su comportamiento. En este sentido, un buen AG puede obtener pobres resultados si los parámetros no han sido establecidos correctamente. No obstante, suelen presentar un rango de valores para cada parámetro que permite dotar de cierto grado de estabilidad e independencia de los parámetros.

- La representación de los individuos para resolver un problema no es única. Esto hace que un mismo problema pueda ser resuelto con AGs que posean diferentes representaciones de la solución. Así, por ejemplo, la BLG ha sido resuelta con un AG unidimensional con individuos codificados en coma flotante, mientras que se podía haber planteado otra solución codificando los individuos en binario.

- La existencia de diferentes bibliotecas [30] para la programación de AGs permite facilitar el desarrollo de sistemas con AGs. No obstante, la mayor parte de estas bibliotecas suelen limitar el tipo de codificaciones que pueden emplearse en los individuos. Sin embargo, la incorporación de las principales estrategias de selección y operadores de reproducción, permite codificar rápidamente el AG.

- La determinación de la función de aptitud es un elemento clave en el AG. Aunque en algunos problemas, la determinación de la función de aptitud suele estar relacionada con el problema a resolver, en otros, debe pensar una estrategia que permita medir la calidad de la solución que representa. Así, mientras que la BLG, BGC y BGV presentan funciones de aptitud igual a la función a optimizar, la BGM ha requerido diseñar una función para poder valorar la calidad de los mutantes generados.

9.3. Líneas de investigación futuras

Una vez descrita las conclusiones obtenidas tras la realización de la tesis, se pueden realizar diferentes ampliaciones y líneas de investigación futuras. De los cuatro métodos propuestos, sólo GAmera permite ampliaciones e investigaciones futuras, de ahí que la mayor parte de las distintas propuestas estén relacionadas con ella:

- Dotar de una interfaz gráfica amigable a GAmera. Actualmente la herramienta sólo posee una interfaz en línea de órdenes, donde los distintos parámetros de configuración se proporcionan mediante ficheros. En este sentido, se propone mejorar el entorno de GAmera, de forma similar al de otras herramientas como Mothra [81], MuClipse [111] o MuJava [92].

- Mejorar la eficiencia de GAmera. La herramienta desarrollada actualmente no es especialmente eficiente y, en este sentido, es susceptible de diversas mejoras. Actualmente, cuando se ejecuta un mutante sobre un conjunto de casos de prueba, esta ejecución se realiza sobre todos los casos, independientemente de si es muerto o no. No obstante, la versión actual de GAmera sólo se tiene en cuenta el primer caso que mata al mutante para poder elaborar la matriz de ejecución (8.2).

  Sería conveniente que cuando el mutante sea muerto no continuar con la ejecución del mutante. Para poder realizar esta mejora, se requiere modificar BPELUnit que se encarga del paso de ejecución.

- Diseño de un generador de casos de prueba que permita realimentar a GAmera, el generador automático de mutantes desarrollado en esta tesis. Este generador de casos actuaría sobre los mutantes equivalentes potenciales, de manera que permitiese mejorar el conjunto de casos de prueba inicial. En este sentido, los trabajos [37, 47, 48, 124] recogen distintos modos de realizar casos de prueba en composiciones WS-BPEL.

- Análisis de la efectividad de los distintos operadores de mutación propuestos para el lenguaje WS-BPEL. Una vez realizada la generación de mutantes con GAmera para una composición, es posible comprobar el grado de efectividad de los distintos operadores planteados. En este sentido, se pueden usar las métricas propuestas por Mresa y Bottaci [110].

- Diseño de nuevos operadores de mutación para el lenguaje WS-BPEL que se ajusten a determinados criterios de cobertura, etc. Estos nuevos operadores se incorporarían a GAmera, modificándose el analizador y el AG encargado de la búsqueda de mutantes.

- Análisis de la calidad de distintos métodos de generación de conjuntos de casos de prueba mediante la comparativa de la puntuación de mutación (8.1) obtenida al aplicar dichos conjuntos a GAmera.

Parte IV

Anexos

Apéndice A

Funciones

En este capítulo se recogen las funciones empleadas para la verificación y comprobación de las técnicas descritas. Para cada una de ellas se aportará la fuente bibliográfica, su descripción, así como el óptimo de la misma y el valor de la función en el mismo. Además, se mostrará las gráficas de su superficie y contorno, que permitirán dar una visión de la dificultad de la localización del óptimo.

A.1. Función de Branin

Esta función bidimensional aparece en el artículo [75]. La definición de la misma es:

$$f(\bar{x}) = \left(\left(x_2 - \frac{5{,}1}{4 \times \pi^2}\right) \times x_1^2 + \frac{5 \times x_1}{\pi} - 6\right)^2 + 10 \times \left(1 - \frac{1}{8 \times \pi}\right) \times \cos(x_1) + 10 \tag{A.1}$$

El mínimo dentro del intervalo $[-5, 10] \times [0, 15]$ vale 0.397887, y se encuentra en los siguientes tres puntos: (-3.14159, 12.275), (9.42478, 2.275) y (3.14159, 2.275).

Las figuras A.1 y A.2 muestran la superficie y el contorno de la función de Branin en un intervalo $[-50, 50] \times [-50, 50]$. En ellas podemos apreciar cómo existe un valle en forma de "plátano". Si analizamos en más detalle el intervalo $[-5, 10] \times [0, 15]$ (ver figuras A.3 y A.4) que es donde se encuentran los óptimos, podemos ver cómo existen tres valles, uno por cada solución.

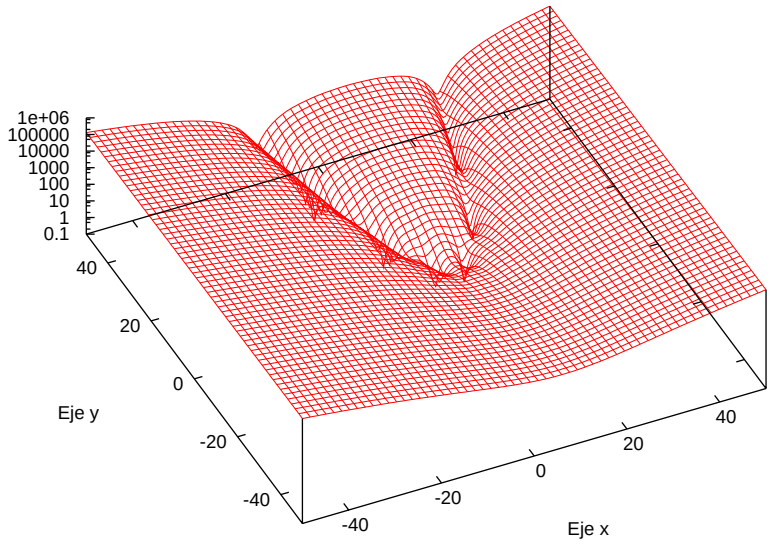


Figura A.1: Superficie de la función de Branin en el intervalo $[-50, 50] \times [-50, 50]$

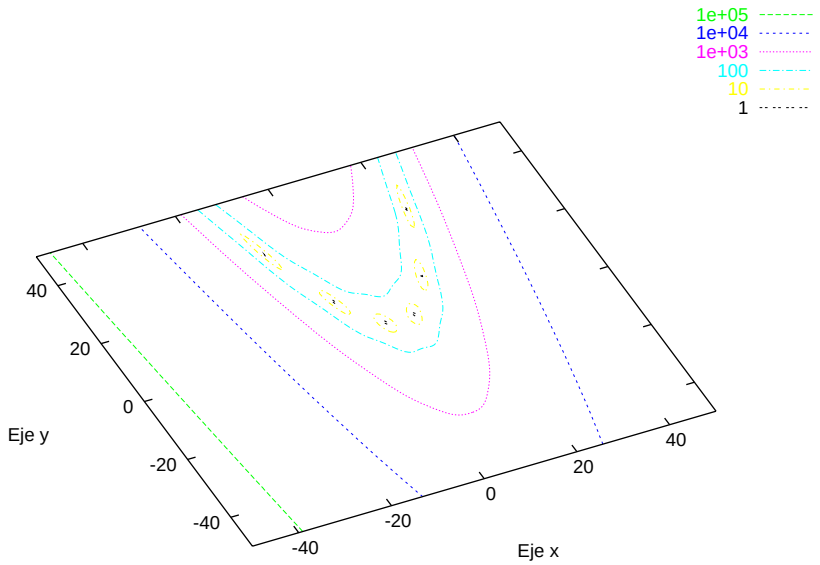


Figura A.2: Contorno de la función de Branin en el intervalo $[-50, 50] \times [-50, 50]$

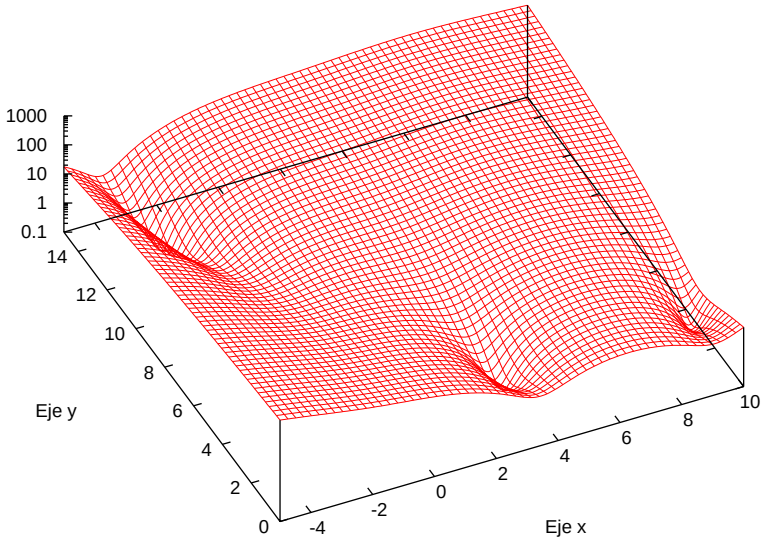


Figura A.3: Superficie de la función de Branin en el intervalo $[-5, 10] \times [0, 15]$

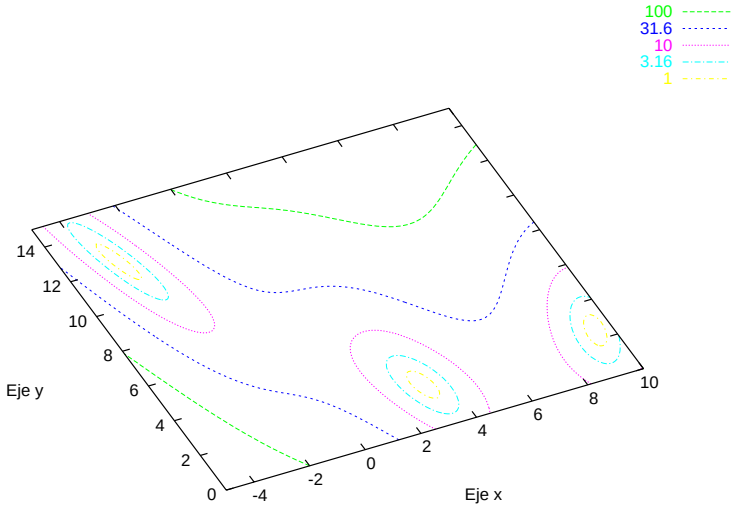


Figura A.4: Contorno de la función de Branin en el intervalo $[-5, 10] \times [0, 15]$

A.2. Función de Chichinadze

Esta función de dos dimensiones aparece en el artículo [75]. La definición de ésta es:

$$f(\bar{x}) = x_1^2 - 12 \times x_1 + 11 + 10 \times \cos\left(\frac{\pi}{2} \times x_1\right) + 8 \times \sin(5 \times \pi \times x_1) - \frac{1}{\sqrt{(5)}} \times \exp\left(\frac{-(x_2 - \frac{1}{2})^2}{2}\right) \tag{A.2}$$

El valor del mínimo dentro del intervalo $[-30, 30] \times [-10, 10]$ es igual a -43.315862, y se encuentra en el punto (5.90133, 0.5).

Las figuras A.5 y A.6 muestran la superficie y el contorno de la función de Chichinadze en un intervalo $[-30, 30] \times [-10, 10]$. En ellas podemos apreciar cómo existe un valle en la zona central de la superficie. Si analizamos en más detalle el intervalo $[0, 10] \times [-5, 5]$ (ver figuras A.7 y A.8) que es donde se encuentra el óptimo, podemos apreciar cómo la superficie está compuesta por una sucesión continuada de múltiples y pequeños valles a lo largo de la variable x_2.

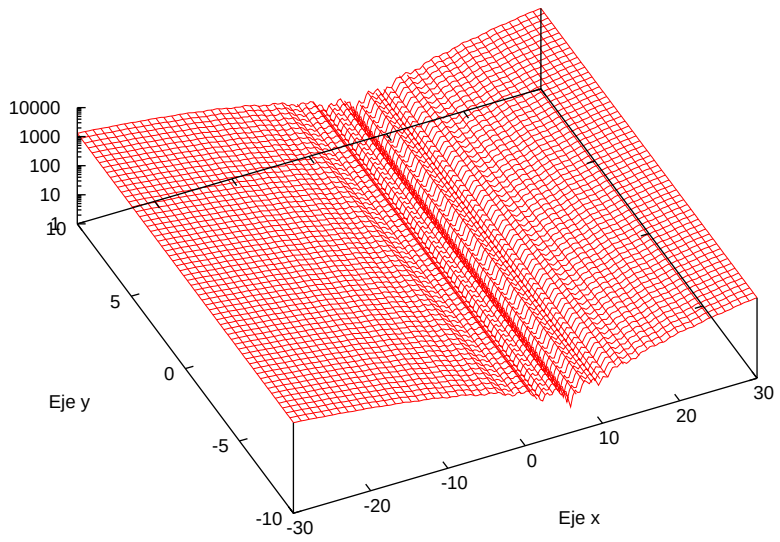


Figura A.5: Superficie de la función de Chichinadze en el intervalo $[-30, 30] \times [-10, 10]$

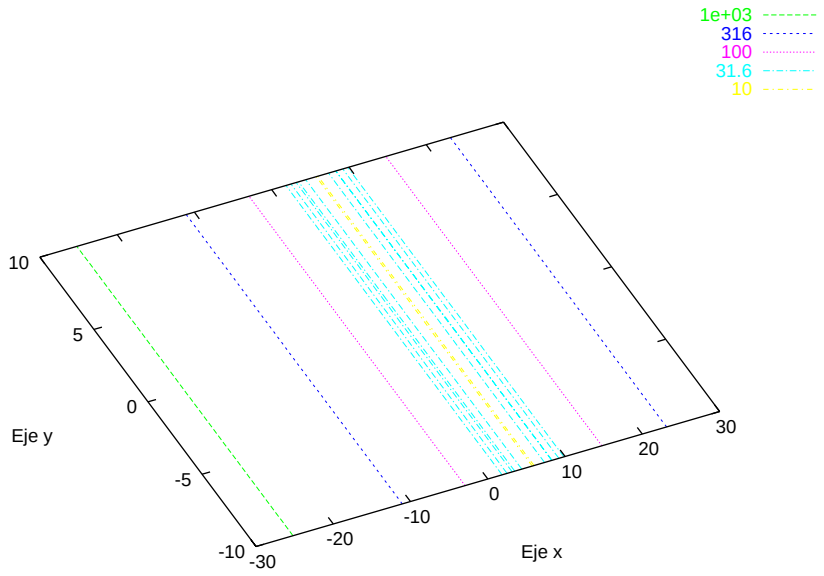


Figura A.6: Contorno de la función de Chichinadze en el intervalo $[-30, 30] \times [-10, 10]$

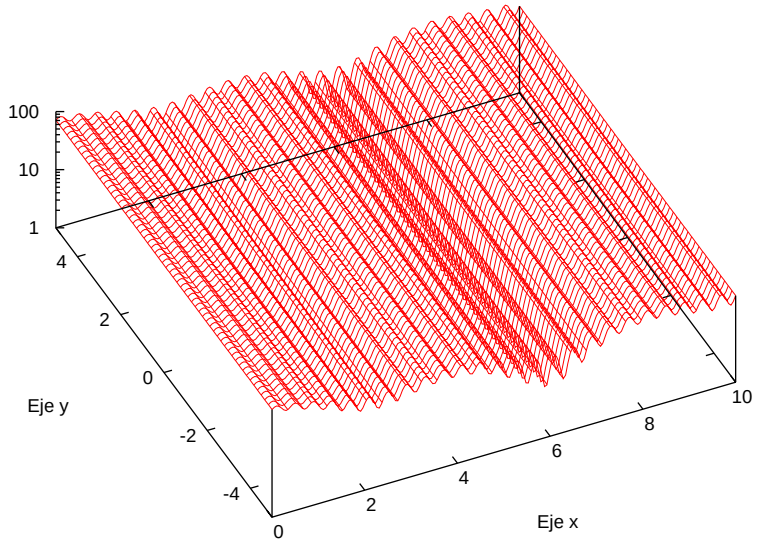


Figura A.7: Superficie de la función de Chichinadze en el intervalo $[0, 10] \times [-5, 5]$

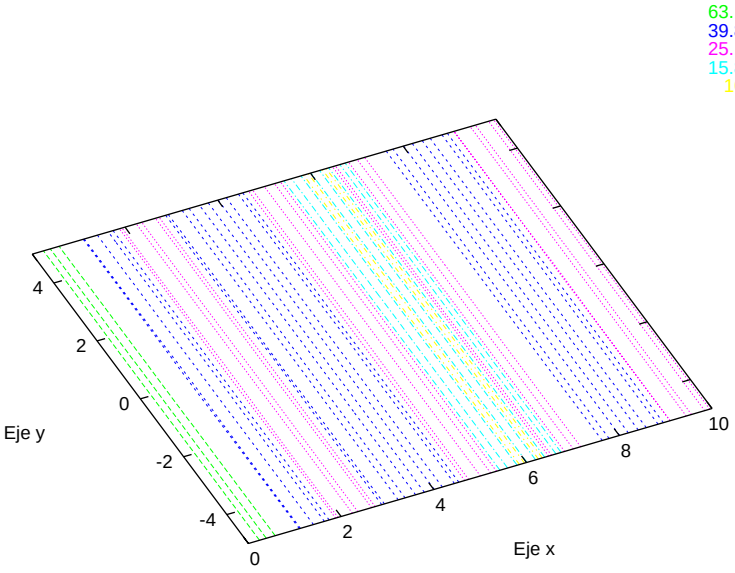


Figura A.8: Contorno de la función de Chichinadze en el intervalo $[0, 10] \times [-5, 5]$

A.3. Función de Griewank de 2 variables

Esta función de dos dimensiones aparece en el artículo [75]. La definición de ésta es:

$$f(\bar{x}) = \frac{x_1^2 + x_2^2}{200} - \cos(x_1) \times \cos\left(\frac{x_2}{\sqrt{2}}\right) + 1 \tag{A.3}$$

El punto mínimo dentro del intervalo $[-100, 100] \times [-100, 100]$ se encuentra en el punto (0,0) y vale 0.

Las figuras A.9 y A.10 muestran la superficie y el contorno de la función de Griewank de dos variables en el intervalo $[-100, 100] \times [-100, 100]$. En ellas podemos apreciar cómo existe un valle en la zona central de la superficie. Si analizamos en más detalle el intervalo $[-10, 10] \times [-10, 10]$ donde se encuentra el óptimo (figuras A.7 y A.8), podemos apreciar cómo la superficie está compuesta por una serie de valles, siendo el central el que posee más profundidad, encontrándose el óptimo en él.

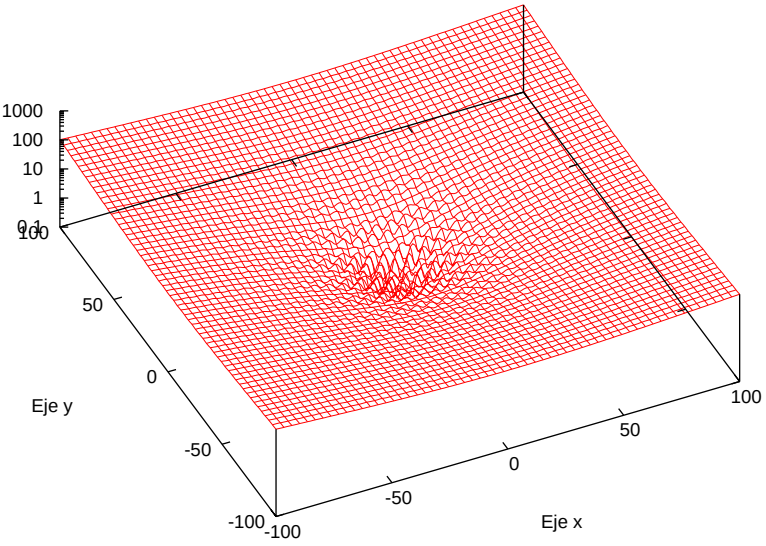


Figura A.9: Superficie de la función de Griewank en el intervalo $[-100, 100] \times [-100, 100]$

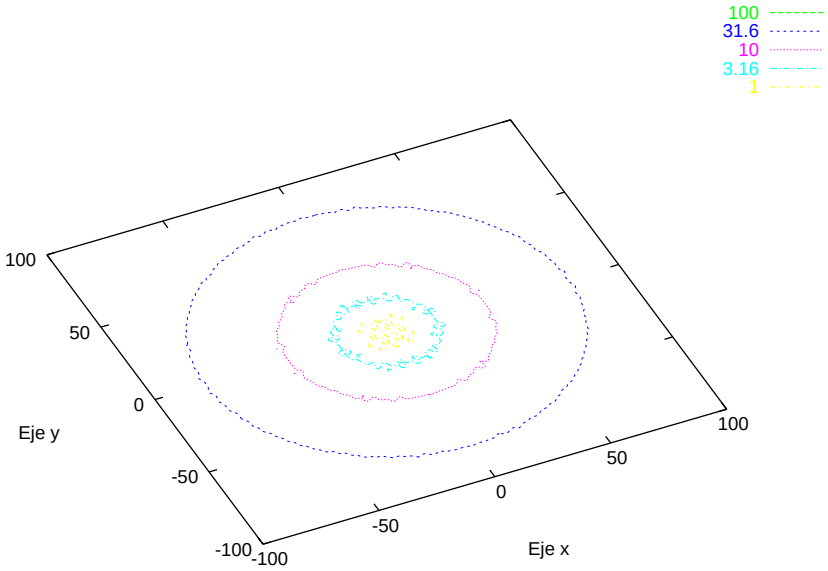


Figura A.10: Contorno de la función de Griewank en el intervalo $[-100, 100] \times [-100, 100]$

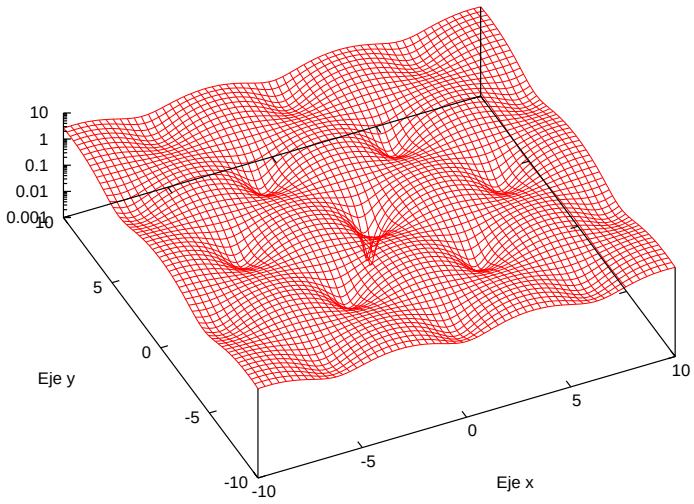


Figura A.11: Superficie de la función de Griewank en el intervalo $[-10, 10] \times [-10, 10]$

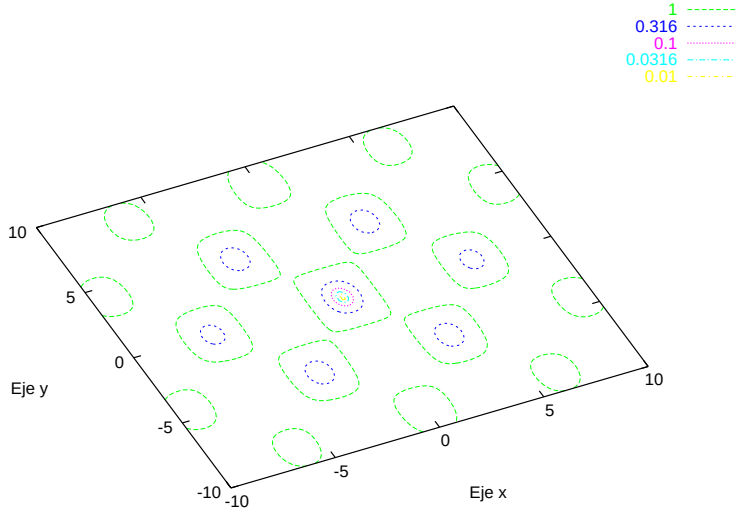


Figura A.12: Contorno de la función de Griewank en el intervalo $[-10, 10] \times [-10, 10]$

A.4. Función de las seis jorobas de camello inversas

Esta función[1] de dos dimensiones aparece en el artículo [75]. La definición de ésta es:

$$f(\bar{x}) = 4 \times x_1^2 - 2{,}1 \times x_1^4 + \frac{1}{3} \times x_1^6 + x_1 \times x_2 - 4 \times x_2^2 + 4 \times x_2^4 \qquad \text{(A.4)}$$

El valor del mínimo dentro del intervalo $[-5, 5] \times [-5, 5]$ vale -1.0316285, y se encuentra en los puntos $(0{,}09094, -0{,}712656)$ y $(-0{,}08984, 0{,}712656)$.

Las figuras A.13 y A.14 muestran la superficie y el contorno de esta función en el intervalo $[-5, 5] \times [-5, 5]$. En ellas podemos apreciar cómo existe un valle en la zona central de la superficie con forma de joroba de camello, que es la que proporciona el nombre a esta función. Observando el contorno de la función es posible vislumbrar las ubicaciones de los dos puntos óptimos que posee la función.

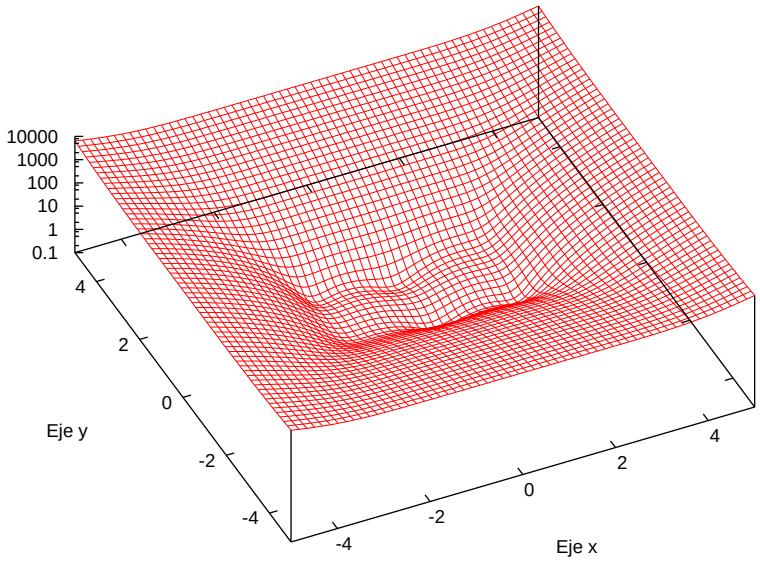


Figura A.13: Superficie de la función de seis jorobas de camello en el intervalo $[-5, 5] \times [-5, 5]$

[1] Su nombre en lengua inglesa es *six hump camel back function*.

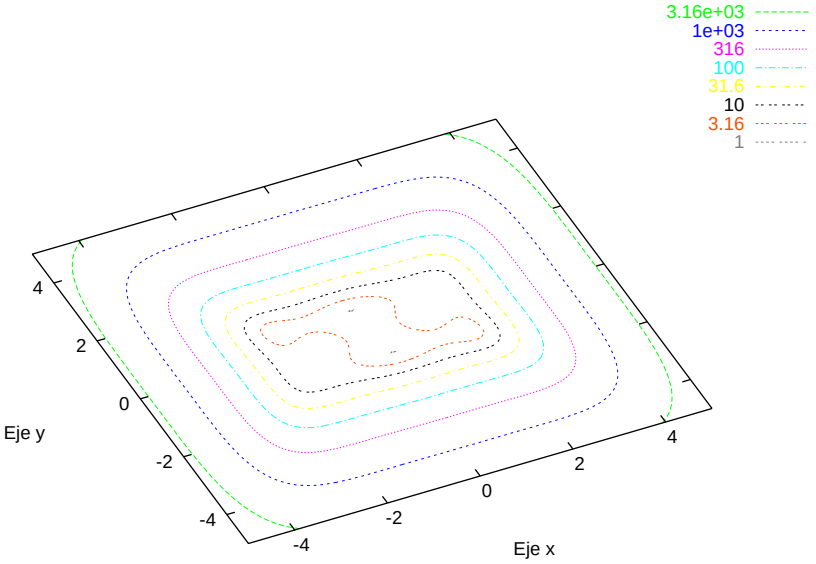


Figura A.14: Contorno de la función de seis jorobas de camello en el intervalo $[-5, 5] \times [-5, 5]$

A.5. Función de Rastrigin

Esta función de dos dimensiones aparece en el artículo [75]. La definición de ésta es:

$$f(\bar{x}) = x_1^2 + x_2^2 - \cos(18 \times x_1) - \cos(18 \times x_2) \tag{A.5}$$

El punto mínimo dentro del intervalo $[-50, 50] \times [-50, 50]$ se encuentra en el punto (0,0) y vale -2.

La figura A.15 muestra la superficie de esta función en el intervalo $[-50, 50] \times [-50, 50]$. En ella se puede observar la existencia de un valle en la zona centra de la superficie donde se encuentra el óptimo. Si analizamos en más detalle el intervalo $[-1, 1] \times [-1, 1]$ (figuras A.16 y A.17), podemos apreciar cómo la superficie está compuesta por múltiples valles simétricos pero de distintas profundidades, lo que dificulta la realización del proceso de optimización.

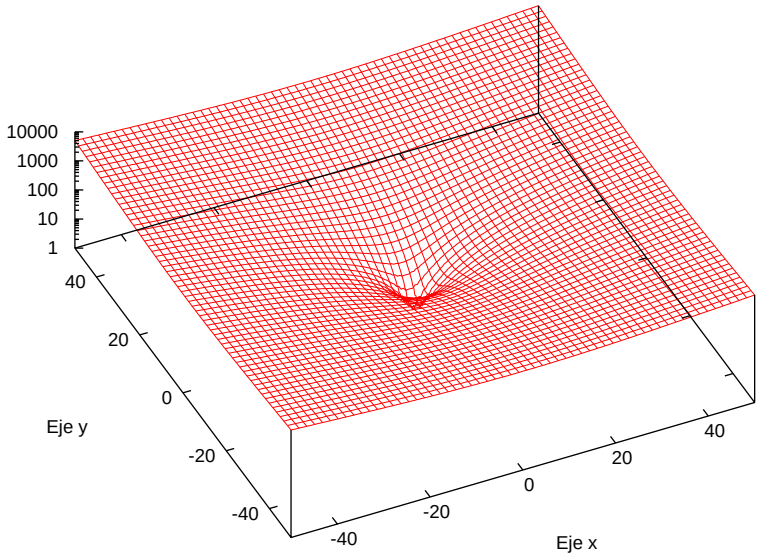


Figura A.15: Superficie de la función de Rastrigin en el intervalo $[-50, 50] \times [-50, 50]$

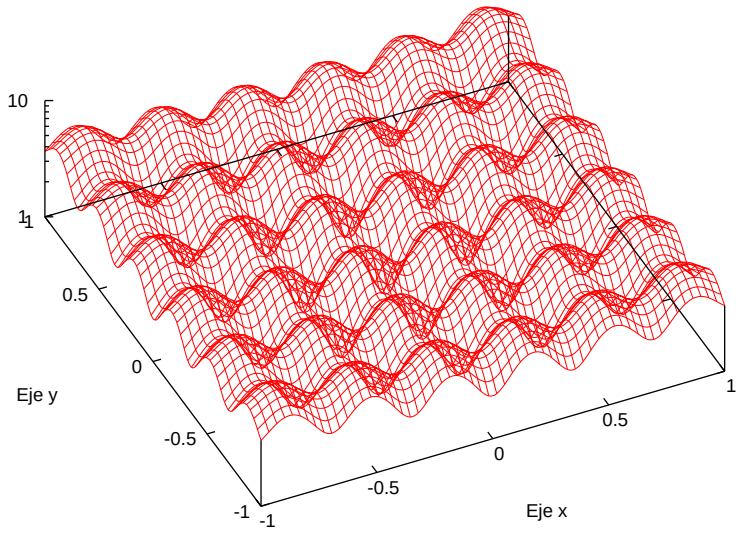


Figura A.16: Superficie de la función de Rastrigin en el intervalo $[-1, 1] \times [-1, 1]$

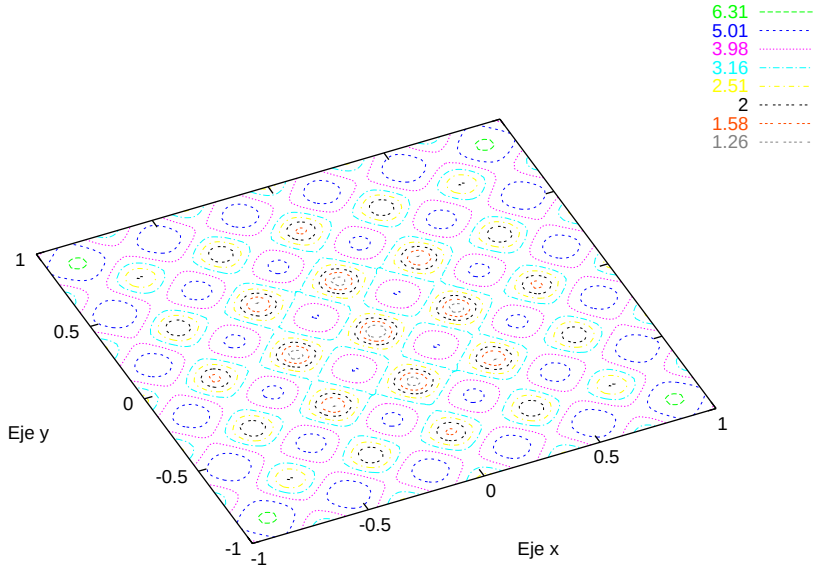


Figura A.17: Contorno de la función de Rastrigin en el intervalo $[-1, 1] \times [-1, 1]$

A.6. Función de Levy número 5

Esta función bidimensional aparece en el artículo [75]. La definición de la misma es:

$$f(\bar{x}) = \sum_{i=1}^{5} i \times \cos((i+1) \times x_1 + i) \times \sum_{j=1}^{5} j \times \cos((j+1) \times x_2 + j) + (x_1 + 1{,}42513)^2 + (x_2 + 0{,}80032)^2 \quad \text{(A.6)}$$

El mínimo dentro del intervalo $[-20, 20] \times [-20, 20]$ vale -176.138, y se encuentra en el punto $(-1{,}30685, -1{,}42485)$.

Las figuras A.18 y A.19 muestran la superficie y el contorno de la función de Levy número 5 en un intervalo $[-20, 20] \times [-20, 20]$. En ellas podemos apreciar cómo la superficie se curva hacia el centro del intervalo, aunque su superficie está compuesta por numerosos valles que siguen una cierta simetría.

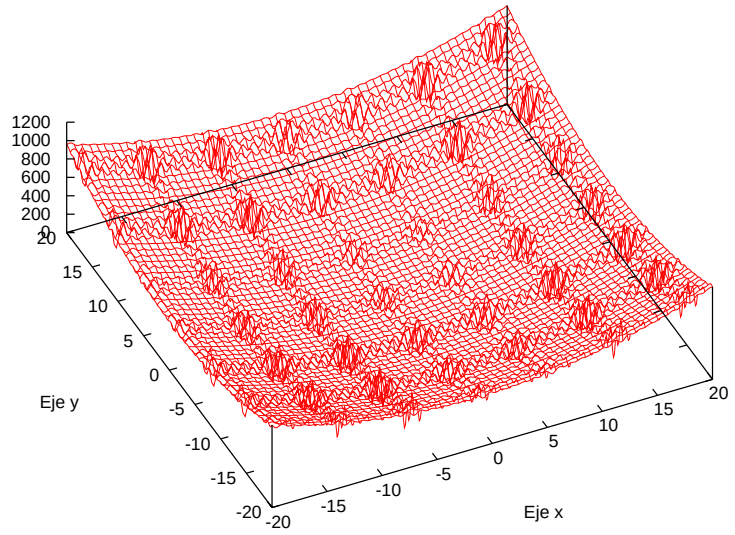


Figura A.18: Superficie de la función de Levy número 5 en el intervalo $[-20, 20] \times [-20, 20]$

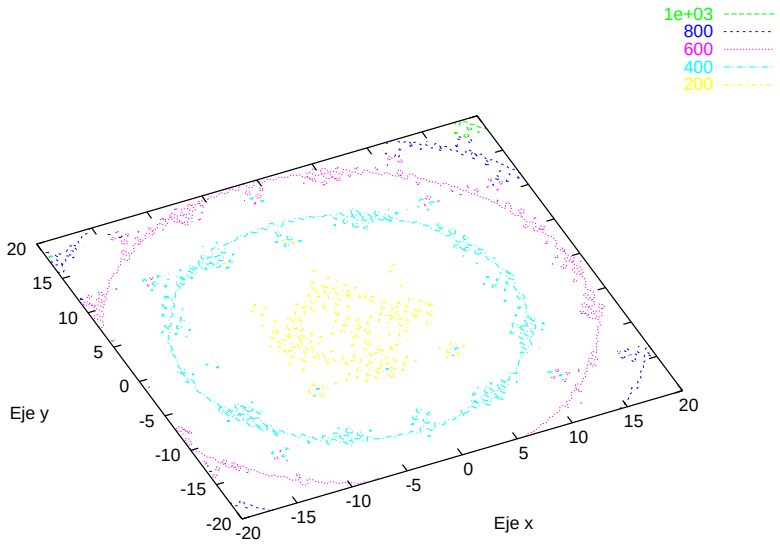


Figura A.19: Contorno de la función de Levy número 5 en el intervalo $[-20, 20] \times [-20, 20]$

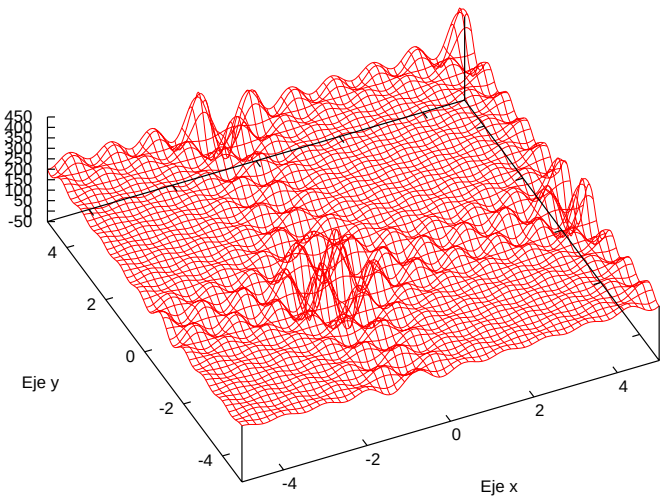


Figura A.20: Superficie de la función de Levy número 5 en el intervalo $[-5, 5] \times [-5, 5]$

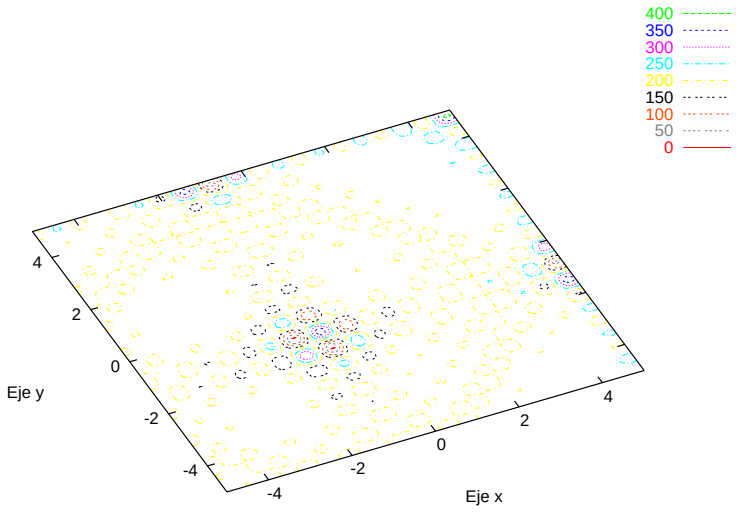


Figura A.21: Contorno de la función de Levy número 5 en el intervalo $[-5, 5] \times [-5, 5]$

Si se observa en detalle el intervalo $[-5,5] \times [-5,5]$ se puede apreciar cómo la superficie (figura A.20) está compuesta de numerosos valles que siguen cierta simetría. Además, cada uno dispone de una profundidad diferente, tal como se puede apreciar en el contorno de la misma (figura A.21).

A.7. Función de Beale

Esta función bidimensional, referenciada en el artículo [108], presenta la siguiente definición:

$$f(\bar{x}) = (1{,}5 - x_1 \times (1 - x_2))^2 + (2{,}25 - x_1 \times (1 - x_2^2))^2 + (2{,}625 - x_1 \times (1 - x_2^3))^2 \tag{A.7}$$

El mínimo global se encuentra en el punto $(3, 0{,}5)$, tomando la función un valor de 0.

Las figuras A.22 y A.23 muestran la superficie y el contorno de la función de esta función en un intervalo $[-100, 100] \times [-100, 100]$. En ellas podemos apreciar cómo la superficie se curva hacia el centro del intervalo.

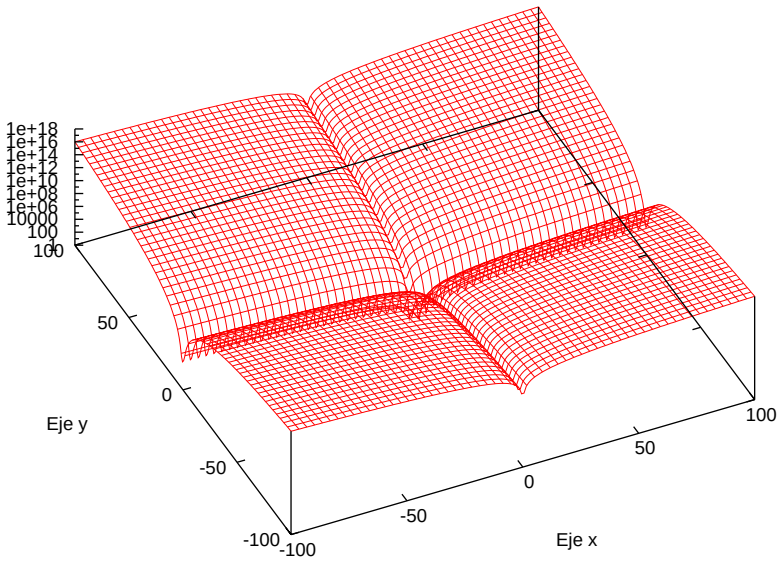


Figura A.22: Superficie de la función de Beale en el intervalo $[100, 100] \times [-100, 100]$

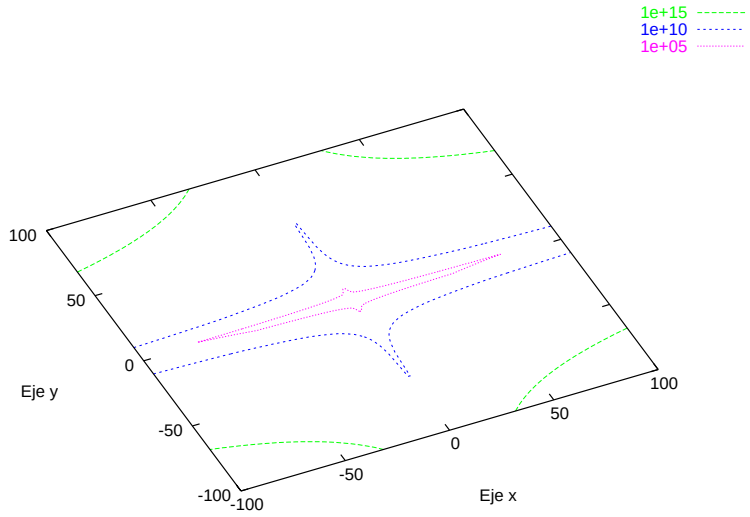


Figura A.23: Contorno de la función de Beale en el intervalo $[-100, 100] \times [-100, 100]$

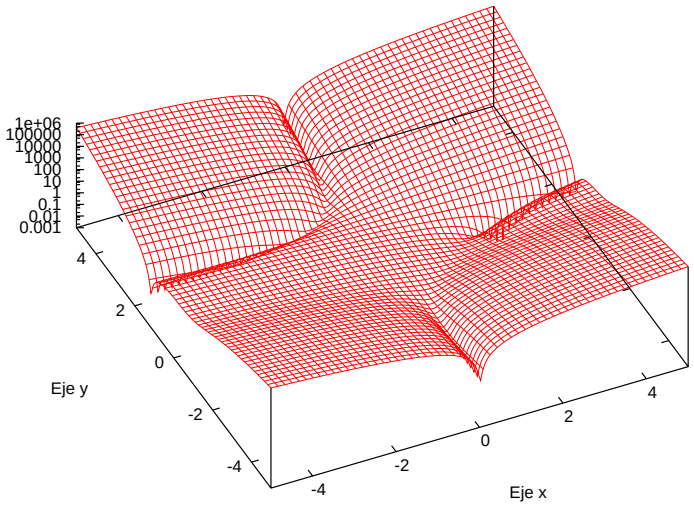


Figura A.24: Superficie de la función de Beale en el intervalo $[-5, 5] \times [-5, 5]$

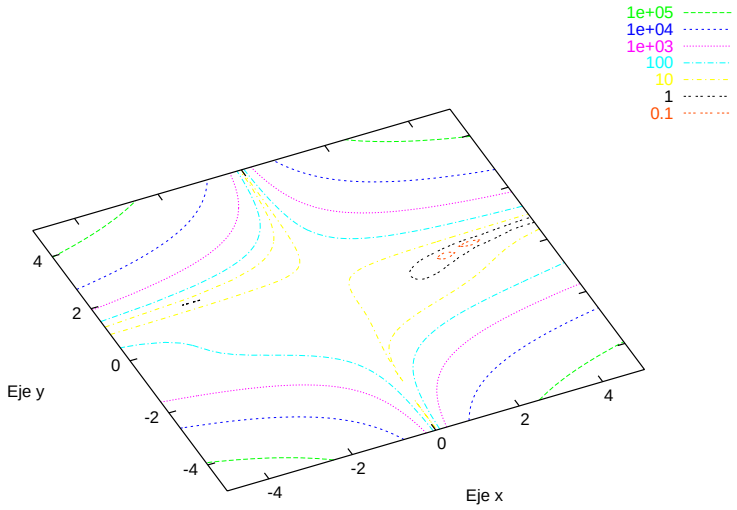


Figura A.25: Contorno de la función de Beale en el intervalo $[-5, 5] \times [-5, 5]$

Si se observa en detalle el intervalo $[-5, 5] \times [-5, 5]$ se puede apreciar cómo la superficie (figura A.24) está compuesta de un valle amplio en forma de estrella. Sin embargo, analizando su contorno (figura A.25) es posible observar la localización de diferentes zonas donde es posible localizar a óptimos locales.

A.8. Función de Rosenbrock de n variables

Definición

$$f(\bar{x}) = \sum_{j=1}^{n-1} \left[100 \times (x_j^2 - x_{j+1})^2 + (x_j - 1)^2\right] \tag{A.8}$$

Dominio de búsqueda $-5 < x_j < 10, \quad \forall j$

Mínimo global $(1, 1, \ldots, 1)$

Valor de la función en el mínimo global 0

Fuente bibliográfica [19]

A.9. Función de Zakharov de n variables

Definición

$$f(\bar{x}) = \left(\sum_{j=1}^{n} x_j^2\right) + \left(\sum_{j=1}^{n} 0{,}5 \times j \times x_j\right)^2 + \left(\sum_{j=1}^{n} 0{,}5 \times j \times x_j\right)^4 \quad \text{(A.9)}$$

Dominio de búsqueda $-5 < x_j < 10, \quad \forall j$

Mínimo global $(0, 0, \ldots, 0)$

Valor de la función en el mínimo global 0

Fuente bibliográfica [19]

A.10. Función trigonométrica

Definición

$$f(\bar{x}) = \left(\sum_{j=1}^{n} f_i(\bar{x})^2\right)$$

donde: (A.10)

$$f_i(\bar{x}) = n - \sum_{j=1}^{n} \cos(x_j) + i \times (1 - \cos(x_i)) - \sin(x_i)$$

Dominio de búsqueda $-50 < x_j < 50, \quad \forall j$

Mínimo global $(\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n})$

Valor de la función en el mínimo global 0

Fuente bibliográfica [108]

Apéndice B

Métodos de optimización continua

En este capítulo se recoge la descripción de otros métodos de optimización continua citados en la clasificación de métodos propuesta en el capítulo 2, pero que no fueron abordadas dado que su comprensión no se ha considerado importante para la lectura de la tesis.

B.1. Optimización unidimensional

Además de la búsqueda dicotómica, búsqueda de Fibonacci y la búsqueda de la sección áurea, existen otros métodos basados en la interpolación polinomial. A continuación pasamos a detallar los aspectos más singulares de los mismos.

B.1.1. Interpolación polinomial

Los métodos anteriores suponen que la función es unimodal en un intervalo dado. Es posible desarrollar procedimientos de búsqueda más potentes para funciones que, además de ser unimodales, presentan cierto grado de suavidad. De esta forma, se puede realizar una aproximación a la función mediante un polinomio.

B.1.1.1. Método de Newton

Consiste en realizar a la función f a minimizar una aproximación cuadrática f en un punto x^k. De este modo:

$$f(x) = f(x^k) + f'(x^k)(x - x^k) + \frac{1}{2}f''(x^k)(x - x^k)^2 \qquad \text{(B.1)}$$

La estimación del punto x^{k+1} se toma del punto que anula la derivada de la cuadrática f, es decir;

$$f'(x^{k+1}) = f'(x^k) + f''(x^k)(x^{k+1} - x^k) = 0 \tag{B.2}$$

De este modo,

$$x^{k+1} = x^k - \frac{f'(x^k)}{f''(x^k)} \tag{B.3}$$

Una posible implementación de este método se puede ver en el algoritmo B.1, donde además de un punto inicial se da como parámetro un valor ϵ que nos permite finalizar el procedimiento de búsqueda.

Algoritmo B.1 Método de Newton

Newton : $a \times \epsilon \longrightarrow a \times b$
$x_k \longleftarrow a$
$x_{k+1} \longleftarrow x_k - \dfrac{f'(x_k)}{f''(x_k)}$
Mientras $\big((|x_{k+1} - x_k| \geq \epsilon) \quad \| \quad (|f'(x_k)| < \epsilon)\big)$
$x_k \longleftarrow x_{k+1}$
$x_{k+1} \longleftarrow x_k - \dfrac{f'(x_k)}{f''(x_k)}$
Devolver $[x_{k+1}, x_k]$

B.1.1.2. Método de la posición falsa

Consiste en realizar también una aproximación cuadrática, al igual que el método de Newton, pero empleando para ello la información de dos puntos. Así,

$$f(x) = f(x^k) + f'(x^k)(x - x^k) + \frac{f'(x^{k-1}) - f'(x^k)}{x^{k-1} - x^k} \frac{(x - x^k)^2}{2} \tag{B.4}$$

De este modo, sólo se requiere la información de la primera derivada. De forma similar al método de Newton, la estimación del punto x_{k+1} se toma del punto que anula la derivada de la cuadrática $\bar{f}$, es decir;

$$x^{k+1} = x^k - f'(x^k) \left[\frac{x^{k-1} - x^k}{f'(x^{k-1}) - f'(x^k)} \right] \tag{B.5}$$

Este método se puede considerar como una aproximación al de Newton, donde la segunda derivada se sustituye por la diferencia de las dos primeras. Una posible implementación se puede ver en el algoritmo B.2.

Algoritmo B.2 Método de la posición falsa

Posición-falsa : $a \times b \times \epsilon \longrightarrow a \times b$

$x^{k-1} \longleftarrow a$

$x^{k} \longleftarrow b$

$x^{k+1} \longleftarrow x^k - f'(x^k)\left[\dfrac{x^{k-1}-x^k}{f'(x^{k-1})-f'(x^k)}\right]$

Mientras $\left(\left(|x^{k+1}-x^k| \geq \epsilon\right) \quad \| \quad \left(|f'(x^k)| < \epsilon\right)\right)$

$x^{k-1} \longleftarrow x^k$

$x^{k} \longleftarrow x^{k+1}$

$x^{k+1} \longleftarrow x^k - f'(x^k)\left[\dfrac{x^{k-1}-x^k}{f'(x^{k-1})-f'(x^k)}\right]$

Devolver $\left[x^k, x^{k+1}\right]$

B.1.1.3. Ajuste cúbico

En este caso, se realiza una aproximación a la función f que se desea minimizar mediante un polinomio cúbico. De este modo, conociendo dos puntos x^{k-1} y x^k, así como sus respectivos valores en la función y de la primera derivada, el siguiente punto se puede determinar como:

$$x^{k+1} = x^k - (x^k - x^{k-1})\left[\frac{f'(x^k) + w - z}{f'(x^k) - f'(x^{k-1}) + 2w}\right] \tag{B.6}$$

donde:

$$z = f'(x^{k-1}) + f'(x^k) - 3\frac{f(x^{k-1}) - f(x^k)}{x^{k-1} - x^k}$$
$$w = \sqrt{z^2 - f'(x^{k-1})f'(x^k)}$$

El algoritmo B.3 muestra una posible implementación de este método.

Algoritmo B.3 Método de ajuste cúbico

Ajuste-cúbico : $a \times b \times \epsilon \longrightarrow a \times b$

$x^k \longleftarrow a$

$x^{k-1} \longleftarrow b$

$z \longleftarrow f'(x^{k-1}) + f'(x^k) - 3\dfrac{f(x^{k-1}) - f(x^k)}{x^{k-1} - x^k}$

$w \longleftarrow \sqrt{z^2 - f'(x^{k-1})f'(x^k)}$

$x^{k+1} \longleftarrow x^k - (x^k - x^{k-1})\left[\dfrac{f'(x^k) + w - z}{f'(x^k) - f'(x^{k-1}) + 2w}\right]$

Mientras $\left(\left(|x^{k+1} - x^k| \geq \epsilon\right) \| \left(|f'(x^k)| < \epsilon\right)\right)$

$x^k \longleftarrow x^{k+1}$

$z \longleftarrow f'(x^{k-1}) + f'(x^k) - 3\dfrac{f(x^{k-1}) - f(x^k)}{x^{k-1} - x^k}$

$w \longleftarrow \sqrt{z^2 - f'(x^{k-1})f'(x^k)}$

$x^{k+1} \longleftarrow x^k - (x^k - x^{k-1})\left[\dfrac{f'(x^k) + w - z}{f'(x^k) - f'(x^{k-1}) + 2w}\right]$

Devolver $\left[x^{k+1}, x^k\right]$

B.1.1.4. Ajuste cuadrático

Los métodos anteriores hacen uso de la primera derivada, y en el caso del método de Newton hasta de la segunda derivada. Sin embargo, es posible hacer una aproximación a la función f empleando la aproximación de un polinomio cuadrático que sólo requiere de valores de la función. En este caso se necesitan tres puntos que cumplan la siguiente condición:

$$\begin{aligned} x_1 > & x_2 > x_3 \\ f(x_1) > & f(x_2) \\ f(x_2) < & f(x_3) \end{aligned}$$

A partir de ellos y sus valores en la función se construye el polinomio cuadrático siguiente, que pasa por dichos puntos:

$$\bar{f}(x) = \sum_{i=1}^{3} f(x_i) \frac{\prod_{j \neq i}(x - x_j)}{\prod_{j \neq i}(x_i - x_j)'} \tag{B.7}$$

De este modo, el nuevo punto se obtiene donde la derivada de dicho polinomio se anula, es decir:

$$x_4 = \frac{1}{2} \frac{b_{23} f(x_1) + b_{31} f(x_2) + b_{12} f(x_3)}{a_{23} f(x_1) + a_{31} f(x_2) + a_{12} f(x_3)} \tag{B.8}$$

donde:

$$a_{ij} = x_i - x_j$$
$$b_{ij} = x_i^2 - x_j^2$$

Algoritmo B.4 Método de ajuste cuadrático

Ajuste-cuadrático : $a \times b \times \epsilon \longrightarrow a$
$x_2 \longleftarrow (b-a)/2$
$x_1 \longleftarrow a$
$x_3 \longleftarrow b$
Mientras $(f(x_1) \leq f(x_2)) \quad \| \quad (f(x_2) \geq (f(x_3))$
  Si $f(x_1) \leq f(x_2)$
    $x_3 \longleftarrow x_2$
    $x_2 \longleftarrow (x_3 - x_1)/2$
  Si $f(x_2) < f(x_1)$
    $x_3 \longleftarrow (x_2 - x_1) \times 2$
calcular_bij_y_aij(x_1, x_2, x_3)
$x_4 = \frac{1}{2} \frac{b_{23} f(x_1) + b_{31} f(x_2) + b_{12} f(x_3)}{a_{23} f(x_1) + a_{31} f(x_2) + a_{12} f(x_3)}$
Mientras $\big((|x_1 - x_3| \geq \epsilon) \quad \| \quad \big(|f'(x_4)| < \epsilon\big)\big)$
  Si $(x_4 == x_2)$
  $x_4 \longleftarrow x_4 + \epsilon$
  Si $(x_4 > x_2)$
    Si $(f(x_4) \geq f(x_2))$
      $x_3 \longleftarrow x_4$
    Si no
      $x_1 \longleftarrow x_2$
      $x_2 \longleftarrow x_4$
  Si $(x_4 < x_2)$
    Si $(f(x_4) \geq f(x_2))$
      $x_1 \longleftarrow x_4$
    Si no
      $x_2 \longleftarrow x_4$
      $x_3 \longleftarrow x_2$
  calcular_bij_y_aij(x_1, x_2, x_3)
  $x_4 = \frac{1}{2} \frac{b_{23} f(x_1) + b_{31} f(x_2) + b_{12} f(x_3)}{a_{23} f(x_1) + a_{31} f(x_2) + a_{12} f(x_3)}$
Devolver x_2

En [91] podemos ver un análisis detallado de la convergencia de este

método. El algoritmo B.4 muestra una posible implementación de este método. Se puede observar que para obtener los tres puntos necesarios o bien se subdivide el intervalo inicial o bien se aumenta el doble.

B.1.2. Interpolación polinomial salvaguardada

En los apartados anteriores hemos visto una serie de métodos que no emplean la información de las derivadas, como es el caso de la búsqueda dicotómica, de Fibonacci, o de la razón áurea, y otros que hacen uso de la derivada con objeto de obtener un mayor grado de convergencia, como en el caso del método de Newton.

Sin embargo, en esas técnicas es posible la aparición de problemas debido al mal condicionamiento de la función. Así, es posible con los métodos anteriores que dos iteraciones consecutivas sean numéricamente indiferenciables, aún estando lejos del óptimo.

Los métodos de interpolación polinomial salvaguardada intentan agrupar ambos métodos, permitiendo una rápida convergencia. Para ello, la interpolación polinomial se combina con un método sin derivadas como la búsqueda de Fibonacci, la razón áurea o la búsqueda dicotómica.

En una interpolación polinomial disponemos de un intervalo $[a, b]$ a optimizar, que se va reduciendo en cada iteración, y una mejor aproximación x^k. Sea x^{k+1} el siguiente punto obtenido en el procedimiento de optimización con interpolación. Evaluamos $\bar{x}^{k+1}$ como una aproximación a la búsqueda de la razón áurea:

$$\bar{x}^{k+1} = \begin{cases} \beta(a - x^k), & \text{Si} \quad x \geq \frac{1}{2}(a+b) \\ \beta(b - x^k), & \text{Si} \quad x < \frac{1}{2}(a+b) \end{cases} \tag{B.9}$$

donde $\beta = 1 - \frac{1}{2}(\sqrt{5} - 1)$. De este modo, si $\bar{x}^{k+1} > x^{k+1}$, se empleará en la siguiente iteración $\bar{x}_{k+1}$, evitando de este modo que dos iteraciones consecutivas sean muy cercanas.

B.2. Optimización multidimensional

A continuación detallamos otros métodos de optimización multidimensional, independientes del método de descenso visto en el capítulo 2.

B.2.1. La búsqueda lineal

En los métodos de descenso, la búsqueda lineal puede efectuarse también por otros métodos diferentes a la regla de Armijo. A continuación pasamos a detallarlos.

Paso constante

Se establece un tamaño de paso constante $\alpha > 0$, que se emplea en todas las iteraciones del método de descenso. Es el esquema más simple. Sin embargo, si este paso es muy grande el método puede no converger, y si es muy pequeño, la convergencia es demasiado lenta. Este esquema es válido sólo para problemas donde puede determinarse un valor de paso válido.

Retroceso

Consiste en ir decrementando la longitud de paso de forma sucesiva en cada iteración, empezando en un punto inicial, hasta que la condición de descenso (2.14) se produzca. Esta estrategia es fácil de implementar y sirve incluso para problemas con restricciones.

Regla de Goldstein

Otra regla que asegura la condición de descenso es la regla de Goldstein. Se establece un escalar $\sigma \in (0, 0{,}5)$, y el paso α escogido debe cumplir la relación:

$$\sigma \leq \frac{f(x^k + \alpha d^k) - f(x^k)}{\alpha \nabla f(x^k)^T d^k} \leq 1 - \sigma \tag{B.10}$$

Regla de Wolfe

Una búsqueda linea inexacta que suele emplearse bastante está basada en las condiciones de Wolfe, donde además de asegurar la condición de descenso se busca asegurar la convergencia. En este caso:

$$f(x^k + \alpha d^k) - f(x^k) \leq \sigma \alpha \nabla f(x^k)^T d^k \tag{B.11}$$

$$\nabla f(x^k + \alpha d^k)^T d^k \geq \tau \nabla f(x^k)^T d^k \tag{B.12}$$

donde $\tau \in (0, 1), \sigma \in (0, 1), \sigma \leq \tau$, empleándose normalmente $\sigma \leq 0{,}5$ [165]. Es posible realizar una pequeña modificación en esta condición para obtener las *condiciones de Wolfe fuerte*, donde (B.12) se sustituye por:

$$|\nabla f(x^k + \alpha d^k)^T d^k| \leq -\tau \nabla f(x^k)^T d^k \tag{B.13}$$

donde τ establece el grado de exactitud en la búsqueda lineal a emplear. Así, si $\tau = 0$ nos encontramos en una búsqueda lineal exacta.

Se han propuesto diversos algoritmos que satisfagan dichas condiciones. Estos algoritmos presentan un esquema general. En primer lugar se busca un intervalo de puntos aceptables. A continuación, va subdividiendo el intervalo generando una secuencia de intervalos que tienden a cero. Para ello, en cada iteración se comprueba si el intervalo converge al paso α que asegura las condiciones de Wolfe. En [42, 109, 130] podemos ver las implementaciones de los algoritmos más empleados.

B.2.1.1. Los métodos del gradiente

El método de Newton modificado

Es igual al método de Newton expuesto en el apartado 2.4.1.2, pero con objeto de disminuir la sobrecarga de tener que evaluar el Hessiano de la función f en cada iteración, ésta sólo se recalcula cada $p > 1$ iteraciones. De este modo, la matriz D^k vendrá dada por:

$$D^{ip+j} = (\nabla^2 f(x^{ip}))^{-1}, \qquad j = 0, 1, \ldots, p-1; i = 0, 1, \ldots \tag{B.14}$$

En este caso, la reducción de tiempo de computación en la evaluación del Hessiano de la función en cada iteración conlleva una degradación en la velocidad de convergencia.

Una variante simplificada de este método consiste en utilizar siempre el Hessiano del punto inicial, sin hacer ninguna evaluación del mismo cada p iteraciones.

El método de Newton discretizado

Es igual al método de Newton, es decir:

$$D^k = (H(x^k))^{-1}, \qquad \forall k \geq 0 \tag{B.15}$$

pero la matriz H^k es una aproximación simétrica definida positiva del Hessiano de la función, que se forma utilizando las aproximaciones de diferencias finitas de la segunda derivada, basada en los valores de la primera derivada de la función f.

El método de máximo descenso escalado diagonalmente

En este caso, la matriz D^k es una matriz diagonal de la forma:

$$D^k = \begin{pmatrix} d_1^k & 0 & 0 & \ldots & 0 & 0 & 0 \\ 0 & d_2^k & 0 & \ldots & 0 & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & \ldots & 0 & d_{n-1}^k & 0 \\ 0 & 0 & 0 & \ldots & 0 & 0 & d_n^k \end{pmatrix} \tag{B.16}$$

donde d_i^k son escalares positivos para asegurar que la matriz D^k sea definida positiva. Una de las elecciones más conocidas es tomar como escalares d_i^k una aproximación a la inversa de la segunda derivada parcial de f respecto a x_i, es decir:

$$d_i^k \approx \left(\frac{\partial^2 f(x^k)}{(\partial x_i)^2} \right)^{-1}$$

En este caso, el método se conoce como *aproximación diagonal al método de Newton*.

El método de Gauss-Newton

Cuando la función que se pretende minimizar viene dado por la suma de cuadrados de una serie de funciones g_i, es decir, si $g = (g_1, \ldots, g_m)$, el problema se puede enunciar como:

$$\text{mín } f(x) = \frac{1}{2}\|g(x)\|^2 = \frac{1}{2}\sum_{m}^{i=1}(g_i(x))^2, \qquad x \in R^n \tag{B.17}$$

En este caso se puede emplear el método de Gauss-Newton, donde la matriz D^k viene dada por:

$$D^k = (\nabla g(x^k)\nabla g(x^k)^T)^{-1}, \qquad \forall k \geq 0 \tag{B.18}$$

siendo la matriz $\nabla g(x^k)\nabla g(x^k)^T$ invertible. Dado que:

$$\nabla f(x^k) = \nabla g(x^k)g(x^k)$$

el método de Gauss-Newton presenta la forma:

$$x^{k+1} = x^k - \alpha^k(\nabla g(x^k)\nabla g(x^k)^T)^{-1}\nabla g(x^k)g(x^k) \tag{B.19}$$

Este tipo de problemas suele presentarse en el análisis de datos estadísticos y en el entrenamiento de redes neuronales retroalimentadas.

B.2.1.2. Métodos de direcciones conjugadas

Los métodos cuasi-Newton permiten una rápida convergencia sin necesidad de calcular el Hessiano de la función. Sin embargo, el inconveniente principal que presentan es la necesidad de tener que almacenar en cada iteración una matriz. Cuando el número de dimensiones que se dispone es elevado, hace inviable el método. Por este motivo, surgen los métodos de direcciones conjugadas que intentan conseguir tanto eficiencia como fiabilidad, sin grandes requisitos de almacenamiento como los métodos cuasi-Newton.

Métodos del gradiente conjugado

Fueron originalmente desarrollados para resolver el problema cuadrático, es decir:

$$\text{mín } f(x) = \frac{1}{2}x^TQx - b^Tx, \qquad x \in R^n \tag{B.20}$$

donde Q es una matriz definida positiva. Sin embargo, también pueden ser empleados para la resolución de problemas no cuadráticos.

Siguen el modelo (2.15), donde la dirección es generada según:

$$\begin{aligned} d^0 &= -\nabla f(x^0) \\ d^k &= -\nabla f(x^k) + \beta^k d^{k-1}, \qquad \forall k \geq 1 \end{aligned} \tag{B.21}$$

Los diferentes modelos de direcciones conjugadas se diferencian en la elección del parámetro β^k. De este modo:

Método de Fletcher-Reeves Fue desarrollada por Fletcher y Reeves (1964) a partir del método de gradientes conjugados de Hestenes y Stiefel para la resolución de sistemas lineales. En este método, se define β^k como:

$$\beta^k = \frac{\nabla f(x^k)^T \nabla f(x^k)}{\nabla f(x^{k-1})^T \nabla f(x^{k-1})} \tag{B.22}$$

Método de Polak-Ribière Toma como alternativa a β^k:

$$\beta^k = \frac{\nabla f(x^k)^T (\nabla f(x^k) - \nabla f(x^{k-1}))}{\nabla f(x^{k-1})^T \nabla f(x^{k-1})} \tag{B.23}$$

Aunque estos métodos son menos eficiente y menos robustos que los métodos cuasi-Newton, la ventaja principal es que no requieren operaciones con matrices $n \times n$ (siendo n el número de dimensiones del problema), de forma que el método de Fletcher-Reeves requiere sólo almacenar tres vectores, mientras que el método de Polak-Ribière necesita cuatro. Sin embargo, estudios experimentales demuestran un mejor rendimiento de este último.

Cuando este método se emplea en un problema de optimización no cuadrático (2.1), es necesario reiniciar el método tras una serie de generaciones tomando en ese momento como dirección la del máximo descenso (2.22). Este reinicio se puede realizar [11]:

1. Cada n iteraciones.

2. Cada k iteraciones después de un primer reinicio efectuado en n iteraciones, siendo $k < n$.

3. Reiniciar cada n iteraciones o bien si se cumple:

$$|\nabla f(x^k)^T \nabla f(x^{k-1})| > \gamma \|\nabla f(x^{k-1})\|^2, \qquad 0 < \gamma < 1$$

Método del gradiente conjugado precondicionado

Es un método que sigue el esquema (2.15), definiéndose las direcciones de búsqueda como sigue:

$$\begin{aligned} d^0 &= -H\nabla f(x^0) \\ d^k &= -H\nabla f(x^k) + \beta^k d^{k-1}, \qquad \forall k \geq 1 \end{aligned} \tag{B.24}$$

donde H es una matriz fija definida positiva y simétrica, y β^k puede derivarse fácilmente de la fórmula de Fletcher-Reeves (B.22):

$$\beta^k = \frac{\nabla f(x^k)^T H \nabla f(x^k)}{\nabla f(x^{k-1})^T H \nabla f(x^{k-1})}$$

o bien de la fórmula de Polak-Ribière (B.23).

Métodos de memoria limitada

La idea principal de estos métodos es la de generar métodos cuasi-Newton sin necesidad de tener que almacenar una matriz en cada iteración. Se basan en la capacidad de poder obtener una aproximación al Hessiano a partir de la suma de una matriz diagonal y un número de matrices de rango uno.

El primer método de memoria limitada fue desarrollado por Nocedal (1980), y se denominó el *método SQN*. Éste es idéntico al método BFGS (apartado 2.4.1.3), pero la diferencia está en la actualización de la matriz D^k. En este caso, el usuario especifica con un parámetro m el número de actualizaciones que serán mantenidas, que permitirá almacenar los vectores (2.25) de las m iteraciones anteriores. De este modo, durante las primeras m iteraciones el método funciona igual al método BFGS. Para las siguientes iteraciones, el Hessiano se aproxima con la información de los vectores (2.25) de las m iteraciones previas, actualizando la aproximación inicial H^0. El algoritmo B.5 muestra una posible implementación del mismo [87], que recibe como parámetros de entrada, además del punto inicial y el número de actualizaciones a almacenar m, la aproximación al Hessiano inicial H^0 que será una matriz simétrica definida positiva. La notación empleada en el algoritmo es la siguiente:

$$\begin{aligned} \delta^k &= x^{k+1} - x^k \\ \gamma^k &= \nabla f(x^{k+1}) - \nabla f(x^k) \\ \rho^k &= \frac{1}{{\gamma^k}^T \delta^k} \\ V^k &= I - \rho^k \gamma^k {\delta^k}^T \end{aligned}$$

Algoritmo B.5 BFGS de memoria limitada

L-BFGS : $x^0 \times m \times H^0 \longrightarrow x^*$
$k \longleftarrow 0$
$d^k \longleftarrow -H^k \nabla f(x^k)$
$\alpha^k \longleftarrow longitud\_paso(x^k, d^k)$
$x^{k+1} \longleftarrow x^k + \alpha^k d^k$
Mientras $\left(\frac{\|x^{k+1} - x^k\|}{\|x^{k+1}\|} \leq \epsilon \right)$
  $\hat{m} \longleftarrow \min\{k, m-1\}$
  $H^{k+1} \longleftarrow (V^{k^T} \cdots V^{k-\hat{m}^T}) H^0 (V^{k-\hat{m}} \cdots V^k)$
  $i \longleftarrow 0$
  Mientras $(i < \hat{m} - 1)$
    $H^{k+1} \longleftarrow \rho^{k-\hat{m}+i} (V^{k^T} \cdots V^{k-\hat{m}+i+1^T}) \delta^{k-\hat{m}+i} \delta^{k-\hat{m}+i^T} (V^{k-\hat{m}+i} \cdots V^k)$
    $i \longleftarrow i + 1$
  $k \longleftarrow k + 1$
Devolver x^k

Existen otros algoritmos como el de Bucley y LeNir [15]. Este algoritmo se desarrolla en dos fases. En las primeras $k \leq m$ iteraciones, la aproximación al Hessiano D^k se construye igual que en el algoritmo de Nocedal. Para las siguientes iteraciones se aplica un método del gradiente conjugado precondicionado, empleando D^m como matriz en el cálculo de las direcciones (B.24). Esta segunda fase termina cuando se cumpla la condición:

$$|\nabla f(x^k)^T \nabla f(x^{k-1})| \geq 0{,}2 \|\nabla f(x^k)\|^2$$

A continuación se reinicia la primera fase, eliminando todos los vectores de diferencias (2.25), salvo δ^{k-1} y γ^{k-1}. El método continúa alternando las dos fases, hasta cumplir el criterio de convergencia.

En [87] se dispone de una comparativa entre ambos métodos, observándose que cuando el problema a resolver es grande, el método de Nocedal obtiene mejores resultados.

Otros métodos de memoria limitada, son los denominados *métodos de Hessiano reducido*, que se caracterizan por usar una pequeña matriz reducida que incrementa su dimensión en cada iteración. Esta matriz incorpora toda la información que ha sido acumulada durante las primeras iteraciones, permitiendo que las direcciones de búsquedas se calculen en un sistema linear más pequeño que en el caso de un método convencional. Dentro de éstos, podemos encontrarnos un algoritmo propuesto por Fletcher [43], o el de Gill y Leonard [51].

B.2.2. Métodos de región de confianza

La idea básica de estos métodos es la de aproximar la función objetivo mediante un modelo cuadrático obtenido tras una expansión de la serie de Taylor:

$$f(x^k + \delta) \approx f(x^k) + \nabla f(x^k)^T \delta + \frac{1}{2}\delta^T \nabla^2 f(x^k)\delta = q^k(\delta) \tag{B.25}$$

donde $\delta = x - x^k$ y $q^k(\delta)$ es la aproximación cuadrática a la función f en la iteración k. De este modo, en lugar de minimizar la función objetivo se intenta minimizar este modelo aproximado. El nombre de región de confianza es que ese problema de minimización se restringe únicamente a una vecindad del punto actual, que se denomina *región de confianza*, y consiste en los puntos más cercanos al punto de la iteración actual, es decir:

$$\Omega^k = \left\{ x : \|x - x^k\| \leq \Delta^k \right\} \tag{B.26}$$

donde Δ^k establece el paso máximo a dar en cada iteración[1]. A continuación se busca un punto δ^k dentro de dicha región, tal que sea solución al subproblema de minimizar la aproximación cuadrática:

$$\min_{\delta} q^k(\delta), \qquad \text{con} \quad \|\delta\| \leq \Delta^k \tag{B.27}$$

de forma que la siguiente iteración vendrá dada por $x^{k+1} = x^k + \delta^k$. Es posible obtener una solución aproximada a dicho subproblema de optimización con una restricción de forma rápida [11].

El algoritmo B.6 muestra una posible implementación de estos métodos. Las constantes $\tau_i, (i = 0, \ldots, 4)$ deben cumplir las siguientes relaciones:

$$\begin{aligned} 0 < \tau_3 < \tau_4 < 1 < \tau_1 \\ 0 \leq \tau_0 \leq \tau_2 < 1 \\ \tau_2 > 0 \end{aligned} \tag{B.28}$$

siendo el usuario el que determina sus valores. Los valores típicos que se suelen emplear son $\tau_0 = 0, \tau_1 = 2, \tau_2 = \tau_3 = 0{,}25$, y $\tau_4 = 0{,}5$ [166]. Para la evaluación del Hessiano de la matriz es posible incorporar una aproximación al mismo, para lo cuál el algoritmo podría recibir como parámetro de entrada una matriz $B \in R^{n \times n}$ simétrica que aproxime al Hessiano en el punto inicial. En cada iteración se requiere además que dicha matriz sea actualizada, pudiendo emplearse algún método de métrica variable como el BFGS o DFP (apartado 2.4.1.3).

[1] Dado que el paso a dar en cada iteración viene dado por el tamaño de la región de confianza, algunos autores utilizan el nombre de *métodos de paso restringido* para referenciar estas técnicas [42].

Algoritmo B.6 Método de región de confianza

Región-confianza : $x^0 \times \Delta^1 \times \epsilon \times \tau_0 \times \tau_1 \times \tau_2 \times \tau_3 \times \tau_4 \longrightarrow x^*$

$k \longleftarrow 1$

$x^k \longleftarrow x^0$

Mientras $(\|\nabla f(x^k)\| \leq \epsilon)$

$\delta^k \longleftarrow \min_{\delta} \nabla f(x^k)^T \delta + \frac{1}{2}\delta^T \nabla^2 f(x^k)\delta = \phi^k(\delta)$ con $\|\delta\| \leq \Delta^k$

$r^k \longleftarrow \frac{f(x^k) - f(x^k + \delta^k)}{\phi^k(0) - \phi^k(\delta^k)}$

Si $(r^k \leq \tau_0)$

$x^{k+1} \longleftarrow x^k$

Si no

$x^{k+1} \longleftarrow x^k + \delta^k$

Si $(r^k < \tau_2)$

$\Delta^{k+1} \longleftarrow \tau_3 \|\delta^k\|$

Si no Si $\Big((\|\delta^k\| = \Delta^k) \quad \&\& \quad (r^k \geq \tau_4)\Big)$

$\Delta^{k+1} \longleftarrow \tau_1 \Delta^k$

Si no

$\Delta^{k+1} \longleftarrow \Delta^k$

$k \longleftarrow k + 1$

Devolver x^k

Las ventajas principales de estos métodos son poseer unas propiedades fuertes de convergencia global [42] y su robustez al aplicarse a cualquier tipo de problemas.

B.2.2.1. El método de Levenberg-Marquard

Este método es específico para la resolución de sumas de cuadrados de funciones (B.17), como el método de Gauss-Newton (apartado B.2.1.1), pero puede ser visto como un método de región de confianza [164].

El método de Levenberg-Marquard es un método iterativo donde el paso a realizar en cada iteración viene dado por:

$$d^k = -(\nabla^2 F(x^k)\nabla^2 F(x^k)^T + \lambda^k I)^{-1} \nabla F(x^k) F(x^k) \tag{B.29}$$

donde $\lambda^k \geq 0$ es un parámetro que se actualiza en cada iteración, y la función $F(x)$ se define como $F(x^k) = (g_1(x^k), g_2(x^k), \ldots, g_m(x^k))^T$, siendo $g_i(x)$ cada una de las funciones en las que puede descomponerse la función original, según (B.17).

El algoritmo B.7 muestra una posible implementación de los métodos de región de confianza basada en el método de Levenberg-Marquard, donde se actualiza en cada iteración el parámetro Δ^k que define la región de confianza, en lugar del λ^k como el método de Levenberg-Marquard.

Algoritmo B.7 Método de región de confianza para la suma de cuadrados

Región-confianza-suma-cuadrados : $x^0 \times \Delta^1 \longrightarrow x^*$
$k \longleftarrow 1$
$x^k \longleftarrow x^0$
$\delta^k \longleftarrow \min_{\delta} \left\{ \|F(x^k) + \nabla f(x^k)^T \delta\|^2 \right\}$ con $\|\delta\| \leq \Delta^k$
Mientras $\left(\|F(x^k)\| \neq \|F(x^k) + \nabla F(x^k)^T \delta^k \right)$
$r^k \longleftarrow \dfrac{\|F(x^k)\| - \|F(x^k + \delta^k)\|}{\|F(x^k)\| - \|F(x^k) + \nabla F(x^k)^T \delta^k\|}$
Si $(r^k \leq 0)$
$x^{k+1} \longleftarrow x^k$
Si no
$x^{k+1} \longleftarrow x^k + \delta^k$
Si $(r^k < 0{,}1)$
$\Delta^{k+1} \longleftarrow \|\delta^k\|$
Si no Si $\left((r^k > 0{,}9) \quad \&\& \quad (\|\delta^k\| > \dfrac{\Delta^k}{2}) \right)$
$\Delta^{k+1} \longleftarrow 2\Delta^k$
Si no
$\Delta^{k+1} \longleftarrow \Delta^k$
$k \longleftarrow k + 1$
$\delta^k \longleftarrow \min_{\delta} \left\{ \|F(x^k) + \nabla f(x^k)^T \delta\|^2 \right\}$ con $\|\delta\| \leq \Delta^k$
Devolver x^k

B.2.3. Métodos de búsqueda de trayectoria

Estos métodos se caracterizan por generar direcciones de descenso de curvatura negativa. La idea es determinar un *par de direcciones de descenso*, s^k y d^k, en cada iteración, de forma que s^k representa la dirección de descenso según la información de curvatura positiva en $\nabla^2 f(x^k)$, y d^k es una dirección de descenso de curvatura negativa, tal que:

$$d^{kT} \nabla^2 f(x^k) d^k < 0$$

De este modo, si $\nabla^2 f(x^k)$ es semidefinida, entonces $d^k = 0$. Este par de direcciones se emplean de forma similar a un método de descenso, de forma que las iteraciones siguen el esquema:

$$x^{k+1} = x^k + \alpha d^k + \alpha^2 s^k \tag{B.30}$$

Moré y Sorensen [43] demuestran que la secuencia de puntos generadas en las sucesivas iteraciones conducen a un mínimo.

B.2.4. Métodos de búsqueda directa

Un método de búsqueda directa es «cualquier algoritmo que dependa únicamente de la función objetivo a través de un rango de valores de la función de un conjunto finito» [149]. Estos métodos surgen en la década de los 60, y aunque han aparecido técnicas de optimización más sofisticadas, estos métodos siguen empleándose hoy en día.

Estos métodos se pueden clasificar en tres categorías, como muestra la figura B.1 [85]. Aunque ésta se basa en los métodos de búsqueda directa que surgen en la década de los 60, las nuevas técnicas que han ido apareciendo son modificaciones de éstas.

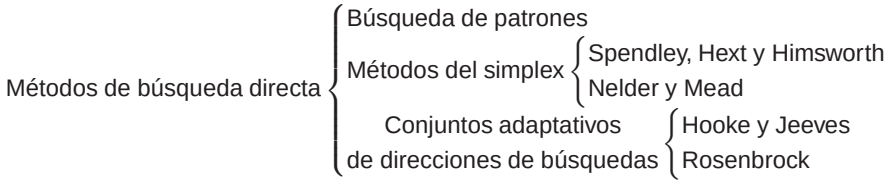


Figura B.1: Clasificación de los métodos de búsqueda directa

A pesar de ser simples de implementar y no necesitar el gradiente de la función, presentan el inconveniente de disponer de unas propiedades de convergencia débiles, aunque en [161] se cambia esta última percepción al realizarse un análisis más detallado de su convergencia.

B.2.4.1. Métodos de búsqueda de patrones

Estos métodos se caracterizan por realizar una serie de *movimientos exploratorios* que consideran el comportamiento de la función objetivo como un modelo de puntos. Ese movimiento consiste en realizar una estrategia sistemática que visite los puntos "vecinos" a la actual iteración. El concepto inicial de vecindad viene definido por un tamaño de paso Δ_k, que se va reduciendo cuando los incrementos o disminuciones de las variables no

producen ninguna disminución de la función objetivo. Este procedimiento continua hasta que el tamaño de paso sea lo suficientemente pequeño.

Una de las características principales de estos métodos es que no realizan un modelado de la función.

B.2.4.2. Métodos del simplex

Un *simplex* se define como un conjunto de $n+1$ puntos en R^n, de forma que el volumen de la figura que definen es no vacío. De este modo, en R^2 sería un triángulo, en R^3 un tetraedro, etc. El procedimiento que siguen estos métodos es, dado un simplex, generar otro, de forma que en cada iteración se vaya sustituyendo siempre el peor vértice. Estos métodos obtienen buenos resultados en problemas pequeños ($n < 10$). En [131] podemos encontrar un análisis detallado de la convergencia de estos métodos.

Estos métodos no tienen que confundirse con los métodos del simplex existentes en la programación lineal. Esto motiva que algunos autores [52] los denominen *métodos politopos*.

Método de Spendley, Hext y Himsworth

Fue el primer método del simplex que aparece (1962). Se caracteriza porque posee un único movimiento de *reflexión*. Éste, dado un simplex $v^o, v^1, v^2, \ldots, v^n$, identifica en primer lugar el peor vértice del conjunto, $v^p, 0 \leq p \leq n$, es decir:

$$f(v^p) \geq f(v^i) \qquad i = 0, 1, \ldots, n; \qquad i \neq m \tag{B.31}$$

A continuación refleja este peor vértice a través del centro del lado opuesto, es decir:

$$\hat{v} = \frac{2}{n}\left(-v^p + \sum_{i=0; i\neq p}^{n} v^i\right) \tag{B.32}$$

Este proceso se iría repitiendo hasta que se obtiene un vértice tal que $f(\hat{v}) \geq f(v^p)$. La figura B.2 muestra de forma gráfica este movimiento. En ella, v^3 sería considerado el peor vértice. Tras el movimiento de reflexión, el nuevo simplex quedaría formado por el mismo conjunto de puntos, salvo el peor vértice v^3, que se sustituiría por v^4 según (B.32).

Método de Nelder y Mead

Este algoritmo sigue el esquema iterativo del método anterior, pero cuando obtiene el punto de reflexión, según el valor de la función en éste procede a generar el nuevo simplex. Para ello, introduce tres movimientos: reflexión, expansión y contracción.

El algoritmo B.8 muestra una posible implementación de éste. En él se realiza un movimiento de reflexión, que viene dado por el punto x^{ref}. Según

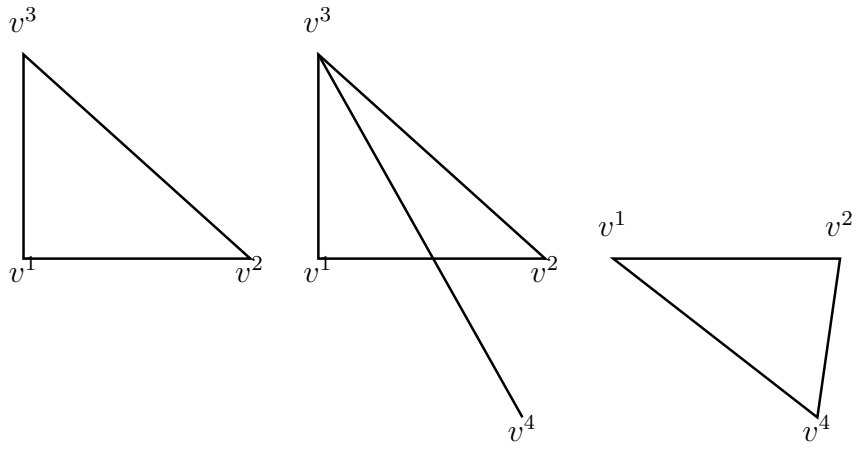


Figura B.2: Movimiento de reflexión del simplex

el valor de éste, se intenta continuar realizando movimientos de reflexión, o realizar un movimiento de expansión al punto x^{exp}, o uno de contracción al punto x^{con}.

Este algoritmo se puede generalizar en uno, donde los distintos movimientos vienen dados según las siguientes relaciones:

$$\begin{aligned} x^{\text{ref}} &= \hat{x} + \beta(\hat{x} - x^{\max}) \\ x^{\text{exp}} &= x^{\text{ref}} + \gamma(x^{\text{ref}} - \hat{x}) \\ x^{\text{con}} &= \begin{cases} \theta x^{\max} + (1-\theta)\hat{x}, & \text{si} \quad f(x^{\max}) \leq f(x^{\text{ref}}) \\ \theta x^{\text{ref}} + (1-\theta)\hat{x}, & \text{en otro caso} \end{cases} \end{aligned} \tag{B.33}$$

donde $\beta > 0$, $\gamma > 0$ y $1 > \theta > 0$, son los denominados *coeficientes de reflexión*, *de expansión* y *de contracción*, respectivamente. El algoritmo de Nelder y Mead, se corresponden con $\beta = 1$, $\gamma = 1$ y $\theta = \frac{1}{2}$.

Algoritmo B.8 Nelder-Mead

Nelder-Mead : $x^0 \times x^2 \times \ldots \times x^N \times \epsilon \longrightarrow x^*$
$k \longleftarrow 0$
Hacer
$k \longleftarrow k+1$
$x^{\text{min}} \longleftarrow \min_{0 \leq i \leq N} \left\{ f(x^i) \right\}$
$x^{\text{max}} \longleftarrow \min_{0 \leq i \leq N} \left\{ f(x^i) \right\}$
$\hat{x} \longleftarrow \frac{1}{N} \left(-x^{\text{max}} + \sum_{i=0}^{N} x^i \right)$
$x^{\text{ref}} \longleftarrow 2\hat{x} - x^{\text{max}}$
Si $(f(x^{\text{min}}) > f(x^{\text{ref}}))$
$x^{\text{exp}} \longleftarrow 2x^{\text{ref}} - \hat{x}$
Si $(f(x^{\text{exp}}) < f(x^{\text{ref}}))$
$x^k \longleftarrow x^{\text{exp}}$
Si no
$x^k \longleftarrow x^{\text{ref}}$
Si no Si $\left(\text{máx} \left\{ f(x^i) | x^i \neq x^{\text{max}} \right\} > f(x^{\text{ref}}) \geq f(x^{\text{min}}) \right)$
$x^k \longleftarrow x^{\text{ref}}$
Si no Si $\left(f(x^{\text{ref}}) \geq \text{máx} \left\{ f(x^i) | x^i \neq x^{\text{max}} \right\} \right)$
Si $(f(x^{\text{max}}) \leq f(x^{\text{ref}}))$
$x^{\text{con}} \longleftarrow \frac{1}{2}(x^{\text{max}} + \hat{x})$
Si no
$x^{\text{con}} \longleftarrow \frac{1}{2}(x^{\text{ref}} + \hat{x})$
$x^k \longleftarrow x^{\text{con}}$
$x^{\text{max}} \longleftarrow x^k$
Hasta que $(f(x^{\text{min}}) - f(x^k) < \epsilon)$
Devolver x^{min}

B.2.4.3. Métodos con conjuntos adaptativos de direcciones de búsquedas

En los métodos de descenso coordenados se realizan búsquedas a lo largo de un conjunto de direcciones fijas, que al ser linealmente independientes garantizan una mejora en el coste de la función. Esta idea se generaliza en los denominados métodos de búsqueda directa, donde se emplea un conjunto diferente de direcciones, y que pueden cambiar en cada iteración.

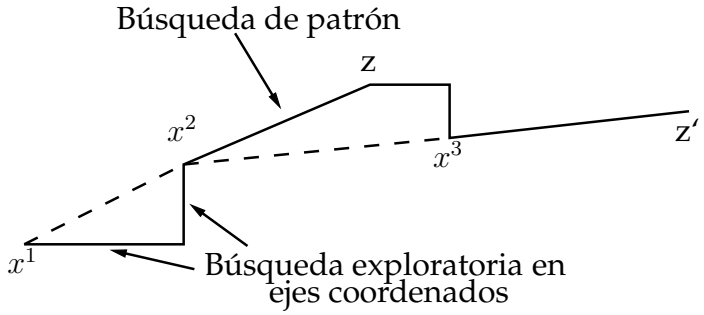


Figura B.3: Esquema de funcionamiento del método de Hooke y Jeeves

Algoritmo B.9 Método de Hooke y Jeeves

Hooke-Jeeves : $x^0 \times \epsilon \longrightarrow x^*$
$k \longleftarrow 1$
$x^k \longleftarrow x^0$
$j \longleftarrow 1$
$(d_1, d_2, \ldots, d_N) \longleftarrow$ direcciones_coordenadas()
Mientras $(j < N)$
  $\alpha \longleftarrow \text{mín}\left\{f(x^k + \alpha d_j)\right\}$
  $x_j^{k+1} \longleftarrow x_j^k + \alpha d_j$
  $j \longleftarrow j + 1$
Mientras $(\|x^{k+1} - x^k\| \geq \epsilon)$
  $d \longleftarrow x^{k+1} - x^k$
  $\alpha \longleftarrow min\left\{f(x^{k+1} + \alpha d)\right\}$
  $x^k \longleftarrow x^{k+1}$
  $k \longleftarrow k + 1$
  $j \longleftarrow 1$
  Mientras $(j \leq N)$
    $\alpha \longleftarrow \text{mín}\left\{f(x^k + \alpha d_j)\right\}$
    $x_j^{k+1} \longleftarrow x_j^k + \alpha d_j$
    $j \longleftarrow j + 1$
Devolver x^k

Método de Hooke y Jeeves

Este método se caracteriza por realizar dos tipos de búsqueda, como se ilustra en la figura B.3. En la primera se realiza una exploración a lo largo de los ejes coordenados, de forma, que dado el punto x^1, se genera el punto x^2. A continuación, se realiza una búsqueda a través de la dirección $x^2 - x^1$, obteniendo un punto z. Se continúa realizando una búsqueda exploratoria hasta obtener el punto x^3, para continuar buscando en la dirección $x^3 - x^2$, y así sucesivamente.

En el algoritmo B.9 podemos ver una implementación de éste, donde se realizan búsquedas lineales en cada dirección de búsqueda. Sin embargo, la propuesta original se caracterizaba por realizar pasos discretos a lo largo de éstas [8].

Algoritmo B.10 Método de Rosenbrock

Rosenbrock : $x^0 \times \epsilon \longrightarrow x^*$
$k \longleftarrow 1$
$x^k \longleftarrow x^0$
$j \longleftarrow 1$
$(d_1, d_2, \ldots, d_N) \longleftarrow$ direcciones_coordenadas()
Mientras $(j \leq N)$
  $\alpha \longleftarrow \text{mín}\left\{f(x^k + \alpha d_j)\right\}$
  $x_j^{k+1} \longleftarrow x_j^k + \alpha d_j$
  $j \longleftarrow j + 1$
Mientras $\left(\|x^{k+1} - x^k\| \geq \epsilon\right)$
  $x^k \longleftarrow x^{k+1}$
  $k \longleftarrow k + 1$
  $j \longleftarrow 1$
  construir_vectores$(d_1, d_2, \ldots, d_N)$
  Mientras $(j \leq N)$
    $\alpha \longleftarrow \text{mín}\left\{f(x^k + \alpha d_j)\right\}$
    $x_j^{k+1} \longleftarrow x_j^k + \alpha d_j$
    $j \longleftarrow j + 1$
Devolver x^k

Método de Rosenbrock

Este método realiza una búsqueda exploratoria a lo largo de n direcciones ortogonales. En la primera iteración, las direcciones corresponden con

los ejes coordenados. Tras realizar la exploración en cada eje, se va generando un nuevo conjunto de direcciones. De este modo, si x^k es el punto actual, $d_1, d_2, \ldots, d_N$ son los vectores de direcciones lineales, cada uno con norma igual a 1, y además ortogonales, es decir, $d_i^T d_j = 0, \forall i \neq j$, el nuevo punto x^{k+1} se obtiene de minimizar la función f a lo largo de cada una de las direcciones, de forma que: $x^{k+1} - x^k = \sum_{j=1}^{N} \alpha_j d_j$, siendo α_j el paso dado en la dirección d_j. El nuevo conjunto de direcciones ortogonales para reiniciar la búsqueda, $d_1, d_2, \ldots, d_N$, se obtiene mediante un procedimiento de Gram-Schmidt o de ortogonalización de vectores, de forma que:

$$
\begin{aligned}
a_j &= \begin{cases} d_j & \text{si} \quad \alpha_j = 0 \\ \sum_{i=j}^{N} \alpha_i d_i & \text{si} \quad \alpha_j \neq 0 \end{cases} \\
b_j &= \begin{cases} a_j & \text{si} \quad j = 1 \\ a_j - \sum_{i=1}^{j-1} (a_j^T d_i) d_i & \text{si} \quad j \geq 2 \end{cases} \\
d_j &= \frac{b_j}{\|b_j\|}
\end{aligned}
\tag{B.34}
$$

El algoritmo B.10 muestra una posible implementación de este método donde se realizan búsquedas lineales a lo largo de cada dirección. En la versión original, al igual que en el método de Hooke y Jeeves se realizaban pasos discretos a lo largo de las direcciones [8]. El procedimiento construir_vectores es el encargado de obtener los nuevos vectores ortogonales, según (B.34).

B.2.5. Métodos de ramificación y acotación

Es un método ampliamente usado en problemas complejos de optimización. Básicamente realiza una ramificación que consiste en dividir en una serie de subconjuntos el conjunto de soluciones posibles, y a continuación realiza la acotación, determinando las cotas inferiores y superiores de cada uno de ellos.

Sea el problema de optimización global siguiente:

$$
\text{mín}\, f(x), \quad x \in D \subset R^n \tag{B.35}
$$

Algoritmo B.11 Ramificación y acotación

Ramificación-acotación : $M_0 \times \beta_0 \times S_{M_0} \times D \times \epsilon \longrightarrow \bar{x}$
$\alpha_0 \longleftarrow \text{mín}(f(S_{M_0}))$
$\beta(M_0) \longleftarrow \beta_0$
$N_k \longleftarrow M_0$
Si $(\alpha_0 < \infty)$
  Escoger $x^0 \in \arg\text{mín}\, f(S_{M_0})$ tal que $f(x^0) = \alpha_0$
$k \longleftarrow 0$
Mientras $(\alpha_k - \beta_k < \epsilon)$
  $k \longleftarrow k + 1$
  $R_k \longleftarrow N_{k-1} - M \in N_{k-1}$ tal que $\beta(M) \geq \alpha_{k-1}$
  $P_k \longleftarrow$ Seleccionar_conjunto(R_k)
  $P'_k \longleftarrow$ Particionar(P_k)
  $M'_k \longleftarrow P'_k - M \in P'_k$ tal que $M \bigcap D = \varnothing$ o $\text{mín}\, f(D) \notin M$
  Para cada $M \in M'_k$, donde $M \subset M' \in N_{k-1}$ **asignar**
    $S_M \subseteq M \bigcap D$ y $S_M \supseteq M \bigcap S_{M'}$
    Si M es conocido
      $\beta(M') \leq \beta(M) \leq \text{ínf}\, f(M \bigcap D)$
    Si M es incierto
      $\beta(M') \leq \beta(M) \leq \text{ínf}\, f(M)$
  $\alpha(M) \longleftarrow \text{mín}\, f(S_M)$
  $N_k \longleftarrow (R_k/P_k) \bigcup M'_k$
  $\alpha_k \longleftarrow \text{ínf}\, \{\alpha(M); M \in N_k\}$
  $\beta_k \longleftarrow \text{mín}\, \{\beta(M); M \in N_k\}$
  Si $(\alpha_k < \infty)$
    Escoger $x^k \in D$ tal que $f(x^k) = \alpha_k$
Devolver x^k tal que $f(x^k) = \alpha_k$

Este algoritmo consiste en dado un conjunto M, donde se encuentran las soluciones posibles, realizar un particionado del mismo, M_i. A continuación determina para cada subconjunto las cotas inferiores y superiores (si es posible), $\beta(M_i)$ y $\alpha(M_i)$ respectivamente, de forma que satisfagan:

$$\beta(M) \leq \text{ínf}\, f(M_i \bigcap D) \leq \alpha(M_i) \tag{B.36}$$

De este modo:

$$\begin{aligned} \beta &= \min_i \beta(M_i) \\ \alpha &= \min_i \alpha(M_i) \\ \beta &\leq \min f(D) \leq \alpha \end{aligned} \tag{B.37}$$

El criterio de finalización del algoritmo vendrá dado cuando dichas cotas sean prácticamente iguales, es decir: $\alpha - \beta \leq \epsilon$.

El algoritmo B.11 muestra un prototipo de algoritmo de ramificación y acotación ([68]). Éste recibe como parámetros de entrada, el conjunto de soluciones factibles, D, un subconjunto de éste ,$S_{M_0} \subset D$, y otro donde se encuentra, $M_0 \supseteq D$. Además, recibe una aproximación al mínimo, $-\infty < \beta_0 \leq \min f(D)$, y la condición de terminación del algoritmo ϵ, que controla la exactitud del proceso de optimización.

En [68] podemos encontrar un estudio sobre la convergencia del método, así como diversos ejemplos de algoritmos de este tipo.

B.2.6. Métodos Lipschitzianos

Una función real se denomina *Lipschitziana* en un conjunto $M \subset R^n$, si existe una constante $L = L(f, M) > 0$, tal que:

$$|f(x) - f(y)| \leq L\|x - y\|, \forall x, y \in M \tag{B.38}$$

Toda función diferenciable continua con gradientes limitados en M es Lipschitziana en M, donde:

$$L = \sup \{\|\nabla f(x)\| : x \in M\} \tag{B.39}$$

Si una función f es Lipschitziana con una constante L, también lo es con las constantes $L' > L$.

El esquema general de resolver un problema de optimización global representado por (B.35), siendo f una función Lipschitziana se puede apreciar en el algoritmo B.12, propuesto por Piyavskii y Shubert ([68]) para funciones unidimensionales. Una extensión al caso n-dimensional fue propuesto por Mladineo.

Algoritmo B.12 Método Lipschitziano

Lipschitziano : $x^0 \times L \longrightarrow x^*$
$F_0(x) \longleftarrow -L\|x - x^0\| + f(x^0)$
$x^1 \in \arg\min F_0(R^n)$
$k \longleftarrow 0$
Mientras $(\|x^{k+1} - x^k\| < \epsilon)$
$k \longleftarrow k + 1$
$F_k(x) \longleftarrow \max_{0 \leq i \leq k} \left\{ f(x^i - L\|x - x^i\| \right\}$
$x^{k+1} \in \arg\min F_k(R^n)$
Devolver x^{k+1}

Sin embargo, el problema de los métodos de optimización Lipschitzianos requieren el conocimiento de la constante Lipschitziana para las funciones implicadas. Ésta puede ser estimada, pero encontrar una buena estimación de la misma puede ser más incluso más difícil que el problema original.

B.2.7. Métodos de aproximación exterior

Estos métodos, conocidos también como *métodos de relajación*, se encuentran dentro de las herramientas básicas de muchos campos de optimización, y han sido empleado de muchas formas y variantes. Consiste en relajar el conjunto de soluciones posibles D en otro más simple D^1, y optimizar la función sobre dicho conjunto. Si la solución que se encuentra sobre el problema relajado se encuentra en D finaliza. En caso contrario, se establece una nueva relajación D^2 que es mejor aproximación a D que D^1, se sustituye D^1 por D^2 y se repite el proceso.

Si consideramos el problema de optimización (B.35), el método de aproximación exterior que lo resuelve consiste en sustituirlo por una secuencia de problemas relajados:

$$\min f(x), \quad x \in D^k \subset R^n \tag{B.40}$$

donde $R^n \supset D^1 \supset D^2 \supset \ldots \supset D$ y $\min f(D^k) \longrightarrow \min f(D), (k \to \infty)$.

De este modo, los conjuntos D^k crean una familia F con las siguientes características:

- $D^k \subset R^n$ son cerrados, y su problema correspondiente, representado por (B.40), puede ser resuelto.

- Para cualquier conjunto D^k y cualquier punto $x^k \in D^k$ D, se puede definir una restricción $l^k : R^n \longrightarrow R$, tal que:

$$\begin{aligned} & l^k(x) \leq 0, \forall x \in D \\ & l^k(x^k) > 0 \\ & \left\{ x \in D^k : l^k(x) \leq 0 \right\} \in F \end{aligned} \tag{B.41}$$

El algoritmo B.13 muestra una posible implementación del mismo.

Algoritmo B.13 Método de aproximación exterior

Aproximación-exterior : $D^1 \longrightarrow x^*$
$k \longleftarrow 1$
$x^k \in \arg\min f(D^k)$
Mientras $(x^k \notin D)$
  $l^k \longleftarrow Construir\_Restriccion(D^k)$
  $D^{k+1} \longleftarrow D^k \bigcap \left\{ x : l^k(x) \leq 0 \right\}$
  $k \longleftarrow k + 1$
  $x^k \in \arg\min f(D^k)$
Devolver x^k

B.2.8. Métodos de intervalos

Consiste en ir subdividiendo la región de búsqueda en pequeñas cajas, de forma que se va comprobando en cada una si el óptimo se encuentra en ella para continuar subdividiéndola. Este proceso se repite hasta encontrar una caja lo suficientemente pequeña donde se encuentra el óptimo.

El método más conocido es el propuesto por Hansen, que hace uso de los siguientes conceptos:

Caja o intervalo El algoritmo necesita como parámetro de entrada una caja que constituye la región a optimizar. El resultado del mismo serán las distintas cajas que contienen los óptimos de la función. Una caja B se define como:

$$B = \{x \in R^n | \underline{x} \leq x \leq \overline{x}\} \tag{B.42}$$

Rango de una función El rango de una función definida sobre un conjunto de puntos $Y \subseteq X$, siendo X la caja que constituye la región a optimizar, se define como:

$$R(f(Y)) = \{f(y) | y \in Y\} \tag{B.43}$$

Función de inclusión Una función $F : \Pi(X) \longrightarrow \Pi(R)$, donde $\Pi(X)$ constituye el conjunto de todas las cajas contenidas en X, se denomina función de inclusión de f sobre la caja X, si:

$$R(f(Y)) \subseteq F(Y), \quad \forall Y \in \Pi(X) \tag{B.44}$$

Diámetro de una caja Dada una caja $X \subseteq \Pi(R)$ se define como:

$$diamX = \overline{x} - \underline{x} \tag{B.45}$$

Diámetro relativo de una caja Dada una caja $X \subseteq \Pi(R)$ sedefine como:

$$diam_r X = \begin{cases} diamX & \text{si} \quad 0 \in X \\ \frac{diamX}{|Centro(X)|} & \text{en otro caso} \end{cases} \tag{B.46}$$

Centro de una caja Dada una caja $X \subseteq \Pi(R)$, se define elpunto central de la caja X, como:

$$Centro(X) = \frac{1}{2}(\overline{x} - \underline{x}) \tag{B.47}$$

En el caso de una caja $X \subseteq \Pi(R^n)$, el diámetro, el diámetrorelativo y el centro de la caja se define componente a componente.

El algoritmo B.14 [75] muestra una posible implementación del método de intervalos de Hansen. Para poder utilizarlo es necesario definir la función de inclusión F de la función f. La entrada del algoritmo es la caja que constituye la región de optimización, así como el diámetro relativo máximo de cada subcaja generada. La salida que se obtendrá es una lista de pares $(Y^i, F(Y^i))$, tal que:

$$X^* \subseteq \bigcup_i Y^i, \quad f^* \in \bigcup_i F(Y^i) \tag{B.48}$$

Algoritmo B.14 Intervalos de Hansen

Intervalos : $B^0 \times \epsilon \longrightarrow S$
$S \longleftarrow \varnothing$
$\tilde{f} \longleftarrow \sup F(\text{Centro}(B^0))$
$L \longleftarrow (B^0, F(B^0))$
Repetir
  $(Y, F(Y)) \longleftarrow$ Sacar_Primer_Elemento_Lista(L)
  $k \longleftarrow$ Seleccionar_Direccion(Y)
  $(V_1, V_2) \longleftarrow$ Biseccionar_Caja(Y, k)
  $W_1 \longleftarrow F(V_1)$
  $W_2 \longleftarrow F(V_2)$
  Hacer para V_1 y V_2
    Si $\left(\text{ínf}(V_j) \leq \tilde{f}\right)$
      Si $(diam_r V_j \leq \epsilon)$
        $S \longleftarrow$ Introduce_Elemento_Lista(V_j, W_j)
      Si no
        $L \longleftarrow$ Introduce_Elemento_Lista(V_j, W_j)
  Hacer para $\forall (Y, F(Y)) \in L$
    Si $(\text{ínf}\, Y > \tilde{f})$
      Quitar_Elemento_Lista$(L, (Y, F(Y)))$
  Si $(L \neq \varnothing)$
    $(Y, F(Y)) \longleftarrow$ Sacar_Primer_Elemento_Lista(L)
    $\tilde{f} \longleftarrow \text{mín}\left\{\tilde{f}, \sup F(\text{Centro}(Y))\right\}$
Hasta que no se puedan unir dos subcajas de S en una sola
Devolver (S)

B.2.9. Análisis estadístico y muestreo Bayesiano

Estos métodos se basan en el conocimiento a priori de cierta información, que permita modelar la función a optimizar. Durante la optimización, estas características de la función se van adaptando y actualizando.

Estos métodos siguen el esquema que aparece en el algoritmo B.15. Podemos observar que cada vez que se obtiene un nuevo punto se vuelve a modelar la función.

En el primer paso de selección de puntos a muestrear para dar el conocimiento de la función suele ser útil emplear un esquema de muestreo de hipercubo latino [97], dado que garantiza cubrir el dominio de la región a optimizar. El número de puntos a muestrear será un parámetro de entrada, siendo recomendado disponer de al menos de 10 puntos por variable [140].

Esta estrategia asegura la convergencia si se genera un conjunto denso de puntos de búsquedas.

Algoritmo B.15 Búsqueda Bayesiana

Bayesiano : $n \times \epsilon \longrightarrow x$
$\{x_1, x_2, \ldots x_n\} \longleftarrow$ Seleccionar_puntos()
$\{y_1, y_2, \ldots y_n\} \longleftarrow$ Evaluar_puntos($\{x_1, x_2, \ldots x_n\}$)
$k \longleftarrow n$
Repetir
    Modelar_funcion($\{x_1, x_2, \ldots x_k\}, \{y_1, y_2, \ldots y_k\}$)
    $x_{k+1} \longleftarrow$ Buscar_maximo_mejora_esperada()
    $y_{k+1} \longleftarrow f(x_{k+1})$
    $k \longleftarrow k + 1$
Hasta que (Mejora_esperada($\{x_1, x_2, \ldots x_k\}, \{y_1, y_2, \ldots y_k\}$) $< \epsilon$)
Devolver x_k

Durante el modelado de la función se empleará un proceso estocástico que se ajuste a la estructura de puntos proporcionada. Este modelo de la función tendrá que estar cambiando según vayamos obteniendo más información de la función. En [140] podemos encontrar una discusión de diversos modelos de funciones. Una de las ventajas de estos métodos estadísticos es la elección de un modelo apropiado para la clase de problemas a la que se aplica. El inconveniente que presenta es la dificultad de implementar un modelo riguroso, que sea computacionalmente eficiente cuando nos enfrentamos a un problema de optimización de muchas dimensiones.

El éxito del algoritmo de optimización Bayesiana depende del modelo de la función. Por este motivo, cuando el modelo no se ajusta bien es posible realizar un ajuste del mismo para responder a los nuevos puntos de muestreo. En [140] podemos encontrar varias estrategias que permiten evaluar la calidad del modelo escogido.

La búsqueda del punto que obtiene la máxima mejora esperada en la función, se basa en la idea de que cualquier evaluación de la función adicional constituye una reducción potencial de la función. Han sido propuestos diversos criterios, que pueden encontrarse en [140]. Conviene reseñar que en [106] realiza este criterio apoyándose con una técnica de optimización local.

El criterio de terminación del algoritmo suele escogerse de forma que el máximo de la mejora esperada sea más pequeño que una tolerancia introducida previamente como parámetro del algoritmo [140].

B.2.10. Métodos de múltiple inicio

Estos métodos consisten en realizar múltiples ejecuciones individuales de un método de descenso (véase apartado 2.4.1) desde distintos puntos de inicio. La mejor solución que se encuentra tras todas las ejecuciones es el óptimo que proporciona. Es fácil verificar que una solución óptima tiene una probabilidad de 1 de encontrarse cuando se realizan infinitas ejecuciones. El algoritmo B.16 muestra el esquema de estos métodos.

Algoritmo B.16 Múltiple inicio

Múltiple_Inicio : $n \times \epsilon \longrightarrow x$
$k \longleftarrow 0$
$x^k \longleftarrow$ Generarp̂untoâleatorio()
$x^{min} \longleftarrow$ MetodoD̂escenso(x^k, ϵ)
Mientras $(k < n)$
$k \longleftarrow k + 1$
    $x^k \longleftarrow$ Generarp̂untoâleatorio()
    $\bar{y}^k \longleftarrow$ MetodoD̂escenso(x^k, ϵ)
    Si $\left(f(x^{min}) > f(\bar{y}^k)\right)$
        $x^{min} \longleftarrow x^k$
Devolver x^{min}

Los estudios de estos métodos se centran en la eficacia de éstos en el procesamiento paralelo y la elección de una distribución probabilística para los puntos a seleccionar. En [72] se encuentra un estudio de estas características tanto en un modelo de optimización continua como uno discreto.

B.2.11. Métodos de agrupamiento y enlace único multinivel

Un método de agrupamiento comienza con un conjunto de puntos de la región a optimizar y crea grupos cerrados de puntos que corresponden a regiones de atracción. Un grupo cerrado C es un conjunto de puntos de una región de atracción, con las siguientes características:

- C contiene exactamente un mínimo local x^*.
- Cualquier algoritmo de descenso que toma como punto de partida un punto $x \in C$ converge al óptimo x^*.
- Las propiedades anteriores implican que C contenga exactamente un único punto estacionario.

Aunque los métodos de agrupamiento disponen de varias técnicas para identificar esas regiones de atracción, el algoritmo básico es el mismo. Se escoge un punto como referencia del conjunto, normalmente el que posee un valor más bajo de la función. El resto de punto se añaden al grupo C mediante una regla de agrupamiento o hasta que se satisfaga el criterio de terminación. Los distintos métodos difieren en la regla de agrupamiento y en el criterio de terminación.

Los dos métodos más comunes [153] se basan en la densidad del grupo C o en la distancia entre los puntos. En el primer método, el criterio de terminación viene dado por el número de puntos dentro del grupo C por unidad de volumen. Esta relación no puede estar por encima de un cierto valor. En el segundo método, el criterio de terminación viene dado por una distancia crítica que deben cumplir los puntos de un mismo grupo C.

B.2.11.1. Enlace único

Es un nombre específico para una estrategia de agrupamiento. Estos métodos empiezan evaluando la función en un conjunto de puntos. Posteriormente aplican un método de optimización local (normalmente se emplea un método de descenso) al punto que posea un valor de la función más pequeño (en el caso de un proceso de maximización sería el más alto), y se crea un grupo C sobre el resultado del óptimo local alcanzado. Los puntos cuya distancia al grupo C se encuentran por debajo de una cierta distancia se añaden al grupo. La distancia entre un punto x y un grupo C se define como la mínima distancia entre x y cualquier punto del grupo. Cuando no pueden ser añadidos más puntos se dice que el grupo ha sido terminado. Si existen todavía puntos sin agrupar se vuelve a repetir el proceso con el siguiente punto cuyo valor de la función sea el más bajo.

Estos métodos son descritos en [136]. Sin embargo, actualmente han sufrido ligeras variaciones. La principal es que los puntos se agrupan en grupos de un tamaño fijo M, y continuamente son reagrupados en cada iteración según los nuevos grupos que se van formando [153].

B.2.11.2. Enlace único multinivel

Estos métodos se caracterizan por aplicar el algoritmo de enlace único al conjunto de puntos completo, de forma que el método de optimización local es aplicado a todos los puntos disponibles, salvo que exista un punto cercano a él (según la distancia crítica) que posea un valor de la función más pequeño.

El método descrito no forma agrupamientos, y se aplica al conjunto completo de puntos disponibles sin reducción ni concentración. Se entiende por reducción el hecho de descartar los puntos con los valores de la función más alto. La concentración consiste en transformar un conjunto de

puntos en uno o unos pocos donde aplicar el algoritmo de optimización local.

Esta variación del método anterior garantiza que, si se ejecuta el algoritmo ejecutarse de forma infinita, encontrará todos los óptimos globales con una probabilidad de 1 [153].

B.2.12. Búsqueda aleatoria pura

Es el método más simple. Consiste en generar aleatoriamente puntos dentro del espacio de soluciones posibles. Para cada una de ellas se determina el valor de la función a optimizar, quedándose con la mejor solución encontrada. Es un método simple, pero que no se garantiza la convergencia ni la calidad de la solución encontrada. El algoritmo B.17 muestra una posible implementación de este método.

Algoritmo B.17 Búsqueda Aleatoria

Búsqueda-Aleatoria : $N \longrightarrow x$
$x \longleftarrow$ Genera_Solucion_Aleatoria()
$t \longleftarrow 1$
Mientras $(t < N)$
  $\bar{y} \longleftarrow$ Genera_Solucion_Aleatoria()
  Si $(f(\bar{y}) < f(x))$
    $x \longleftarrow \bar{y}$
  $t \longleftarrow t + 1$
Devolver x

Huntch y col.[153] muestran que si la función es evaluada en puntos que siguen una distribución uniforme sobre la región a optimizar, se puede demostrar que el método converge al óptimo global con una probabilidad de 1.

B.2.13. Búsqueda adaptativa pura

Estos métodos son un proceso iterativo, donde cada iteración consta de dos fases, una de construcción de una solución y otra de búsqueda local [40]. El algoritmo B.18 muestra un esquema de estos métodos.

Algoritmo B.18 Búsqueda adaptativa

BúsquedaÂdaptativa : $n \times \epsilon \longrightarrow x$
$k \longleftarrow 0$
$x^k \longleftarrow$ ConstruirŜolucionÂleatoria()
$x^{min} \longleftarrow$ BusquedaL̂ocal(x^k, ϵ)
Mientras $(k < n)$
  $k \longleftarrow k + 1$
  $x^k \longleftarrow$ ConstruirŜolucionÂleatoria()
  $x^k \longleftarrow$ BusquedaL̂ocal(x^k, ϵ)
  Si $\left(f(x^{min}) > f(x^k)\right)$
    $x^{min} \longleftarrow x^k$
Devolver x^{min}

Estos métodos se caracterizan porque la solución se va construyendo paso a paso, de forma que en cada iteración intenta determinar un componente de la solución actual. De este modo, en la fase de construcción una solución válida se construye de forma iterativa, con un componente cada vez. En cada iteración, se tiene que determinar cuál será el siguiente elemento a construir. El algoritmo B.19 muestra de forma esquemática esta fase. La selección de un elemento es realizada de forma aleatoria, pero se tiene en cuenta el valor de la función con la solución que se está construyendo.

Algoritmo B.19 Construcción de una solución adaptativa

Construcción :$\longrightarrow x$
$x \longleftarrow \{\}$
Mientras (Solucion_no_construida)
  $s \longleftarrow$ Seleccionar_Elemento()
  $x \longleftarrow x \cup s$
Devolver x

La fase de búsqueda local intenta mejorar la solución construida. Para ello suele emplearse un método de descenso, de forma que se obtenga el óptimo local más cercano a la solución construida.

B.2.14. Enfriamiento lento simulado

El enfriamiento lento simulado se clasifica dentro de los algoritmos de búsqueda local denominados *algoritmos de umbral*. Un pseudocódigo de éstos viene dado por el algoritmo B.20. El procedimiento Inicializar() selecciona una solución inicial del espacio de soluciones, Generar() selecciona una solución de la vecindad de la solución actual, y el procedimiento Terminar() evalúa el criterio que define la terminación del algoritmo. Los algoritmos de umbral continuamente seleccionan un vecino a la solución actual y compara la diferencia en coste entre ambos y un umbral. Si se encuentra debajo del umbral, el vecino reemplaza a la solución actual. En caso contrario, se continúa la búsqueda con la solución actual. La secuencia t_k denota los distintos umbrales usados en las distintas iteraciones.

Algoritmo B.20 Algoritmo de umbral

Umbral $:\longrightarrow x$
$x \longleftarrow$ Inicializar()
$k \longleftarrow 0$
Repetir hasta (Terminar())
  $y \longleftarrow$ Generar(x)
  Si $(f(y) - f(x) < t^k)$
    $x \longleftarrow y$
  $k \longleftarrow k + 1$
Devolver x

Existen tres clases de algoritmos de umbral, dependiendo de la naturaleza de éste [1]:

Mejora iterativa En este caso, el umbral $t^k = 0, \quad \forall k$. Esta clase de algoritmos sólo aceptan al vecino si realmente produce una disminución del coste de la función. También reciben el nombre de *métodos de búsqueda local*.

Aceptación de umbral Utilizan una secuencia decreciente de umbrales deterministas, de forma que $t^k \geq 0, t^k \geq t^{k+1}, \quad \lim_{k\to\infty} t^k = 0$. Debido a que los umbrales son positivos, es posible aceptar vecinos peores que la solución actual. Sin embargo, dado que éste tiende a decrecer gradualmente hasta 0, llega un momento en el que sólo se aceptarán mejoras de las soluciones.

Enfriamiento lento simulado El umbral es una variable aleatoria que sigue una distribución probabilística. En la práctica, la función que de-

termina el umbral se escoge de forma que las soluciones que producen un gran empeoramiento en la solución actual tienen una probabilidad pequeña de ser aceptada, mientras que las soluciones que producen pequeños incrementos en el coste tienen una gran probabilidad de ser seleccionadas.

El enfriamiento lento simulado debe su nombre a la analogía que presenta con la termodinámica, y más en concreto con el proceso natural del recocido de un sólido en busca del estado de mínima energía correspondiente al equilibrio térmico. Por ejemplo, para generar una figura de cristal se empieza calentando los materiales hasta un estado de moldeado. A continuación se reduce la temperatura del cristal, hasta que la estructura de cristal esté congelada. Si el enfriamiento es muy rápido, el efecto sobre el cristal es perjudicial, pudiendo aparecer irregularidades en el mismo. En metalurgia por ejemplo, la clave reside en enfriar lentamente el sólido previamente fundido para que a cada temperatura se llegue al equilibrio y se recoloquen todos los átomos en los diferentes niveles de energía. Si el enfriamiento se hace correctamente, al final se llega al denominado estado fundamental, en el que las partículas forman retículas perfectas (cristal) y el sistema está en su nivel energético más bajo. En el caso contrario, si se enfría rápidamente, la sustancia resultante puede formar una estructura cristalina con múltiples defectos, con estructuras óptimas restringidas a zonas locales muy concretas.

Algoritmo B.21 Enfriamiento lento simulado

ELS : $T_{max} \times T_{min} \times K \times R \longrightarrow x$
$T \longleftarrow T_{max}$
$x \longleftarrow$ Escoger_Punto_Inicial_Aleatorio$()$
Mientras $(T \geq T_{min})$
  $k \longleftarrow 0$
  Mientras $(k < K)$
    $y \longleftarrow$ Escoger_Vecino(x)
    Si $(f(y) < f(x))$
      $x \longleftarrow y$
    Si no
      $\alpha \longleftarrow$ Genera_Aleatorio$(0, 1)$
      Si $\left(\alpha < e^{\frac{f(y)-f(x)}{T}}\right)$
        $x \longleftarrow y$
    $k \longleftarrow k + 1$
  $T \longleftarrow R \times T$
Devolver x

Este concepto es aplicado a los problemas de optimización. Así, Kirkpatrick y otros [82] proponen un método que básicamente conjuga una búsqueda aleatoria que permite los movimientos en cualquier dirección durante las etapas iniciales, con una especie de descenso por gradiente en las últimas iteraciones, cuando la temperatura ya es muy baja y se acerca al valor nulo. De esta forma, se define una temperatura que nos indicará la cercanía a un óptimo. Inicialmente se empezará con un valor elevado para producir una búsqueda aleatoria, y posteriormente decrecer el valor de dicha temperatura. Al final de la ejecución, el valor de ésta será pequeño.

El procedimiento de enfriamiento lento simulado se puede observar en el algoritmo B.21. Los parámetros de entrada corresponde a la temperatura inicial (T_{max}), la temperatura de congelación (T_{min}), el ratio de enfriamiento (R) y el número máximo de iteraciones a realizar (K).

Bibliografía

[1] Aarts, E. y Lenstra, J. K., editors.

Local search in combinatorial optimization.

John Wiley & Sons, 1997.

[2] ActiveVOS.

Activebpel WS-BPEL and BPEL4WS engine.

http://sourceforge.net/projects/activebpel, 2008.

[3] Adamopoulos, K., Harman, M. y Hierons, R. M.

How to overcome the equivalent mutant problem and achieve tailored selective mutation using co-evolution.

GECCO (2): Proceedings of the Genetic and Evolutionary Computation Conference, pág. 1338–1349, 2004.

[4] Al-Sawafi, M.M.S. y Jervase, J.A.

A micro-genetic algorithm-based cdma multi-user detector.

Proceedings of the Conference on Communication Networks and Services Research, pág. 175–180, Fredericton (Canadá), mayo 2004.

[5] Alander, J. T., Mantere, T. y Turunen, P.

Genetic Algorithm Based Software Testing

http://citeseer.ist.psu.edu/40769.html

[6] Baker, J. E.

Reducing bias and inefficiency in the selection algorithm.

Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application, pág. 14–21, Hillsdale, NJ, USA, 1987. L. Erlbaum Associates Inc.

[7] Battiti, R.

First- and second-order methods for learning: between steepest descent and newton's method.

Neural Comput., 4(2):141–166, 1992.

[8] Bazaraa, M. S., Sherali, H. D. y Shetty, C. M.

Nonlinear programming. Theory and algorithms.

John Wiley & Sons, Inc., 2ª edition, 1993.

[9] Berthiau, G. y P. Siarry.

A genetic algorithm for globally minimizing functions of several continuous variables.

Second International Conference on Metaheuristics, Sophia-Antipolis (France), 1997.

[10] Bertsekas, D. P.

Constrained Optimization and Lagrange Multiplier Methods.

Academic Press, New York, 1982.

[11] Bertsekas, D. P.

Nonlinear programming.

Athena Scientific, 1995.

[12] Bland, R.G. y Shallcross, D.F.

Large traveling salesman problems arising from experiments in x-ray crystallography: a preliminary report on computation.

Operations Research Letters, (8):125–128, 1989.

[13] Bottaci, L.

Instrumenting programs with flag variables for test data search by gen etic algorithms.

GECCO '02: Proceedings of the Genetic and Evolutionary Computation Conference, pág. 1337–1342, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[14] Brady, R. M.

Optimization strategies gleaned from biological evolution.

Nature, 317:804–806, 1985.

[15] Buckley, A. y LeNir, A.

Qn-like variable storage conjugate gradients.

Mathematical programming, (27):155–175, 1983.

[16] Bui, T.Ñ. y Moon, B. R.

A new genetic approach for the traveling salesman problem.

Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on, pág. 7–12 vol.1, 1994.

[17] Canfora, G. y Di Penta, M.

Testing services and service-centric systems: challenges and opportunities.

IT Professional, 8(2):10–17, 2006.

[18] Cantú-paz, E.

A survey of parallel genetic algorithms.

Calculateurs Paralleles, Reseaux et Systems Repartis, 10:141–171, 1997.

[19] Chelouah, R. y Siarry, P.

A continuous genetic algorithm designed for the global optimization of multimodal functions.

Journal of heuristics, (6):191–213, 2000.

[20] Coello, C.A., Van Veldhuizen, D. A. y Lamont, G.B.

Evolutionary algorithms for solving multi-objective problems.

Kluwer Academica Publishers, New York, 2002.

[21] I. S. O. E. Commision.

ISO/IEC 12207. International Standard. Software Life Cycle processes.

1995.

[22] Davis, L.

Applying adaptive algorithms to epistatic domains.

Proceedings of the International Joint Conference on Artificial Intelligence, pág. 162–164, 1985.

[23] Davis, T. E. y Principe, J. C.

A markov chain framework for the simple genetic algorithm.

Evolutionary Computation, 1(3):269–288, 1993.

[24] De Jong, K.

An analysis of the behaviour of a class of genetic adaptive systems.

PhD thesis, University of Michigan, 1975.

[25] Delamaro, M. y Maldonado, J.

Proteum–a tool for the assessment of test adequacy for c programs.

Proceedings of the Conference on Performability in Computing System (PCS 96), pág. 79–95, Julio 1996.

[26] DeMillo, R. A., Lipton, R. J. y Sayward, F. G.

Hints on test data selection: Help for the practicing programmer.

Computer, 11(4):34–41, 1978.

[27] Denton, J. W. y Hung, M. S.

A comparison of nonlinear optimization methods for supervised learning in multilayer feedforward neural networks.

European Journal of Operational Research, 93(2):358–368, September 1996.

[28] Diego-Mas, J.

Optimización de la distribución en planta de instalaciones industriales mediante algoritmos genéticos. Aportación al control de la geometría de las actividades.

PhD thesis, Universidad Politécnica de Valencia, Enero 2006.

[29] Dijkstra, E. W.

Notes on structured programming.

Structured programming, pág. 1–82. Academic Press Ltd., London, UK, 1972.

[30] Diversos autores.

Bibliotecas de algoritmos genéticos.

http://geneticalgorithms.ai-depot.com/Libraries.html, 2007.

[31] Domínguez Jiménez, J. J., Lozano Segura, S. y Calle Suárez, M.

Búsqueda genética en vecindades: Aplicación al problema del viajante simétrico.

Actas del primer Congreso Español de Algoritmos Evolutivos y Bioinspirados — AEB'02, pág. 89–95, Mérida, España, 2002.

[32] Domínguez Jiménez, J.J, Estero Botaro, A. y Medina Bulo, I.

A framework for mutant genetic generation for ws-bpel.

Proceedings of the 35th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2009, Špindlerův Mlýn (República Checa), 2009.

(Pendiente de publicar).

[33] Domínguez Jiménez, J.J, Estero Botaro, A., Medina Bulo, I. y García Domínguez, A.

Gamera: An automatic mutant generation system for ws-bpel compositions.

In *The 18th International World Wide Web Conference (WWW09)*, Madrid (España), 2009.

(Pendiente de aceptación).

[34] Domínguez, J.J., Lozano, S., Calle, M. y Smith, K.

A new method for combinatorial optimization: genetic neighborhood search.

eural Network World, International Journal on Non-Standard Computing and Artificial Intelligence, 12(6):533–548, 2002.

[35] Drechsler, R., Becker, B. y Drechsler, N.

Genetic algorithm for minimisation of fixed polarity reed-muller expressions.

IEE Proceedings - Computers and Digital Techniques, volume 147, pág. 349–353, septiembre 2000.

[36] Dumitrescu, D., Lazzerini, B., Jain, L.C. y Dumitrescu, A.

Evolutionary computation.

CRC Press LLC, Florida, 2000.

[37] Díaz, E., Blanco, R. y Tuya, J.

Los métodos de generación de casos de prueba y su automatización.

Técnicas cuantitativas para la gestión en la Ingeniería del software, pág. 291–322. Netbiblio, 2007.

[38] Estero Botaro, A., Palomo Lozano, F. y Medina Bulo, I.

Mutation operators for WS-BPEL 2.0.

ICSSEA 2008: Proceedings of the 21th International Conference on Software & Systems Engineering and their Applications, Paris, France, (December 2008). To be published, 2008.

[39] Fensel, D. y Bussler, C.

The Web service modeling framework WSMF.

Electronic Commerce Research and Applications, pág. 113–137, 2002.

[40] Feo, T. A. y Resende, M. G. C.

Greedy randomized adaptive search procedures.

Journal of global optimization, , vol. 6:104–133, 1995.

[41] Fleming, P., Pohlheim, H. y Fonseca, C.

Genetic algorithm toolbox user's guide, acse.

Technical report, 1994.

[42] Fletcher, R.

Practical methods of optimization.

John Wiley, 2ª edition, 1987.

[43] Fletcher, R.

An overview of unconstrained optimization.

Numerical analysis report NA/149, Dept. of Mathematics and Computer Science, University of Dundee, Escocia, Junio 1993.

[44] Fox, B. R. y McMahon, M. B.

Genetic operators for sequencing problems.

Foundations of genetic algorithms, pág. 284–300. Morgan Kaufmann Publishers, San Mateo, 1991.

[45] Freisleben, B. y Merz, P.

A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems.

International Conference on Evolutionary Computation, pág. 616–621, 1996.

[46] Freisleben. B. y Merz, P.

New genetic local search operators for the traveling salesman problem.

Proceedings of theFourthConference on Parallel Problem Solving from Nature(PPSN IV), vol. 1141, pág. 890–899, Berlin, 1996. Springer.

[47] García-Fanjul, J., Tuya, J. y de la Riva, C.

Generating test cases specifications for compositions of web services.

International Workshop on Web Services, Modeling and Testing (WS-MaTe), pág. 83–94, 2006.

[48] García-Fanjul, J., Tuya, J. y de la Riva, C.

Generación sistemática de pruebas para composiciones de servicios utilizando criterios de suficiencia basados en transiciones.

JISBD 2007: Actas de las XII Jornadas de Ingeniería del Software y Bases de Datos, 2007.

[49] Garey, M. y Johnson, D. S.

Computers and intractibility; a guide to the theory of NP-completeness.

W. H. Freeman and Co., 1979.

[50] Gerstoft, P.

Inversion of acoustic data using a combination of genetic algorithms and the Gauss-Newton approach.

Acoustical Society of America Journal, 97:2181–2190, Apr. 1995.

[51] Gill, P. E. y Leonard, M. W.

Limited-memory reduced-hessian methods for large-scale unconstrained optimization.

Report NA97-1, Dept. of Mathematics, Santa Clara Universitiy, California, Estados Unidos, 1997.

[52] Gill, P. E., Murray, W. y Wright, M. H.

Practical optimization.

Academic Press, 1981.

[53] Glover, F.

Ejection chains, reference structures and alternating path methods for traveling salesman problems.

Discrete Appl. Math., 65(1-3):223–253, 1996.

[54] Goldberg, D. E.

A meditation on the application of genetic algorithms.

Technical Report 98003, Illinois Genetic Algorithms laboratory, Febrero 1998.

[55] Goldberg, D. E.

Genetic algorithms in search, optimization and machine learning.

Addison-Wesley, Reading, 1989.

[56] Goldberg, D. E. y Lingle, Jr. R.

Alleles, loci and the traveling salesman problem.

Proceedings of the First International Conference on Genetic Algorithms and Their Applications, pág. 154–159, 1985.

[57] Grefenstette, J.

A user's guide to genesis, version 5.0.

http://www.genetic-programming.com/c2003genesisgrefenstette.txt, octubre 1990.

[58] Hamlet, R. G.

Testing programs with the aid of a compiler.

IEEE Transactions Software Engineering, 3(4):279–290, 1977.

[59] Hassoun, M. H.

Fundamentals of artificial neural networks.

MIT Press, 1995.

[60] Haupt, R.

Comparison between genetic and gradient-based optimization algorithms for solving electromagnetics problems.

IEEE Transactions on Magnetics, 31(3):1932–1935, mayo 1995.

[61] Haupt, R.L. y Haupt, S.E.

Practical genetic algorithms.

John Wiley & Sons, New York, 1998.

[62] Haykin, S.

Neural Networks: A Comprehensive Foundation (2nd Edition).

Prentice Hall, July 1998.

[63] Hecht-Nielsen, R.

Neurocomputing.

Addison-Wesley, 1989.

[64] Heffner, R.

Real-world soa: Soa platform case studies.

Forrester Research, Septiembre 2005.

[65] Holland, J. H.

Adaptation in natural and artificial systems.

Technical report, University of Michigan Press, Ann Arbor, MI, 1975.

[66] Holland, J. H.

Adaptation in natural and artificial systems.

MIT Press, Cambridge, 1992.

[67] Homaifar, A., Guan, S. y Liepins, G. E.

A new approach on the traveling salesman problem by genetic algorithms.

Proceedings of the 5th International Conference on Genetic Algorithms, pág. 460–466, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[68] Horst, R. y Tuy, H.

Global optimization. Deterministic approaches.

Springer-Verlag, 2ª edition, 1993.

[69] Houck, C.R., Joines, J.A. y Kay, M.G.

A genetic algorithm for function optimization: a matlab implementation.

Technical Report NCSU-IE-TR-95-09, North Carolina State University, octubre 1995.

[70] Howden, W. E.

Symbolic testing and the dissect symbolic evaluation system.

IEEE Transactions Software Engineering, 3(4):266–278, 1977.

[71] Hu, X. y Xie, C.

Niche genetic algorithm for robot path planning.

ICNC '07: Proceedings of the Third International Conference on Natural Computation (ICNC 2007), pág. 774–778, Washington, DC, USA, 2007. IEEE Computer Society.

[72] Hu, X., Shonkwiler, R. y Spruill, M. C.

Random restarts in global optimization.

Technical report, School of Mathematics, Georgia Institute of Technology, Atlanta, Estados Unidos, Enero 1994.

[73] IDC.

Research reports.

http://www.idc.com, 2008.

[74] Janikow, C. Z. y Michalewicz, Z.

An experimental comparison of binary and floating point representations in genetic algorithms.

ICGA, pág. 31–36, 1991.

[75] Jansson, C. y Knuppel, O.

A global minimization method: the multi-dimensional case.

Technical report, Technische Universitat Hamburg-Harburg, Hamburgo, Alemania, 1992.

Bericht 92.1.

[76] Johansson, E.M., Dowla, F.U. y Goodman, D.M.

Backpropagation learning for multi-layer feed-forward neural networks using the conjugate gradient method.

International Journal of Neural System, 2(4):291–302, 1991.

[77] Johnson, J. M. y Rahmat-Samii, V.

Genetic algorithms in engineering electromagnetics.

IEEE Antennas Propagation Magazine, 39(4):7–25, agosto 1997.

[78] Johnson, J.M. y Rahmat-Samii, Y.

Genetic algorithm optimization of wireless communication networks.

Proceedings of the IEEE Antennas and Propagation Society International Symposium, vol. 4, pág. 1964–1967, California (USA), Junio 1995.

[79] Jones, D.F., Mirrazavi, S.K. y Tamiz, M.

Multi-objective meta-heuristics: An overview of the current state of the art.

European Journal of Operational Research, (137):1–9, 2002.

[80] Kennedy, J. y Eberhart, R.C.

Swarm intelligence.

Morgan Kaufmann Publishers, San Francisco, 2001.

[81] King, K. N. y Offutt, A. J.

A Fortran language system for mutation-based software testing.

Software - Practice and Experience, 21(7):685–718, 1991.

[82] Kirkpatrick, S., Gelatt, C.D., y Vecchi, M.P.

Optimization by simulated annealing.

Science, 220(4598):671–680, mayo 1983.

[83] Korte, B.

Mathematical Programming: Recent Developments and applications, chapter Application of combinatorial optimization, pág. 1–55.

Kluwe, Dordrecht, 1989.

[84] Lawler, E.L., Lenstra, L.K., Rinnooy Kan, A.H.G. y Shmoys, D.B., editors.

The Traveling Salesman Problem: A guided tour in combinatorial optimization.

John Wiley & Sons, 1985.

[85] Lewis, R. M., Torczon, V. y Trosset, M. W.

Direct search methods: then and now.

Report 2000-26, Institute for Computer Applications in Science and Engineering, NASA Langle Research Center, NASA Langley Research Center, Hampton, Virginia, Estados Unidos, Mayo 2000.

[86] Lippmann, R. P.

An introduction to computing with neural nets.

SIGARCH Computer Architecture News, 16(1):7–25, 1988.

[87] Liu, D. C. y Nocedal, J.

On the limited memory bfgs method for large scale optimization.

Mathematical programming, (45):503–528, 1989.

[88] Lozano, S., Domínguez, J. J., Guerrero, F., Onieva L. y Larrañeta, J.

Intelligent Engineering Systems Through Artificial Neural Networks, vol. 7, chapter Training Feedforward Neural Network Using a Genetic Line Search, pág. 119–124.

ASME Press, 1997.

[89] Lozano, S., Domínguez, J. J., Guerrero, F. y Smith, K.

A new optimization method: Genetic line search.

Intelligent Systems Design and Applications (ISDA 2001), volume 2074 of *Computational Science – ICCS 2001, Lecture notes in computer science*, pág. 318–326, San Francisco, Estados Unidos, 2001.

[90] Lozano, S., Domínguez Jiménez, J. J., Guerrero, F., Onieva L. y Larrañeta J.

Advances in Soft Computing. Engineering Design and Manufacturing, chapter Unconstrained Optimization Using Genetic Box Search, pág. 379–389.

Springer-Verlag, Londres, 1999.

[91] Luenberger, D. E.

Programación lineal y no lineal.

Addison-Wesley Iberoamericana, 1989.

[92] Ma, Y. S., Offutt, J. y Kwon, Y. R.

MuJava: an automated class mutation system.

Software Testing, Verification & Reliability, 15(2):97–133, 2005.

[93] Ma, J., Tian, P. y Zhang, D.

Global optimization by darwin and boltzmann mixed strategy.

Computers & operations research, (27):143–159, 2000.

[94] Mantere, T. y Alander, J. T.

Evolutionary software engineering, a review.

Applied Soft Computing, 5(3):315–331, 2005.

[95] Mayer, P.

Design and Implementation of a Framework for Testing BPEL Compositions.

PhD thesis, Gottfried Wilhelm Leibniz Universität Hannover, 2006.

[96] Mayer, P. y Lübke, D.

Towards a bpel unit testing framework.

TAV-WEB '06: Proceedings of the 2006 workshop on Testing, analysis, and verification of web services and applications, pág. 33–42, New York, NY, USA, 2006. ACM.

[97] McKay, M. D., Beckman, R. J. y Conover, W. J.

A comparison of three methods for selecting values of input variables in the analysis of output from a computer code.

Technometrics, (21):239–245, 1979.

[98] Mcminn, P.

Search-based software test data generation: A survey

Software Testing, Verification and Reliability, pág. 105–156, vol. 14, 2004.

[99] Merz, P. and Freisleben, B.

Genetic local search for the tsp: New results.

Proceedings of the 1997 IEEE International Conference on Evolutionary Computation, pág. 159–164. IEEE Press, 1997.

[100] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M.Ñ., Teller, A. H. y Teller, E.

Equation of state calculations by fast computing machines.

The Journal of Chemical Physics, 21(6):1087–1092, 1953.

[101] Michael, C. C., McGraw, G., Schatz, M. y Walton, C. C.

Genetic algorithms for dynamic test data generation.

Automated Software Engineering, pág. 307–308, 1997.

[102] Michalewicz, Z.

Genetic algorithms + data structures = Evolution programs.

Springer-Verlag, 1992.

[103] Michalewicz, Z. y Fogel, D. B.

How to solve it: modern heuristics.

Springer-Verlag, 1998.

[104] Miller, B. L. y Shaw, M. J.

Genetic algorithms with dynamic niche sharing for multimodal function optimization.

Illigal Report 95010, Department of General Engineering, University of Illinois at Urbana-Champaign, 117 Transportation Building, 104 South Mathews Avenue, Urbana, IL 61801, Diciembre 1995.

[105] Mitchell, M.

An introduction to genetic algorithms.

Massachusetts Institute of Technology, 1996.

[106] Mockus, J.

Bayesian approach to global optimization.

Kluwer Academic Publishers, Dordrecht, Holanda, 1989.

[107] Montana, D. y Davis, L.

Training feedforward neural networks using genetic algorithms.

Proceedings of the International Joint Conference on Artificial Intelligence, vol. 1, pág. 762–767, 1989.

[108] More, J. J., Garbow, B. S. y Hillstrom, K. E.

Testing unconstrained optimization software.

ACM Transactions on mathematical software, , vol. 7(1):17–41, marzo 1981.

[109] More, J. J. y Thuente, D. J.

Line search algorithms with guaranteed sufficient decrease.

Report MCS-P330-1992, Argonne National Laboratory, Mathematics and Computer Science Division, 9700 South Cass Avenue, Argonne, Illinois, Estados Unidos, Octubre 1992.

[110] Mresa, E. S. y Bottaci, L.

Efficiency of mutation operators and selective mutation strategies: An empirical study.

Software Testing, Verification and Reliability, 9(4):205–232, 1999.

[111] MuClipse.

http://muclipse.sourceforge.net, 2007.

[112] Muhlenbein, H.

Evolution in time and space - the parallel genetic algorithm.

Foundations of Genetic Algorithms, pág. 316–337. Morgan Kaufmann, 1991.

[113] Muhlenbein, H. y Schlierkamp-voosen, D.

Analysis of selection, mutation and recombination in genetic algorithms.

Neural Network World, 3:907–933, 1993.

[114] Muhlenbein, H., Schomisch, M. y Born, J.

The parallel genetic algorithm as function optimizer.

Proceedings of the Fourth International Conference on Genetic Algorithms, pág. 271—-278, San Diego, CA, 1991.

[115] Musil, M., Wilmut, M.J. y Chapman, N.R.

A hybrid simplex genetic algorithm for estimating geoacousticparameters using matched-field inversion.

IEEE Journal of Oceanic Engineering, 24(3):358–369, 1999.

[116] Myers, G.J., Sandler, C., Badgett, T. y Thomas, T. M.

The Art of Software Testing, 2nd ed.

Wiley - Interscience, 2004.

[117] Nocedal, J.

Theory of algorithms for unconstrained optimization.

Acta Numérica, pág. 199–242, 1992.

[118] OASIS.

Web services business process execution language 2.0, 2007.

Organization for the Advancement of Structured Information Standards.

[119] Offutt, A. J., Rothermel, G. R. y Zapf, C.

An experimental evaluation of selective mutation.

Proceedings of the 15th International Conference on Software Engineering, pág. 100–107, 1993.

[120] Offutt, A. J., Lee, A., Rothermel, G.,R., Untch, H. y Zapf, C.

An experimental determination of sufficient mutant operators.

ACM Transactions on Software Engineering and Methodology, 5(2):99–118, 1996.

[121] Offutt, A. J. y Untch, R. H.

Mutation 2000: uniting the orthogonal.

Mutation testing for the new century, pág. 34–44. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[122] Oliver, I.M., Smith, D.J. y Holland, J.R.C.

A study of permutation crossover operators on the TSP.

Genetic Algorithms and Their Applications: Proceedings of the Second International Conference, pág. 224–230, 1987.

[123] Osowski, S., Bojarczak, P. y Stodolski, M.

Fast second order learning algorithm for feedforward multilayer neural networks and its applications.

Neural Netw., 9(9):1583–1596, 1996.

[124] Palacios, M., García-Fanjul, J., Tuya, J. y de la Riva, C.

Estado del arte en la investigación de métodos y herramientas de pruebas para procesos de negocio bpel.

Actas de las IV Jornadas Científico-Técnicas en Servicios Web y SOA (JSWEB-08), pág. 132–137, 2008.

[125] Palomo-Duarte, M., García-Domínguez, A. y Medina-Bulo, I.

Takuan: A dynamic invariant generation system for WS-BPEL compositio ns.

Proceedings of the 6th IEEE European Conference on Web Services, D ublin, Ireland, (November 2008). To be published, 2008.

[126] Papazoglou, M.

Web services technologies and standards.

Computing Surveys (enviado para su revisión).

[127] Pargas, R. P., Harrold, M. J. y Peck, R.
Test-data generation using genetic algorithms.
Software Testing, Verification & Reliability, 9(4):263–282, 1999.

[128] Parker, R. G. y Rardin, R. L.
Discrete optimization.
Academic Press, Inc., 1988.

[129] Petridis, V., Kazarlis, S. y Papaikonomou, A.
A genetic algorithm for training recurrent neural networks.
vol. 3, pág. 2706–2709, Nagoya (Japón), octubre 1993.

[130] Potra, F. A. y Shi, Y.
Efficient line search algorithm for unconstrained optimization.
Journal of optimization. Theory and applications, , vol. 85(3):679–704, Junio 1995.

[131] Powell, M. J. D.
Direct search algorithms for optimization calculations.
Acta numerica, vol. 7, pág. 287–336. Cambridge University Press, 1998.

[132] Rahmat-Samii, Y. y Michielssen, E.
Electromagnetic optimization by genetic algorithms.
John Wiley & Sons, New York, 1999.

[133] Reinelt, G.
TSPLIB, a traveling salesman problem library.
http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95, 1995.

[134] Rice, R.
Surviving the top 10 challenges of software test automation.
CrossTalk: The Journal of Defense Software Engineering, pág. 26–29, mayo 2002.

[135] Rinnooy Kan, A. H. G. y Timmer, G. T.
Stochastic methods for global optimization.
American Journal of Mathematical and Management Sciences, 2:1–2, 1984.

[136] Rinnooy Kan, A. H. G. y Timmer, G. T.

A stochastic approach to global optimization.

Byrd, R. H. and Schnabel, R. B., editors, *Numerical optimization. SIAM*. 1984.

[137] Rumelhart, D. E., Hinton, G. E. y Williams, R. J.

Learning representations by back-propagating errors.

pág. 696–699, 1988.

[138] Salomon, R.

A hybrid method for evolving neural network topologies.

Intelligent Engineering Systems Through Artificial Neural Networks, volume 4, pág. 147–152. ASME Press, 1994.

[139] Saxonica.

The Saxon XSLT and XQuery Processor.

http://saxon.sourceforge.net/, Oct. 2008.

[140] Schonlau, M., Welch, W. J. y Jones, D. R.

A data-analytic approach to bayesian global optimization.

Proceedings of the ASA, 1997.

[141] Sinha, A. y Goldberg, D. E.

A survey of hybrid genetic and evolutionary algorithms.

Illigal Report 2003004, Illinois Genetic Algorithms Laboratory, 2003.

[142] Smith, R.E., Goldberg, D.E. y Earickson, J.A.

SGA-C: A C-language implementation of a simple genetic algorithm.

Technical Report 91002, The Clearinghouse for Genetic Algorithms, 1991.

[143] Stender, J.

Parallel Genetic Algorithms: Theory and Applications.

IOS Press, 1993.

[144] Suh, J. Y. y Gucht, D. V.

Incorporating heuristic information into genetic search.

Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application, pág. 100–107, Hillsdale, NJ, USA, 1987. L. Erlbaum Associates Inc.

[145] Syswerda, G.

Handbook of Genetic Algorithms, chapter Schedule optimization using genetic algorithms, pág. 332–349.

Van Nostrand Reinhold, New York, 1991.

[146] Syswerda, G.

A study of reproduction in generational and steady-state genetic algorithms.

Foundations of genetic algorithms, pág. 94–101. Morgan Kaufmann Publishers, San Mateo, 1991.

[147] Tang, A. Y. C. y Leung, K. S.

A modified edge recombination operator for the travelling salesman problem.

PPSN III: Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature, pág. 180–188, London, UK, 1994. Springer-Verlag.

[148] Torn, A. y Zilinskas, A..

Global Optimization.

Springer, Berlin, 1989.

[149] Trosset, M. W.

I know it when i see it: toward a definition of direct search methods.

SIAG/OPT Views-and-News, (9):7–10, 1997.

[150] Tuya, J., Cabal, M. J. S. y de la Riva, C.

Mutating database queries.

Information and Software Technology, 49(4):398–417, 2007.

[151] Ulder, N. L. J., Aarts, E. H. L., Bandelt, H. J., van Laarhoven, P. J. M. y Pesch, E.

Genetic local search algorithms for the travelling salesman problem.

PPSN I: Proceedings of the 1st Workshop on Parallel Problem Solving from Nature, pág. 109–116, London, UK, 1991. Springer-Verlag.

[152] Untch, R. H., Offutt, A. J. y Harrold, M. J.

Mutation analysis using mutant schemata.

International Symposium on Software Testing and Analysis, pág. 139–148, 1993.

[153] Van Iwaarden, R. J.

An improved unconstrained global optimization algorithm.

PhD thesis, University of Colorado, Denver, Estados Unidos, 1996.

[154] Viswanadha, S. y Sankar, S.

JavaCC.

https://javacc.dev.java.net/, Sept. 2008.

[155] Voudouris, C. y Tsang, E.

Guided local search.

Technical Report CSM-247, Department of Computer Science, University of Essex, 1995.

[156] Wall, M.

GAlib: A C++ Library of Genetic Algorithm Components.

Mechanical Engineering Department, Massachusetts Institute of Technology, 1996.

http://lancet.mit.edu/ga/.

[157] Whitley, D. y Kauth, J.

Genitor: a different genetic algorithm.

Proceedings of the Rocky Mountain Conference on Artificial Intelligence, pág. 118–130, 1988.

[158] Whitley, D., Starkweather, T. y Shaner, D.

Handbook of Genetic Algorithms, chapter The traveling salesman and sequence scheduling: Quality solutions using genetic edge recombination, pág. 350–372.

Van Nostrand Reinhold, New York, 1991.

[159] Wilf, H. S.

Algorithms and complexity.

Prentice-Hall, 1986.

[160] Wright, A. H.

Genetic algorithms for real parameter optimization.

Foundations of genetic algorithms, pág. 205–218. Morgan Kaufmann Publishers, San Mateo, 1991.

[161] Wright, M. H.

Direct search methods: once scorned, now respectable.

Numerical analysis, pág. 191–208. Addison Wesley Longman, 1996.

[162] Xanthakis, S., Ellis, C., Skourlas, C., Gall, A. L., Katsikas, S. y Karapoulios, K.

Application of genetic algorithms to software testing (application des algorithmes génétiuqes au test des logiciels).

Proceedings of the 5th International Conference on Software Engineering, pág. 625–636. Toulouse, Francia, 1992.

[163] Yen, J. y Lee, B.

A simplex genetic algorithm hybrid.

IEEE International Conference on Evolutionary Computation, pág. 175–180, 1997.

[164] Yuan, Y.

Nonlinear optimization: trust region algorithms.

Proceedings of the Third Chinese SIAM Conference, pág. 84–102. Tsinghua University Press, Beijing, 1994.

[165] Yuan, Y.

Numerical linear algebra and optimization, chapter 9: Problems on convergence of unconstrained optimization algorithms, pág. 95–107.

Science Press, Nueva York, 1999.

[166] Yuan, Y.

A review of trust region algorithms for optimization.

Report ICM–99–038, Institute of Computational Mathematics and Scientific/Engineering Computing, Chinese Academy of Sciences, Beijing, China, 1999.

[167] Zhu, H., Hall, P. y May, J.

Software unit test coverage and adequacy.

ACM Computing Surveys, 29(4):366–427, December 1997.