Metodología de optimización de los parámetros de control de un algoritmo genético

Jorge Alberto Jaramillo Garzón

Universidad Nacional de Colombia
Facultad de Ingeniería y Arquitectura
Departamento de Ingeniería Eléctrica, Electrónica y Computación
Manizales
2007

Metodología de optimización de los parámetros de control de un algoritmo genético

Jorge Alberto Jaramillo Garzón

Trabajo de grado para optar al título de
Magíster en Ingeniería — Automatización Industrial

Director
Prof. César Germán Castellanos Domínguez

Universidad Nacional de Colombia
Facultad de Ingeniería y Arquitectura
Departamento de Ingeniería Eléctrica, Electrónica y Computación
Manizales
2007

Optimization Methodology for Control Parameters of Genetic Algorithms

Jorge Alberto Jaramillo Garzón

Thesis for the degree of
Master in Engineering — Industrial Automation

Supervisor
Prof. César Germán Castellanos Domínguez

National University of Colombia
Faculty of Engineering and Architecture
Department of Electrical, Electronic and Computing Engineering
Manizales
2007

Este trabajo se realiza en el marco del proyecto "*DETECCIÓN DE LOS NIVELES DE COMPROMISO DE RESONANCIA EN NIÑOS CON LABIO Y/O PALADAR HENDIDO*", financiado por la Dirección de Investigaciones DIMA, de la Universidad Nacional de Colombia Sede Manizales

Índice

Índice I

Lista de tablas IV

Lista de figuras V

I Preliminares 1

Resumen 2

Abstract 3

Introducción 4

Objetivos 6

II Marco teórico 7

1. Algoritmos genéticos 8
- 1.1. El algoritmo genético simple . . . 8
- 1.2. Fundamentos teóricos de los algoritmos genéticos . . . 12
  - 1.2.1. Teoría de esquemas . . . 12
  - 1.2.2. Caminos reales (*royal roads*) . . . 13
- 1.3. Mejoramientos al modelo básico . . . 15
  - 1.3.1. Métodos de escalamiento de la calidad . . . 15
  - 1.3.2. Métodos de ordenamiento (*Ranking*) . . . 16
  - 1.3.3. Nichos y especies . . . 17
- 1.4. Sintonización de los parámetros de control . . . 18

2. Optimización con restricciones utilizando algoritmos genéticos 21
- 2.1. Funciones de penalización . . . 22
  - 2.1.1. Penalizaciones estáticas . . . 22
  - 2.1.2. Penalizaciones dinámicas . . . 23
  - 2.1.3. Rechazo de las soluciones no viables (*Death penalty*) . . . 25

2.2. Separación de las restricciones y las funciones objetivo . . . 25
2.2.1. Co-evolución . . . 25
2.2.2. Superioridad de los puntos viables . . . 26

3. Superficies de regresión por vectores de soporte 28
3.1. Máquinas de vectores de soporte . . . 28
3.1.1. Caso linealmente separable . . . 29
3.1.2. Caso linealmente no separable . . . 30
3.1.3. Máquinas no lineales de vectores de soporte . . . 32
3.2. Superficies de regresión por vectores de soporte . . . 33
3.2.1. Regresión εSV . . . 33
3.2.2. Selección del modelo de una SVR . . . 35

III Marco experimental 37

4. Metodología propuesta 38
4.1. Función de evaluación para medir el desempeño obtenido con cada conjunto de parámetros . . . 39
4.2. Selección del modelo para la SVR . . . 41
4.2.1. Ancho de la zona ε-insensible . . . 41
4.2.2. Coeficiente de penalización de los errores . . . 42
4.2.3. Selección y sintonización del kernel . . . 43
4.3. Algoritmo genético de alto nivel . . . 43
4.3.1. Algoritmo de identificación dinámica de nichos . . . 44
4.3.2. Asignación de la calidad a las muestras según los nichos identificados 47
4.3.3. Esquema de apareamiento restringido . . . 48

5. Experimentos y resultados 49
5.1. Funciones de evaluación comparativa . . . 49
5.2. Parámetros de control a sintonizar . . . 50
5.3. Construcción de las superficies de regresión y ejecución del algoritmo genético de alto nivel . . . 53
5.4. Metodologías de referencia . . . 53
5.4.1. Parámetros recomendados por la literatura . . . 54
5.4.2. Meta-algoritmo genético con redes neuronales . . . 54
5.5. Resultados . . . 55

6. Discusión y conclusiones 61

IV Apéndices 63

A. Método de búsqueda Fibonacci 64

B. Algoritmo de entrenamiento SVM^{light} 66

Bibliografía **68**

Lista de tablas

5.1. Funciones de evaluación comparativa 50
5.2. Número de generaciones obtenido con cada metodología 55
5.3. Porcentaje de error respecto al máximo verdadero 55
5.4. Resultados obtenidos para $\alpha = 0.95$ 56

Lista de figuras

1.1. Proceso de selección y recombinación en una generación del GA 10
1.2. Recombinación de un solo punto 11
1.3. Cadena óptima, dividida en ocho bloques constructores (esquemas) 14

3.1. Ajuste de margen para una SVM lineal 29
3.2. Ajuste de *margen suave* para una SVM lineal 31
3.3. Función de pérdida en la regresión εSV 34

4.1. Distribución de las muestras en una función multimodal con dos óptimos 45
4.2. Secuencia de las muestras, basada en su distancia al centroide 45
4.3. Secuencia de las muestras, basada en distancias encadenadas 46
4.4. Corrección de la interferencia entre nichos 47

5.1. Funciones de evaluación comparativa f1 a f4 51
5.2. Funciones de evaluación comparativa f5 a f8 52
5.3. Modelos obtenidos con las SVR para f1 y f2 57
5.4. Modelos obtenidos con las SVR para f3 y f4 58
5.5. Modelos obtenidos con las SVR para f5 y f6 59
5.6. Modelos obtenidos con las SVR para f7 y f8 60

Parte I

Preliminares

Resumen

En este trabajo se presenta una metodología para la sintonización simultánea de los parámetros de control de un algoritmo genético, enfocada a la reducción del número de generaciones empleadas por el algoritmo para encontrar un óptimo.
La metodología incluye un meta-algoritmo genético en combinación con una superficie de regresión por vectores de soporte que reduce la demanda computacional inherente al método del meta-algoritmo.
Cada elemento que compone la metodología fue escogido de forma que cuente con sustento suficiente para la sintonización de sus propios parámetros de control y se cumpla con el propósito principal de este trabajo que es la automatización del proceso de selección de los parámetros de control, sin generar parámetros adicionales.
El algoritmo genético de alto nivel incorpora técnicas de generación de nichos para aumentar la exploración del espacio de búsqueda. El método utilizado para la definición de los nichos es uno de los principales aportes de este trabajo, así como una novedosa técnica de apareamiento restringido.
La optimización de los parámetros de control del algoritmo genético de bajo nivel, es planteada como un problema de optimización sujeto a restricciones, para lo cual se plantea una nueva forma de penalización estática que no requiere el ajuste de coeficientes de penalización.
Las pruebas y resultados presentados incluyen la sintonización de los tres parámetros de control fundamentales de un algoritmo genético: la probabilidad de mutación, la probabilidad de cruce y el tamaño de la población inicial. Además, se sintoniza un parámetro adicional como un ejemplo de los parámetros resultantes al incluir otros operadores especiales, en este caso, un método de escalamiento de calidad.
Cada prueba fue desarrollada sobre un conjunto de ocho funciones de evaluación comparativa (*benchmarking*) que incluye casos continuos, discontinuos, convexos, no convexos, unimodales, multimodales, cuadráticos, no cuadráticos, de baja y de alta dimensionalidad, determinísticos y no determinísticos.

Abstract

A methodology for simultaneous tuning of the control parameters of genetic algorithms is presented. This methodology is focused on reducing the number of generations the algorithm takes to find an optimal solution.
The inherent computational costs due to the meta-algorithm are reduced by incorporating a support vector regression that models the interactions among genetic algorithm parameters.
Each element in the methodology was selected in such a way that its theoretical background provides the sufficient basis for tuning its own parameters and thus, we can achieve the automatic tuning of control parameters in genetic algorithms, without adding new parameters, which is the main purpose of this work.
The high level genetic algorithm was designed encompassing niche and speciation techniques, in order to increase exploration in the search space. The method for locating the niches and a novel restricted mating procedure are two of the mayor contributions of this work.
Parameter optimization of low level genetic algorithm is viewed as a restricted optimization problem, which is solved by a novel static penalization method with no penalty coefficients.
The results include tuning of the three main parameters: mutation probability, recombination probability and size of the initial population. In addition, an extra parameter for fitness scaling is included, as an example of parameters resulting from the application of special operators.
Each test was developed over a set of eight benchmark functions, including continuous, discontinuous, convex, non-convex, unimodal, multimodal, quadratic, non-quadratic, low dimensional, high dimensional, deterministic and nondeterministic functions.

Introducción

Los algoritmos genéticos (GA) son una familia de modelos computacionales inspirados en la evolución. Se basan en la codificación de una solución potencial a un problema específico, en una estructura de datos simple y la aplicación de diferentes operadores de recombinación a estas estructuras para preservar la información crítica. Estos algoritmos han mostrado una gran robustez en diversas tareas de optimización y aprendizaje de máquina gracias a su capacidad de búsqueda en pseudo-paralelo, lo que les permite trabajar en espacios de búsqueda altamente no lineales y de gran extensión.

Aunque los algoritmos genéticos son un método robusto para resolver un amplio rango de problemas, se requiere la sintonización de sus parámetros de control para obtener máxima eficiencia [17]. La optimización de estos parámetros es una tarea que consume gran cantidad de tiempo y que demanda a menudo la intervención subjetiva de un experto [52], por lo cual la mayoría de investigadores opta por utilizar conjuntos de parámetros reportados en trabajos anteriores, que pueden no funcionar bien en otros problemas [46, 73].

Un método de auto-sintonización, denominado meta-algoritmo, ha demostrado ser una forma acertada de resolver este problema; ésta consiste en correr un GA de alto nivel que busca un conjunto de parámetros óptimos para un GA de bajo nivel. El GA de alto nivel, o meta-GA, corre sobre una población cuyos individuos son codificaciones de los parámetros de control, mientras que el GA de bajo nivel es un GA regular que usa los parámetros encontrados por el GA de alto nivel. No obstante, la ejecución del meta-algoritmo sigue siendo computacionalmente costosa y no entrega un modelo de la influencia de los parámetros de control sobre el desempeño del algoritmo genético [52].

Para simplificar la ejecución del meta-algoritmo, éste puede combinarse con otras herramientas computacionales, como se hizo en [65], donde se introdujo una red neuronal que se entrenaba para predecir el desempeño del GA. Aunque esta red neuronal permite reducir el tiempo de cómputo empleado por el algoritmo de alto nivel, no presenta una sustentación acerca de la función utilizada para medir el desempeño del GA de bajo nivel, ni el criterio usado para escoger el modelo de la red neuronal (cantidad de neuronas, número de capas, etc.). Además, no permite explicar la interacción entre los parámetros, debido a la misma naturaleza de las redes neuronales.

Los métodos de regresión por vectores de soporte (SVR) son una alternativa robusta para el cálculo de un modelo de los parámetros de control del algoritmo genético y cuentan con fundamentos teóricos que permiten un análisis detallado. Por tanto, se propone una metodología de optimización de los parámetros de control del algoritmo genético, usando para ello un meta-algoritmo, en combinación con superficies de regresión por vectores de soporte para reemplazar el algoritmo de bajo nivel y así reducir el tiempo de cómputo

necesario para la determinación del conjunto óptimo de parámetros. Esto también permite modelar las diferentes relaciones entre los parámetros de control, para además contar con un modelo de la influencia de éstos sobre el desempeño del algoritmo genético.
Este documento se encuentra dividido en tres partes. La Parte I contiene los resúmenes en español e inglés del documento, la presente introducción y los objetivos planteados en la propuesta de tesis. La Parte II está conformada por tres capítulos que abarcan los fundamentos teóricos del trabajo desarrollado. El primero, describe los principios básicos que rigen los algoritmos genéticos, su fundamentación teórica y algunas de las variaciones que se han planteado sobre el modelo básico; el segundo capítulo contiene una revisión de algunas de las principales técnicas para optimización sujeta a restricciones, utilizadas con algoritmos evolutivos; el tercer capítulo presenta los fundamentos sobre las máquinas de vectores de soporte y su expansión hacia la regresión por vectores de soporte. La Parte III contiene tres capítulos donde se presenta el desarrollo del trabajo. El primer capítulo describe la metodología propuesta para la optimización de los parámetros de control de un algoritmo genético; el segundo capítulo describe los experimentos realizados para probar dicha metodología y presenta los resultados obtenidos; el tercer capítulo contiene las conclusiones y trabajo futuro. Por último, se incluyen dos apéndices acerca del algoritmo de entrenamiento SVM^{light}, sobre el cual está fundamentado el *toolbox* con el que se corrieron las pruebas, y el método de optimización de Fibonacci, el cual constituye una parte fundamental en la selección del modelo de la regresión por vectores de soporte tal como se propone en este trabajo.

Objetivos

Objetivo general

Desarrollar una metodología para la optimización de los parámetros de control del algoritmo genético, orientada a la minimización del tiempo de convergencia del algoritmo genético, usando un meta-algoritmo en combinación con una superficie de regresión con vectores de soporte.

Objetivos específicos

- Implementar y analizar un meta-algoritmo que optimice los 3 parámetros de control fundamentales de un GA: La probabilidad de mutación (μ); la probabilidad de cruce (χ) y el tamaño de la población inicial (r).
- Reducir la demanda computacional del meta-algoritmo, incorporando una superficie de regresión por vectores de soporte que modele las interacciones de los parámetros a optimizar, reemplazando al GA de bajo nivel al momento de buscar la convergencia del GA de alto nivel.
- Encontrar los parámetros óptimos del GA en la solución de las funciones de evaluación comparativa (benchmarking) propuestas para algoritmos genéticos y comparar el desempeño conseguido con el obtenido usando los valores clásicos propuestos en la literatura.

Parte II

Marco teórico

Capítulo 1

Algoritmos genéticos

Los Algoritmos Genéticos (GA) son métodos computacionales basados en los procesos de selección y evolución natural, que han sido satisfactoriamente aplicados sobre una gran cantidad de problemas en diferentes áreas de investigación (bibliografías indexadas de implementaciones y artículos sobre GA, pueden encontrarse en [2] y [1] respectivamente).
Su utilización puede ser dividida en dos enfoques principales: el desarrollo de modelos computacionales que ayuden en el estudio de sistemas biológicos [19, 33, 31, 45], y como herramientas para la resolución de problemas de optimización [15, 44, 4, 22]. Este último es el punto de vista bajo el cual serán entendidos en este documento.
En este capítulo se exponen los principios básicos que rigen los GA y algunas de las mejoras que se han aplicado sobre el modelo inicial, para convertirlo en una de las herramientas de búsqueda y optimización computacional más robustas que existen en el momento.

1.1. El algoritmo genético simple

Los algoritmos genéticos fueron desarrollados por John Holland y sus estudiantes y colegas de la Universidad de Michigan en los 60's y 70's. El algoritmo genético de Holland, llamado algunas veces *Algoritmo Genético Simple (SGA)*, es un método para moverse de una población de *cromosomas* a una nueva población, usando una especie de *selección natural* y los operadores de *cruce* y *mutación*, inspirados en la genética.
La ejecución de un SGA para resolver un problema de optimización, se puede sintetizar en las siguientes etapas:

Definición de la función de evaluación: La codificación y la función de evaluación son los dos únicos componentes que dependen de la naturaleza del problema a resolver. En optimización, la función de evaluación corresponde simplemente a la función que se quiere optimizar. De ahora en adelante, se considerará únicamente el problema de maximización, teniendo en cuenta que un problema de minimización puede ser expresado como uno de maximización, de la forma:

$$\min_{\lambda_i} g(\lambda_1, \lambda_2, ..., \lambda_M) = \max_{\lambda_i}\{1 - g(\lambda_1, \lambda_2, ..., \lambda_M)\},\ i = 1, 2, ..., M \tag{1.1}$$

siendo $g(\lambda_1, ..., \lambda_M) \in [0, 1]$ la función a optimizar normalizada, y $\lambda_1, ..., \lambda_M$ los parámetros sobre los cuales se quiere hacer la optimización, que pueden ser expresados como un vector $\boldsymbol{\lambda} = [\lambda_1, ..., \lambda_M]$ de $1 \times M$.

Codificación de los parámetros de entrada y generación de la población inicial: En segundo lugar se encuentra la tarea de cómo codificar el vector de parámetros de entrada $\boldsymbol{\lambda} = [\lambda_1, ..., \lambda_M]$, el cual representa las diferentes combinaciones de parámetros que pueden existir y cada una de las cuales es una solución potencial al problema de optimización. Cada posible solución debe ser codificada en una cadena de longitud finita, denominada *cromosoma*, conformada por la concatenación de las codificaciones de todos los parámetros λ_i, $i = 1, ..., M$

$$\mathbf{c} = [\mathcal{C}(\lambda_1), \mathcal{C}(\lambda_2), ..., \mathcal{C}(\lambda_M)] \tag{1.2}$$

donde $\mathbf{c}$ será el cromosoma resultante y $\mathcal{C}$ representa el operador de codificación.
Estas cadenas están divididas en unidades denominadas *genes*, cada una de las cuales puede tomar diferentes valores llamados *alelos*. Holland [30] demostró a través de la teoría de esquemas (ver sección 1.2.1) que es preferible tener cromosomas de gran longitud con pocos posibles alelos por cada gen, lo que conlleva a que el mejor caso son las cadenas de genes binarios. El operador $\mathcal{C}$ queda entonces definido como una conversión a sistema binario

$$\mathcal{C} : \mathbb{R} \rightarrow \{0, 1\}^{l_i},\ i = 1, 2, ..., M \tag{1.3}$$

donde l_i es la longitud de la palabra usada para representar al parámetro λ_i y depende del planteamiento del problema.
Cuando se ha definido la forma de codificación, es necesario generar una *población inicial* P_0 de r cromosomas

$$P_0 = \{\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_r\} \tag{1.4}$$

En el SGA, esta población es generada de una forma completamente aleatoria. El parámetro r, denominado *longitud de la población inicial*, es el primer parámetro de control que es necesario sintonizar para el buen desempeño del GA.

Selección de los mejores individuos: Con el fin de emular el proceso de selección natural de los sujetos más aptos, es necesario asignar a cada individuo un valor de *calidad* (*fitness*), a partir del cual se hará dicha selección. Cuando los individuos más aptos han sido seleccionados, se forma la llamada *población intermedia*, que es un paso medio entre la población actual y la siguiente. Este proceso se ilustra en la Figura 1.1 Aunque algunas veces se utilizan los términos *evaluación* y *calidad* sin ninguna distinción, es útil distinguir los dos términos al trabajar con algoritmos genéticos. La función de evaluación provee información sobre el desempeño de un individuo (cromosoma) en la solución del problema planteado. La calidad transforma esta evaluación en una asignación de oportunidades de reproducción. Aún más, la evaluación corresponde a un individuo y es calculada independientemente del resto de la población, mientras la asignación de calidad depende de las evaluaciones de

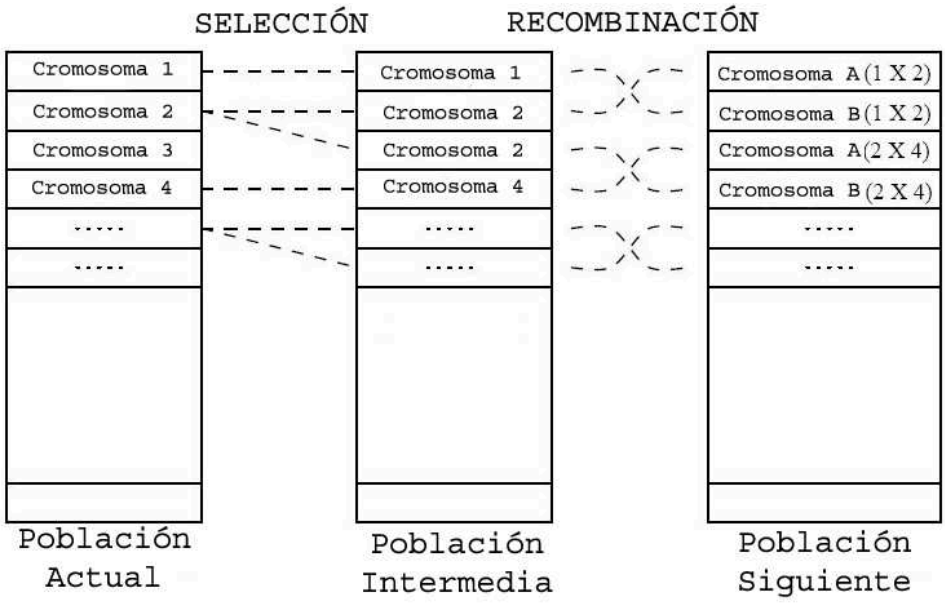


Figura 1.1: Proceso de selección y recombinación en una generación del GA

toda la población [73].
En el SGA, la calidad está definida por:

$$f(\mathbf{c}_i) = \frac{g(\mathbf{c}_i)}{\overline{g}},\ i = 1, 2, ..., r \tag{1.5}$$

donde $g(\mathbf{c}_i)$ es la evaluación del *i-ésimo* individuo y $\overline{g}$ es el promedio de las evaluaciones de toda la población:

$$\overline{g} = \frac{1}{r}\sum_{i=1}^{r} g(\mathbf{c}_i) \tag{1.6}$$

Una vez se ha asignado un valor de calidad a cada uno de los individuos de la población, se escogen los más aptos a través de un muestreo estocástico, lo que le da oportunidad de reproducirse a todos los individuos, en proporción a su calidad. El primer método planteado es el de *Muestreo Estocástico con reemplazo*, que puede imaginarse como si se tuviera a todos los individuos en una ruleta, cada uno representado por un espacio proporcional al valor de su calidad. Girando repetitivamente la ruleta, se van escogiendo individuos hasta completar la población intermedia.
Otro método que puede concordar más con lo esperado a partir de los valores de calidad, es el de *Muestreo Estocástico sobre el restante* (*Remainder Stochastic Sampling*). Para cada cadena $\mathbf{c}_i$, $i = 1, 2, ..., r$, cuando $g(\mathbf{c}_i)$ sea mayor a 1.0, la parte entera de este número indica cuántas copias de esa cadena pasarán directamente a la población intermedia. Luego, todas las cadenas, incluyendo las que ya fueron copiadas directamente, tendrán una probabilidad de ser copiadas a la población intermedia, correspondiente a la parte decimal de su valor de calidad. El *Muestreo Estocástico sobre el restante* es implementado más eficientemente si, para la parte

de la selección no directa de copias, se utiliza un método conocido como *Muestreo Estocástico Universal* (*Universal Stochastic Sampling*). Supóngase que se tienen todos los individuos ubicados al azar en una gráfica tipo *torta*, cada uno con un espacio proporcional a la parte decimal de su valor de calidad. Ahora, se superpone una ruleta de k lugares de igual tamaño, siendo k la cantidad de lugares disponibles en la población intermedia después de pasar las copias directas; entonces, con un solo giro de la ruleta, se escogen los k individuos que hacen falta.

Recombinación y mutación: La recombinación es el proceso por medio del cual se crea la población siguiente a partir de la población intermedia. La recombinación es aplicada a cromosomas apareados al azar con una probabilidad denotada por χ (se considera que la población ya ha sido suficientemente mezclada con el proceso de selección al azar). Lo que se hace es tomar un par de cromosomas y con dicha probabilidad χ decidir si los cromosomas se cruzarán para formar nuevos individuos o pasarán tal cual a la siguiente generación.
El SGA utiliza la recombinación de un solo punto, que consiste en partir las dos cadenas por un punto escogido al azar e intercambiar los trozos de ambas, como se muestra en la Figura 1.2

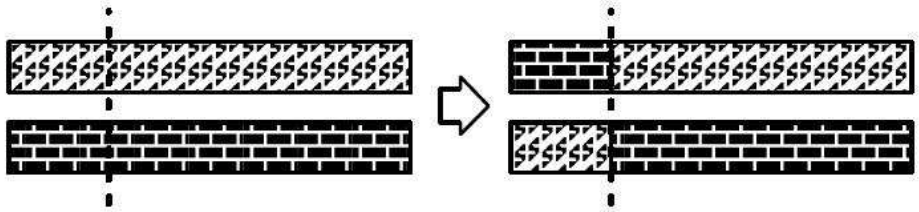


Figura 1.2: Recombinación de un solo punto

Después de la recombinación, se aplica el operador de mutación. Para cada bit de la población, se escoge si se aplica mutación o no, con una probabilidad μ. La mutación consiste en la *negación* del bit, es decir, se cambia la condición del bit de cero a uno o de uno a cero según sea el caso.
Las probabilidades de recombinación (χ) y mutación (μ), constituyen otros dos parámetros de control fundamentales para el buen desempeño del GA.

Las etapas de selección, recombinación y mutación, son repetidas iterativamente hasta que la población haya convergido, es decir, hasta que el 95 % de los individuos compartan los mismos genes [15].

1.2. Fundamentos teóricos de los algoritmos genéticos

A pesar de que la descripción e implementación los GA es simple, los fundamentos teóricos acerca de por qué realmente funcionan, es aún un campo abierto. En esta sección se presentan dos aproximaciones que intentan describir el funcionamiento macroscópico de los GA: la *teoría de esquemas* utilizada por John Holland para describirlos originalmente [30] y los *caminos reales (Royal Roads)*, desarrollados más recientemente por Melanie Mitchell, Stephanie Forrest y John Holland como una alternativa para investigar más a fondo el funcionamiento de los GA [46].

1.2.1. Teoría de esquemas

Esta es la teoría tradicional de los GA, donde se asume su funcionamiento a través del descubrimiento y recombinación de "bloques constructores" de soluciones, en una forma de proceso paralelo. La noción de "bloques constructores" puede ser formalizada por medio de la introducción del concepto de *esquemas*, que pueden ser vistos como plantillas de unos, ceros y posiciones de "no importa" denotadas por asteriscos. Por ejemplo, el esquema $H = 1 * * * * 1$ representa el conjunto de todas las cadenas de seis bits de longitud, que empiezan y terminan en 1. Las cadenas que cumplen con la plantilla H, son llamadas *ejemplos* de H. El número de bits que están dados en un esquema (que no son asteriscos) determina el *orden* del esquema y la distancia entre sus bits definidos más externos, es denominada *distancia definitiva*.
Una cadena dada de longitud l, será entonces un ejemplo de 2^l esquemas diferentes. Por lo tanto, cuando un GA procesa r cadenas de una población, está procesando un número de esquemas entre 2^l (si toda la población es idéntica) y $2^l r$. Esto significa que, en una generación dada, mientras el GA evalúa explícitamente la calidad de r cadenas, está estimando implícitamente la calidad promedio de un número de esquemas mucho mayor, siendo la calidad promedio de un esquema, el promedio de la calidad de todos los ejemplos que cumplen con ese esquema. Esta idea es denominada "paralelismo implícito".
Desde este punto de vista, la dinámica aproximada de incremento y decremento de los ejemplos de un esquema determinado, puede calcularse como se explica a continuación. Supóngase que H es un esquema con al menos un ejemplo presente en la población P_t de la generación t. Sea $m(H,t)$ el número de ejemplos del esquema H, en la generación t y sea $\hat{u}(H,t)$ la calidad promedio observada de H en la generación t (es decir, el promedio de la calidad de los ejemplos de H presentes en P_t).
Se busca calcular el valor esperado de ejemplos de H, $E(m(H,t+1))$ en la generación $t+1$. Utilizando las ecuaciones (1.5) y (1.6) definidas en la sección 1.1, el número esperado de copias de todos los ejemplos $\{\mathbf{c}|\mathbf{c} \in H\}$ después de pasar por el proceso de selección, será:

$$\begin{aligned} E(m(H,t+1)) &= \sum_{\mathbf{c}\in H} \frac{f(\mathbf{c})}{\bar{f}} \\ &= \frac{\hat{u}(H,t)}{\bar{f}} m(H,t) \end{aligned} \tag{1.7}$$

donde se puede observar que el incremento o decremento de los ejemplos de determinado esquema H en la generación t, depende de $\hat{u}(H,t)$ aunque este valor no sea calculado explícitamente.

La recombinación y la mutación pueden destruir o crear ejemplos de H. A continuación se considerarán los efectos destructivos de ambas. Sea χ la probabilidad de recombinación y supóngase que un ejemplo de H es escogido para ser uno de los padres. Se dice que el esquema H sobrevivió a la recombinación, si al menos una de las 2 cadenas resultantes pertenece a H. Una cota superior de la probabilidad $S_c(H)$ de que H sobreviva a la recombinación de un solo punto, estará dada por:

$$S_c(H) \geq 1 - \chi \frac{d(H)}{l-1} \tag{1.8}$$

donde $d(H)$ es la distancia definitiva de H y l es la longitud de las cadenas en el espacio de búsqueda. Esto significa que las recombinaciones que ocurran dentro de la distancia definitiva de H, pueden destruir el esquema. Esta probabilidad, multiplicada por la probabilidad de recombinación χ, entrega la probabilidad de destrucción, por lo tanto, al restarla a uno, se tiene la probabilidad de supervivencia. Observando la ecuación (1.8), se puede deducir entonces que la probabilidad de supervivencia es mayor para los esquemas de menor longitud.
A continuación, sea μ la probabilidad de que algún bit sea mutado. La probabilidad $S_m(H)$ de que el esquema H sobreviva después de la etapa de mutación, será:

$$S_m(H) = (1-\mu)^{o(H)} \tag{1.9}$$

donde $o(H)$ es el orden del esquema. Esto es, para cada bit definido del esquema, la probabilidad de que no ocurra mutación es $1-\mu$, así que la probabilidad de que ninguno de los bits definidos del esquema sea mutado, será esta cantidad multiplicada $o(H)$ veces. Examinando la ecuación (1.9), se puede deducir que los esquemas de menor orden tendrán más probabilidad de sobrevivir.
Estos efectos destructivos, puesden ser usados para completar la ecuación (1.7) de la forma:

$$E(m(H,t+1)) \geq \frac{\hat{u}(H,t)}{\bar{f}} m(H,t) \left(1 - \chi \frac{d(H)}{l-1}\right) (1-\mu)^{o(H)} \tag{1.10}$$

Esta ecuación es conocida como el *Teorema de Esquemas* [30] y describe el crecimiento de un esquema dado, de una generación a otra. Este teorema es un límite inferior, dado que sólo considera los efectos destructivos de la recombinación y la mutación, más no sus efectos constructivos. Comúnmente es interpretado como una implicación de que los esquemas cortos, de orden bajo, cuya calidad promedio se encuentre por encima de la media, recibirán un número exponencialmente creciente de ejemplos a través de la ejecución del GA, mientras el número de ejemplos de aquellos esquemas que no sean destruidos y permanezcan sobre el promedio de calidad, será incrementado por un factor de $\hat{u}(H,t)/\bar{f}$ en cada generación. Algunas críticas a esta interpretación, han surgido durante años, pero aún no hay un consenso definitivo sobre su validez o no [46, 73].

1.2.2. Caminos reales (*royal roads*)

El teorema de esquemas tiene en cuenta los efectos destructivos de la recombinación y la mutación, pero no sus efectos constructivos, aunque para muchos esta es una de

las principales fuentes del poder de búsqueda de los GA. Los caminos reales, son un conjunto de funciones de calidad desarrolladas por Melanie Mitchell, Stephanie Forrest y John Holland para tratar de capturar la esencia de los bloques constructores, de una forma idealizada [47, 20, 48]. Como ejemplo, la función R_1, mostrada en la figura 1.3, se encuentra definida como:

$$R_1(x) = \sum_i \delta_i(x),\ donde\ \delta(x) = \begin{cases} 1\ si\ x \in s_i \\ 0\ otro\ caso \end{cases} \tag{1.11}$$

La calidad de la cadena x será entonces la sumatoria de los coeficientes c_i asignados a todos los esquemas s_i a los cuales pertenece (ver figura 1.3). Por ejemplo, si x pertenece a 2 de los esquemas, su calidad será 16, mientras para la cadena óptima mostrada en 1.3, $R_1(s_{opt}) = 64$.

s_1 = 11111111**********; $c_1 = 8$
s_2 = ********11111111**********; $c_2 = 8$
s_3 = ****************11111111**********; $c_3 = 8$
s_4 = ************************11111111**********; $c_4 = 8$
s_5 = ********************************11111111**********; $c_5 = 8$
s_6 = **11111111**********; $c_6 = 8$
s_7 = **11111111********; $c_7 = 8$
s_8 = **11111111; $c_8 = 8$
s_{opt}=11111111111111111111111111111111111111111111111111111111111111111111

Figura 1.3: Cadena óptima, dividida en ocho bloques constructores (esquemas)

La comparación del desempeño de los GA en la solución de estas funciones, contra otros métodos de optimización bien conocidos, permitió plantear lo que sus autores llaman el *Algoritmo genético ideal IGA* de la siguiente forma:

- En cada iteración, escójase una nueva cadena aleatoriamente, con probabilidad uniforme por cada bit.
- La primera vez que sea encontrada una cadena que contenga uno o más esquemas deseados, consérvese esa cadena.
- Cuando se encuentre una cadena que contenga uno o más esquemas no descubiertos aún, recombínese la nueva cadena con la almacenada de forma que la cadena resultante conserve todos los esquemas que han sido descubiertos hasta el momento.

Nótese que el IGA no maneja una población, sino que trabaja en una sola cadena cada vez. La selección completamente independiente de las cadenas, permite muestrear independientemente los esquemas. Además, la forma de recombinación que utiliza, implementa perfectamente la idea del paralelismo implícito.
Este GA es, por supuesto, no implementable en un problema práctico dado que en la realidad no se tiene información sobre cuáles son los esquemas deseados o no deseados, pero su análisis puede dar ideas sobre cómo hacer más eficiente un GA. Para hacer un GA lo más parecido posible al IGA, se pueden tomar en cuenta las siguientes ideas:

Muestras Independientes: La población debe ser lo suficientemente grande, el proceso de selección lo suficientemente lento y la tasa de mutación lo suficientemente alta para asegurar que ningún gen particular tenga el mismo valor en toda la población o incluso en la gran mayoría.

Almacenamiento de los esquemas deseados: La selección debe ser lo suficientemente fuerte para preservar los esquemas deseados que han sido descubiertos.

Recombinación instantánea: La tasa de recombinación debe ser de tal manera que el tiempo en que se recombinan los esquemas deseados sea menor al tiempo con que se descubren nuevos esquemas.

Estos mecanismos no son todos mutuamente compatibles (por ejemplo la alta mutación no permite el almacenamiento de los esquemas deseados) y por lo tanto deben ser cuidadosamente balanceados.

1.3. Mejoramientos al modelo básico

Además de los operadores clásicos de selección, recombinación y mutación descritos en el SGA, existen otra serie de operadores inspirados en fenómenos naturales, que buscan aumentar la robustez del SGA. A continuación serán descritos algunos de esos operadores, que serán utilizados en el presente trabajo.

1.3.1. Métodos de escalamiento de la calidad

Estos métodos son utilizados para evitar dos problemas comunes en los GA. El primero es la *convergencia prematura*, que consiste en que en las primeras generaciones del algoritmo, algunos individuos con una calidad muy alta en comparación con el resto de la población, pueden dominar el proceso de selección e impedir el proceso de exploración que debe realizar el GA. El segundo, es el *terminado lento*, que se presenta en las generaciones donde la población tiene una alta convergencia y por lo tanto todos los individuos tienen valores de calidad similares. En este caso, la simulación tiende a moverse sin que predomine ninguno de los individuos, y no converge a alguno de ellos.
En el primer caso, la función objetivo debe escalarse para evitar grandes diferencias entre la calidad de los individuos, mientras en el segundo caso se debe buscar una acentuación de estas diferencias para promover la convergencia [22].

Algunos de los métodos de escalamiento más usados son:

Escalamiento lineal: Como su nombre lo indica, la función de calidad es obtenida a partir de la función objetivo, usando una ecuación lineal de la forma:

$$f(\mathbf{c}_i) = ag(\mathbf{c}_i) + b,\ i = 1, 2, ..., r \tag{1.12}$$

En esta ecuación, los valores a y b son escogidos para hacer dos cosas: igualar los valores promedio de la función escalada y la función original, y hacer que el máximo valor de calidad asignado sea un múltiplo específico del valor promedio (usualmente es de 2, pero en general puede ser tratado como otro parámetro de control). Estas dos condiciones aseguran que los miembros promedio reciban un valor esperado de copias igual a uno, mientras los mejores reciben el número de copias que se haya asignado.

Escalamiento sigma: En este método de escalamiento, se utiliza la desviación estándar de la población como una medida de la variación en la información contenida en esta. Un ejemplo de escalamiento sigma es el siguiente [46]:

$$f(\mathbf{c}_i) = \begin{cases} 1 + \frac{g(\mathbf{c}_i) - \overline{g}}{2\sigma} & \text{si } \sigma \neq 0 \\ 1 & \text{si } \sigma = 0 \end{cases} \tag{1.13}$$

donde $\overline{g}$ es la media de la evaluación de la población y σ es la desviación estándar de la evaluación. Esta función le da a un individuo que tenga calidad una desviación estándar por encima de la media, 1.5 de descendencia esperada.
Al principio de las iteraciones, cuando la desviación estándar de la calidad es típicamente alta, los individuos mejor calificados no tendrán una calidad que esté muchas desviaciones estándar por encima de la media y por lo tanto no se llevarán la parte más grande de la descendencia. Igualmente, cuando el número de iteraciones sea elevado, la convergencia será típicamente alta y la desviación estándar será pequeña, por lo tanto los individuos mejor calificados sobresaldrán más, permitiendo que la evolución continúe.

Escalamiento exponencial: En este método, la calidad es dada por una potencia específica de la función objetivo, de la forma:

$$f(\mathbf{c}_i) = g(\mathbf{c}_i)^k \tag{1.14}$$

donde el exponente k se convierte en un parámetro de control que debe ser sintonizado.

1.3.2. Métodos de ordenamiento (*Ranking*)

La naturaleza de los procesos de escalamiento condujo a la proposición de procesos de selección no paramétricos en los que los individuos son ordenados de acuerdo a su evaluación y el valor esperado de cada individuo depende de este rango y no del valor de calidad [6]. Eliminando la importancia de las diferencias grandes o pequeñas en la calidad, el algoritmo permite más oportunidades de reproducción a los individuos no muy altamente calificados, disminuyendo el riesgo de convergencia prematura.

Ordenamiento lineal: El método lineal propuesto en [6] consiste en lo siguiente: Todos los individuos son organizados desde 1 hasta r según su valor de calidad, siendo 1 el peor calificado y r el mejor. A continuación, el diseñador escoge el valor esperado de copias $1 \leq s \leq 2$ para el individuo mejor calificado, y calcula el resto aplicando la fórmula:

$$f(\mathbf{c}_i) = 2 - s + \frac{2(i-1)(s-1)}{r-1},\ i = 1, 2, ..., r \tag{1.15}$$

donde i corresponderá al lugar que ocupa el cromosoma $\mathbf{c}_i$, de forma que para el individuo peor calificado reciba un valor esperado de $2 - s$.
Aunque en [6] se recomienda $s = 1.1$, en general este valor es un parámetro de control adicional.

Ordenamiento Exponencial: En este método, después de ordenar los individuos desde 1 hasta r según su valor de calidad, siendo 1 el peor calificado y r el mejor, el valor de calidad es asignado según:

$$f(\mathbf{c}_i) = s^{r-i} \tag{1.16}$$

donde s es un valor cercano a 1 que debe ser ajustado y se convierte en un parámetro de control. En este caso, la presión de selección es proporcional a $1 - s$ por lo tanto $s = 0{,}994$ proporciona dos veces la tasa de convergencia que proporciona $s = 0{,}998$. Con $s = 0{,}968$, la tasa de convergencia es idéntica a la obtenida con ordenamiento lineal y $s = 1{,}8$ [25].

1.3.3. Nichos y especies

Los GA con nichos buscan encontrar el mayor número de puntos óptimos que sea posible, por medio de la conservación de la diversidad en la población. La mayoría de estos modelos están basados en la idea de preservar la diversidad por medio de alteraciones en los operadores genéticos, para evitar la convergencia a un solo óptimo.
El método más común y mejor estudiado es el de *Calidad Compartida (Fitness Sharing)* [23]. Este método añade una penalización a la calidad $f(\mathbf{x}_i)$, $i = 1, 2, ..., r$ de todos los individuos de la población, en la medida que sean similares entre sí de acuerdo a una métrica $\|\mathbf{.}, \mathbf{.}\|$. Para ello, se define una distancia σ_{share} por debajo de la cual los individuos compartirán su calidad, definiendo una nueva calidad compartida f':

$$f'(\mathbf{c}_i) = \frac{f(\mathbf{c}_i)}{\sum_{j=0}^{r} s(\mathbf{c}_i, \mathbf{c}_j)} \tag{1.17}$$

donde $s(\mathbf{c}_i, \mathbf{c}_j)$ es una función que define qué tanto deben compartir su calidad las muestras $\mathbf{x}_i$ y $\mathbf{c}_j$, dada por:

$$s(\mathbf{x}_i, \mathbf{x}_j) = \begin{cases} 1 - \frac{\|\mathbf{c}_i, \mathbf{c}_j\|}{\sigma_{share}} & \text{si } \|\mathbf{c}_i, \mathbf{c}_j\| \leq \sigma_{share} \\ 0 & \text{en otro caso} \end{cases} \tag{1.18}$$

Esta penalización pretende conseguir una distribución uniforme de la población, alrededor de áreas altamente calificadas y evitar la convergencia de toda la población a un solo óptimo. Desafortunadamente, tiene una serie de inconveniente, como se menciona en [69]:

- Se define un radio fijo σ_{share} para todos los individuos en la población. Por lo tanto, todos los óptimos deben estar aproximadamente equidistantes.
- La selección del radio σ_{share} requiere de conocimiento *a priori* acerca del espacio de búsqueda.
- El uso de funciones de escalamiento puede evitar que el algoritmo converja al máximo real, por lo cual se hace necesario un algoritmo de búsqueda local para mejorar el desempeño [13].

En [75] se propone un *Algoritmo Adaptativo de Clustering* para evitar la estimación *a priori* de σ_{share}. En lugar de las ecuaciones (4.21) y (4.22), se proponen las ecuaciones:

$$f'(\mathbf{c}_i) = \frac{f(\mathbf{c}_i)}{s(\mathbf{c}_i)} \tag{1.19}$$

$$s(\mathbf{c}_i) = n_{z_j} - n_{z_j} \left(\frac{\|\mathbf{c}_i, \mathbf{c}_j\|}{2d_{max}} \right)^{\alpha} \tag{1.20}$$

donde α es una constante, $\|\mathbf{c}_i, \mathbf{z}_j\|$ es la distancia entre la muestra $\mathbf{c}_i$ y el centroide $\mathbf{z}_j$ del cluster al cual pertenece $\mathbf{c}_i$, y n_{z_j} es el número de individuos asociados a dicho cluster. Aunque este algoritmo elimina la necesidad de definir un parámetro σ_{share}, introduce dos nuevas variables d_{min} y d_{max} que representan los radios mínimo y máximo permitidos a un cluster. Sin embargo, este trabajo ha sido el origen de muchas otras investigaciones que asocian algoritmos de clustering a algoritmos genéticos con nichos y especies, como es el caso de [69] y [39].

1.4. Sintonización de los parámetros de control

Como se ha visto en las secciones anteriores, existe un número de parámetros que es necesario sintonizar en el GA para su correcto funcionamiento. Estos parámetros son comúnmente denominados *parámetros de control* e incluyen valores como el tamaño de la población inicial (r), la probabilidad de recombinación (χ), la probabilidad de mutación (μ) (ver sección 1.1) y los parámetros resultantes de los métodos de escalamiento y ordenamiento como el exponente k del escalamiento exponencial o el número esperado de copias s para el ordenamiento exponencial, entre otros (ver sección 1.3.2).
Actualmente existen dos tendencias acerca del establecimiento de los parámetros de control. La primera apunta hacia la *sintonización* de los parámetros, es decir, el establecimiento de unos valores estáticos que se mantendrán a lo largo de toda la ejecución del algoritmo; mientras la segunda sugiere realizar un *control* de los parámetros, es decir, tener parámetros dinámicos que cambien a través de la ejecución del algoritmo, ya sea de forma determinista o por medio de una retroalimentación que entregue información

del desempeño del algoritmo [17]. Este trabajo se encuentra planteado desde el punto de vista de la sintonización, ya que actualmente existen aún muchas aplicaciones que utilizan parámetros estáticos y hacen falta metodologías que ayuden a escoger correctamente estos valores. Ya que estos parámetros generan fuertes interacciones no lineales, no pueden ser sintonizados individualmente sino en conjunto, y la mayoría de investigadores utiliza conjuntos de parámetros que han sido reportados en trabajos anteriores.
Uno de los primeros trabajos realizados en este campo, es el presentado en [15] donde se realizó un estudio sistemático de cómo afectan las variaciones de los parámetros de control al desempeño *en línea* y *fuera de línea* del GA (el desempeño en línea en la generación t se refiere al promedio de la calidad de todos los individuos que han sido analizados a través de las t generaciones, mientras el desempeño fuera de línea es el promedio de los mejores valores de calidad observados a través de las t generaciones). Los resultados obtenidos en este trabajo se pueden resumir como:

- Tamaño de la población: 50 a 100 individuos
- Probabilidad de recombinación: 60 %
- Probabilidad de mutación: 0.1 %

Estos valores siguen siendo ampliamente utilizados a pesar de que no existe una claridad acerca de qué tan bien se puede desempeñar un GA con estos valores, en problemas aparte de los usados en el trabajo original [46].
Otro conjunto de parámetros ampliamente usado, es el propuesto en [24], donde se usó por primera vez un *meta-algoritmo* genético. Dado que los GA son herramientas de optimización, se utilizó un GA de alto nivel para optimizar los parámetros de otro GA que se encontraba realizando una tarea específica (se usaron los mismos problemas de optimización usados en [15]). El algoritmo de alto nivel fue sintonizado con los parámetros encontrados en [15], y obtuvo un mejor desempeño en línea para los valores:

- Tamaño de la población: 30 individuos
- Probabilidad de recombinación: 95 %
- Probabilidad de mutación: 1 %

Sin embargo, no se encontró ningún conjunto de parámetros que superara el desempeño fuera de linea obtenido por [15]. Uno de los principales aportes de este trabajo, consistió en plantear el uso de meta-algoritmos, que han sido usados eficientemente en otros trabajos como [8], donde se introdujeron modificaciones a los métodos de selección de poblaciones iniciales y a los operadores de mutación para mejorar el desempeño en la optimización de funciones, y [74] donde se aplicó el meta-algoritmo para optimizar los parámetros de GA en la optimización de problemas mixtos no lineales. Estos trabajos demuestran que el método de meta-algoritmo resulta ser adecuado pero muy costoso computacionalmente.
En [58], se hicieron esfuerzos adicionales en la determinación experimental de parámetros óptimos, probando un amplio rango de combinaciones de parámetros. En este caso, establecieron los siguientes rangos:

- Tamaño de la población: $20 - 30$ individuos
- Probabilidad de recombinación: $75 - 95\,\%$
- Probabilidad de mutación: $1\,\%$

El pequeño tamaño de las poblaciones obtenido, puede deberse al método usado para evaluar el desempeño del algoritmo (se utilizó solamente el desempeño en línea).
En [27] se propone una técnica de auto-sintonización denominada "GA sin parámetros", que realmente es una sintonización estática estableciendo los valores de la probabilidad de recombinación y la tasa de selección, mediante análisis teóricos.
De todas maneras, a pesar de que estos estudios son una muy buena base para decidir el valor de los parámetros a utilizar, la mayoría de los investigadores concuerdan en que la selección de los parámetros depende del problema específico que se esté tratando y en que no hay una certeza al rededor de este tema [46].
Recientemente, en [65] se tomó el modelo de meta-algoritmo para optimizar los parámetros de control de un GA aplicado a la solución del problema de la mayor subgráfica común. En dicho trabajo se introdujo una red neuronal para la evaluación de la función de adaptabilidad, con el fin de simplificar la ejecución del meta-algoritmo, entrenando la red para predecir el desempeño del algoritmo genético. La introducción de la red resultó ser una buena estrategia para disminuir el tiempo de ejecución pero no presenta una sustentación acerca de la función utilizada para medir el desempeño del GA de bajo nivel, ni el criterio usado para escoger el modelo de la red neuronal (cantidad de neuronas, número de capas, etc.) además de que no permite explicar la interacción entre los parámetros, debido a la misma naturaleza de las redes neuronales.
En [57] se realizó un estudio estadístico usando análisis de varianza (ANOVA) para encontrar los parámetros que tienen una mayor influencia estadística sobre el comportamiento y el desempeño del GA, teniendo en cuenta las interacciones que se presentan entre ellos. Este estudio es un análisis exhaustivo pero no sugiere una manera de sintonizar los parámetros para otras aplicaciones.
Finalmente, cabe señalar el hecho de que se han llevado a cabo una considerable cantidad de estudios teóricos y experimentales sobre el efecto de cada parámetro de control por separado, por ejemplo acerca de la influencia del tamaño de la población: [21, 3, 26], la tasa mutación: [18, 50, 5] y la tasa de recombinación: [67, 72], sin embargo, como se mencionó anteriormente, debido a las relaciones no lineales que se originan entre los parámetros de control, no es recomendable sintonizarlos independientemente.

Capítulo 2

Optimización con restricciones utilizando algoritmos genéticos

Un problema de optimización sujeto a restricciones, consiste en la maximización o minimización de una función $f(\boldsymbol{\lambda})$, sujeta a restricciones que pueden ser de desigualdad $h(\boldsymbol{\lambda}) \geq 0$, o de igualdad $h(\boldsymbol{\lambda}) = 0$. De ahora en adelante se considerarán sólo los problemas de maximización con restricciones de desigualdad, dado que un problema de minimización puede ser convertido en uno de maximización, de la forma:

$$\min_{\boldsymbol{\lambda}} f(\boldsymbol{\lambda}) = \max_{\boldsymbol{\lambda}} 1 - f(\boldsymbol{\lambda}) \tag{2.1}$$

siendo $f(\boldsymbol{\lambda}) \in [0, 1]$ la función a optimizar normalizada y además una restricción de igualdad puede ser convertida en una restricción de desigualdad, de la forma:

$$|h(\boldsymbol{\lambda})| - \epsilon \leq 0 \tag{2.2}$$

donde ϵ representa la tolerancia permitida [12].
Cuando se utiliza un meta-algoritmo para optimizar los parámetros de control de un GA, es necesario definir una función que refleje qué tan bueno es el desempeño de éste último con determinada combinación de parámetros. Esta función de evaluación debe tener en cuenta fundamentalmente dos parámetros: la velocidad de convergencia y la precisión de la solución encontrada. Desde este punto de vista, el problema puede plantearse como un problema de optimización sujeto a restricciones, de la forma:

$$\begin{array}{ll} minimizar & t \\ sujeto\, a & g(\mathbf{c}_{opt}) \geq \alpha \end{array} \tag{2.3}$$

donde t representa el tiempo de convergencia del algoritmo, $g(\mathbf{c}_{opt})$ es la evaluación del mejor cromosoma encontrado por el algoritmo y α es un parámetro que representa la exactitud requerida en la calidad de la solución encontrada.
En este capítulo se presenta una revisión de algunas de las principales técnicas para optimización sujeta a restricciones, utilizadas con algoritmos evolutivos.

2.1. Funciones de penalización

La idea básica de esta técnica consiste en transformar el problema de optimización con restricciones, en un problema no restringido, mediante la sustracción de un cierto valor a la función objetivo, con base en una medida de qué tanto son violadas las restricciones. La formulación general de una función de penalización, tiene la forma:

$$\phi(\boldsymbol{\lambda}) = f(\boldsymbol{\lambda}) - \sum_{i=1}^{n} r_i h_i(\boldsymbol{\lambda}) \tag{2.4}$$

donde $\phi(\boldsymbol{\lambda})$ es la nueva función objetivo expandida, $f(\boldsymbol{\lambda})$ es la función objetivo original, $h_i(\boldsymbol{\lambda})$, $i = 1, ..., n$ son las restricciones y r_i, $i = 1, ..., n$ son constantes positivas, denominadas "coeficientes de penalización".
La penalización debe ser tan pequeña como sea posible, justo sobre el límite en el cual las soluciones no viables (las que no cumplen con alguna o varias de las restricciones) son óptimas. Esto es llamado la *regla de la mínima penalización* [12,64]. Sin embargo, aunque esta regla es conceptualmente simple, no es necesariamente fácil de implementar. Varios autores han buscado desarrollar diferentes técnicas para construir buenas funciones de penalización. A continuación se describen las más importantes.

2.1.1. Penalizaciones estáticas

En esta categoría se consideran las técnicas en las que los coeficientes de penalización no dependen del número de la generación actual y por lo tanto permanecen constantes durante toda la ejecución del algoritmo.
En [32] se propuso una aplicación en la que el usuario define varios niveles de violación de las restricciones y se asigna un coeficiente a cada nivel, de tal manera que el coeficiente aumente conforme la violación sea más alta. Cada individuo es evaluado como:

$$\phi(\boldsymbol{\lambda}) = f(\boldsymbol{\lambda}) - \sum_{i=1}^{n} R_{k,i} \,\text{máx}\{0, h_i(\boldsymbol{\lambda})\}^2 \tag{2.5}$$

donde $R_{k,i}$ son los coeficientes de penalización usados, n es el número total de restricciones y $k = 1, 2, ..., l$ son los niveles de violación definidos.
Otra forma de plantear las penalizaciones, consiste en hacerlas depender del número de constantes violadas, sin tener en cuenta una medida de la violación. Una aplicación de este tipo es presentada en [49], donde la calidad de cada individuo es determinada mediante:

$$\phi(\boldsymbol{\lambda}) = \begin{cases} f(\boldsymbol{\lambda}) & \textit{si cumple todas las restricciones} \\ K - \sum_{i=1}^{s} (\frac{K}{n}) & \textit{en otro caso} \end{cases} \tag{2.6}$$

donde s es el número de restricciones satisfechas, n es el número total de restricciones y K es un valor alto (en los experimentos de [49] se usó $K = 1 \times 10^9$).
Finalmente, en [29] se propuso la siguiente función de penalización:

$$\phi(\boldsymbol{\lambda}) = f(\boldsymbol{\lambda}) + \sqrt{\sum_{i=1}^{n} H(-h_i(\boldsymbol{\lambda}))h_i(\boldsymbol{\lambda})^2} \tag{2.7}$$

donde h_i, $i = 1, ...n$ son las restricciones y $H : \mathbb{R} \rightarrow \{0, 1\}$ es la función de Heaviside, definida como:

$$H(y) = \begin{cases} 1 & si\ y > 0 \\ 0 & si\ y \leq 0 \end{cases} \tag{2.8}$$

La principal ventaja de esta aplicación es la carencia de coeficientes de penalización a sintonizar, sin embargo hace la suposición de que todos los puntos viables poseen en general una mayor evaluación que los puntos no viables, lo cual no siempre es cierto [12].

2.1.2. Penalizaciones dinámicas

En esta categoría se encuentran las aplicaciones en las que el número de la generación actual influye sobre el cálculo de los coeficientes de penalización (generalmente la penalización es mayor conforme avanza la ejecución del algoritmo).
En [35] se propone una técnica donde los individuos son evaluados, en la generación g, por medio de la ecuación:

$$\phi(\boldsymbol{\lambda}) = f(\boldsymbol{\lambda}) - (Ct)^{\alpha} s(\beta, \boldsymbol{\lambda}) \tag{2.9}$$

donde C,α y β son constantes definidas por el usuario (los autores usaron $C = 0{,}5$, $\alpha = 1$ ó 2 y $\beta = 1$ ó 2), y $s(\beta, \boldsymbol{\lambda})$ está definida como:

$$s(\beta, \boldsymbol{\lambda}) = \sum_{i=1}^{n} d_i(\boldsymbol{\lambda})^{\beta} \tag{2.10}$$

siendo

$$d_i(\boldsymbol{\lambda}) = \begin{cases} 0, & h_i(\boldsymbol{\lambda} \leq 0) \\ |h_i(\boldsymbol{\lambda})|, & en\ otro\ caso \end{cases} \tag{2.11}$$

Esta dinámica aumenta la penalización conforme avanza la ejecución del algoritmo.
En [36] se llevó a cabo un estudio detallado de una función de la forma:

$$\phi(\boldsymbol{\lambda}) = f(\boldsymbol{\lambda}) - v(g) \left(A \sum_{i=1}^{n} (\delta_i w_i \Phi(d_i(\boldsymbol{\lambda}))) + B \right) \delta_{\boldsymbol{\lambda}} \tag{2.12}$$

donde A es un "coeficiente de severidad", n es el número total de restricciones, w_i es un factor de ponderación para la restricción $h_i(\boldsymbol{\lambda})$, $d_i(\boldsymbol{\lambda})$ es una medida de la violación de la restricción $h_i(\boldsymbol{\lambda})$, B es un coeficiente de penalización de umbral, $v(g) \in [0, 1]$ es una función creciente del número de la generación actual g, y

$$\delta_i = \begin{cases} 1, & si\ h_i(\boldsymbol{\lambda})\ es\ violada \\ 0, & en\ otro\ caso \end{cases} \tag{2.13}$$

$$\delta_{\boldsymbol{\lambda}} = \begin{cases} 1, & si\ \boldsymbol{\lambda}\ es\ viable \\ 0, & en\ otro\ caso \end{cases} \tag{2.14}$$

Utilizando diferentes funciones de prueba, se experimentó con diferentes formas de la función $v(g)$ (lineal, cuadrática, cúbica, de cuarto orden, exponencial y *5-step*) y se encontró que le mejor desempeño se obtiene para:

$$v(g) = \left(\frac{g}{G} \right)^2 \tag{2.15}$$

siendo G el número total de generaciones.
Otra forma de plantear una penalización dinámica, es bajo la idea del *recocido simulado (simulated annealing)*. Con esta técnica, los coeficientes de penalización son cambiados una vez en muchas generaciones (cuando el algoritmo quede atrapado en un óptimo local). La penalización se incrementa con el tiempo, de forma que los individuos no viables sean fuertemente penalizados en las últimas generaciones. El método presentado en [43] requiere que las restricciones sean divididas en cuatro grupos: igualdades lineales, desigualdades lineales, igualdades no lineales y desigualdades no lineales. Además, debe crearse un conjunto de restricciones activas $\mathfrak{A}$, donde se incluyan todas las igualdades no lineales y las desigualdades no lineales que sean violadas. Cada individuo es entonces evaluado como:

$$\phi(\boldsymbol{\lambda}) = f(\boldsymbol{\lambda}) - \frac{1}{2\tau}\sum_{i\in\mathfrak{A}}\varphi(\boldsymbol{\lambda})^2 \tag{2.16}$$

donde τ es la tasa de enfriamiento y

$$\varphi(\boldsymbol{\lambda}) = \text{máx}\{0, (h_i(\boldsymbol{\lambda}))\} \tag{2.17}$$

Un aspecto interesante de esta aplicación es que la población inicial no es realmente diversa, sino que se compone de múltiples copias de un individuo que satisfaga todas las restricciones lineales. En cada iteración, la temperatura τ es disminuida y la nueva población es conformada por el mejor individuo de la anterior generación. El proceso se detiene cuando se alcanza una *temperatura de congelación* τ_f.
Una propuesta similar a la anterior se encuentra en [62], donde cada individuo es evaluado por:

$$\phi(\boldsymbol{\lambda}) = K(f(\boldsymbol{\lambda})) \tag{2.18}$$

donde K depende de dos parámetros: M, que es una medida de la violación de la restricción, y t, que depende del número de la generación actual g. Usando el principio básico del recocido simulado, K fue definida como:

$$K = e^{-\frac{M}{\tau(g)}} \tag{2.19}$$

de forma que el coeficiente de penalización inicial sea pequeño y se incremente con el tiempo. La tasa de enfriamiento $t(g)$, fue definida como:

$$t(g) = \frac{1}{\tau(g-1)} \tag{2.20}$$

De la misma manera, en la aplicación presentada en [35], también se experimentó con una función de penalización inspirada en el recocido simulado, de la forma:

$$\phi(\boldsymbol{\lambda}) = f(\boldsymbol{\lambda}) - e^{(Cg)^{\alpha}s(\beta\boldsymbol{\lambda})} \tag{2.21}$$

donde $s(\beta\boldsymbol{\lambda})$ está definida como en la ecuación (2.10), $C = 0{,}05$, $\alpha = \beta = 1$.

Algunos investigadores han argumentado que las penalizaciones dinámicas son mejores que las estáticas. Sin embargo, es tan difícil encontrar buenas funciones de penalización

dinámicas, como encontrar buenos coeficientes para funciones de penalización estáticas [12]. Los principales problemas residen en la necesidad de sintonizar varios parámetros para los que no hay un respaldo teórico suficiente (como los parámetros A y B de la aplicación presentada en [36] o el coeficiente C de la aplicación en [35]) y la sensibilidad de la solución encontrada ante los cambios de los parámetros involucrados en las funciones de penalización (este es el caso de la aplicación presentada en [35] ante los parámetros α y β y las aplicaciones que usan la idea de recocido simulado, ante la tasa de enfriamiento τ).
En general, los problemas asociados a las penalizaciones estáticas, también están presentes en las penalizaciones dinámicas: Si se escoge inadecuadamente el factor de penalización, el GA puede converger a soluciones viables no óptimas (si la penalización es muy alta) o a soluciones no viables (si la penalización es muy baja) [63].

2.1.3. Rechazo de las soluciones no viables (*Death penalty*)

Esta técnica consiste en rechazar las soluciones no viables, es decir, asignar un valor de calidad igual a cero para todos aquellos individuos que violen alguna de las restricciones. Dado que no es necesario hacer ningún cálculo adicional para calcular el grado de violación, es muy eficiente desde el punto de vista del gasto computacional.
Sin embargo, al no explotar ninguna información sobre el grado de violación de las soluciones no viables, el algoritmo no podrá avanzar si la población inicial no contiene ningún individuo viable, dado que todos sus individuos tendrán una calidad igual a cero.
Por lo tanto, esta técnica se encuentra limitada a las aplicaciones donde el espacio viable sea convexo y constituya una porción razonablemente grande del espacio de búsqueda total [12].

2.2. Separación de las restricciones y las funciones objetivo

Existen varias aplicaciones donde las restricciones y las funciones objetivo son manejadas por separado, es decir, sin combinar el valor de la función objetivo con la medida de violación de las restricciones. Algunas de las propuestas más representativas, se describen a continuación.

2.2.1. Co-evolución

En [55], se propone una técnica basada en un modelo co-evolutivo en el que hay dos poblaciones: una contiene las restricciones que deben cumplirse (realmente esta no es una población en el sentido convencional ya que su contenido no cambia a través del tiempo) y la otra contiene las posibles soluciones al problema específico que se quiere resolver. Usando una analogía con el modelo de predador-presa, la presión de selección en los miembros de una población, depende de la calidad de los miembros de la otra [55].

Un individuo con una alta calidad en la segunda población, representa un individuo que satisface una gran cantidad de restricciones, mientras un individuo con calidad alta en la primera población, representa una restricción que es violada por una gran cantidad de posibles soluciones. La calidad de cada individuo es calculada como la suma de sus evaluaciones durante las últimas n generaciones (en [55] se usó $n = 25$). La idea es aumentar la calidad de las restricciones difíciles de satisfacer, para que el algoritmo se concentre en éstas.
Un posible problema de esta técnica, como se observa en [12], es que la búsqueda puede dejar de progresar en el caso en que todas las restricciones sean igual de difíciles de satisfacer.

2.2.2. Superioridad de los puntos viables

En [56] se incorporó una regla heurística para tratar las soluciones no viables: la evaluación de las soluciones viables se mapean en el intervalo $(-\text{ínf}, 1)$, mientras la evaluación de las soluciones no viables es mapeada en el intervalo $(1, \text{ínf})$. Los individuos son evaluados mediante:

$$\phi(\boldsymbol{\lambda}) = \begin{cases} f(\boldsymbol{\lambda}) & \text{si el punto es viable} \\ 1 + r\sum_{i=1}^{n} h_i(\boldsymbol{\lambda}) & \text{en otro caso} \end{cases} \tag{2.22}$$

donde $f(\boldsymbol{\lambda}) \in (-\text{ínf}, 1)$, $h_i(\boldsymbol{\lambda}) \in (1, \text{ínf})$, $i = 1, 2, ..., n$ y r es una constante. Nótese que la función objetivo y la medida de violación de las restricciones no se combinan cuando la solución es no viable. En esta aplicación se utilizó escalamiento por ordenamiento lineal (ver sección 1.3.2), de forma que en las primeras generaciones existiera una baja convergencia y más adelante se forzara la convergencia de los individuos mejor calificados.
En [14] se propuso un método similar, en el cual cada individuo es evaluado de la forma:

$$\phi(\boldsymbol{\lambda}) \begin{cases} f(\boldsymbol{\lambda}) & \text{si } h_i(\boldsymbol{\lambda}) \geq 0,\ i = 1, 2, ..., n \\ f_{worst} - \sum_{i=1}^{n} h_i(\boldsymbol{\lambda}) & \text{en otro caso} \end{cases} \tag{2.23}$$

donde f_{worst} es el valor de la función objetivo para la peor solución viable en la población. Si no hay soluciones viables, f_{worst} es establecido en cero.
Usando selección por torneo binario, se aplican las siguientes reglas para comparar dos individuos:

- Una solución viable es siempre preferida sobre una no viable.
- Entre dos soluciones viables, es preferida la que tenga un valor de función objetivo mayor.
- Entre dos soluciones no viables, es preferida la que tenga menor violación de las restricciones.

No se necesitan coeficientes de penalización dado que el proceso de selección utilizado sólo realiza comparaciones por parejas. Por lo tanto, las soluciones viables tienen una calidad igual a la evaluación entregada por la función objetivo. El uso de la medida de violación en la comparación, guía las soluciones no viables hacia la frontera de la región viable. Debido al hecho de que las restricciones generalmente tienen diferentes dimensiones, se realizó una

normalización para evitar alguna clase de sesgo hacia alguna de ellas.

La principal diferencia entre las dos técnicas mencionadas anteriormente es que en la segunda no se necesita un factor de penalización debido a que las comparaciones se hacen por parejas. Sin embargo, en la propuesta en [14] se hace necesario recurrir a técnicas de nichos y especies (ver sección 1.3.3) para mantener la diversidad.

Capítulo 3

Superficies de regresión por vectores de soporte

Las superficies de regresión por vectores de soporte (SVR), son una poderosa técnica para el análisis predictivo de datos [70, 66], con muchas aplicaciones en diversas áreas de estudio [28,38,51]. Su origen se encuentra en las Máquinas de Vectores de Soporte (SVM), una técnica de clasificación nacida a partir de la teoría de aprendizaje estadístico desarrollada por Vladimir Vapnik [70].
En este capítulo se presentan los fundamentos sobre las SVM y su expansión hacia las SVR.

3.1. Máquinas de vectores de soporte

La teoría de las *Máquinas de Vectores de Soporte* (SVM), es una técnica de clasificación que ha tomado mucha atención en años recientes [9, 59]. Tal vez la principal razón de su popularidad, se encuentra en el fuerte respaldo teórico con que cuenta, desarrollado por Vapnik [70] y que se basa en la idea de minimización del riego estructural.
En muchas aplicaciones, las SVM han mostrado tener gran desempeño, más que las máquinas de aprendizaje tradicional como las redes neuronales y han sido introducidas como herramientas poderosas para resolver problemas de clasificación [9]. Una SVM primero mapea los puntos de entrada a un espacio de características de una dimensión mayor y encuentra un hiperplano que los separe y maximice el margen entre las clases en este espacio, como se aprecia en la figura 3.1.
Maximizar el margen es un problema de programación cuadrática (QP) y puede ser resuelto por su problema dual, introduciendo multiplicadores de Lagrange. Sin ningún conocimiento del mapeo, la SVM encuentra el hiperplano óptimo utilizando el producto punto con funciones en el espacio de características que son llamadas kernels. La solución del hiperplano óptimo puede ser escrita como la combinación de unos pocos puntos de entrada que son llamados *vectores de soporte*.

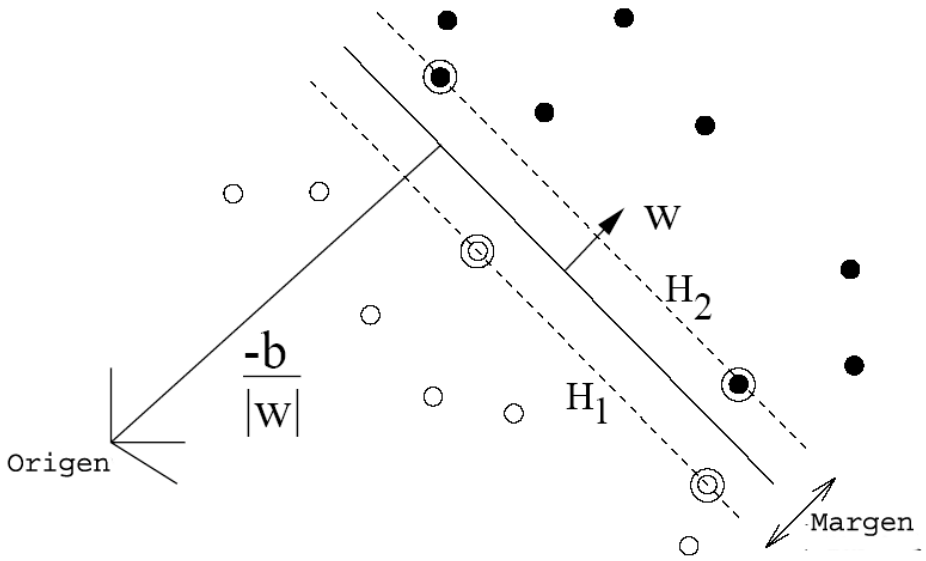


Figura 3.1: Ajuste de margen para una SVM lineal

3.1.1. Caso linealmente separable

Supóngase que se tiene un conjunto S de puntos etiquetados (puntos blancos y negros en la figura 3.1)

$$S = \{(y_1, \mathbf{x}_1), ..., (y_N, \mathbf{x}_N)\} \tag{3.1}$$

donde cada punto de entrenamiento $\mathbf{x}_i \in \mathbb{R}^n$, $i = 1, 2, ..., N$ pertenece a alguna de dos clases y se le ha dado una etiqueta $y_i \in \{1, -1\}$, $i = 1, 2, ..., N$. El objetivo es encontrar un hiperplano de separación de forma tal que aquellos puntos con igual etiqueta queden al mismo lado del hiperplano. Esto significa encontrar un $\mathbf{w}$ y un b tal que

$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) > 0, \; i = 1, ..., N \tag{3.2}$$

donde $\langle \cdot, \cdot \rangle$ denota el producto interno. Si existe un hiperplano que satisfaga (3.2), se dice que los datos son *linealmente separables* y el hiperplano estará descrito por el vector normal $\mathbf{w}$, con una distancia al origen dada por $\frac{b}{\|\mathbf{w}\|}$.
Sin embargo, con el fin de aumentar la capacidad de generalización de la SVM, es necesario encontrar un hiperplano que no sólo separe correctamente los datos, sino que proporcione el mayor margen de separación posible entre las clases. Esto equivale a maximizar la distancia entre los hiperplanos H_1 y H_2 mostrados en la figura 3.1. La ecuación (3.2) se puede escalar, para reescribirse como

$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1, \; i = 1, ..., N \tag{3.3}$$

lo cual conlleva a que la distancia entre los hiperplanos H_1 y H_2 sea igual a $\frac{2}{\|\mathbf{w}\|}$. De esta forma, encontrar el máximo margen equivale a minimizar la norma de $\mathbf{w}$, es decir:

$$\begin{aligned} &\textit{minimizar} && \tfrac{1}{2}\langle \mathbf{w}, \mathbf{w} \rangle \\ &\textit{sujeto a} && y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 \; \forall i \end{aligned} \tag{3.4}$$

Mientras $\langle \mathbf{w}, \mathbf{w} \rangle$ sea convexo, es posible solucionar el problema (3.4) utilizando multiplicadores de Lagrange [9]. Sean $\boldsymbol{\alpha} = \{\alpha_1, ..., \alpha_N\}$, los N multiplicadores de Lagrange no negativos asociados a (3.3). Para minimizar (3.4) se debe encontrar el punto de silla de la función de Lagrange:

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\langle w, w \rangle - \sum_{i=1}^{N} \alpha_i [y_i(\langle w, x_i \langle + b) - 1] \tag{3.5}$$

Para encontrar dicho punto, hay que minimizar la función (3.5) respecto a $\mathbf{w}$ y b, y luego maximizarla sobre los multiplicadores de Lagrange $\alpha_i \geq 0,\ 1 = 1, 2, ..., N$. El punto de silla debe satisfacer las condiciones de Karush-Kuhn-Tucker (KKT):

$$\frac{\partial L(\mathbf{w}, b, \boldsymbol{\alpha})}{\partial b} = \sum_{i=1}^{N} y_i \alpha_i = 0 \tag{3.6}$$

$$\frac{\partial L(\mathbf{w}, b, \boldsymbol{\alpha})}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i \tag{3.7}$$

Substituyendo (3.6) y (3.7) en (3.5) el problema de optimización toma su forma dual:

$$\begin{array}{ll} \textit{maximizar} & \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j}^{N} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \\ \textit{sujeto a} & \sum_{i=1}^{N} y_i \alpha_i = 0 \; y \; \alpha_i \geq 0, \forall i \end{array} \tag{3.8}$$

el cual puede ser solucionado utilizando métodos de programación cuadrática estándar [9]. Una vez el vector $\boldsymbol{\alpha}^0 = \{\alpha_i^0, ..., \alpha_N^0\}$ solución de (3.8) ha sido encontrado, el hiperplano de separación tiene la siguiente forma:

$$\mathbf{w}_0 = \sum_{i=1}^{N} \alpha_i^0 y_i \mathbf{x}_i \tag{3.9}$$

mientras que b puede ser calculado a partir de las condiciones (3.6) y (3.7) como:

$$b_0 = y_i - \langle \mathbf{w}_0, \mathbf{x}_i \rangle \tag{3.10}$$

El problema de clasificar un nuevo punto $\mathbf{x}$, es resuelto examinando el signo de $\langle \mathbf{w}_0, \mathbf{x} \rangle + b_0$.

3.1.2. Caso linealmente no separable

Si los datos no son linealmente separables, buscar un hiperplano óptimo carece completamente de sentido. Con la finalidad de posibilitar las violaciones, se pueden introducir variables de relajación $(\xi_1, ..., \xi_N)$, $\xi_i \geq 0$ [59], de manera que la expresión (3.3) se puede escribir como:

$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 + \xi_i, \forall i \tag{3.11}$$

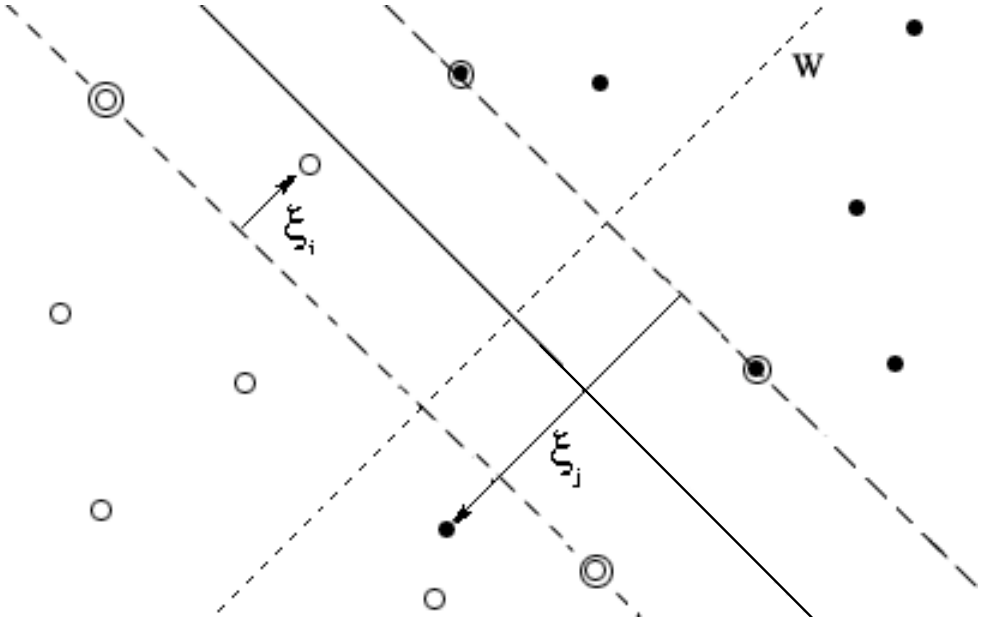


Figura 3.2: Ajuste de *margen suave* para una SVM lineal

El propósito de estas variables es permitir puntos erróneamente clasificados, los cuales correspondan a $\xi_i > 1$, por lo tanto, $\sum_i \xi_i$ puede tomarse como una cota superior del número de errores de entrenamiento. Este clasificador es conocido como un *hiperplano de margen suave* y puede observarse en la figura 3.2.

El hiperplano de separación generalizado, puede ser obtenido como la solución del siguiente problema:

$$\begin{aligned} &\textit{minimizar} && \tfrac{1}{2}\langle \mathbf{w}, \mathbf{w}\rangle + C\sum_{i=1}^{N} \xi_i \\ &\textit{sujeto a} && y_i(\langle \mathbf{w}, \mathbf{x}_i\rangle + b) \geq 1 + \xi_i \text{ y } \xi_i \geq 0 \; \forall i \end{aligned} \tag{3.12}$$

El primer término es minimizado para controlar la capacidad de aprendizaje del mismo modo que en el caso separable; el segundo término permite mantener bajo control el número de clasificaciones erróneas. El parámetro C es elegido por el usuario de manera que un valor grande es equivalente a asignar una alta penalización a los errores. En analogía con el caso separable, la utilización de multiplicadores de Lagrange deriva en el siguiente problema de optimización:

$$\begin{aligned} &\textit{maximizar} && \sum_{i=1}^{N} \alpha_i - \tfrac{1}{2}\sum_{i,j}^{N} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j\rangle \\ &\textit{sujeto a} && \sum_{i=1}^{N} y_i \alpha_i = 0 \; y \; 0 \leq \alpha_i \leq C, \; \forall i \end{aligned} \tag{3.13}$$

De la ecuación (3.13) se puede notar que la única diferencia hasta el momento con el caso separable es que ahora $\boldsymbol{\alpha}$ tiene una cota superior C.

3.1.3. Máquinas no lineales de vectores de soporte

El principio de SVM no lineal consiste en mapear el espacio de entrada a un espacio de representación de dimensión alta a través de una función no lineal elegida a priori [70]. Sin embargo, surge un problema computacional, dado que la dimensión del espacio de representación puede ser muy alta y la dificultad radica en cómo construir un hiperplano de separación en este espacio. La respuesta al problema parte de que para construir dicho hiperplano, el mapeo $z = \phi(\mathbf{x})$ no necesita ser explícito, de manera que reemplazando $\mathbf{x}$ por $\phi(\mathbf{x})$ en (3.8), se tiene:

$$\begin{aligned} &maximizar \quad \textstyle\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j}^{N} \alpha_i \alpha_j y_i y_j \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle \\ &sujeto\ a \qquad \textstyle\sum_{i=1}^{N} y_i \alpha_i = 0 \ y \ \alpha_i \geq 0, \forall i \end{aligned} \tag{3.14}$$

de lo anterior, el algoritmo de entrenamiento solo depende de los datos a través de los productos punto en el espacio de representación, esto es, funciones de la forma $\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$. Sea dada una función kernel simétrica K tal que $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)' \phi(\mathbf{x}_j)$, de modo que el algoritmo de entrenamiento dependa solo de K y el mapeo ϕ no sea usado explícitamente. Dado $\phi : \mathbb{R}^d \rightarrow \mathfrak{H}$, el kernel K es $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$, pero de manera inversa, dado un kernel K se deben establecer las condiciones para que el mapeo exista. Tales condiciones son aseguradas por las condiciones de Mercer:

Teorema 3.1 *Sea $K(\mathbf{x}, \mathbf{y})$ una función simétrica continua en $L^2(\mathbb{C})$, luego, existe un mapeo ϕ y una expansión, tal que:*

$$K(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{\infty} \langle \phi(\mathbf{x})_i, \phi(\mathbf{y})_i \rangle \tag{3.15}$$

si y solo si, para algún $f \in L^2(\mathbb{C})$, tal que

$$\int_{\mathbb{C} \times \mathbb{C}} K(\mathbf{x}, \mathbf{y}) f(\mathbf{x}) f(\mathbf{y}) dx dy \geq 0 \tag{3.16}$$

Nótese que para casos específicos, puede no ser fácil mostrar cuándo son cumplidas las condiciones de Mercer, mientras que (3.16) debe mantenerse para algún $g \in L^2(\mathbb{C})$. Sin embargo, es fácil probar que la condición se cumple para el kernel polinomial $K(\mathbf{x}, \mathbf{y}) = \langle \mathbf{x}, \mathbf{y} \rangle^p$ (ver [9]).

Las principales son:

- Polinomial: $K(\mathbf{x}, \mathbf{y}) = (\langle \mathbf{x}, \mathbf{y} \rangle + c)^d$ para $c > 0$
- Función de base radial (RBF): $K(\mathbf{x}, \mathbf{y}) = exp(-\gamma \|\mathbf{x} - \mathbf{y}\|^2)$ para $\gamma > 0$
- Sigmoide: $tanh(\kappa \langle \mathbf{x}, \mathbf{y} \rangle + \nu)$

El primero resulta en un clasificador con función de decisión polinomial, el segundo un clasificador con función de base radial y el último un tipo particular de red sigmoidal de dos capas. Para el caso de RBF, el número de centros (número de vectores de soporte), los centros (vectores de soporte), los pesos (α_i) y el desplazamiento (b) son generados automáticamente por la SVM en la etapa de entrenamiento y dan excelentes resultados en comparación a la red RBF clásica [61].

3.2. Superficies de regresión por vectores de soporte

Las superficies de regresión por vectores de soporte, surgen como una generalización de las SVM cuando el problema pasa del reconocimiento de patrones, que maneja objetivos $y \in \{+1, -1\}$, a la estimación de funciones reales $y \in \mathbb{R}$.
Esto puede lograrse a través de la llamada *función de pérdida ε-insensitiva* [66]:

$$|y - f(\mathbf{x})|_\varepsilon := max\{0, |y - f(\mathbf{x})| - \varepsilon\} \tag{3.17}$$

la cual no penaliza los errores que estén por debajo de un cierto $\varepsilon \geq 0$.

3.2.1. Regresión εSV

Supóngase que se tienen los datos de entrenamiento $\{(y_1, \mathbf{x_1}), ...(y_N, \mathbf{x}_N)\}$ pero en este caso, a cada punto de entrenamiento $\mathbf{x}_i \in \mathbb{R}^n$, $i = 1, 2, ..., N$ se le asigna un valor real $y \in \mathbb{R}$. La meta es encontrar una función que tenga como máximo una desviación ε de los objetivos obtenidos actualmente y al mismo tiempo sea lo más plana posible. En otras palabras, no importan los errores mientras sean inferiores a ε, pero no se aceptará ninguna desviación más grande que esto. En el caso de una función lineal, se tiene:

$$f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b \tag{3.18}$$

Que la función sea lo más plana posible, denota que se quiere buscar el menor parámetro $\mathbf{w}$ posible. Una forma de asegurar esto es por medio de la minimización de la norma $\langle \mathbf{w}, \mathbf{w} \rangle$. Este problema se puede reescribir como un problema de optimización convexa:

$$\begin{array}{ll} Minimizar & \frac{1}{2}\langle \mathbf{w}, \mathbf{w} \rangle \\ Sujeto\, a & \begin{cases} y_i - \langle \mathbf{w}, \mathbf{x_i} \rangle - b \leq \varepsilon \\ \langle \mathbf{w}, \mathbf{x_i} \rangle + b - y_i \leq \varepsilon \end{cases} \end{array} \tag{3.19}$$

La suposición tácita que se ha hecho es que dicha función g existe, o en otras palabras, que el problema de optimización convexa es realizable. Sin embargo, algunas veces este no es el caso, o se quiere poder permitir algunos errores. Análogamente al problema de "margen suave" de función de pérdida que fue adaptada para máquinas de vectores de soporte, se pueden introducir variables de flexibilidad ξ_i y ξ_i^* que puedan manejar restricciones del

problema, que de otra forma serían no realizables. De esta forma se llega a la formulación establecida en [70]:

$$\begin{array}{ll} Minimizar & \frac{1}{2}\langle \mathbf{w}, \mathbf{w}\rangle + C\sum_{i=1}^{m}(\xi_i + \xi_i^*) \\ Sujeto\ a & \begin{cases} y_i - \langle \mathbf{w}, \mathbf{x_i}\rangle - b \leq \varepsilon + \xi_i \\ \langle \mathbf{w}, \mathbf{x_i}\rangle + b - y_i \leq \varepsilon + \xi_i^* \\ \\ \xi_i,\ \xi_i^* \geq 0 \end{cases} \end{array} \tag{3.20}$$

donde la constante C determina la compensación entre qué tan plana es g y el porcentaje de error sobre el cual las desviaciones mayores a ε son permitidas. Esto corresponde a tratar con el llamado problema de insensibilidad ε con función de pérdida $|\xi|_\varepsilon$ descrito por:

$$|\xi|_\varepsilon = \begin{cases} 0\ si\ |\xi| \leq \varepsilon \\ |\xi| - \varepsilon\ en\ otro\ caso \end{cases} \tag{3.21}$$

La figura 3.3 muestra la situación de forma gráfica. Nuevamente, es posible solucionar el

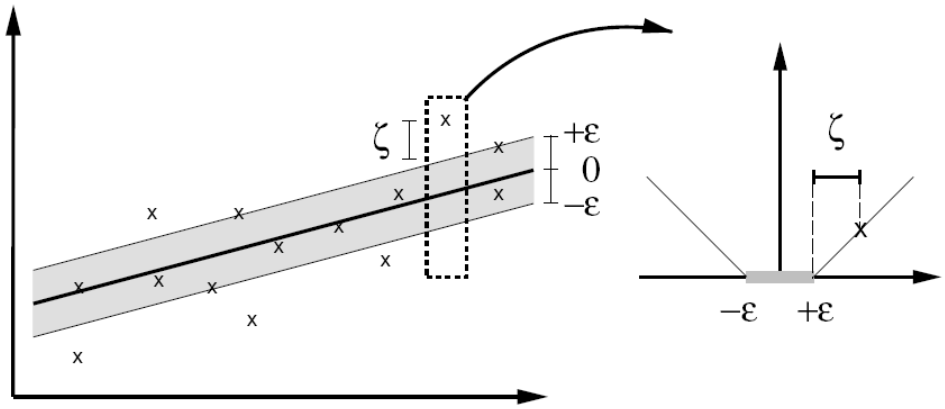


Figura 3.3: Función de pérdida en la regresión εSV

problema (3.20) encontrando el punto silla en la función de Lagrange:

$$\begin{array}{rl} L = & \frac{1}{2}\langle \mathbf{w}, \mathbf{w}\rangle + C\sum_{i=1}^{N}(\xi_i + \xi_i^*) - \sum_{i=1}^{N}(\eta_i\xi_i + \eta_i^*\xi_i^*) \\ & - \sum_{i=1}^{N}\alpha_i(\varepsilon + \xi_i - y_i + \langle \mathbf{w}, \mathbf{x}_i\rangle + b) \\ & + \sum_{i=1}^{N}\alpha_i^*(\varepsilon + \xi_i^* + y_i - \langle \mathbf{w}, \mathbf{x}_i\rangle - b) \end{array} \tag{3.22}$$

donde $\eta_i, \eta_i^*, \alpha_i, \alpha_i^*$ son los multiplicadores de Lagrange. Para encontrar el punto silla, las derivadas respecto a las variables primarias deben ser cero, por lo tanto:

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{N}(\alpha_i^* - \alpha_i) = 0 \tag{3.23}$$

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)\mathbf{x}_i = 0 \tag{3.24}$$

$$\frac{\partial L}{\partial \xi_i} = C - \alpha_i - \eta_i = 0 \tag{3.25}$$

$$\frac{\partial L}{\partial \xi_i^*} = C - \alpha_i^* - \eta_i^* = 0 \tag{3.26}$$

Reemplazando (3.23), (3.24), (3.25) y (3.26) en (3.22), se obtiene el problema de optimización dual:

$$\begin{array}{ll} \text{maximizar} & \left\{ \begin{array}{l} -\frac{1}{2}\sum_{i,j}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\langle \mathbf{x}_i, \mathbf{x}_j\rangle \\ -\varepsilon\sum_{i=1}^{N}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{N} y_i(\alpha_i - \alpha_i^*) \end{array} \right. \\ \text{sujeto a} & \sum_{i=1}^{N}(\alpha_i - \alpha_i^*) = 0 \text{ y } \alpha_i, \alpha_i^* \in [0, C] \end{array} \tag{3.27}$$

La ecuación (3.24) puede reescribirse como:

$$\mathbf{w} = \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)\mathbf{x}_i \tag{3.28}$$

por lo tanto:

$$f(\mathbf{x}) = \sum_{i} = 1^N(\alpha_i - \alpha_i^*)\langle \mathbf{x}_i, \mathbf{x}\rangle + b \tag{3.29}$$

Esta es la llamada *expansión en vectores de soporte* que muestra que $\mathbf{w}$ puede ser completamente descrito por los vectores de entrenamiento $\mathbf{x}_i$ y además, dado que el algoritmo completo puede ser descrito en función de productos internos entre los datos, se puede llegar a la formulación de la regresión no lineal a través de la inclusión de funciones kernel:

$$f(\mathbf{x}) = \sum_{i} = 1^N(\alpha_i - \alpha_i^*)K(\mathbf{x}_i, \mathbf{x}) + b \tag{3.30}$$

3.2.2. Selección del modelo de una SVR

Es sabido que la capacidad de generalización de las SVR (la exactitud de la estimación), depende de una buena selección de los parámetros ε, C y los parámetros del kernel escogido [66, 11, 68].

El parámetro C determina el compromiso entre la complejidad del modelo (suavidad de la regresión) y el grado en que las desviaciones mayores a ε son toleradas en la formulación del problema de optimización (3.27). Si C es infinito, el objetivo es minimizar el riesgo empírico sin importar qué tan complejo sea el modelo.

Por otro lado, el parámetro ε representa la amplitud de la zona $\varepsilon - insensible$, usada para

ajustar los datos de entrenamiento. Este valor puede afectar la cantidad de vectores de soporte usados para construir la función de regresión. Valores altos de ε resultan en pocos vectores de soporte seleccionados y por lo tanto regresiones más suaves (menos complejas).
Finalmente, la selección del kernel depende usualmente del dominio de la aplicación y puede reflejar la distribución de los parámetros de entrada $\mathbf{x}$ [11, 68].

Las principales aplicaciones existentes para la selección del modelo de las SVR, pueden ser resumidas como:

- La selección del modelo por parte del usuario o un experto con base en conocimiento *a priori*, es mencionada en [70, 60]. Basándose en la observación de que los vectores de soporte se encuentran fuera de la zona insensible de radio ε, en [60] se sugiere la inclusión de un parámetro ν que represente la fracción de puntos por fuera de esta zona. Estas aplicaciones por supuesto no son apropiadas para usuarios no expertos.

- En [66] y [37], se han propuesto valores asintóticamente óptimos de ε, que son proporcionales a la varianza del ruido, en concordancia con fuentes generales en SVM. Sin embargo, según [11], la principal desventaja de estas aplicaciones es que la selección de ε no depende de la cantidad de muestras de entrenamiento. Por tal razón, proponen una técnica de selección analítica de ε a partir del tamaño de la muestra y el nivel de ruido presente en los datos de entrenamiento.

- En [42] se propone la selección del parámetro C igual al rango de los valores de salida. En [11] se propone una modificación, en la que se tiene en cuenta la presencia de muestras anómalas en los datos de entrenamiento.

- El uso de validación cruzada es una forma de selección del modelo poderosa y sencilla [66, 59]. Se basa en la idea de que el valor esperado del error en un subconjunto de las muestras de entrenamiento, que no se utilice para entrenar, es igual al error esperado en sí mismo. Un caso particular es el error *leave one out (LOO)*, que es una validación cruzada con un número de particiones igual a N (el número de muestras de entrenamiento).

- Finalmente, dado que la validación cruzada resulta muy costosa computacionalmente (se necesita entrenar y validar k veces, siendo k el número de particiones) se han propuesto diversas estimaciones del error LOO como en [10] y [40].

Parte III

Marco experimental

Capítulo 4

Metodología propuesta

En este capítulo se describe la metodología propuesta para la optimización de los parámetros de control de un algoritmo genético. Esta optimización está enfocada a la reducción del tiempo de cómputo, entendido como el número de generaciones que tarda el GA en encontrar una solución, pero buscando una mínima degradación en la calidad de la solución encontrada.
Como se describió en la sección 1.4, el modelo del meta-algoritmo genético es adecuado para la sintonización de los parámetros de control de un GA, ya que realiza la optimización de todos los parámetros simultáneamente. Un meta-algoritmo está constituido por:

- Un GA denominado *de bajo nivel* con un vector de parámetros de control $\boldsymbol{\gamma}^{\ell} = [\gamma_1^{\ell}, ..., \gamma_n^{\ell}]$ que está optimizando una función específica $g^{\ell}(\boldsymbol{\lambda}^{\ell})$, siendo $\boldsymbol{\lambda}^{\ell} = [\lambda_1^{\ell}, ..., \lambda_M^{\ell}]$ el vector de parámetros de entrada al GA.

- Un GA denominado *de alto nivel* con un vector de parámetros de control $\boldsymbol{\gamma}^{h} = [\gamma_1^{h}, ..., \gamma_n^{h}]$ que está optimizando una función $g^{h}(\boldsymbol{\lambda}^{h})$. Esta función g^{h} es la encargada de entregar una evaluación del desempeño del GA de bajo nivel, por lo tanto las variables de las que depende, coinciden con los parámetros de control de este último $\boldsymbol{\lambda}^{h} := \boldsymbol{\gamma}^{\ell}$

Los superíndices ℓ y h serán utilizados de ahora en adelante para diferenciar los parámetros asociados al algoritmo de bajo nivel y al algoritmo de alto nivel, respectivamente.
Sin embargo, la ejecución de un meta-algoritmo requiere la ejecución del GA de bajo nivel r^h veces por cada generación del GA de alto nivel (siendo r^h la longitud de la población del de alto nivel), lo que lo hace un método muy costoso computacionalmente. Para reducir esta demanda computacional, se propone incorporar una SVR que modele el desempeño del GA de bajo nivel ante diferentes combinaciones de sus parámetros de control, definiendo un conjunto de entrenamiento similar a (3.1):

$$S = \{(y_1, \boldsymbol{\gamma}_1^{\ell}), ...(y_N, \boldsymbol{\gamma}_N^{\ell})\}, \tag{4.1}$$

donde $y_i,\ i = 1, 2, ...N$ es el conjunto de objetivos que representa el desempeño de cada conjunto de parámetros $\boldsymbol{\lambda}_i,\ i = 1, 2, ..., N$ y N es el número de diferentes combinaciones usadas para el entrenamiento de la SVR.

De esta forma, resolviendo el problema de optimización cuadrática (3.27), la función de evaluación de alto nivel, g^h, quedará definida según la ecuación (3.30):

$$g^h(\boldsymbol{\gamma}^\ell) = \sum_{i=1}^{N}(\alpha_i - \alpha_i^*)K(\boldsymbol{\gamma}_i^\ell, \boldsymbol{\gamma}^\ell) + b \tag{4.2}$$

donde α_i y α_i^* son los multiplicadores de Lagrange introducidos para la solución del problema cuadrático (ver sección 3.2) y $K(\mathbf{.},\mathbf{.})$ es la función kernel escogida para la regresión. Esta función constituye una superficie que modela el comportamiento del GA de bajo nivel según la variación de sus parámetros de control, y la evaluación de (4.2) es computacionalmente menos costosa que la ejecución de un GA para obtener una medida de su desempeño.

Para poner en práctica esta metodología, es necesario definir principalmente tres cosas de las cuales depende su eficiencia:

- La función $g^h(\boldsymbol{\gamma}^\ell)$ encargada de generar los objetivos y con los que se entrenará la SVR, de forma que refleje el desempeño de cada conjunto de parámetros de entrenamiento.
- El modelo de la SVR, es decir, los parámetros como el ancho de la zona $\varepsilon-insensible$ (ε), el grado penalización de los errores por fuera de esta zona (C) y el kernel utilizado.
- El modelo del GA de alto nivel, es decir, sus parámetros de control $\boldsymbol{\gamma}^h$ y sus operadores genéticos.

A continuación se explica en detalle cada uno de los anteriores puntos.

4.1. Función de evaluación para medir el desempeño obtenido con cada conjunto de parámetros

Como se mencionó al comienzo del capítulo 2, cuando se utiliza un meta-algoritmo para optimizar los parámetros de control de un GA, es necesario definir una función que refleje qué tan bueno es el desempeño de éste último con determinada combinación de parámetros. Teniendo en cuenta que la metodología está orientada en particular hacia la reducción del tiempo de cómputo, esta función de evaluación debe estar definida de tal manera que las combinaciones de parámetros que gasten un menor número de generaciones en llegar a una solución óptima, reciban una mejor evaluación que las combinaciones que gasten un tiempo mayor. Sin embargo, no se debe descuidar la calidad de la solución encontrada, de forma que el problema se convierte en un problema de optimización sujeto a restricciones, de la forma:

$$\begin{array}{ll} minimizar & t(\boldsymbol{\gamma}^\ell) \\ sujeto\,a & g^\ell(\mathbf{c}_{opt}) \geq \eta \end{array} \tag{4.3}$$

donde $t(\boldsymbol{\gamma}^{\boldsymbol{\ell}})$ representa el tiempo de convergencia del algoritmo usando los parámetros de control $\boldsymbol{\gamma}^{\boldsymbol{\ell}}$, $g^{\ell}(\mathbf{c}_{opt})$ es la evaluación del mejor cromosoma encontrado por el algoritmo de bajo nivel y η es un parámetro que indica qué tan precisa debe ser la solución encontrada. Este parámetro puede ser expresado como un porcentaje de la mejor solución encontrada en el conjunto de entrenamiento:

$$\eta = \alpha \max_{i}\{g^{\ell}(\mathbf{c}_{opt_i})\}, \quad i = 1, 2.., N \tag{4.4}$$

donde c_{opt_i} es el mejor individuo encontrado por el GA de bajo nivel, para el *i-ésimo* conjunto de parámetros de control del conjunto de entrenamiento y α es un valor entre 0 y 1. En el capítulo 2 se presentó una revisión de las principales técnicas para el manejo de esta clase de problemas. Sin embargo, como se observa en la mayoría de las funciones presentadas, surge la necesidad de sintonizar varios parámetros para los que no hay un respaldo teórico suficiente y la solución encontrada es sensible a los cambios en dichos parámetros. La función presentada en [29] resalta por la carencia de coeficientes de penalización que deban ser ajustados, ya que penaliza las soluciones no viables en proporción su distancia euclidiana medida desde el límite de la zona viable, de la forma:

$$f(\mathbf{x}) = g(\mathbf{x}) + \sqrt{\sum_{i=1}^{K} H(-h_i(\mathbf{x}))h_i(\mathbf{x})^2} \tag{4.5}$$

donde h_i, $i = 1, ...K$ son las restricciones y $H(\mathbf{x})$ es la función de Heaviside, definida como:

$$H(y) = \begin{cases} 1 & si\ y > 0 \\ 0 & si\ y \leq 0 \end{cases} \tag{4.6}$$

Sin embargo, como se mencionó antes, esta función supone que todos los puntos viables poseen en general una mayor evaluación que los puntos no viables, lo cual no siempre es cierto [12].
En este trabajo, se propone entonces una función similar, pero que supera estos inconvenientes:

$$f(\mathbf{x}) = (1 - H(-h_i(\mathbf{x})))g(\mathbf{x}) + \sqrt{\sum_{i=1}^{K} H(-h_i(\mathbf{x}))h_i(\mathbf{x})^2} \tag{4.7}$$

Esta función cumple con las propiedades enunciadas en [54]:

- Una solución viable siempre tiene una evaluación más alta que una solución inviable.
- A las soluciones viables, se les asigna una evaluación dependiente únicamente de la función $g(\mathbf{x})$.
- A una solución no viable con un valor pequeño de violación de las restricciones, se le asigna un valor de evaluación más alto que a uno con un valor alto de violación.
- A las soluciones no viables se les asigna una evaluación dependiente solamente de la medida de violación de las restricciones. Esto elimina la necesidad de coeficientes de penalización independientes.

4.2. Selección del modelo para la SVR

La selección del modelo consiste en la elección de los parámetros ε (ancho de la zona insensitiva), C (compromiso entre la complejidad del modelo penalización a los errores por fuera de la zona $\varepsilon - insensible$) y el kernel $K(\mathbf{x}, \mathbf{y})$. Según la revisión presentada en la sección 3.2, estos parámetros pueden ser escogidos de varias formas:

- Por intervención de un experto con conocimiento *a priori*
- Utilizando un algoritmo de búsqueda que minimize el error de validación cruzada (en particular LOO)
- Mediante métodos analíticos

Teniendo en cuanta que el objetivo fundamental de este trabajo es disminuir al máximo la intervención subjetiva del usuario, se descartaron las dos primeras opciones (un algoritmo de búsqueda para minimizar el error LOO también debe tener parámetros de control por sintonizar) y se optó por el método analítico presentado en [11], para la sintonización de ε y C.
En cuanto al kernel, se seleccionó el kernel RBF gaussiano, por sus conocidas buenas propiedades [71], el cual tiene un único parámetro γ a ser sintonizado. Para dicha tarea, se decidió utilizar el algoritmo de búsqueda de Fibonacci, teniendo en cuenta el estudio realizado en [71], donde se demuestra por comparación con la *teoría de espacios escalados* que la variación de la capacidad de generalización de una SVR con kernel RBF gaussiano, forma un espacio de búsqueda semiconvexo, dependiente de γ.

4.2.1. Ancho de la zona ε-insensible

El parámetro ε representa la amplitud de la zona usada para ajustar los datos de entrenamiento. Es bien sabido que el valor de éste debe ser proporcional al nivel de ruido que contengan los datos [66, 70, 37]:

$$\varepsilon \propto \sigma_{noise} \tag{4.8}$$

siendo σ_{noise} la desviación estándar del ruido. Esta desviación debe ser conocida o se debe poder estimar a partir de los datos de entrenamiento.
Sin embargo, en [11] se argumenta que el valor de ε también debe depender del número de muestras de entrenamiento, como se describe a continuación. Considérense los errores de estimación:

$$\delta_i = \hat{y}_i - y_i,\ i = 1, 2, ..., N \tag{4.9}$$

donde y_i son los objetivos reales y $\hat{y}_i$ son los objetivos estimados por la regresión. La media muestral de estos errores, puede considerarse como una estimación del ruido presente en los datos de entrenamiento:

$$\bar{\delta} = \frac{1}{N} \sum_{i=1}^{N} \delta_i \tag{4.10}$$

La constante ε debe ser entonces proporcional a la desviación estándar de $\bar{\delta}$. Considerando a las componentes de la ecuación (4.10) con media cero y varianza σ^2_{noise}, de acuerdo al teorema del límite central, $\hat{\delta}$ debe aproximarse a una variable gaussiana con media cero y varianza σ^2_{noise}/N. Por lo tanto, (4.8) debe convertirse en:

$$\varepsilon \propto \frac{\sigma_{noise}}{\sqrt{N}} \tag{4.11}$$

Basados en comparaciones empíricas y en la teoría de la dimension de Vapnik-Chervonenkis (VC) [70], los autores de [11] reportan un buen desempeño para diferentes tamaños de la muestra de entrenamiento, diferentes niveles de ruido y diferentes funciones objetivo, utilizando la siguiente relación:

$$\varepsilon = 3\sigma_{noise}\sqrt{\frac{\ln N}{N}} \tag{4.12}$$

donde σ_{noise} es la desviación estándar del ruido de los datos y N es el número de muestras de entrenamiento.
Finalmente, la estimación de σ_{noise} se puede realizar a través de una *regresión de k-vecinos más cercanos*, utilizando la ecuación también propuesta en [11]:

$$\hat{\sigma_{noise}}^2 = \frac{N^{1/5}k}{N^{1/5}k - 1}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{4.13}$$

donde k es el orden de la regresión de k-vecinos más cercanos y $\hat{y}_i$, $i = 1, ..., N$ son las estimaciones realizadas por dicha regresión.

4.2.2. Coeficiente de penalización de los errores

El parámetro C determina el compromiso entre la complejidad del modelo (suavidad de la regresión) y el grado en que las desviaciones mayores a ε son toleradas.
Considérese la expansión en vectores de soporte presentada en la ecuación (3.30). También supóngase sin pérdida de generalidad, que la función kernel es acotada en el dominio de entrada (este es el caso, por ejemplo, del kernel RBF gaussiano). Bajo estas suposiciones, se puede asociar el parámetro C al rango de salida de los datos de entrada [42]. Específicamente, recuérdese de la sección 3.1 que el parámetro C acota el rango de valores para los multiplicadores de Lagrange:

$$0 \leq \alpha, \alpha^* \leq C \tag{4.14}$$

Sin embargo, esta selección es sensible a posibles datos anómalos en el conjunto de entrenamiento, por lo cual, en [11] se presenta la siguiente prescripción:

$$C = \text{máx}\,|\bar{y} + 3\sigma_y|, |\bar{y} + 3\sigma_y| \tag{4.15}$$

donde $\bar{y}$ y σ_y son respectivamente la media y la desviación estándar de los objetivos de entrenamiento.

4.2.3. Selección y sintonización del kernel

Se seleccionó el kernel RBF gaussiano, por sus conocidas buenas propiedades [71], el cual está dado por:

$$K(x, y) = exp(-\gamma\|\mathbf{x} - \mathbf{y}\|^2) \tag{4.16}$$

donde γ puede ser expresado como:

$$\gamma = \frac{1}{2\sigma^2} \tag{4.17}$$

y σ es un parámetro que debe ser sintonizado y representa la dispersión de la RBF.
Para esta sintonización, no se cuenta con una forma analítica que pueda ser implementada, así que es necesario utilizar un algoritmo de búsqueda que minimice el error de generalización de la SVR. Sin embargo, en [71] se demuestra por comparación con la *teoría de espacios escalados* que la variación de la capacidad de generalización de la SVR, es una función cuasiconvexa dependiente de σ. Por lo tanto, σ puede ser encontrada utilizando algún método de búsqueda para funciones cuasiconvexas, los cuales generalmente son sencillos y poseen pocos parámetros de control. Se propone utilizar el algoritmo de búsqueda Fibonacci, un procedimiento de búsqueda en línea para la minimización de funciones estrictamente cuasiconvexas, en un intervalo cerrado (el algoritmo se presenta en el apéndice A).
Hace falta entonces una forma de estimar el error de generalización obtenido por la SVR para cada valor de σ. La forma más común de realizar esta estimación, es a través del error LOO, pero esta estimación resulta muy costosa computacionalmente ya que se necesita entrenar un número de superficies de regresión igual al número de muestras de entrenamiento [40]. Por lo tanto, se propone utilizar el límite $\xi\alpha$ enunciado en [40]:

$$LOO_{error} \leq \varepsilon + \frac{2R^2 \sum_{i=1}^{N}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{N}(\xi_i + \xi_i^*)}{N} \tag{4.18}$$

donde α_i, α_i^*, $i = 1, 2, ..., N$ son los multiplicadores de Lagrange introducidos para solucionar el problema (3.27), ξ_i, ξ_i^*, $i = 1, 2, ..., N$ son las variables de relajación introducidas en la formulación del problema (3.20) y R^2 es un límite superior en la similitud entre dos muestras diferentes y para el kernel gaussiano está dado por:

$$R^2 = \exp(-\frac{d_{max}}{2\sigma^2}) \tag{4.19}$$

donde d_{max} es la máxima distancia hallada entre los vectores de entrenamiento:

$$d_{max} = \max_{i,j} \|\mathbf{x}_i - \mathbf{x}_j\|^2 \tag{4.20}$$

4.3. Algoritmo genético de alto nivel

Para que la metodología sea eficiente, el GA de alto nivel debe llevar a cabo una búsqueda amplia a través de la superficie generada. Además, teniendo en cuenta que diferentes combinaciones de parámetros pueden entregar evaluaciones de desempeño similares, la función a optimizar es de carácter multimodal. Por estas razones, se decidió que el GA de

alto nivel tenga la habilidad de crear nichos y especies según las técnicas presentadas en la sección 1.3.
Estas técnicas permiten mantener una distribución uniforme de la población alrededor de áreas altamente calificadas (nichos) y evitar la convergencia de toda la población a un solo óptimo, pero presentan algunos problemas debidos en particular a la selección del parámetro σ_{share} que representa el radio de los nichos encontrados. Por esta razón se propone una técnica de identificación dinámica de los nichos, que es una mejora de la presentada en [39]. Además, para mejorar el desempeño del algoritmo, se propone una técnica de apareamiento restringido para la etapa de cruce.

4.3.1. Algoritmo de identificación dinámica de nichos

Como se mencionó en la sección 1.3, el método más utilizado para generar nichos en la población de un GA es el de *Calidad Compartida (Fitness Sharing)* [23]. En este método se define una distancia σ_{share} por debajo de la cual los individuos compartirán su calidad, definiendo una nueva calidad compartida f':

$$f'(\mathbf{x}_i) = \frac{f(\mathbf{x}_i)}{\sum_{j=0}^{r} s(\mathbf{x}_i, \mathbf{x}_j)} \tag{4.21}$$

donde $s(\mathbf{x}_i, \mathbf{x}_j)$ es una función que define qué tanto deben compartir su calidad las muestras $\mathbf{x}_i$ y $\mathbf{x}_j$, dada por:

$$s(\mathbf{x}_i, \mathbf{x}_j) = \begin{cases} 1 - \frac{\|\mathbf{x}_i, \mathbf{x}_j\|}{\sigma_{share}} & \text{si } \|\mathbf{x}_i, \mathbf{x}_j\| \leq \sigma_{share} \\ 0 & \text{en otro caso} \end{cases} \tag{4.22}$$

Sin embargo, esta formulación acarrea varios problemas debidos al radio σ_{share}, como la dificultad de su determinación sin un conocimiento *a priori* sobre el espacio de búsqueda, o la necesidad de que los nichos deban ser equidistantes.
Un método reciente para la identificación dinámica de nichos, es el propuesto en [39]. Si una función multimodal es mustreada con un número suficientemente alto de puntos, el número de nichos puede ser calculado a partir de la observación de su topografía, mediante la información extraída de las muestras. En la figura 4.1 se muestra la distribución de una población de once individuos, sobre una función multimodal con dos óptimos. Si la calidad $f(\mathbf{x})$ de alguna muestra es mayor que la de sus dos muestras adyacentes (como es el caso de x_3 en la figura 4.1), este punto es considerado como un óptimo relativo. Esto significa que es un óptimo dentro de la vecindad existente, más no necesariamente coincide con el óptimo real del nicho. Por otro lado, si la calidad de una muestra es inferior a sus dos muestras adyacentes (como en el caso de x_6 en la figura 4.1), se considera que esas dos muestras pertenecen a nichos diferentes (en este caso x_5 y x_7 pertenecerán a nichos diferentes).
El algoritmo de [39] procede entonces de la siguiente manera. La muestra con mayor calidad entre toda la población, es asignada como el centro para el primer nicho, el cual es considerado como una hiperesfera cuyo radio es necesario determinar. A continuación, los individuos son ordenados según su distancia al centroide que se acaba de asignar y se busca

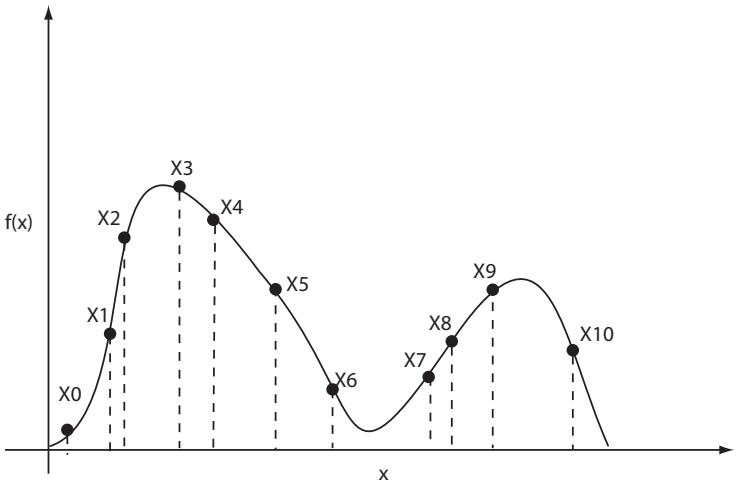


Figura 4.1: Distribución de las muestras en una función multimodal con dos óptimos

una muestra cuyo valor de calidad sea más alto en comparación al valor inmediatamente anterior, y la distancia desde el centroide hasta este punto representará el radio del nicho. Este procedimiento, sin embargo, presenta un inconveniente como se muestra en la figura 4.2. Las muestras han sido organizadas según su distancia al individuo x_3, y la muestra

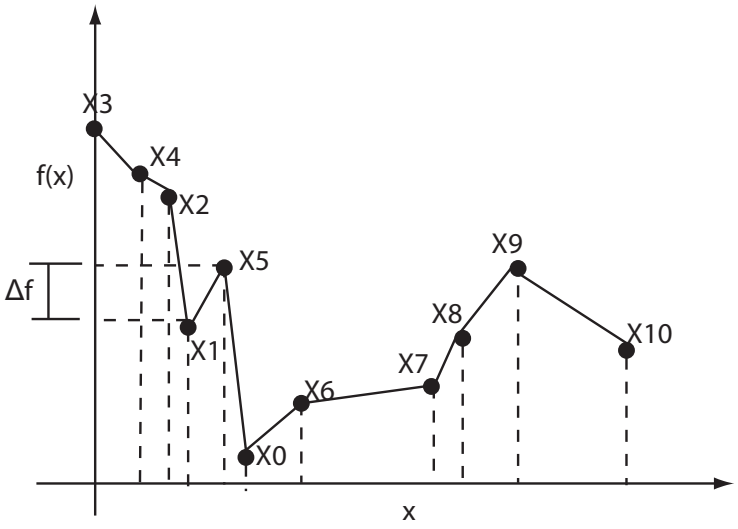


Figura 4.2: Secuencia de las muestras, basada en su distancia al centroide

x_1 aparece como el límite del nicho. No obstante, si se observa la figura 4.1, se puede evidenciar que el límite ha sido mal escogido ya que los límites del nicho deberían ser x_6 ó x_0. Para superar este problema, los autores proponen definir un parámetro Δf que diga la mínima diferencia de calidad por debajo del cual no se considerará un cambio de nicho. Este procedimiento presenta una pobre identificación de los nichos si el parámetro es mal sintonizado (un valor muy grande producirá la fusión de varios nichos en uno solo, mientras

un valor muy pequeño identificará más nichos de los que realmente hay), y no se cuenta con una información suficiente para sintonizarlo (los autores proponen el 10 % de la diferencia entre los valores máximo y mínimo de calidad, sin explicar las razones de esta elección).
Por lo anterior, en este trabajo se propone una modificación del algoritmo, de forma que las muestras no sean ordenadas según su distancia al centroide, sino a través de un algoritmo de distancias encadenadas, es decir, cada muestra es organizada según la distancia a la muestra precedente, comenzando por el centroide. La distribución de las muestras según esta organización, se observa en la figura 4.3. En este caso, el límite es asignado a x_6, sin

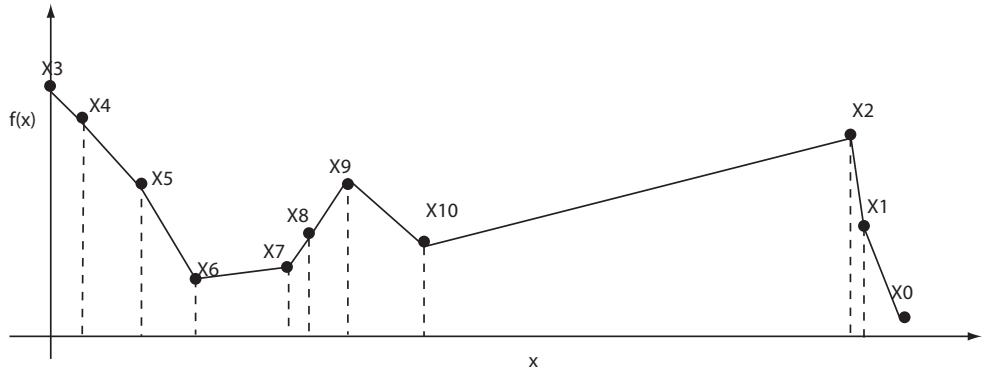


Figura 4.3: Secuencia de las muestras, basada en distancias encadenadas

necesidad de definir ningún parámetro adicional.
Antes de plantear el procedimiento completo para la identificación de nichos, es necesario aclarar dos casos especiales en que dos nichos identificados se traslapan. Los casos posibles son mostrados en las figuras 4.4(a) y 4.4(b).
La figura 4.4(a) presenta el caso donde los nichos se traslapan, y uno de los dos contiene el centroide del otro. En este caso, los dos nichos se fusionan en uno solo, conservando el centroide del nicho más grande y definiendo un nuevo radio R':

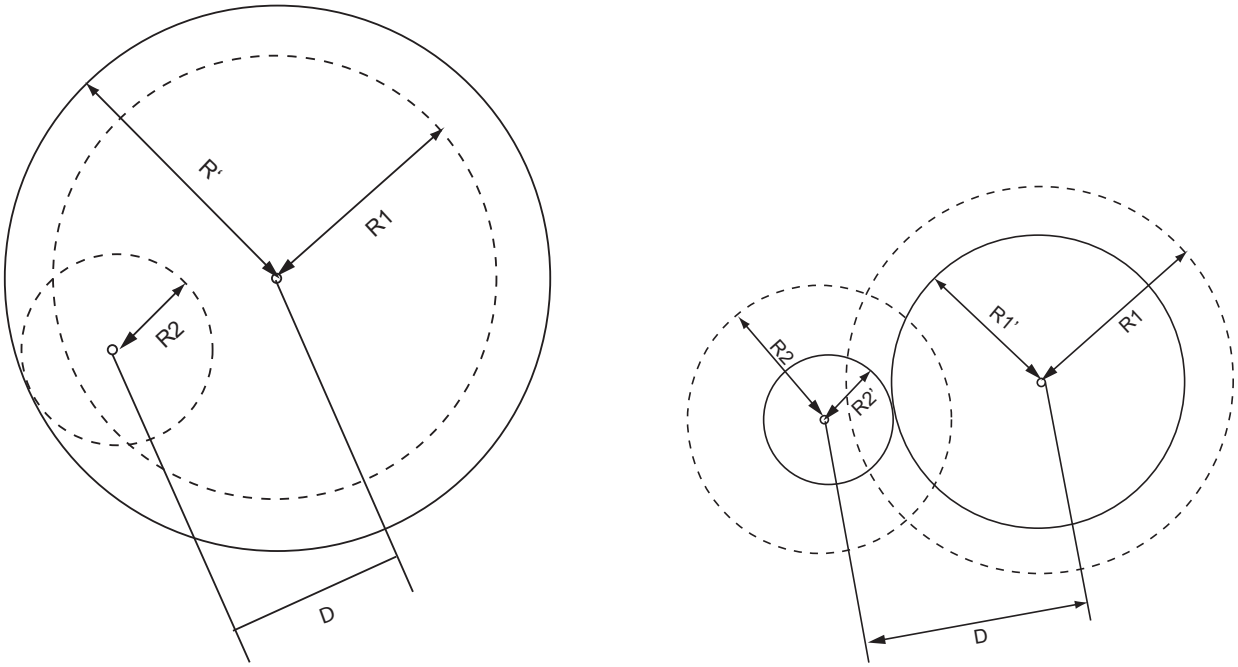
$$R' = \text{máx}\{R_2,\ R_2 + D\} \tag{4.23}$$

donde R_2 es el radio del nicho más pequeño y D es la distancia entre los centroides. De esta manera, si un nicho contiene completamente a otro, el más pequeño desaparece y sólo queda el más grande.
La figura 4.4(b) muestra el caso en que los nichos se traslapan pero ninguno contiene el centroide del otro. En este caso, los dos radios son modificados, de la forma:

$$R'_i = \frac{R_i}{R_1 + R_2} D,\ i = 1, 2 \tag{4.24}$$

donde $R'_i,\ i = 1, 2$ son los nuevos radios y D es la distancia entre los centroides.
El procedimiento de identificación de dinámica de nichos, queda definido de la siguiente forma:

1. El parámetro con la calidad más alta en la población, es escogido como el centroide del primer nicho hiperesférico.



(a) Un nicho contiene el centroide de otro

(b) Los nichos se traslapan sin contener sus centroides

Figura 4.4: Corrección de la interferencia entre nichos

2. Se ordenan las muestras a partir de la muestra marcada como centroide, agregando cada vez la muestra más cercana a la que se acaba de poner en la lista.

3. Se define el radio del nicho, buscando una muestra en la secuencia, cuya calidad sea superior a la de la muestra inmediatamente anterior.

4. Se escoge la muestra con calidad más alta entre la población que aún no pertenece a ningún nicho, y se repiten los pasos 2 y 3 hasta que todas las muestras pertenezcan a algún nicho.

5. Se modifican los nichos que hayan quedado traslapados.

4.3.2. Asignación de la calidad a las muestras según los nichos identificados

Utilizando la información obtenida a partir de la identificación dinámica de nichos, la calidad de cada individuo es asignada según un esquema de calidad compartida propuesto en [39]:

$$f_i' = \frac{f_i}{N_j}\, i = 1, 2, ..., r \tag{4.25}$$

donde f_i' es la nueva calidad asignada a la *i-ésima* muestra de la población, f_i es la calidad de la muestra, N_j es la cantidad de muestras en el nicho al que pertenece la muestra y r es el tamaño de la población.

Según esta definición, cada muestra en el mismo nicho tendrá una penalización igual, mientras las muestras que no han quedado dentro de ningún nicho serán penalizadas con el promedio de las penalizaciones en todos los nichos:

$$f_i' = \frac{f_i}{\frac{1}{m}\sum_{j=1}^{m} N_j} \tag{4.26}$$

siendo m el número total de nichos identificados.

4.3.3. Esquema de apareamiento restringido

Cuando se utilizan algoritmos genéticos con técnicas de nichos y especies, es común combinarlos con métodos de apareamiento restringido. El método más común consiste en permitir que dos individuos se recombinen, solamente si están a una distancia menor a cierto umbral σ_{mating}, el cual en la práctica es escogido igual al radio σ_{share} de los nichos. Sin embargo, como se expone en [41], la recombinación entre clases puede ser útil para eventualmente descubrir nuevos nichos. Por lo tanto, se propone un esquema de apareamiento restringido que no elimina totalmente las oportunidades de cruzamiento entre individuos de nichos diferentes, sino que define la probabilidad de cruzamiento en función de la distancia entre los individuos:

$$\chi = 1 - d(x_1, x_2) \tag{4.27}$$

donde χ es la probabilidad de recombinación para los individuos $\mathbf{x}_1$ y $\mathbf{x}_2$, y $d(\blacksquare, \blacksquare) \in [0, 1]$ es la distancia normalizada entre los dos individuos.

Capítulo 5

Experimentos y resultados

En este capítulo se describen los experimentos realizados para probar la metodología propuesta y se presentan los resultados obtenidos con ésta y con otras metodologías usadas para determinar los parámetros de control de un GA.

5.1. Funciones de evaluación comparativa

Para probar la metodología, se escogió el conjunto de *funciones de evaluación comparativa (benchmarking)* presentado en [16], compuesto de ocho funciones entre las que hay continuas y discontinuas, convexas y no convexas, unimodales y multimodales, de baja y alta dimesionalidad, deterministas y estocásticas. Las primeras cinco pertenecen al conjunto utilizado en [15] para determinar sistemáticamente la influencia de los parámetros de control sobre el desempeño de un GA.
El conjunto completo se encuentra descrito matemáticamente en la tabla 5.1 y es mostrado en las figuras 5.1 y 5.2, donde las funciones han sido reducidas de dimensión par poder ser graficadas e invertidas para mostrarlas como problemas de maximización, tarea realizada por el GA de bajo nivel. A continuación se describe brevemente cada una.

f1: Esfera Esta función es suave, unimodal, fuertemente convexa y simétrica. Representa uno de los problemas de optimización más simples que se pueden tener.

f2: Función de Rosenbrock Esta función es considerada difícil porque presenta un borde elevado muy angosto cuya cima está bien definida y se encuentra alrededor de una parábola. El algoritmo debe descubrir una buena dirección para ser eficiente en este problema.

f3: Escalón Esta función es representativa de las superficies planas, las cuales constituyen obstáculos para los algoritmos de optimización dado que no proporcionan ninguna información sobre cuál dirección es favorable.

f4: Función cuártica Es una función unimodal, adicionada con ruido. El ruido gaussiano asegura que el algoritmo nunca encuentre el mismo valor al evaluar un mismo punto. Representa problemas de optimización altamente ruidosos.

Tabla 5.1: Funciones de evaluación comparativa

Nombre	Función	Límites
f1	$f_1 = \sum_{i=1}^{2} x_i^2$	$-5{,}12 \leq x_i \leq 5{,}12$
f2	$f_2 = 100(x_1^2 - x_2)^2 + (1 - x_1)^2$	$-2{,}048 \leq x_i \leq 2{,}048$
f3	$f_3 = \sum_{i=1}^{5} int.(x_i)$	$-5{,}12 \leq x_i \leq 5{,}12$
f4	$f_4 = \sum_{i=1}^{30}(ix_i^4 + Gauss(0,1))$	$-1{,}28 \leq x_i \leq 1{,}28$
f5	$f_5(x_i) = 0{,}002 + \sum_{j=1}^{25}\left(\frac{1}{j} + \sum_{i=1} 2(x_i - a_{ij})^6\right)$	$-65{,}536 \leq x_i \leq 65{,}536$
f6	$f_6 = 10V + \sum_{i=1}^{10}(-x_i \sin\sqrt{\lvert x_i\rvert}), \quad V = 4189{,}829101$	$-500 \leq x_i \leq 500$
f7	$f_7 = 20A + \sum_{i=1}^{20}(x_i^2 - 10\cos(2\pi x_i)), \quad A = 10$	$-5{,}12 \leq x_i \leq 5{,}12$
f8	$f_8 = 1 + \sum_{i=1}^{10}\left(\frac{x_i^2}{4000} - \prod_{i=1}^{10}\left(\cos\left(\frac{x_i}{\sqrt{i}}\right)\right)\right)$	$-600 \leq x_i \leq 600$

f5: Agujeros (*Foxholes*) Esta función es el ejemplo de un problema de optimización con muchos óptimos locales. Muchos algoritmos de optimización se quedan en el primer pico que encuentran.

f6: Función de Schwefel Es una función multimodal, cuya principal característica es que tiene un segundo mejor máximo que está muy alejado del óptimo.

f7: Función de Rastrigin Esta función tiene un alto número de óptimos locales. Su superficie está determinada por las variables externas A y ω, que controlan la amplitud y la frecuencia de la modulación, respectivamente. Con $A = 10$, el intervalo escogido es dominado por la modulación y la función es altamente multimodal.

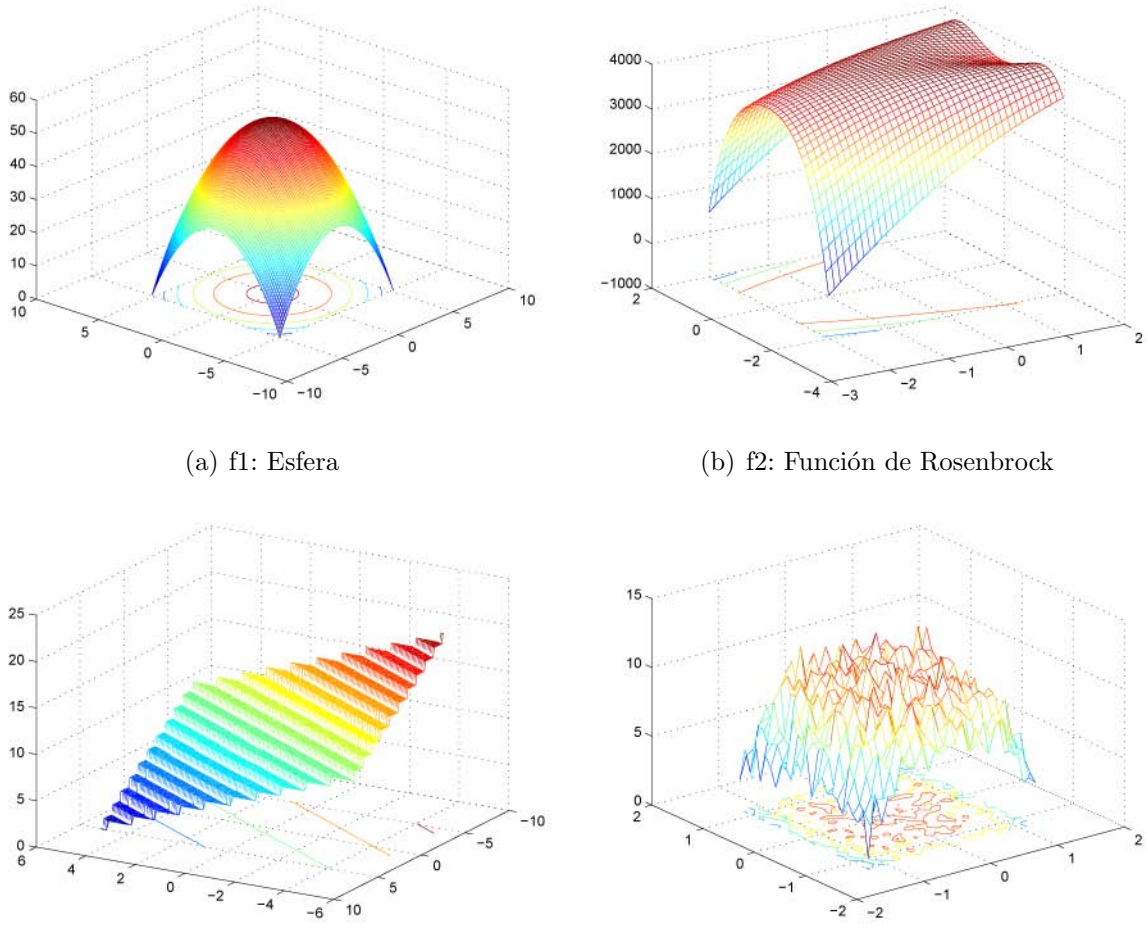
f8: Función de Griewangk En esta función, los términos de la sumatoria producen una parábola y los productos crean subpoblaciones altamente dependientes. Como resultado, se tiene una función altamente multimodal.

5.2. Parámetros de control a sintonizar

Como algoritmo encargado de optimizar las funciones mencionadas en la sección anterior, se escogió un GA con las siguientes características:

- Población inicial generada aleatoriamente.
- Recombinación de un solo punto.
- Mutación por negación del bit.
- Escalamiento de la calidad según el método de ordenamiento exponencial.

Por lo tanto, se tienen cuatro parámetros de control que deben ser sintonizados: tamaño de la población inicial (r), probabilidad de recombinación (χ), probabilidad de mutación (μ) y el parámetro s del ordenamiento exponencial (ver sección 1.3.2), que representa el



(a) f1: Esfera

(b) f2: Función de Rosenbrock

(c) f3: Escalón

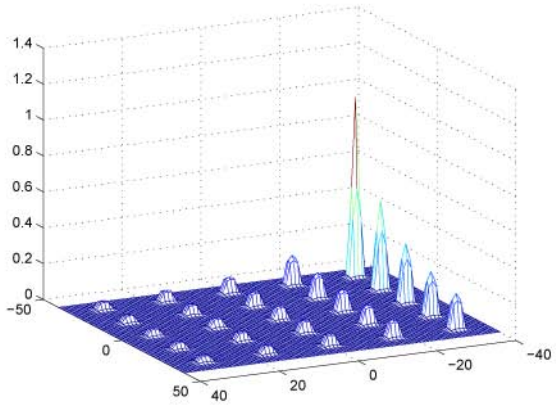
(d) f4: Función cuártica con ruido

Figura 5.1: Funciones de evaluación comparativa f1 a f4

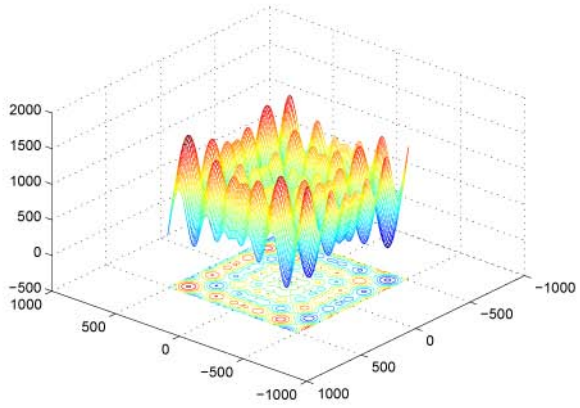
valor esperado de copias para el segundo mejor individuo de cada generación.
Estos parámetros deben codificarse para formar el vector de entrada al GA de alto nivel, para lo cual se escogieron los siguientes rangos:

- Tamaño de la población inicial: Desde 10 hasta 160 individuos, con incrementos de 10 (4 bits).
- Probabilidad de recombinación: Entre 0.3 y 1, con incrementos de 0.1 (3 bits).
- Probabilidad de mutación: 10^{-i}, para $i = 1, ..,8$ (3 bits)
- Parámetro del ordenamiento exponencial: Entre 0.9 y 0.99 con incrementos de 0.006 (4 bits).

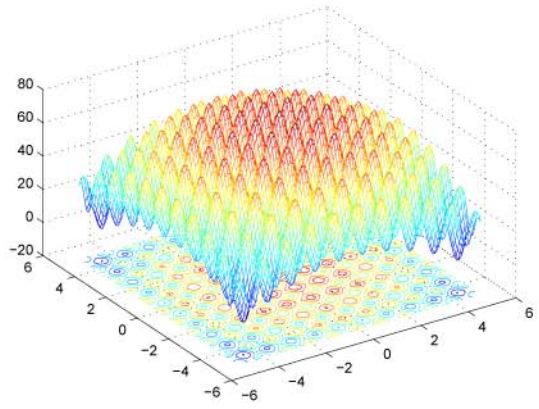
Los primeros tres parámetros constituyen el conjunto fundamental de parámetros de control en un GA, mientras que el último se incluyó como un ejemplo de los parámetros resultantes al adicionar otros operadores especiales, en este caso, un método de escalamiento



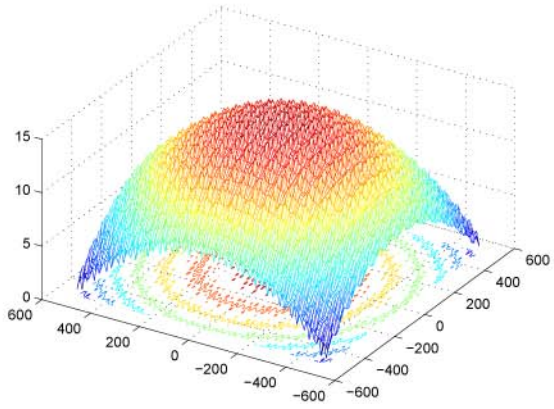
(a) f5: Agujeros (*Foxholes*)



(b) f6: Función de Schwefel



(c) f7: Función de Rastrigin



(d) f8: Función de Griewangk

Figura 5.2: Funciones de evaluación comparativa f5 a f8

de calidad. Los rangos escogidos para la búsqueda del conjunto óptimo, están basados en los valores propuestos por diversos autores como [15, 24, 58] y [65].

5.3. Construcción de las superficies de regresión y ejecución del algoritmo genético de alto nivel

Con la codificación propuesta en la sección anterior, se forma un espacio de búsqueda discreto de 2^{14} combinaciones (más de dieciséis mil) que debe ser modelado por la SVR. Los objetivos para la regresión fueron generados utilizando la función (4.7), que queda definida de la forma:

$$g^h(\boldsymbol{\gamma}^\ell) = (1 - H(-h(\boldsymbol{\gamma}^\ell)))f(\boldsymbol{\gamma}^\ell) + H(-h(\boldsymbol{\gamma}^\ell))h(\boldsymbol{\gamma}^\ell)^2 \tag{5.1}$$

con

$$\begin{aligned} f &= t(\boldsymbol{\gamma}^\ell) \\ h &= g^\ell(\mathbf{c}_{opt}) - \alpha \,\text{máx}_i\{g^\ell(\mathbf{c}_{opt_i})\} \end{aligned} \tag{5.2}$$

siendo $\alpha = 0.99$. De esta forma, se busca minimizar el tiempo de convergencia del algoritmo, sujeto a que las soluciones encontradas no tengan un desempeño inferior al 99 % de la mejor evaluación encontrada en el conjunto de entrenamiento. Sin embargo, dado que esta función es discontinua, sería difícil para la SVR modelar el desempeño del GA de bajo nivel, por lo tanto se decidió construir dos superficies: Una para modelar el número de generaciones y otra para modelar la evaluación obtenida, y con las estimaciones entregadas por estas superficies, evaluar la función (5.1).

Para cada una de las funciones de evaluación comparativa descritas en la sección 5.1, se realizaron entonces los siguientes pasos:

- Se generaron aleatoriamente dos mil conjuntos de parámetros de control $\boldsymbol{\gamma}^\ell$. Utilizando cada uno de estos conjuntos, se corrió el GA de bajo nivel sobre la función de prueba diez veces, y se registraron los promedios de la mejor evaluación obtenida $g^\ell(\boldsymbol{\gamma}^\ell)$ y el número de generaciones empleado $t(\gamma^\ell)$.
- Utilizando los valores obtenidos como objetivos y los conjuntos de parámetros como muestras, se construyeron dos SVR: Una para modelar el número de generaciones empleado, y otra para modelar la mejor evaluación obtenida.
- Se ejecutó el algoritmo genético de alto nivel diez veces, en la optimización de la función (5.1). La solución más repetida representa el conjunto de parámetros óptimo para la solución de la función de prueba.

La construcción de las superficies de regresión fue realizada con el *toolbox* SVM^{light}, desarrollado por Thorsten Joachims [34]. Este toolbox posee un poderoso algoritmo de entrenamiento que permite resolver el problema de optimización cuadrática en un tiempo reducido, y es expuesto en el apéndice B.

5.4. Metodologías de referencia

Con el fin de comparar los resultados arrojados por la metodología propuesta, se realizaron las mismas pruebas usando otras dos metodologías para la sintonización de los parámetros de control del GA de bajo nivel. Éstas son descritas a continuación.

5.4.1. Parámetros recomendados por la literatura

Como se mencionó en la sección 1.4, una de las prácticas más comunes consiste en sintonizar los parámetros de control según los datos reportados por algún otro autor. Dentro de estos, se encuentran los parámetros propuestos por De Jong en [15], donde se reportan los siguientes:

- Tamaño de la población: 100 individuos
- Probabilidad de recombinación: 60 %
- Probabilidad de mutación: 0.1 %

En cuanto al parámetro del ordenamiento exponencial, se utilizó el valor recomendado en [25] de 0.99.

5.4.2. Meta-algoritmo genético con redes neuronales

Esta metodología, propuesta en [65], consiste en modelar el desempeño del GA de bajo nivel a través de una red neuronal, y luego ejecutar un GA de alto nivel cuya función de evaluación es entregada por esta red.
La red neuronal cuenta con cuatro entradas (una para cada uno de los parámetros de control del GA), una capa oculta de seis neuronas y una capa de salida de cuatro neuronas, cada una con función de activación tangente sigmoidal. El método de entrenamiento utilizado fue el *backpropagation* con 400 épocas e inicialización aleatoria.
Las cuatro neuronas de salida codifican la calidad del conjunto de parámetros en dieciséis valores discretos. Para generar los objetivos en el entrenamiento, se utilizó la función:

$$f(\boldsymbol{\gamma}^{\ell}) = round\left(15\frac{9t_1 + t_2}{10}\right) \tag{5.3}$$

donde:

$$t_1 = \frac{g^{\ell}(\mathbf{c}_{opt})}{\text{máx}_i\{g^{\ell}(\mathbf{c}_{opt_i})\}} \tag{5.4}$$

$$t2 = 1 - \frac{t(\boldsymbol{\gamma}^{\ell})}{\text{máx}_i\{t(\boldsymbol{\gamma}_i^{\ell})\}} \tag{5.5}$$

El GA de alto nivel, es un algoritmo genético simple con los siguientes parámetros:

- Tamaño de la población: 100 individuos
- Probabilidad de recombinación: 95 %
- Probabilidad de mutación: 0.25 %

La red neuronal fue entrenada para cada una de las funciones de prueba descritas en la sección 5.1, con los mismos conjuntos de dos mil muestras generados aleatoriamente que se usaron para entrenar las SVR.
Se ejecutó el algoritmo genético de alto nivel diez veces, en la optimización de la función (5.3). La solución más repetida representa el conjunto de parámetros óptimo para la solución de cada función de prueba.

5.5. Resultados

En las tablas 5.2 y 5.3, se muestran el número de generaciones y el porcentaje de error respecto al máximo, respectivamente, para los conjuntos de parámetros obtenidos con las dos metodologías de referencia, y la metodología propuesta. Cada prueba fue repetida diez veces con el fin de disminuir la influencia aleatoria inherente de los GA. Las tablas muestran las medias y desviaciones estándar obtenidas para cada función. Las funciones cuyo número de generaciones esté indicado como > 100 significa que superaron el máximo número de iteraciones, establecido en 100.

Tabla 5.2: Número de generaciones obtenido con cada metodología

	Metodologías		
Funciones	Parámetros recomendados	Meta-GA Red Neuronal	Meta-GA SVR ($\alpha = 0.99$)
f1	42.8 ± 9.6	15.3 ± 3.2	6.6 ± 3.6
f2	48.9 ± 17	10 ± 0.8	6.2 ± 2.0
f3	96.1 ± 9.2	57.7 ± 25.5	83.9 ± 5.9
f4	> 100	> 100	77.4 ± 24.7
f5	62.6 ± 11.7	30.6 ± 4.3	33.9 ± 4.22
f6	70.5 ± 6.3	15.6 ± 3.7	47.2 ± 18.8
f7	> 100	> 100	44.3 ± 9.9
f8	94.5 ± 0.6	47.3 ± 14.9	32.5 ± 10.6

Tabla 5.3: Porcentaje de error respecto al máximo verdadero

	Conjuntos de Parámetros		
Funciones	Parámetros recomendados	Meta-GA Red Neuronal	Meta-GA SVR ($\alpha = 0.99$)
f1	$2.28e-4 \pm 2.15e-4$	$8.39e-4 \pm 2.2e-4$	$6.90e-2 \pm 2.2e-2$
f2	$3.27e-3 \pm 4.45e-3$	$1.41e-3 \pm 1.13e-3$	$6.99e-3 \pm 5.3e-3$
f3	1.15 ± 1.01	3.84 ± 1.58	1.15 ± 0.81
f4	1.64 ± 0.40	2.55 ± 0.14	$5.33e-1 \pm 3.34e-1$
f5	19.94 ± 8.84	14.20 ± 5.37	1.71 ± 5.40
f6	7.61 ± 1.14	8.47 ± 1.42	7.03 ± 1.60
f7	21.01 ± 1.56	4.14 ± 0.69	2.98 ± 1.03
f8	9.28 ± 0.21	$6.91e-2 \pm 0.02$	$6.21e-1 \pm 3.06-1$

De las tablas 5.2 y 5.3, es posible observar que la metodología propuesta, mejora el tiempo de cómputo en la mayoría de funciones, conservando unos porcentajes de error muy similares e incluso inferiores a los de la metodología propuesta en [65].
A pesar de que en algunas funciones la metodología de [65] parece haber alcanzado un buen resultado en un menor número de generaciones, como en el caso de $f3$ y $f5$, es necesario observar que los porcentajes de error en esas mismas funciones, son de 3.84

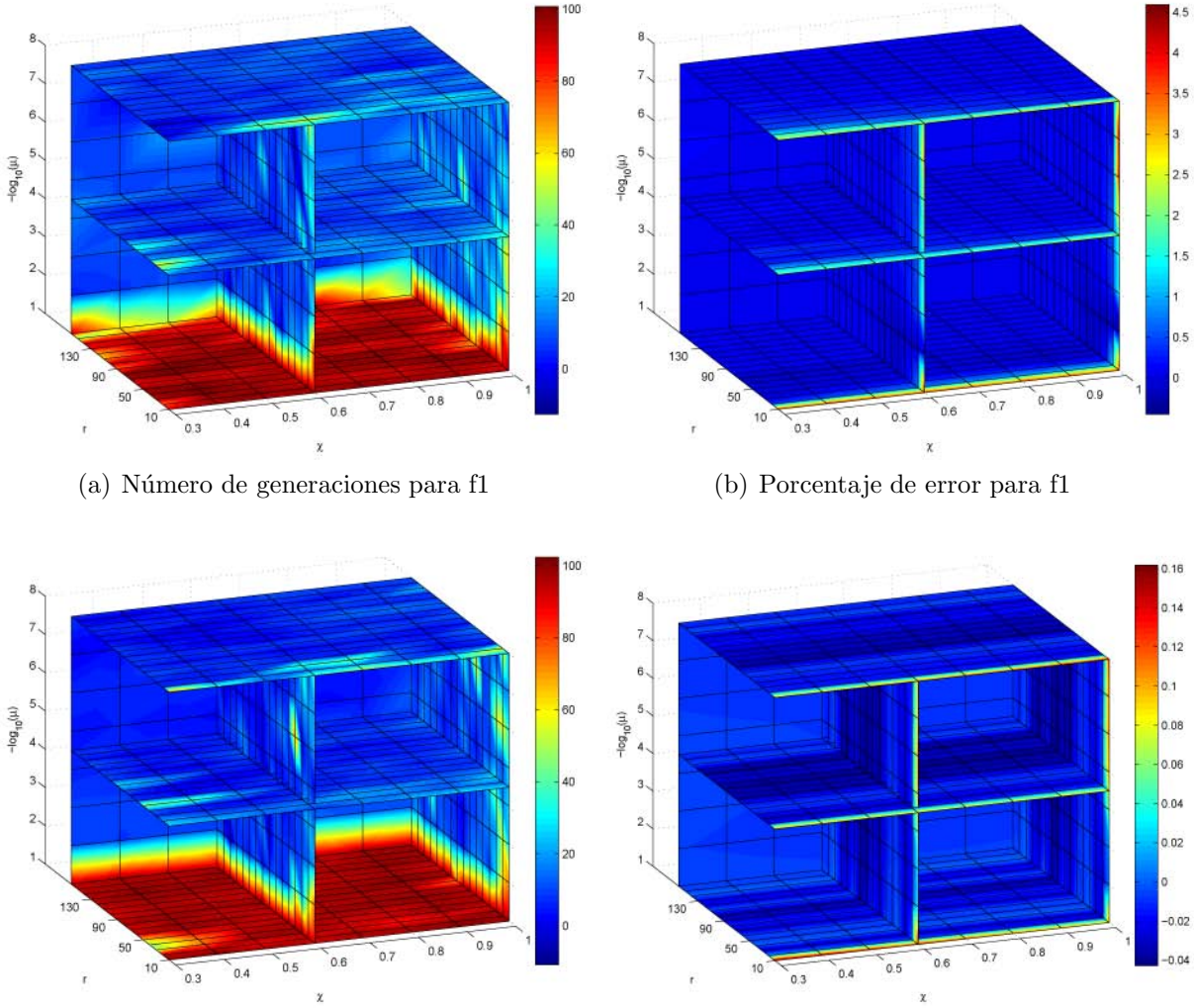
y 14.2 respectivamente, mientras que para la metodología propuesta los errores son de apenas 1.15 y 1.71. Esto es debido principalmente a la restricción impuesta sobre la función de evaluación. Recuérdese de las ecuaciones (5.1) y (5.1), que el haber seleccionado $\alpha = 0.99$ significa que se le está pidiendo al GA de alto nivel *encontrar la configuración de los parámetros de control que minimice el número de generaciones requerido para obtener una solución con un error no inferior al* $1\,\%$. Aunque los errores fueron superiores al $1\,\%$, esto es debido a que el algoritmo de entrenamiento no conoce el máximo verdadero sino la mayor evaluación dentro del conjunto de entrenamiento, sin embargo, los errores se aproximan al valor pedido.
Al repetir las pruebas pero esta vez escogiendo $\alpha = 0.95$, es decir, permitiendo errores cercanos al $5\,\%$, se obtuvieron los datos mostrados en la tabla 5.4.

Tabla 5.4: Resultados obtenidos para $\alpha = 0.95$

	Meta-GA SVR $\alpha = 0.95$	
Funciones	Número de generaciones	Porcentaje de Error
f1	8.2 ± 4.9	$8.9e-3 \pm 1.59$
f2	4.2 ± 2.0	$4.6e-3 \pm 1.45e-3$
f3	23.2 ± 26.4	3.37 ± 2.07
f4	49.9 ± 26.1	$2.82 \pm 8.70e-1$
f5	18.9 ± 3.5	7.18 ± 1.36
f6	8.8 ± 2.3	9.33 ± 1.71
f7	35.2 ± 0.6	$5.28 \pm 7.81e-1$
f8	37.3 ± 14.9	$5.21e-1 \pm 3.33e-1$

Se observa que esta vez el número de generaciones obtenido fue de 23.2 y 18.9 para $f3$ y $f5$ respectivamente, con errores del $3.37\,\%$ y $7.18\,\%$. Esto supera ampliamente los resultados obtenidos por la metodología de [65], con porcentajes de error incluso inferiores. En general, el número de generaciones bajó para las funciones en las cuales la disminución de α permitió encontrar soluciones con mayor porcentaje de error, el cual es el caso de las funciones $f3$ a $f7$. Sin embargo, para $f1$, $f2$ y $f8$, los porcentajes de error no aumentaron. Esto se debe a que α es un *límite inferior* de la exactitud, lo cual no impide que se encuentren soluciones de una exactitud mayor. Por lo tanto, el número de generaciones permaneció alrededor de los mismos valores. Se puede concluir entonces, que el parámetro α, proporciona una medida confiable sobre el porcentaje de error que se permitirá cometer al GA, a cambio de una reducción significativa en el tiempo de cómputo.

Utilizando las superficies de regresión entrenadas, se construyeron las gráficas mostradas en las figuras 5.3, 5.4 y 5.6, las cuales presentan el número de generaciones y el porcentaje de error para cada una de las ocho funciones de prueba, ante las variaciones de los tres parámetros principales. Con el fin de poder llevar los modelos a una representación gráfica, la dependencia respecto al parámetro s del ordenamiento exponencial no fue incluida, sino que éste se dejó fijo en 0,99. Además, la probabilidad de mutación μ fue incluida en escala logarítmica para su mejor visualización.
Cada combinación de r, χ y μ está representada por un punto cuyo color indica el número



(a) Número de generaciones para f1

(b) Porcentaje de error para f1

(c) Número de generaciones para f2

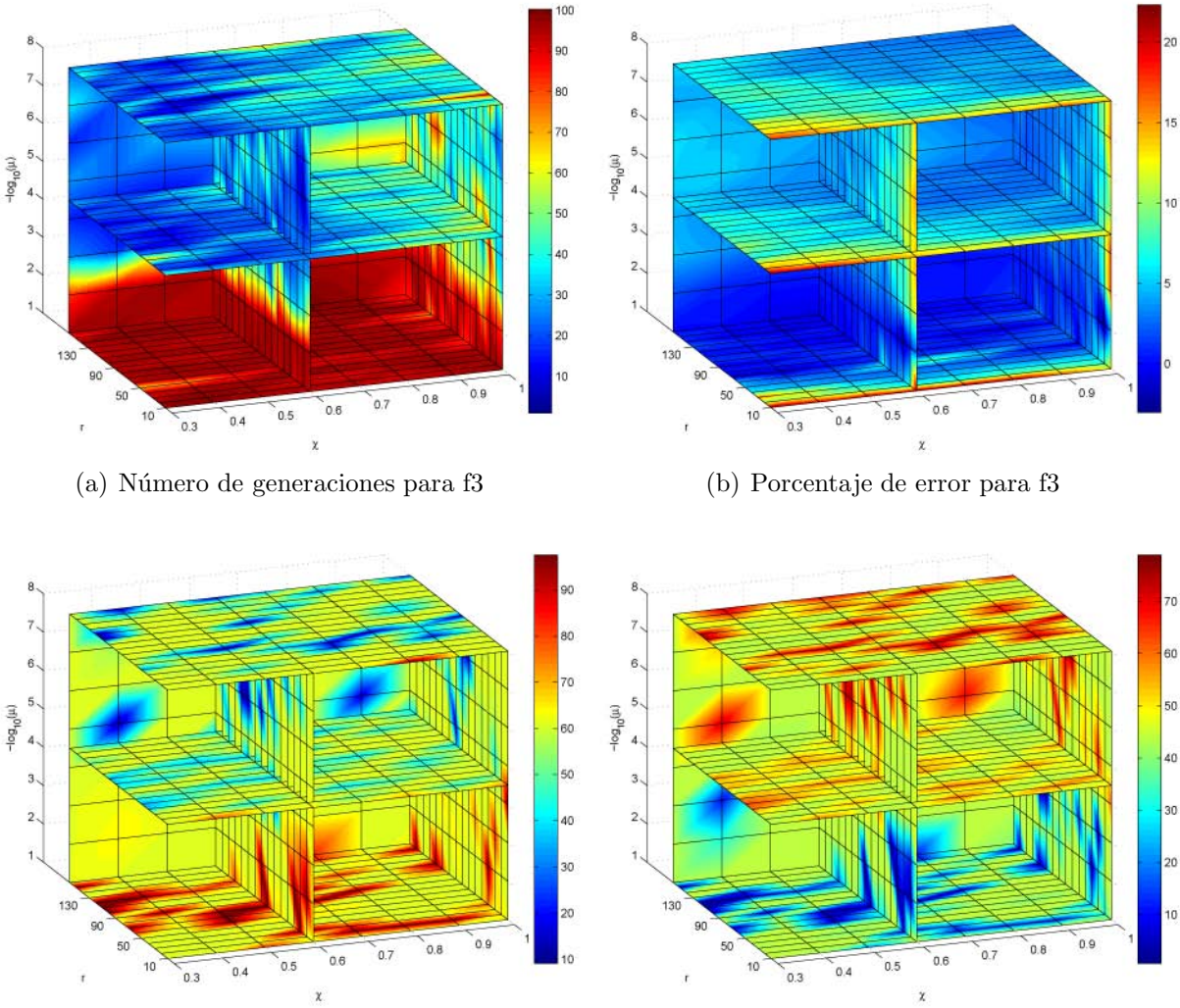
(d) Porcentaje de error para f2

Figura 5.3: Modelos obtenidos con las SVR para f1 y f2

de generaciones o el porcentaje de error obtenido con dicha combinación. Esto generaría una distribución cúbica con todas las posibles combinaciones. Las gráficas muestran cortes longitudinales de estos cubos, de manera que sea posible ver su interior.

Observando el conjunto total de gráficas, es posible distinguir comportamientos comunes a todas las funciones. En primer lugar, como era de esperarse, es evidente que el aumento de la probabilidad de mutación produce un aumento en el número de generaciones que el algoritmo necesita para converger. Además, las zonas en las que se presenta aumento del número de generaciones, concuerdan para la mayoría de funciones con las zonas en las cuales hay menor porcentaje de error, lo que respalda los resultados obtenidos con la variación del parámetro α.

Analizando cada caso individualmente, es posible darse cuenta que para las funciones unimodales y continuas como $f1$ (figura 5.1(a)) y $f2$ (figura 5.1(b)), el porcentaje de error es muy uniforme a través de todo el espacio de parámetros (ver la figura 5.3), de modo



(a) Número de generaciones para f3

(b) Porcentaje de error para f3

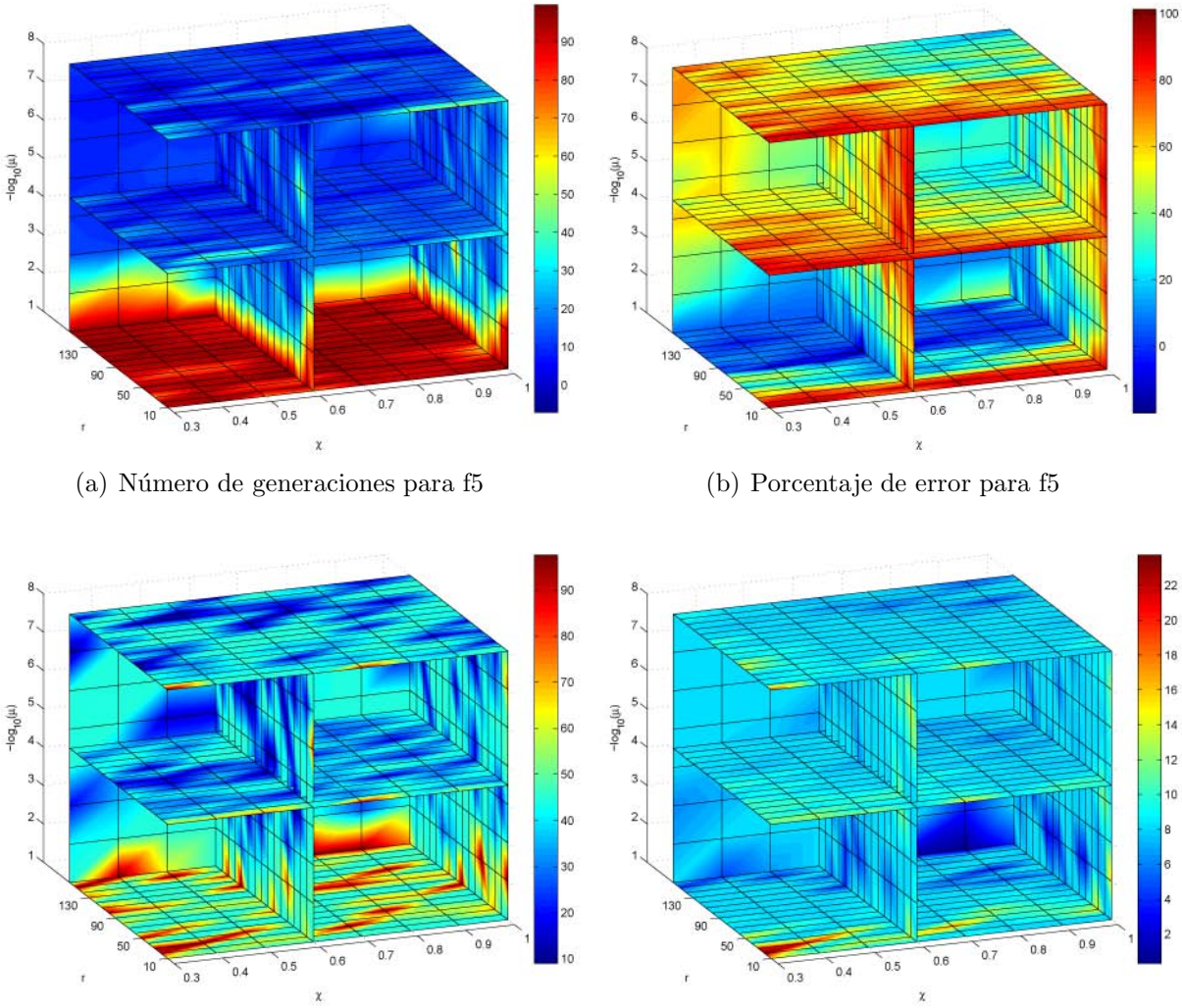
(c) Número de generaciones para f4

(d) Porcentaje de error para f4

Figura 5.4: Modelos obtenidos con las SVR para f3 y f4

que la optimización puede ser reducida a la minimización del tiempo de convergencia. Los parámetros óptimos se encuentran para probabilidades de mutación inferiores al 1 % y poblaciones entre 50 y 100 individuos, mientras la probabilidad de cruce no tiene una mayor relevancia.

Para las funciones con superficies planas, que no proporcionan información sobre las direcciones favorables, como es el caso de $f3$, la probabilidad de cruce cobra especial importancia. Si se observan las figuras 5.4(a) y 5.4(b), se puede comprobar que probabilidades de cruce bajas hacen que el algoritmo converja rápidamente pero a la vez coinciden con las zonas que presentan errores más altos. Lo que se necesita es entonces aumentar la capacidad de exploración del algoritmo, por medio de probabilidades de cruce altas y poblaciones grandes, pero teniendo un control sobre qué tanto error se permitirá a la solución final. Probabilidades de mutación cercanas a 0,6 como se propone en [15] permiten errores cercanos al 5 ó 10 % mientras una probabilidad de cruce de 0,95 como la propuesta en [24]



(a) Número de generaciones para f5

(b) Porcentaje de error para f5

(c) Número de generaciones para f6

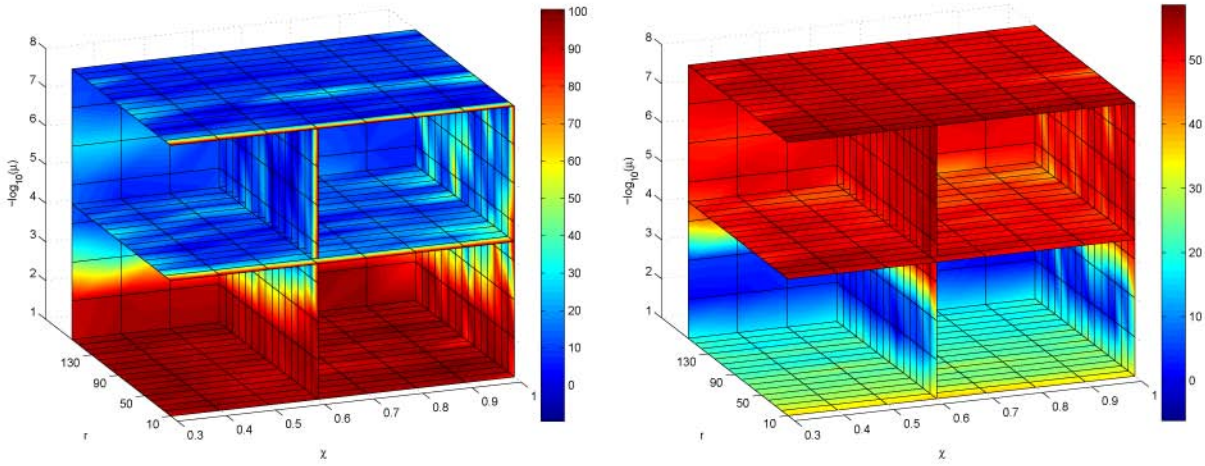
(d) Porcentaje de error para f6

Figura 5.5: Modelos obtenidos con las SVR para f5 y f6

disminuye los errores a menos del 5 %. La metodología propuesta en el presente trabajo, permite un control directo sobre este error.

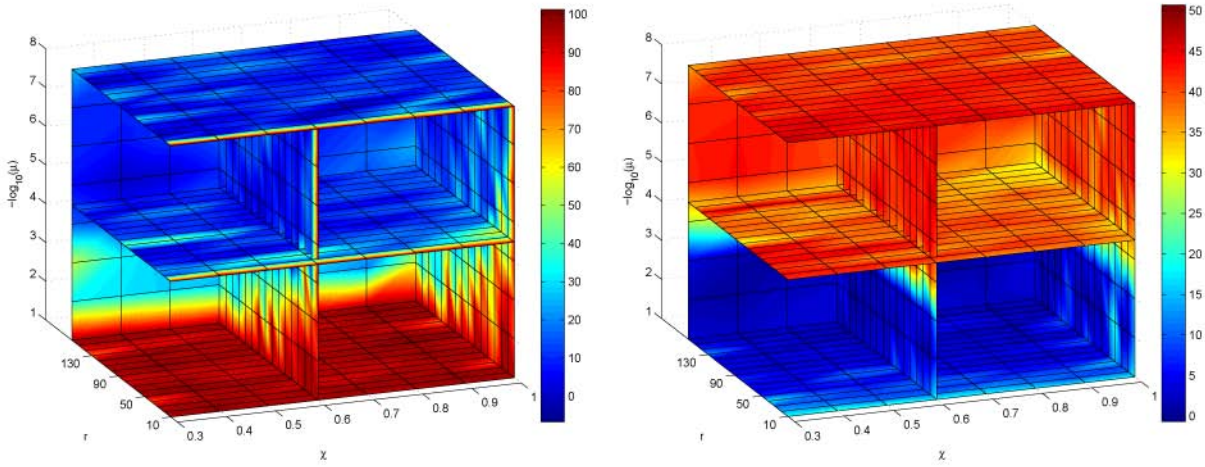
En el caso de funciones con componentes aleatorias, como $f4$, se cumplen los mismos lineamientos generales, pero existe también una aleatoriedad en las zonas que presentan mejores características (ver figuras 5.4(c) y 5.4(d)). El mismo comportamiento es observable en funciones altamente multimodales como $f6$ (ver figuras 5.5(c) y 5.5(d)), donde el desempeño del algoritmo dependerá fuertemente de la ubicación de las soluciones iniciales. Consecuentemente, en estos dos casos se observan las mayores desviaciones estándar como se muestra en la tabla 5.2.

El caso de $f5$ es donde se hace más evidente la necesidad de poblaciones grandes cuando el espacio de búsqueda es multimodal. De la figuras 5.5(b) es claro que el algoritmo necesita contar con una gran capacidad exploratoria para encontrar el verdadero máximo de la función (probabilidades de cruce altas y poblaciones grandes), mientras que la figura



(a) Número de generaciones para f7

(b) Porcentaje de error para f7



(c) Número de generaciones para f8

(d) Porcentaje de error para f8

Figura 5.6: Modelos obtenidos con las SVR para f7 y f8

5.5(a) muestra que para obtener un número de generaciones bajo, vasta con reducir la probabilidad de mutación a menos del 1 %, lo cual no disminuye su capacidad de exploración.

Respecto a las funciones $f7$ y $f8$, muestran un comportamiento muy similar, reflejando su complejidad en la alta presencia del color rojo (alto número de generaciones y alto porcentaje de error) en las gráficas de la figura 5.6. De nuevo se evidencia la correlación entre los porcentajes de error bajos con el número de iteraciones elevado. Es claro que se necesitan probabilidades de mutación altas para encontrar los máximos verdaderos, pero esta necesidad es más evidente en $f8$ que en $f7$.

Capítulo 6

Discusión y conclusiones

Se desarrolló una metodología para la optimización de los parámetros de control de un algoritmo genético, enfocada a la reducción del número de generaciones empleado para encontrar una solución. Para esto, su usó un meta-algoritmo, una técnica consistente en la sintonización de los parámetros del algoritmo genético por medio de otro algoritmo genético denominado "de alto nivel", cuyos parámetros de entrada son los parámetros de control del primero. Teniendo en cuenta el costo computacional que representa el empleo de esta técnica, se incorporó una superficie de regresión por vectores de soporte para modelar el desempeño del algoritmo genético de bajo nivel, a partir de un conjunto de entrenamiento, lo cual representa un costo computacional mucho menor.

Teniendo en cuenta que el principal objetivo de este trabajo es la automatización de las tareas de sintonización, se buscó que todos los elementos que componen la metodología cuenten con un sustento suficiente para la sintonización de sus propios parámetros, de tal manera que el usuario no esté en la necesidad de ajustar ningún valor adicional para conseguir el mejor rendimiento.
Así, para la elección del modelo de la superficie de regresión por vectores de soporte, se escogieron técnicas analíticas desarrolladas recientemente, de forma que todos los parámetros son escogidos de manera automática. En cuanto al algoritmo genético de alto nivel, se escogió un modelo que incorpora la creación de nichos para aumentar la exploración sobre el espacio de búsqueda. El método utilizado para la definición de los nichos es uno de los principales aportes de este trabajo, ya que es una mejora de otro método existente en la literatura, pero elimina la necesidad de parámetros qué sintonizar. Asimismo, se propone una novedosa técnica de apareamiento restringido basado en distancias fenotípicas, que ayuda a la exploración del espacio de búsqueda y a su vez elimina la necesidad de sintonizar la probabilidad de recombinación. Además, dada la naturaleza del modelo, el operador de mutación no es utilizado, hecho que suprime otro parámetro de control.
La optimización de los parámetros de control del algoritmo genético de bajo nivel, es planteada como un problema de optimización sujeto a restricciones, para lo cual se plantea una nueva forma de penalización estática que no requiere el ajuste de coeficientes de penalización, lo cual es otro aporte significativo. Sin embargo, esta misma función incorpora un parámetro que permite al usuario tener un control sobre el porcentaje de error que le será permitido al algoritmo genético, a cambio de la reducción en el tiempo de cómputo.

Esto representa un dominio por parte del usuario, que no proporcionan la mayoría de técnicas para la sintonización de los parámetros de control.

Los modelos obtenidos a través de las SVR permiten arrojar una mayor comprensión de la interacción entre los parámetros de control y sobre todo proporcionan la certidumbre de que para cada espacio de búsqueda existen diferentes implicaciones en el desempeño del GA, por lo cual es necesario ajustar los parámetros de control en cada caso particular en lugar de utilizar un mismo conjunto para todos los problemas de optimización. Se observó que es posible distinguir comportamientos comunes a todas las funciones, como el hecho de que el aumento de la probabilidad de mutación produce un aumento en el número de generaciones y que las zonas en las que se presenta mayor número de generaciones, concuerdan con las zonas en las cuales hay menor porcentaje de error. Sin embargo, hay rasgos específicos que dependen de la naturaleza del espacio de búsqueda, por ejemplo, para las funciones unimodales y continuas el porcentaje de error es muy uniforme a través de todo el espacio de parámetros de modo que la optimización de los parámetros de control puede ser entendida simplemente como la minimización del tiempo de convergencia, mientras para las funciones altamente multimodales se hace más evidente la necesidad de poblaciones grandes dado que el algoritmo necesita contar con una gran capacidad exploratoria para encontrar el verdadero máximo de la función (probabilidades de cruce altas y poblaciones grandes).

Queda abierta la investigación sobre el tamaño de la población para el algoritmo genético de alto nivel. Por la naturaleza del algoritmo, debe ser un valor suficientemente alto para reflejar la topología del espacio de búsqueda, pero hace falta un posterior análisis para determinar cotas sobre este valor.
Además, es necesario realizar pruebas para determinar la eficiencia de las técnicas de nichos y apareamiento restringido propuestas en este trabajo, por medio de un mayor análisis y fuera del ambiente de esta metodología.

Parte IV

Apéndices

Apéndice A

Método de búsqueda Fibonacci

El algoritmo de búsqueda Fibonacci es un procedimiento de búsqueda en línea para la minimización de funciones estrictamente cuasiconvexas $f(.)$, en un intervalo cerrado. Este algoritmo realiza dos evaluaciones de la función en la primera iteración y una sola evaluación en las iteraciones siguientes.
El procedimiento está basado en la secuencia de Fibonacci $\{F_i\}$, definida de la siguiente forma:

$$\begin{array}{l} F_{i+1} = F_i + F_{i-1} \quad i = 1, 2, ... \\ F_0 = F_1 = 1 \end{array} \tag{A.1}$$

En la iteración k del algoritmo, supóngase que el intervalo de búsqueda ha sido restringido a $[a_k,\ b_k]$. Considérense a continuación los dos puntos λ_k y μ_k descritos por:

$$\lambda_k = a_k + \frac{F_{n-k-1}}{F_{n-k+1}}(b_k - a_k) k = 1, 2, ..., n-1 \tag{A.2}$$

$$\mu_k = a_k + \frac{F_{n-k}}{F_{n-k+1}}(b_k - a_k) k = 1, 2, ..., n-1 \tag{A.3}$$

donde n es el número total de evaluaciones planeadas.
Teniendo en cuenta que $f(.)$ es cuasiconvexa, se cumple que el intervalo de búsqueda para la siguiente iteración, puede restringirse a [7]:

$$[a_{k+1},\ b_{k+1}] = \begin{cases} [\lambda_k,\ b_k] & \text{si } f(\lambda_k) > f(\mu_k) \\ [a_k,\ \mu_k] & \text{si } f(\lambda_k) \leq f(\mu_k) \end{cases} \tag{A.4}$$

En el primer caso, teniendo en cuenta la ecuación (A.2), y haciendo $i = n - k$, se obtiene:

$$\begin{aligned} b_{k+1} - a_{k+1} &= b_k - \lambda_k \\ &= b_k - a_k - \frac{F_{n-k-1}}{F_{n-k+1}}(b_k - a_k) \\ &= \frac{F_{n-k}}{F_{n-k+1}}(b_k - a_k) \end{aligned} \tag{A.5}$$

En el segundo caso, considerando (A.3), se obtiene:

$$\begin{aligned} b_{k+1} - a_{k+1} &= \mu_k - a_k \\ &= \frac{F_{n-k}}{F_{n-k+1}}(b_k - a_k) \end{aligned} \tag{A.6}$$

Por lo tanto, en cualquier caso el intervalo de búsqueda es reducido por un factor F_{n-k}/F_{n-k+1}. Ahora se mostrará que en la iteración $k+1$, se cumple que $\lambda_{k+1} = \mu_k$ o $\mu_{k+1} = \lambda_k$, de forma que sólo es necesario hacer una evaluación de la función. Supóngase que se cumple la condición superior de (A.4), entonces, aplicando (A.2) y reemplazando k por $k+1$, se tiene:

$$\begin{aligned}\lambda_{k+1} &= a_{k+1} + \tfrac{F_{n-k-2}}{F_{n-k}}(b_{k+1} - a_{k+1}) \\ &= \lambda_k + \tfrac{F_{n-k-2}}{F_{n-k}}(b_k - \lambda_k)\end{aligned}$$

Sustituyendo λ_k por (A.2), se tiene:

$$\lambda_{k+1} = a_k + \frac{F_{n-k-1}}{F_{n-k+1}}(b_k - a_k) + \frac{F_{n-k-2}}{F_{n-k}}\left(1 - \frac{F_{n-k-1}}{F_{n-k+1}}\right)(b_k - a_k)$$

Haciendo $i = n-k$ en (A.1) y reemplazando en la ecuación anterior, se tiene:

$$\lambda_{k+1} = a_k + \left(\frac{F_{n-k-1} + F_{n-k-2}}{F_{n-k+1}}\right)(b_k - a_k)$$

Finalmente, haciendo $i = n-k-1$ en (A.1) y teniendo en cuenta (A.3), se llega a:

$$\lambda_{k+1} = a_k + \frac{F_{n-k}}{F_{n-k+1}}(b_k - a_k) = \mu_k$$

Similarmente, se puede demostrar que cuando se cumple la segunda condición de (A.4), $\mu_{k+1} = \lambda_k$. Por lo tanto, en cualquier caso, sólo es necesaria una evaluación de la función en la iteración $k+1$.
Para resumir, en la primera iteración se hacen dos evaluaciones de la función, y en cada iteración subsecuente sólo se necesita una evaluación. Por lo tanto, al final de la iteración $n-2$, se han hecho $n-1$ evaluaciones de la función. Aún más, en la iteración $n-1$, se obtiene a partir de las ecuaciones (A.2) y (A.3) que $\lambda_{n-1} = \mu_{n-1} = \frac{1}{2}(a_{n-1} + b_{n-1})$. Dado que $\lambda_{n-1} = \mu_{n-2}$ ó $\mu_{n-1} = \lambda_{n-2}$, teóricamente no es necesario hacer más evaluaciones en este punto. Sin embargo, para reducir aún más el intervalo de búsqueda, la última observación es puesta levemente a la izquierda o a la derecha del punto medio $\lambda_{n-1} = \mu_{n-1}$, de forma que la longitud del intervalo de búsqueda final $[a_n,\ b_n]$ es $\frac{1}{2}(b_{n-1} - a_{n-1})$.

Por último, es necesario aclarar cómo definir el numero de iteraciones n. De las ecuaciones (A.5) y (A.6), la longitud del intervalo de búsqueda es reducida en la iteración k por un factor F_{n-k}/F_{n-k+1}. Por lo tanto, al final de la iteración $n-1$, la longitud del intervalo de búsqueda será:

$$b_n - a_n = \frac{b_1 - a_1}{F_n} \tag{A.7}$$

Por lo tanto, n debe ser escogido de forma que $(b_1 - a_1)/F_n$ refleje la exactitud requerida en la búsqueda.

Apéndice B

Algoritmo de entrenamiento SVM^{light}

Como se planteó en la sección 3.1, en [70], se muestra que el problema de entrenar una máquina de vectores de soporte para clasificación, lleva al siguiente problema de optimización cuadrática:

$$\begin{aligned} &maximizar && \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j}^{N} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \\ &sujeto\, a && \sum_{i=1}^{N} y_i \alpha_i = 0 \; y \; \alpha_i \geq 0, \forall i \end{aligned} \tag{B.1}$$

El número de muestras de entrenamiento está denotado por N, α es un vector de N variables donde cada α_i está ligado a una muestra (x_i, y_i) y la solución a (B.1) es el vector α^0 que minimiza la función de Lagrange, cumpliendo con las restricciones. Definiendo la matriz Q como $(Q)_{ij} = y_i y_j k x_i, x_j$, (B.1) puede reescribirse como:

$$\begin{aligned} &maximizar && W(\alpha) = \alpha^T 1 - \frac{1}{2} \alpha^T Q \alpha \\ &sujeto\, a && \alpha^T y = 0 \; y \; 0 \leq \alpha \leq C1 \end{aligned} \tag{B.2}$$

El tamaño del problema de optimización depende del número de muestras de entrenamiento N. Dado que el tamaño de la matriz Q es N^2, para tareas con más de 10000 muestras de entrenamiento, se vuelve imposible almacenar Q en memoria [34], e incluso si este no es el caso, el tiempo de cómputo crece rápidamente con el número de muestras. Una forma de resolver esto, consiste en descomponer el problema de optimización en tareas de menor tamaño, para lo cual SVM^{light} utiliza la descomposición formulada en [53]. En cada iteración, las variables α_i en (B.1) son separadas en dos categorías:

- El conjunto B de variables libres
- El conjunto M de variables fijas

Las variables libres son aquellas que pueden ser actualizadas en la iteración actual, mientras que las variables fijas son aquellas que están temporalmente en un valor dado. El conjunto de variables libres, también llamado conjunto de trabajo, tiene un tamaño constante q mucho menor que N. El algoritmo funciona como sigue:

`Mientras que las condiciones de optimalidad sean violadas`

`-- Seleccione` q `variables del conjunto de trabajo` B`. Las` $N-q$ `variables restantes, quedan fijas en el valor actual.`

`-- Descomponga el problema y resuelva la tarea de optimización: Maximice` $W(\alpha)$ `en` B`.`

`Termine y retorne` α

El valor óptimo de α se encuentra teniendo en cuenta que (B.1) siempre tiene una hessiana semidefinida positiva Q y todas las restricciones son lineales, por lo tanto (B.1) es un problema de optimización convexo. Para este tipo de problemas, las condiciones de Karush-Kuhn-Tucker (3.6) y (3.7) son necesarias y suficientes para garantizar la optimalidad.
Si las condiciones de optimalidad no se cumplen, el algoritmo descompone el problema separando las variables en un conjunto de trabajo B y un conjunto de variables dadas M. Supóngase que se tienen α, y y Q adecuadamente organizadas respecto a los conjuntos B y M:

$$\alpha = \begin{vmatrix} \alpha_B \\ \alpha_M \end{vmatrix} \quad y = \begin{vmatrix} y_B \\ y_M \end{vmatrix} \quad Q = \begin{vmatrix} Q_{BB} & Q_{BM} \\ Q_{MB} & Q_{MM} \end{vmatrix} \tag{B.3}$$

Dado que Q es simétrica (y en particular $Q_{BM} = Q_{MB}^T$), se puede escribir:

$$\begin{aligned} &\textit{maximizar} \quad W(\alpha) = \alpha_B^T(1 - Q_{BM}\alpha_M) - \tfrac{1}{2}\alpha_B^T Q_{BB}\alpha_B - \tfrac{1}{2}\alpha_M^T Q_{MM}\alpha_M + \alpha_M^T 1 \\ &\textit{sujeto a} \qquad \alpha_B^T y_B + \alpha_M^T y_M = 0 \, y \, 0 \leq \alpha \leq C1 \end{aligned} \tag{B.4}$$

Dado que las variables en M están fijas, los términos $-\frac{1}{2}\alpha_M^T Q_{MM}\alpha_M$ y $\alpha_M^T 1$ son constantes y pueden ser omitidos sin alterar la solución de (B.4). Éste es un problema semidefinido positivo de optimización cuadrática, lo suficientemente pequeño para ser resuelto rápidamente por la mayoría de métodos más comunes.

Bibliografía

[1] J. Alander. Indexed bibliography of genetic algorithm implementations, 1999.

[2] J. Alander. Indexed bibliography of genetic algorithms papers available via ftp and www, 1999.

[3] J. Arabas, Z. Michalewicz, and J. Mulawka. Gavaps - a genetic algorithm with varying population size. In New York: IEEE Press, editor, *1997 IEEE Interational Conference on Evolutionary Computation*, pages 7–12, 1997.

[4] Thomas Back. Optimization by means of genetic algorithms. In E. Köhler, editor, *36th International Scientific Colloquium*, pages 163–169, Technical University of Ilmenau, 1991.

[5] Thomas Bäck. Optimal mutation rates in genetic search. In *ICGA*, pages 2–8, 1993.

[6] James E. Baker. Adaptive selection methods for genetic algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 101–111, Mahwah, NJ, USA, 1985. Lawrence Erlbaum Associates, Inc.

[7] Mokhtar Bazaraa, Hanif Sherali, and C. M. Shetty. *Nonlinear Programming, Theory and Algorithms*. John Wiley and Sons, Inc, 1993.

[8] Mark F. Bramlette. Initialization, mutation and selection methods in genetic algorithms for function optimization. In *ICGA*, pages 100–107, 1991.

[9] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. In *Knowledge Discovery and Data Mining*, volume 2, pages 121–167. 1998.

[10] Ming-Wei Chang and Chih-Jen Lin. Leave-one-out bounds for support vector regression model selection. *Neural Computation*, 17(5):1188–1222, 2005.

[11] Vladimir Cherkassky and Yunqian Ma. Practical selection of SVM parameters and noise estimation for SVM regression. *Neural Networks*, 17(1):113–126, January 2004.

[12] Carlos A. Coello Coello. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering*, 191(11–12):1245–1287, 2002.

[13] Paul Darwen and Xin Yao. How good is fitness sharing with a scaling function. Technical Report CS 8/95, University of New South Wales, 1995.

[14] K. Deb. An efficient constraint handling method for genetic algorithms, 1998.

[15] Kenneth. A. DeJong. *An analysis of the Behavior of a class of Genetic Adaptative System*. PhD thesis, University of Michigan, 1975.

[16] Jason Digalakis and Konstantinos Margaritis. An experimental study of benchmarking functions for evolutionary algorithms. *International Journal of Computer Mathemathics*, 79(4):403–416, April 2002.

[17] Ágoston Endre Eiben, Robert Hinterding, and Zbigniew Michalewicz. Parameter control in evolutionary algorithms. *IEEE Trans. on Evolutionary Computation*, 3(2):124–141, 1999.

[18] Terence C. Fogarty. Varying the probability of mutation in the genetic algorithm. In *ICGA*, pages 104–109, 1989.

[19] Stephanie Forrest and Terry Jones. Modeling complex adaptive systems with echo, 1994.

[20] Stephanie Forrest and Melanie Mitchell. Relative building-block fitness and the building-block hypothesis. In L. Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 109–126. Morgan Kaufmann, San Mateo, CA, 1993.

[21] D. E. Goldberg, K. Deb, and J. H. Clark. Genetic algorithms, noise and the sizing of populations. *Complex Systems*, 6:333–362, 1992.

[22] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1989.

[23] David E. Goldberg and Jon Richardson. Genetic algorithms with sharing for multimodal function optimization. In John J. Grefenstette, editor, *Genetic algorithms and their applications: Proc. of the Second Int. Conf. on Genetic Algorithms*, pages 41–49, Hillsdale, NJ, 1987. Lawrence Erlbaum Assoc.

[24] J. J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE transactions on Systems, Man, and Cybernetics*, 16(1):122–128, 1986.

[25] Peter J.B. Hancock. An empirical comparison of selection methods in evolutionary algorithms. In *Proceedings of the ASIB workshop on Evolutionary Computation*, 1994.

[26] Harik, Cantu-Paz, Goldberg, and Miller. The gambler's ruin problem, genetic algorithms, and the sizing of populations. In *IEEECEP: Proceedings of The IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, 1997.

[27] Georges R. Harik and Fernando G. Lobo. A parameter-less genetic algorithm. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 258–265, Orlando, Florida, USA, 13-17 1999. Morgan Kaufmann.

[28] R. Herbrich, T. Graepel, and K. Obermayer. Support vector learning for ordinal regression.

[29] Frank Hoffmeister and Joachim Sprave. Problem-independent handling of constraints by use of metric penalty functions. In *Evolutionary Programming*, pages 289–294, 1996.

[30] John H. Holland. *Adaptation in Natural and Artifical Systems*. University of Michigan Press, Ann Arbor, MI, 1975.

[31] John H. Holland. Echoing emergence: objectives, rough definitions, and speculations for echo-class models. pages 309–342, 1994.

[32] A. Homaifar, C. Qi, and S. Lai. Constrainted optimization via genetic algorithms. *Simulation*, 62(4):242–253, April 1994.

[33] P. Hraber, T. Jones, and S. Forrest. The ecology of echo, 1997.

[34] T. Joachims. Making large-scale support vector machine learning practical. In *Advances in Kernel Methods: Support Vector Machines*.

[35] Jeffrey A. Joines and Christopher R. Houck. On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GA's. In *International Conference on Evolutionary Computation*, pages 579–584, 1994.

[36] Spiridon A. Kazarlis and Vassilios Petridis. Varying fitness functions in genetic algorithms: Studying the rate of increase of the dynamic penalty terms. In A. E. Eiben, Thomas Bäck, Marc Schoenauer, and Hans-Paul Schwefel, editors, *Parallel Problem Solving from Nature - PPSN V, 5th International Conference, Amsterdam, The Netherlands, September 27-30, 1998, Proceedings*, volume 1498 of *Lecture Notes in Computer Science*, pages 211–220. Springer, 1998.

[37] James T. Kwok. Linear dependency between epsilon and the input noise in epsilon-support vector regression. In *ICANN '01: Proceedings of the International Conference on Artificial Neural Networks*, pages 405–410, London, UK, 2001. Springer-Verlag.

[38] Y. Li, , S. Gong, J. Sherrah, and H. Liddell. Multi-view face detection using support vector machines and eigenspace modelling, 2000.

[39] Chyi-Yeu Lin and Wen-Hong Wu. Niche identification techniques in multimodal genetic search with sharing scheme. *Advances in Engineering Software*, 33(11-12):779–791, November-December 2002.

[40] Jingxu Liu and Yuejin Tan. Estimating the leave-one-out error for support vector regression. In *International Conference on Neural Networks and Brain*, volume 1, pages 208–213, 2005.

[41] Samir W. Mahfoud. *Niching methods for genetic algorithms*. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL, USA, May 1995. IlliGAL Report 95001.

[42] Davide Mattera, Francesco Palmieri, and Simon Haykin. Generalized support vector machines. In *ESANN*, pages 213–218, 1999.

[43] Zbigniew Michalewicz and Naguib Attia. Evolutionary optimization of constrained problems. In *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, pages 98–108, 1994.

[44] Melanie Mitchell. *Genetic Algorithms in Problem Solving*, chapter 2, pages 35–81. Massachusetts Institute of Technology, 1996.

[45] Melanie Mitchell. *Genetic Algorithms in Scientific Models*, chapter 3, pages 85–113. Massachusetts Institute of Technology, 1996.

[46] Melanie Mitchell. *An Introduction to Genetic Algorithms*. Massachusetts Institute of Technology, 1996.

[47] Melanie Mitchell, Stephanie Forrest, and John H. Holland. The royal road for genetic algorithms: Fitness landscapes and GA performance. In Francisco J. Varela and Paul Bourgine, editors, *Towards a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life, 1991*, pages 245–254, Paris, 11–13 1992. A Bradford book, The MIT Press.

[48] Melanie Mitchell, John H. Holland, and Stephanie Forrest. When will a genetic algorithm outperform hill climbing. In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 51–58. Morgan Kaufmann Publishers, Inc., 1994.

[49] A. Kuri Morales and C.V. Quezada. A universal eclectic genetic algorithm for constrained optimization. In *Proceedings of the 6th European Congress on Intelligent Techniques and Soft Computing, EUFIT'98*, pages 518–522, september 1998.

[50] Heinz Mühlenbein. How genetic algorithms really work: Mutation and hillclimbing. In *PPSN*, pages 15–26, 1992.

[51] K. Muller, A. Smola, G. atsch, B. Sch, o Kohlmorgen, and V. Vapnik. Predicting time series with support vector machines, 1997.

[52] Mauricio Orozco. Optimización de los parámetros de control del algoritmo genético simple usando regresión de vectores soporte. In *IX simposio sobre tratamiento de señales, imágenes y visión artificial*. Universidad Nacional de Colombia, Universidad Autónoma de Manizales, Universidad de Manizales, 2004.

[53] E. Osuna, R. Freund, and F. Girosi. Improved training algorithm for support vector machines, 1997.

[54] Ahmet Irfan Oyman, Kalyanmoy Deb, and Hans-Georg Beyer. An alternative constraint handling method for evolution strategies. In *1999 Congress on Evolutionary Computation*, pages 612–619, Piscataway, NJ, 1999. IEEE Service Center.

[55] Jan Paredis. Co-evolutionary constraint satisfaction. In *PPSN*, pages 46–55, 1994.

[56] David Powell and Michael M. Skolnick. Using genetic algorithms in engineering design optimization with non-linear constraints. In *ICGA*, pages 424–431, 1993.

[57] I Rojas, J. González, H. Pomares, J. J. Merelo, P. A. Castillo, and G. Romero. Statistical analysis of the main parameters involved in the design of a genetic algorithm. *IEEE Trans. Syst., Man, Cybern. C*, 32(1):31–37, 2002.

[58] J. David Schaffer, Rich Caruana, Larry J. Eshelman, and Rajarshi Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In *ICGA*, pages 51–60, 1989.

[59] Bernhard Schölkopf and Alex Smola. Kernel methods and support vector machines, 2003.

[60] B. Schoelkopf and A. J. Smola. *Learning with Kernels*. The MIT Press, Cambridge, MA, 2002.

[61] B. Schölkopf, K.-K. Sung, C. J. C. Burges, F. Girosi, P.Ñiyogi, T. Poggio, and V. Vapnik. Comparing support vector machines with Gaussian kernels to radial basis function classiers. *IEEE Transactions on Signal Processing*, 45(11):2758–2765, 1997.

[62] S. Skalak, R. Shonkwiler, S. Babar, and M. Aral. Annealing a genetic algorithm over constraints. In *IEEE International Conference on Systems, Man, and Cybernetics*, volume 4, pages 3931–3936, 1998.

[63] Alice E. Smith and David W. Coit. Penalty functions. In Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors, *Handbook of Evolutionary Computation*, pages C5.2:1–6. Institute of Physics Publishing and Oxford University Press, Bristol, New York, 1997.

[64] Alice E. Smith and David M. Tate. Genetic optimization using A penalty function. In Stephanie Forrest, editor, *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, pages 499–505, San Mateo, CA, 1993. Morgan Kaufmann.

[65] Stephen F. Smith and Vincent A. Cicirello. Modeling GA performance for control parameter optimization. In Darrell Whitley, David Goldberg, Erick Cantu-Paz, Lee Spector, Ian Parmee, and Hans-Georg Beyer, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pages 235–242, Las Vegas, Nevada, USA, 10-12 2000. Morgan Kaufmann.

[66] A.J. Smola and B. Schoelkopf. A tutorial on support vector regression, 1998.

[67] William M. Spears. Adapting crossover in evolutionary algorithms. In J. R. McDonnell, R. G. Reynolds, and D. B. Fogel, editors, *Proc. of the Fourth Annual Conference on Evolutionary Programming*, pages 367–384, Cambridge, MA, 1995. MIT Press.

[68] B. Üstün, W. J. Melssen, M. Oudenhuijzen, and L. M. C. Buydens. Determination of optimal support vector regression parameters by genetic algorithms and simplex optimization. In *Analytica Chimica Acta*, volume 544, pages 292–305, 2005.

[69] Streichert, Stein, Ulmer, and Zell. A clustering based niching EA for multimodal search spaces. In *International Conference on Artificial Evolution, Evolution Artificielle, LNCS*, volume 6, 2003.

[70] Vladimir Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998.

[71] Wenjian Wang, ZongBen Xu, Weizhen Lu, and Xiaoyun Zhang. Determination of the spread parameter in the gaussian kernel for classification and regression. *Neurocomputing*, 55(3-4):643–663, 2003.

[72] Tony White and Franz Oppacher. Adaptive crossover using automata. In *PPSN*, pages 229–238, 1994.

[73] Darrell Whitley. An overview of evolutionary algorithms: practical issues and common pitfalls. *Information and Software Technology*, 43(14):817–831, 2001.

[74] S. J. Wu and P. T. Chow. Genetic algorithms for nonlinear mixed discrete-integer optimization problems via meta-genetic parameter optimization. *Engeenering Optimization*, 24(2):137–159, 1995.

[75] Xiaodong Yin and Noël Germay. Improving genetic algorithms with sharing through cluster analysis. In Stephanie Forrest, editor, *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, page 100, San Mateo, CA, 1993. Morgan Kaufmann.