



Universidad de Murcia
Departamento de Ingeniería de la Información y las Comunicaciones

**DISEÑO Y EVALUACIÓN DE
ALGORITMOS EVOLUTIVOS MULTI OBJETIVO
EN OPTIMIZACIÓN Y MODELIZACIÓN DIFUSA**

Gracia Sánchez Carpena

2002

DISEÑO Y EVALUACIÓN DE ALGORITMOS EVOLUTIVOS MULTIOBJETIVO EN OPTIMIZACIÓN Y MODELIZACIÓN DIFUSA

Tesis Doctoral

Presentada por:

Gracia Sánchez Carpena

Dirigida por:

Dr. Antonio F. Gómez Skarmeta

Dr. Fernando Jiménez Barrionuevo

Noviembre 2002

DEPARTAMENTO DE INGENIERÍA DE LA INFORMACIÓN Y LAS COMUNICACIONES
FACULTAD DE INFORMÁTICA
UNIVERSIDAD DE MURCIA

La memoria titulada

**Diseño y Evaluación de
Algoritmos Evolutivos Multiobjetivo
en Optimización y Modelización Difusa**

que presenta Gracia Sánchez Carpena para optar al grado de **Doctor en Informática**, ha sido realizada en el Departamento de Ingeniería de la Información y las Comunicaciones de la Universidad de Murcia, bajo la dirección del Dr. Antonio F. Gómez Skarmeta, Profesor Titular de Universidad y el Dr. Fernando Jiménez Barrionuevo, Profesor Titular de Universidad, ambos del Departamento en el que se ha realizado la tesis.

Murcia, Noviembre de 2002

Fdo: Gracia Sánchez Carpena

Fdo: Dr. Antonio F. Gómez Skarmeta

Fdo: Dr. Fernando Jiménez Barrionuevo

Agradecimientos

Mi más sincero agradecimiento a quienes, de una forma u otra, han hecho posible la realización de este trabajo. En primer lugar a mis directores *Antonio F. Gómez Skarmeta* y *Fernando Jiménez Barrionuevo*, cuya ayuda ha sido completamente imprescindible para el desarrollo de esta Tesis Doctoral.

En general, mi agradecimiento a todo el *Departamento de Ingeniería de la Información y las Comunicaciones de la Universidad de Murcia*, y en particular a aquellos más cercanos que han sabido soportarme en el día a día y cuyas aportaciones y consejos, tanto en el campo científico como humano, han resultado de gran valía.

También agradecer a aquellas personas ajenas a la universidad de Murcia, *Hans Roubos* y *Robert Babuška* del *Control Engineering Laboratory, Delft University of Technology, the Netherlands* y también a *Carlos Coello* y a *Kalyanmoy Deb* por el interés y la colaboración prestada en la realización de este trabajo.

Mi agradecimiento especial a mi esposo *Cipriano Ángel* y a *mis padres*, quienes con su cariño y apoyo en todo momento han hecho posible que esta memoria vea finalmente la luz. Por último, pero no menos importante, no puedo olvidar a mi hija *María del Pilar*, quien me hace ver cada día que la vida existe más allá del trabajo.

A mi esposo Cipriano Ángel
A mi hija María del Pilar
A mis padres

**DISEÑO Y EVALUACIÓN DE
ALGORITMOS EVOLUTIVOS MULTIOBJETIVO
EN OPTIMIZACIÓN Y MODELIZACIÓN DIFUSA**

Gracia Sánchez Carpena

Resumen

Los algoritmos evolutivos son técnicas heurísticas basadas en los procesos naturales de reproducción y selección de los individuos mejor adaptados que por su robustez y versatilidad han sido utilizadas para resolver un amplio abanico de problemas. Una de las aplicaciones de los algoritmos evolutivos de mayor actualidad se encuentra en el campo de la optimización multiobjetivo, puesto que, por sus características, los algoritmos evolutivos resultan especialmente adecuados para resolver este tipo de problemas. Otro campo de gran interés es la obtención de modelos difusos. Los algoritmos evolutivos han sido utilizados con éxito para realizar modelos difusos, en muchas ocasiones junto con otras técnicas de optimización numéricas, dando lugar a algoritmos híbridos que obtienen modelos aproximativos, muy precisos, pero poco interpretables. Utilizando algoritmos evolutivos multiobjetivo es posible tener en cuenta no sólo la precisión de los modelos, sino también otros parámetros relacionados con la interpretabilidad de los mismos, dando lugar a modelos precisos pero, al mismo tiempo, interpretables.

Esta tesis está dedicada al diseño y evaluación de algoritmos evolutivos multiobjetivo. Se propone un conjunto de algoritmos evolutivos para optimización multiobjetivo y se realiza la evaluación de los mismos mediante una serie de problemas test, mejorando los resultados obtenidos por algunos de los algoritmos evolutivos multiobjetivo más actuales. En una segunda parte de la tesis se utilizan las técnicas multiobjetivo para diseñar nuevos algoritmos evolutivos que realicen modelos difusos precisos y también interpretables, aplicando dichos algoritmos para resolver diferentes problemas.

Palabras clave: Algoritmos evolutivos, optimización multiobjetivo con restricciones, optimalidad Pareto, modelización difusa.

Indice de Contenidos

Introducción	7
1 Computación Evolutiva	11
1.1 Introducción	12
1.1.1 Programación Evolutiva	15
1.1.2 Estrategias de Evolución	16
1.1.3 Programación genética	17
1.1.4 Algoritmos Genéticos	18
1.2 Enfoque de un Algoritmo Genético	19
1.2.1 Representación	19
1.2.2 Población inicial	22
1.2.3 Función de adecuación	22
1.2.4 Operadores de variación	23
1.2.5 Parámetros	27
1.2.6 Fundamentos teóricos	28
1.3 Computación Evolutiva: Mejoras y Extensiones	31
1.3.1 Representación	32
1.3.2 Población inicial	33
1.3.3 Función de adecuación	34
1.3.4 Esquemas de selección, muestreo y sustitución generacional . .	37
1.3.5 Esquemas de apareamiento	41
1.3.6 Operadores de cruce	41

1.3.7	Operadores de mutación	45
1.3.8	Otros operadores	47
1.3.9	Técnicas basadas en el conocimiento	51
1.4	Sumario	54
2	Optimización Multiobjetivo	55
2.1	Problemas de Optimización Multiobjetivo	57
2.1.1	Descripción del problema	57
2.1.2	Concepto de óptimo Pareto	58
2.1.3	Frente Pareto	59
2.2	Optimización Multiobjetivo mediante Métodos Clásicos	59
2.2.1	Método simplex multiobjetivo	60
2.2.2	Descomposición multiparamétrica	61
2.2.3	Programación fraccional	62
2.2.4	Programación por metas	62
2.2.5	Programación de compromiso	64
2.3	Técnicas Evolutivas para Manejo de Restricciones	67
2.3.1	Funciones de penalización	67
2.3.2	Representación y operadores de variación específicos	75
2.3.3	Separación de objetivos y restricciones	78
2.3.4	Métodos híbridos	83
2.3.5	Otros métodos	85
2.3.6	Conclusiones	88
2.4	Técnicas Evolutivas para Optimización Multiobjetivo	91
2.4.1	Métodos con articulación de preferencias <i>a priori</i>	94
2.4.2	Métodos <i>a posteriori</i> no basados en el concepto Pareto	99
2.4.3	Métodos no elitistas basados en el concepto Pareto	109
2.4.4	Métodos elitistas basados en el concepto Pareto	113
2.4.5	Conclusiones	123
2.5	Técnicas Evolutivas para Optimización Multiobjetivo con Restricciones	125

2.5.1	Ignorar las soluciones no factibles	125
2.5.2	Funciones de penalización	125
2.5.3	Separar las soluciones factibles de las no factibles	126
2.5.4	Definir el problema utilizando metas y prioridades	129
2.5.5	Conclusiones	132
2.6	Sumario	133
3	Nuevos Algoritmos Evolutivos para Optimización Multiobjetivo	135
3.1	Algoritmo Evolutivo para Problemas de Optimización Multiobjetivo sin Restricciones	137
3.1.1	Estructura del algoritmo evolutivo	137
3.1.2	Representación de las soluciones	137
3.1.3	Inicialización de los individuos	137
3.1.4	Operadores de variación	138
3.1.5	Diversidad	141
3.1.6	Experimentos y resultados	143
3.2	Algoritmos Evolutivos para Problemas de Optimización Multiobjetivo con Restricciones	158
3.2.1	Algoritmo I. Algoritmo evolutivo con preselección simple . . .	159
3.2.2	Algoritmo II. Algoritmo evolutivo con métricas de diversidad .	161
3.2.3	ENORA. Algoritmo evolutivo con ordenación no dominada por tramos radiales	164
3.2.4	Problemas test	168
3.2.5	Experimentos y resultados	175
3.3	Sumario	200
4	Modelización Difusa	201
4.1	Aspectos de Modelización Difusa	202
4.1.1	Estructura del modelo difuso	202
4.1.2	Estimación de los parámetros de los consecuentes	206
4.1.3	Transparencia en modelos difusos aproximativos	207

4.1.4	Modelización neuro-difusa	209
4.2	Enfoques Evolutivos para Modelización Difusa	214
4.2.1	FuGeNeSys. Sistema neuro-genético para modelización difusa	214
4.2.2	Algoritmo evolutivo para modelización difusa con reducción de la complejidad.	218
4.2.3	Algoritmo evolutivo para modelos lingüísticos	221
4.2.4	Algoritmos evolutivos para modelos con etiquetas fijas	224
4.2.5	Algoritmo evolutivo para entrenar modelos aproximativos y descriptivos en control difuso	230
4.2.6	Optimalidad Pareto en modelización difusa	235
4.3	Sumario	238
5	Nuevos Algoritmos Evolutivos Multiobjetivo para Modelización Difusa	239
5.1	Algoritmo Evolutivo para Selección de Variables	241
5.1.1	Representación de soluciones	243
5.1.2	Población inicial	243
5.1.3	Operadores de variación	244
5.1.4	Selección y reemplazo generacional	245
5.1.5	Experimentos y resultados	246
5.1.6	Conclusiones	249
5.2	Algoritmo Evolutivo Multiobjetivo para Modelización Difusa	250
5.2.1	Identificación del modelo difuso	250
5.2.2	Criterios para la modelización difusa	251
5.2.3	Simplificación del conjunto de reglas difusas	253
5.2.4	Características del algoritmo evolutivo multiobjetivo	255
5.2.5	Representación de soluciones	256
5.2.6	Población inicial	256
5.2.7	Operadores de variación	257
5.2.8	Selección y reemplazo generacional	260

5.2.9	Modelo de optimización y proceso de decisión	261
5.2.10	Experimentos y resultados	263
5.3	Algoritmo Neuro-Evolutivo Multiobjetivo para Modelización Difusa .	272
5.3.1	Identificación del modelo difuso	272
5.3.2	Criterios para la modelización difusa	273
5.3.3	Mejora de la transparencia y la compactitud del modelo difuso	275
5.3.4	Entrenamiento de la red neuronal RBF	276
5.3.5	Representación de soluciones	277
5.3.6	Población inicial	277
5.3.7	Operadores de variación	278
5.3.8	Selección y reemplazo generacional	278
5.3.9	Modelo de optimización y proceso de decisión	279
5.3.10	Experimentos y resultados	279
5.3.11	Adaptación del algoritmo para la obtención de modelos aproxi- mativos	284
5.4	Experimentos y Resultados Adicionales	285
5.5	Sumario	298
Conclusiones y Líneas Futuras		301
A Apendice: Conceptos básicos sobre Teoría de Conjuntos Difusos		309
A.1	Conjuntos difusos	309
A.2	Definiciones básicas	310
A.3	Operaciones con conjuntos difusos	312
A.4	Funciones de pertenencia	313
A.5	Defuzzificación	316
Bibliografía		317

Introducción

Los *algoritmos evolutivos* son métodos de optimización y búsqueda basados en los principios naturales de selección y supervivencia de los individuos más aptos, (Darwin, *On the Origin of Species by Means of Natural Selection*, 1959). A finales de los años sesenta y principios de los setenta, John Holland desarrolla los algoritmos genéticos, que son el primer tipo de algoritmos evolutivos. Desde entonces, los algoritmos evolutivos se han consolidado como técnicas robustas de optimización y búsqueda aplicándose para resolver un amplio abanico de problemas, tanto en el campo científico, como en el campo de la ingeniería o de la empresa.

Cuando un problema de optimización tiene más de una función objetivo, la tarea de encontrar una o más soluciones óptimas se llama *optimización multiobjetivo*. La mayoría de problemas de optimización del mundo real tienen múltiples objetivos, lo que justifica la gran importancia de la optimización multiobjetivo dentro de la investigación actual. La diferencia fundamental entre optimización simple y optimización multiobjetivo consiste en que en optimización simple existe una sola solución óptima, mientras que en optimización multiobjetivo existen múltiples soluciones óptimas para un mismo problema, las llamadas *soluciones Pareto óptimas*, que son aquellas soluciones que no pueden ser mejoradas en uno de sus objetivos sin ser empeoradas en otro. El conjunto de estas soluciones forma el *frente Pareto*.

La resolución de problemas de optimización multiobjetivo mediante algoritmos evolutivos es uno de los campos de investigación de mayor actualidad. Por sus características, los algoritmos evolutivos resultan una aproximación muy adecuada para resolver problemas de optimización multiobjetivo, pues al trabajar con poblaciones de individuos permiten encontrar, en una sola ejecución, múltiples soluciones óptimas al problema para, posteriormente, escoger aquella solución que más nos interese. El mantener la diversidad en la población es, por tanto, uno de los problemas fundamen-

tales al aplicar algoritmos evolutivos en optimización multiobjetivo. Este problema ya surge al resolver problemas de optimización simple, pues es necesario mantener cierta diversidad dentro de la población para evitar una convergencia prematura del algoritmo. Sin embargo, en el campo de la optimización multiobjetivo, el mantener la diversidad se convierte en un objetivo fundamental, pues se desea obtener un conjunto de soluciones no sólo lo más cerca posible del frente Pareto sino, además, que se encuentren uniformemente distribuidas por dicho frente. Por tanto, es necesario mantener una población que evolucione hacia el frente Pareto, al mismo tiempo que mantenga su diversidad.

La inclusión de restricciones, por otra parte, resulta inevitable en la mayoría de problemas de optimización del mundo real, lo que ha llevado a que se desarrollen una gran variedad de técnicas para manejar restricciones. En el caso de optimización multiobjetivo, las restricciones pueden dificultar no sólo el avance hacia el frente Pareto, sino también que dicho avance se realice manteniendo la diversidad, por lo que el manejo de restricciones en optimización multiobjetivo debe realizarse teniendo en cuenta ambos parámetros, lo que resulta una complejidad añadida al problema.

La optimización multiobjetivo ofrece, por tanto, un campo de pruebas idóneo para diseñar y evaluar nuevos algoritmos evolutivos. Tras un detallado estudio del estado del arte, se han diseñado cuatro nuevos algoritmos evolutivos para optimización multiobjetivo. El primer algoritmo propuesto resuelve problemas de optimización multiobjetivo sin restricciones; es el algoritmo más sencillo y sirve como base para el resto de algoritmos diseñados. También se ha utilizado este algoritmo para resolver problemas de optimización con restricciones realizando una transformación de los mismos en problemas sin restricciones. Los otros tres algoritmos planteados a continuación se diferencian fundamentalmente del anterior en que estos sí realizan un manejo de restricciones, por lo que pueden resolver directamente problemas de optimización multiobjetivo con restricciones, sin necesidad de realizar ninguna transformación de los mismos. La diferencia entre estos algoritmos se encuentra en los métodos utilizados para mantener la diversidad. El segundo algoritmo propuesto utiliza un esquema de preselección que permite, de una forma sencilla y eficiente, mantener una cierta diversidad, por lo que se ha utilizado como base para iniciar esta investigación; sin embargo, en algunos problemas el mantener la diversidad resulta especialmente difícil y un esquema de preselección simple no es suficiente, por lo que es necesario en estos casos introducir técnicas más elaboradas para mantener la di-

versidad. En concreto, se ha experimentado con dos técnicas diferentes que son las utilizadas en el tercer y cuarto algoritmo. El tercer algoritmo emplea métricas de diversidad de forma que las soluciones se evalúan teniendo en cuenta, tanto su optimalidad Pareto, como su diversidad; y el cuarto algoritmo aplica un método *ad hoc* que consiste en particionar el espacio de búsqueda en tramos radiales, forzando a que las nuevas soluciones encontradas sean diversas. Este último algoritmo ha conseguido resultados que mejoran los obtenidos por algunos de los algoritmos evolutivos multiobjetivo más actuales. Otro aspecto importante en optimización multiobjetivo son los problemas test. Se han estudiado algunos de los problemas test más importantes diseñados en optimización multiobjetivo, utilizándolos para evaluar la efectividad de los algoritmos propuestos en términos de la bondad de las soluciones obtenidas.

La *modelización difusa* se ha convertido en un área de investigación del máximo interés, extendiéndose su aplicación a prácticamente cualquier campo del mundo real. Se han aplicado diversas técnicas para calcular modelos difusos; algunas de las aproximaciones desarrolladas con éxito utilizan algoritmos evolutivos, en muchas ocasiones realizando hibridación con otras técnicas numéricas tales como clustering, gradiente, etc. En estos casos se obtienen modelos difusos aproximativos muy precisos, pero poco interpretables. Sin embargo, resulta muy interesante obtener modelos que, además de ser precisos, sean interpretables; es decir, tenemos un problema con múltiples objetivos y para resolverlo se propone aplicar las técnicas desarrolladas en la primera parte de esta tesis para diseñar nuevos algoritmos que sean capaces de obtener modelos difusos precisos y al mismo tiempo interpretables. En esta parte de la tesis se han desarrollado tres nuevos algoritmos evolutivos: un primer algoritmo realiza la selección de variables, lo cual es un primer paso dentro del proceso de modelización difusa; los dos algoritmos que se presentan a continuación efectúan el cálculo de los parámetros para modelos difusos del tipo Takagi-Sugeno con consecuentes lineales, diferenciándose en el tipo de conjuntos difusos utilizados en el antecedente, puesto que el primero utiliza conjuntos trapezoidales y el segundo utiliza conjuntos gaussianos asimétricos. En este segundo algoritmo, además, se realiza hibridación con un método de gradiente, resultando un algoritmo neuro-evolutivo, lo que permite mejorar la precisión obtenida. Ambos algoritmos han sido aplicados a diferentes problemas test, obteniendo en todos los casos modelos difusos precisos e interpretables. Los resultados de ambos algoritmos son similares, si bien el algoritmo neuro-evolutivo obtiene una precisión algo mayor. Dados los buenos resultados obtenidos, el algoritmo neuro-

evolutivo se ha aplicado también para resolver un par de problemas adicionales: la obtención de un modelo difuso para un problema real de predicción de riego para la Comunidad Autónoma de la Región de Murcia y una aplicación para resolver problemas de clasificación. Los resultados obtenidos en todos los casos han sido satisfactorios.

Así pues, esta tesis se centra en el diseño y evaluación de algoritmos evolutivos multiobjetivo para optimización y modelización difusa. La experiencia en computación evolutiva del grupo de investigación de Sistemas Inteligentes del Departamento de Ingeniería de la Información y las Comunicaciones de la Universidad de Murcia ha sido un factor determinante a la hora de diseñar los algoritmos. Por otra parte, la evaluación de los algoritmos se ha efectuado teniendo en cuenta la bondad de las soluciones, para lo que ha sido fundamental la elección de problemas test que sean difíciles de resolver y al mismo tiempo permitan una fácil inspección visual de sus espacios de solución. En ambos aspectos de diseño y evaluación ha existido una estrecha colaboración con grupos de investigación de gran relevancia a nivel internacional.

Dentro de esta memoria, por tanto, pueden observarse dos partes diferenciadas: una primera dedicada a optimización multiobjetivo y otra segunda donde se aplican las técnicas desarrolladas en la parte anterior para modelización difusa. La primera parte comprende los tres primeros capítulos. En el primer capítulo se hace un estudio de las técnicas evolutivas más importantes. En el segundo capítulo se expone una amplia revisión de los algoritmos evolutivos utilizados en optimización multiobjetivo, incluyendo el manejo de restricciones. Y en el tercer capítulo se describen los nuevos algoritmos desarrollados para resolver problemas de optimización multiobjetivo. La segunda parte de esta memoria abarca los dos últimos capítulos. En el cuarto capítulo se realiza un estudio básico sobre modelización difusa y algunos de los algoritmos evolutivos utilizados para generar modelos difusos. En el quinto, y último capítulo, se presentan los nuevos algoritmos diseñados para resolver problemas de modelización difusa. Tanto el tercer como el quinto capítulo de la tesis incluyen respectivas secciones de experimentos sobre los que se ha probado la eficacia de los algoritmos diseñados.

La memoria termina recogiendo las principales conclusiones extraídas de los resultados obtenidos e indicando las líneas de investigación futura de mayor interés. Se incluye, además, un apéndice introduciendo algunos de los conceptos básicos sobre teoría de conjuntos difusos. Por último se recopila toda la bibliografía consultada.

Capítulo 1

Computación Evolutiva

La computación evolutiva se ha consolidado como una técnica válida para solucionar múltiples tipos de problemas. Sus características de robustez y paralelismo implícito hacen de la computación evolutiva el paradigma adecuado para resolver algunos problemas donde otras técnicas no consiguen buenos resultados.

La optimización multiobjetivo con funciones no lineales es uno de los campos donde la computación evolutiva ha encontrado una gran aplicación. Este trabajo se centra en el diseño y evaluación de nuevos algoritmos evolutivos para resolver problemas de este tipo. Resulta adecuado, por tanto, realizar en este primer capítulo una discusión sobre computación evolutiva, puesto que es esta técnica la que va a utilizarse en el desarrollo de este trabajo.

En este capítulo se realiza en primer lugar una introducción a la computación evolutiva en la sección 1.1; a continuación, en la sección 1.2, se realiza la descripción de un algoritmo genético, que es el tipo original de algoritmo evolutivo, y se comentan brevemente los fundamentos teóricos de los algoritmos genéticos; en la sección 1.3 se hace un repaso a los diversos métodos contemplados en la literatura para mejorar los algoritmos evolutivos de forma que sean más eficientes para resolver determinados problemas; y por último en la sección 1.4 se resumen los puntos tratados en el capítulo.

1.1 Introducción

Los seres vivos son capaces de adaptarse para sobrevivir en medios ambientes muy diversos, es decir, pueden resolver problemas de forma versátil. Para ello, la naturaleza utiliza, entre otros, los mecanismos de selección y genética. Los *algoritmos evolutivos* son técnicas que se basan en los principios de evolución natural como método de búsqueda robusto para resolver problemas complejos.

El inicio de los algoritmos basados en procesos naturales se sitúa al principio de la década de los cincuenta, cuando diversos biólogos utilizaron ordenadores para simular sistemas biológicos. Sin embargo, fue el trabajo realizado en la universidad de Michigan bajo la dirección de John Holland a finales de los sesenta y principios de los setenta el que dio origen al primer tipo de algoritmo evolutivo, los *algoritmos genéticos*. Holland escribe la principal monografía sobre este tema, *Adaptation in Natural and Artificial Systems* [Hol75]. Desde entonces, se han realizado múltiples trabajos que demuestran la validez de los algoritmos evolutivos en problemas de optimización y aplicaciones de control. Los algoritmos evolutivos han destacado principalmente por su robustez, manteniendo el equilibrio necesario entre eficiencia y eficacia para un buen comportamiento en muchos entornos diferentes. El campo de aplicación de los algoritmos evolutivos se amplía, por tanto, cada vez más: problemas de empresas, científicos, de ingeniería, etc.

Los algoritmos evolutivos mantienen una población de individuos (soluciones potenciales) que se transforman utilizando una serie de operadores de variación unarios (mutación) y de mayor orden (cruce). Se utiliza un esquema de selección basado en la adecuación de los individuos a su entorno (supervivencia de los más aptos, Darwin [Dar59]). Después de un cierto número de iteraciones, el algoritmo converge y es de esperar que el mejor individuo de la población represente la solución óptima al problema.

El funcionamiento básico de un algoritmo evolutivo es el siguiente:

1. Se genera, de forma aleatoria, una población inicial de soluciones potenciales al problema, y
2. se entra en un proceso iterativo que transforma la población a través de:
 - (a) evaluación de las soluciones que forman la población,

procedimiento algoritmo evolutivo $t \leftarrow 0$ inicializar-población(t)evaluar(t)**mientras** (no se cumpla la condición de parada) **hacer** $t \leftarrow t + 1$ seleccionar-padres(t)recombinar(t)mutar(t)evaluar(t)seleccionar-descendientes(t)

Figura 1.1: Estructura de un Algoritmo Evolutivo.

- (b) selección y reproducción de un conjunto de soluciones basándose en su conveniencia o adecuación, y
- (c) recombinación de estas soluciones por medio de operadores de variación para formar una nueva población.

Este proceso continúa durante un número determinado de iteraciones o hasta que se cumpla alguna condición de parada establecida. El resultado obtenido por el algoritmo es el representado por la mejor solución de la última población. Si el algoritmo ha convergido de forma adecuada, dicha solución será la óptima del problema.

La figura 1.1 describe un esquema del funcionamiento de un algoritmo evolutivo sencillo.

La gran ventaja de los algoritmos evolutivos frente a los métodos tradicionales

es su *robustez*, es decir, son algoritmos que mantienen una eficiencia alta para una gran variedad de problemas, mientras que los métodos tradicionales alcanzan una eficiencia mayor que los algoritmos evolutivos, pero sólo para un tipo concreto de problemas. Para conseguir esta robustez, los algoritmos evolutivos se diferencian fundamentalmente de otros métodos de optimización y búsqueda en los siguientes puntos:

- Trabajan con una codificación del conjunto de parámetros, no con los propios parámetros.
- Realizan una búsqueda a partir de una población de puntos, no un solo punto.
- La única información que utilizan es el valor de la función objetivo, sin utilizar funciones derivadas ni otro conocimiento auxiliar.
- Utilizan reglas de transición probabilísticas, no deterministas.

Generalmente se acepta que un algoritmo evolutivo debe contener los siguientes cinco componentes:

- Una *representación* de las soluciones potenciales del problema.
- Una forma de crear una *población inicial* de soluciones potenciales.
- Una *función de adecuación* capaz de medir la bondad de cualquier solución, y que hará el papel de “entorno”, en el cual las mejores soluciones (esto es, aquellas con mejor adaptación o conveniencia) tengan mayor probabilidad de supervivencia.
- Un conjunto de *operadores de variación* como reglas de transición probabilísticas (no deterministas) para guiar la búsqueda, que combinan entre sí las soluciones existentes con el propósito de obtener otras nuevas.
- El valor de unos *parámetros* de entrada que el algoritmo evolutivo usa para guiar su evolución (tamaño de la población, número de iteraciones, probabilidades de aplicación de los operadores de variación, etc.).

Existen diferentes tipos de algoritmos evolutivos. A continuación realizamos una breve discusión de algunos de ellos.

1.1.1 Programación Evolutiva

La programación evolutiva, desarrollada por Fogel et al. [Fog66], se concibe originalmente como una técnica de búsqueda a través de un espacio de pequeñas máquinas de estado finito, y posteriormente usada como optimizador. La programación evolutiva sigue el siguiente esquema:

1. Inicializar y evaluar los N individuos de la población.
2. Mutar los N individuos de la población para producir N hijos.
3. Evaluar los N hijos.
4. Seleccionar N descendientes entre los $2N$ individuos (padres e hijos).

La selección se realiza mediante un torneo probabilista, es decir, se toman varios individuos aleatorios y se selecciona el mejor de ellos.

Se sigue una estrategia elitista, es decir, el mejor individuo siempre es seleccionado para pasar a la siguiente generación.

5. Si se alcanza la condición de parada, terminar el algoritmo, en otro caso, volver al punto 2.

Podemos destacar las siguientes características de la programación evolutiva:

- Utiliza una representación adaptada al problema concreto que trata de resolver; por ejemplo, para problemas de optimización de números reales, una representación con números reales; para el problema del viajante de comercio, utiliza listas ordenadas, o grafos para problemas con máquinas de estado finito.
- La mutación se realiza según la representación utilizada que, como ya hemos visto, puede ser muy variada y se realiza normalmente de forma adaptativa.
- No se realiza cruce entre individuos, pero la mutación es bastante flexible y puede producir un efecto similar al del cruce.

1.1.2 Estrategias de Evolución

Las estrategias de evolución se proponen originalmente por Rechenberg [Rec73] utilizando selección, mutación y población de tamaño uno. Schwefel [Schw81] introduce el cruce y poblaciones con más de un individuo. Se aplican inicialmente para problemas de optimización de parámetros, utilizando representación con números reales. El esquema que siguen las estrategias de evolución es el siguiente:

1. Inicializar y evaluar los N individuos de la población.
2. Seleccionar de forma aleatoria y uniforme los individuos que van a ser padres.
3. Generar λ hijos ($\lambda > N$) mediante cruce de los padres.
4. Mutar los λ hijos.
5. Evaluar los λ hijos.
6. Seleccionar N descendientes de forma determinista mediante alguno de los siguientes métodos:
 - (a) Método (N, λ) . Seleccionar los mejores N hijos.
 - (b) Método $(N + \lambda)$. Seleccionar los mejores N hijos y padres.

El método (N, λ) no es elitista, mientras que el método $(N + \lambda)$ sí lo es.¹

7. Si se alcanza la condición de parada, terminar el algoritmo, en otro caso, volver al punto 2.

Al igual que en la programación evolutiva, se ha realizado un gran esfuerzo para realizar la mutación de forma adaptativa, pero a diferencia de la programación evolutiva, el cruce sí tiene un papel importante, especialmente para adaptar la mutación.

¹Tradicionalmente, a estos métodos se les referencia como (μ, λ) y $(\mu + \lambda)$, pero puesto que μ hace referencia al número de individuos N en la población, hemos preferido poner N en lugar de μ .

1.1.3 Programación genética

La principal característica de la programación genética [Koza92] es la representación de las soluciones. La programación genética no devuelve soluciones concretas a los problemas, sino programas (típicamente en lenguaje lisp, aunque puede ser otro tipo de lenguaje) que resuelven dichos problemas. Cada individuo es un programa concreto y se representa mediante estructuras de funciones y terminales que forman programas. La evaluación de un individuo se realiza ejecutando el programa que representa y asignando como valor de adecuación alguna medida de lo bien que ha resuelto el problema. El esquema que se utiliza en programación genética es el siguiente:

1. Inicializar y evaluar los N individuos de la población de forma proporcional a su adecuación.
2. Seleccionar N individuos para ser padres.

La selección puede realizarse de diversas formas basándose siempre en el valor de adecuación de los individuos. Algunos de los posibles esquemas son la selección proporcional a la adecuación, la selección por torneo, probabilidad proporcional al rango de la adecuación, etc.

3. Generar N hijos mediante cruce y/o mutación de los hijos.

Los operadores de cruce y variación se definen de forma específica según la representación utilizada.

4. Evaluar los N hijos.
5. Reemplazar la población anterior por los N hijos.
6. Si se alcanza la condición de parada, terminar el algoritmo, en otro caso, volver al punto 2.

La programación genética resulta muy potente puesto que su salida no es un valor, sino otro programa. En esencia, esto representa el inicio de los programas que se crean a si mismos. En algunos problemas, la programación genética resulta especialmente adecuada. Por ejemplo, en problemas donde no existe una única solución óptima o donde las variables se encuentran constantemente cambiando. La programación genética no encuentra una sola solución, sino el camino para encontrar soluciones diferentes cuando cambien las variables del problema.

1.1.4 Algoritmos Genéticos

Desarrollados originalmente por Holland [Hol75] utilizan tradicionalmente una representación independiente del problema, en concreto se utilizan cadenas binarias de longitud fija para codificar los individuos y como operadores de variación utilizan el cruce y la mutación binaria.

Puesto que los algoritmos genéticos utilizan una representación y unos operadores de variación independientes del problema, se consideran métodos muy robustos, puesto que pueden aplicarse a una gran diversidad de problemas sin tener que adaptarse a cada problema concreto.

El esquema básico de un algoritmo genético es el siguiente:

1. Inicializar y evaluar los N individuos de la población.
2. Seleccionar N individuos para ser padres.

La selección tradicionalmente se realiza de forma proporcional a su adecuación, es decir, aquellos individuos con una mejor adecuación son seleccionados con mayor probabilidad que aquellos con peor adecuación.

3. Generar N hijos mediante cruce de los padres.
4. Mutar los N hijos.
5. Evaluar los N hijos.
6. Reemplazar la población anterior por los N hijos.
7. Si se alcanza la condición de parada, terminar el algoritmo, en otro caso, volver al punto 2.

Resulta interesante notar que en los algoritmos genéticos históricamente se ha considerado el cruce como el operador principal, mientras que a la mutación se le ha concedido una importancia secundaria. Esto es lo contrario a lo que ocurre en la programación evolutiva. Sin embargo, el interés por la mutación es cada vez mayor, en parte debido a la influencia de la investigación realizada en el campo de la programación evolutiva y de las estrategias de evolución.

1.2 Enfoque de un Algoritmo Genético

En la sección anterior hemos visto que dentro de los algoritmos evolutivos existen diferentes tipos de algoritmos que se diferencian fundamentalmente por la importancia dada a cada tipo de operador y por el tipo de representación utilizada. Esta sección se centra en los algoritmos genéticos, describiendo un algoritmo genético simple y robusto que podrá ser aplicado en muchos dominios diferentes y haciendo una breve descripción de los fundamentos teóricos de los algoritmos genéticos.

La estructura general de un algoritmo genético se muestra en la figura 1.2. Se describe esquemáticamente el funcionamiento de un algoritmo genético para resolver problemas de optimización simple. Sin pérdida de generalidad, se consideran sólo problemas de maximización. Si el problema consiste en minimizar g , puede encontrarse otra función f de forma que resulte equivalente maximizar f a minimizar g , es decir:

$$\max f(\mathbf{x}) = \min g(\mathbf{x})$$

También se asume que la función objetivo f toma sólo valores positivos en su dominio; en otro caso, se suma una constante positiva C de forma que:

$$\max f(\mathbf{x}) = \min g(\mathbf{x}) + C$$

El problema, por tanto, consiste en maximizar una función de k variables $f(x_1, \dots, x_k)$, $\mathbb{R}^k \rightarrow \mathbb{R}$, donde cada variable x_i toma valores en un dominio $D_i = [l_i, u_i] \subset \mathbb{R}$ y $f(x_1, \dots, x_k) > 0$ para toda $x_i \in D_i$.

1.2.1 Representación

Existen diversos términos utilizados para referirse a una solución del problema, por ejemplo, **string** o **estructura**, y, siguiendo el vocabulario de los sistemas biológicos, **cromosoma**. Otros términos utilizados en computación evolutiva son:

Gen: Unidad básica de información. Los cromosomas están formados por genes. Si la representación es binaria, un gen corresponde a una unidad binaria.

Alelo: Valor de un gen determinado dentro del cromosoma.

Locus: Posición de un gen determinado dentro del cromosoma.

procedimiento algoritmo genético $t \leftarrow 0$ inicializar $P(t)$ evaluar $P(t)$ **mientras** (no se cumpla la condición de parada) **hacer** $t \leftarrow t + 1$ seleccionar $P(t)$ a partir de $P(t - 1)$ recombinar $P(t)$ mutar $P(t)$ evaluar $P(t)$

Figura 1.2: Estructura de un Algoritmo Genético.

Genotipo: Paquete genético total; es el cromosoma completo.**Fenotipo:** Interacción del genotipo con su entorno, que se traduce en la decodificación del cromosoma para la obtención de una solución (conjunto de parámetros particular, o un punto en el espacio de búsqueda).Utilizando una codificación binaria, podemos representar un cromosoma \mathbf{b} como:

$$\mathbf{b} = (b_1 \dots b_n) \text{ donde } b_i \in \{0, 1\}, i = 1, \dots, n$$

La decodificación o fenotipo \mathbf{x} , está formada por las k variables de decisión:

$$\mathbf{x} = (x_1 \dots x_k) \text{ donde } x_i \in D_i, i = 1, \dots, k$$

La adecuación f es un valor real que indica la adaptación del individuo a su entorno, es decir, indica cómo de óptimo es el valor alcanzado en la función objetivo

por dicho individuo.

El término **individuo** es frecuentemente utilizado (Goldberg [Gol89]) para referirse al conjunto de información *genotipo-fenotipo-adequación*. Así, podemos representar un individuo X como la terna:

$$\mathbf{X} = (\mathbf{b}, \mathbf{x}, f)$$

donde \mathbf{b} es la representación (genotipo, cromosoma),

\mathbf{x} es la decodificación (fenotipo), y

f es la adecuación de la solución al entorno.

En un algoritmo evolutivo, las soluciones se pueden codificar mediante cualquier estructura de datos adecuada al problema (representación mediante números reales, longitud variable, etc. . .) En un algoritmo genético, las soluciones potenciales al problema se representan mediante cadenas binarias de bits (0's y 1's) de una longitud determinada n que viene impuesta por el número de variables existentes en la solución y por el número de bits necesarios para codificarlas. El número de bits necesario para codificar una variable depende de la precisión que deseemos obtener. Si se desea obtener una precisión de p decimales para la variable i -ésima, el dominio D_i debe dividirse en $(u_i - l_i) \cdot 10^p$ tramos y por tanto el número de dígitos binarios n_i necesarios para codificar la variable i -ésima es el menor entero n_i tal que $(u_i - l_i) \cdot 10^p \leq 2^{n_i} - 1$. El número de bits n necesario para codificar una solución al problema se calcula por tanto como:

$$n = \sum_{i=1}^k n_i$$

.

Cada variable x_i , $i = 1, \dots, k$ se interpreta según la siguiente formula:

$$x_i = l_i + decimal(b_{i_1} \dots b_{i_2}) \frac{u_i - l_i}{2^{n_i} - 1}$$

donde $decimal(b_{i_1} \dots b_{i_2})$ representa el valor decimal de la cadena binaria $b_{i_1} \dots b_{i_2}$,

$$i_1 = \sum_{j=1}^{i-1} n_j + 1, \text{ y}$$

$$i_2 = \sum_{j=1}^{i-1} n_j + n_i.$$

1.2.2 Población inicial

La población en un algoritmo evolutivo está formada por un conjunto de N individuos, donde el número N (tamaño de la población) es un parámetro de entrada al algoritmo. De esta forma, en una generación determinada t , podemos representar una población $P(t)$ como:

$$P(t) = \{\mathbf{X}_1^t, \dots, \mathbf{X}_N^t\}$$

donde el individuo i -ésimo en una generación determinada t se representa como:

$$\mathbf{X}_i^t = (\mathbf{b}_i^t, \mathbf{x}_i^t, f_i^t)$$

siendo $\mathbf{b}_i^t = (b_{i1}^t, \dots, b_{in}^t)$ la representación,

$\mathbf{x}_i^t = (x_{i1}^t, \dots, x_{ik}^t)$ la decodificación, y

f_i^t es la adecuación.

Para obtener una población inicial $P(0) = \{\mathbf{X}_1^0, \dots, \mathbf{X}_N^0\}$ debemos por tanto generar N individuos iniciales. Estos individuos deben generarse de forma aleatoria uniforme en todo su dominio.

En el caso de representación binaria, el procedimiento de inicialización de un individuo \mathbf{X}_i^0 consiste simplemente en asignar, para cada gen de su cromosoma \mathbf{b}_i^0 , un valor aleatorio 0 ó 1.

Con la decodificación del cromosoma \mathbf{b}_i^0 obtendremos su fenotipo \mathbf{x}_i^0 , y la adecuación de la solución al entorno se obtiene a través de la función de adecuación f_i^0 .

1.2.3 Función de adecuación

Una forma sencilla de medir la adecuación de una solución en una generación t consiste simplemente en evaluar su fenotipo a través de la función objetivo f del problema que se esté resolviendo. Así pues, la función *adecuación* se corresponde con la función objetivo f del problema, y entonces, dado un cromosoma \mathbf{b}_i^t y su fenotipo \mathbf{x}_i^t , podemos obtener su adecuación f_i^t como:

$$f_i^t = \text{adecuación}(\mathbf{b}_i^t) = f(\mathbf{x}_i^t)$$

La función objetivo juega por tanto un papel fundamental en un algoritmo evolutivo, puesto que esta es la única información que se usa del entorno, lo cual hace que estos métodos sean muy generales y robustos.

1.2.4 Operadores de variación

Una vez que se han evaluado todas las soluciones de la población en una generación, el proceso evoluciona hacia una nueva generación. La población en la nueva generación t sufrirá una transformación con respecto a la población en la generación anterior $t - 1$. El objetivo de esta transformación es guiar a la población de una forma probabilística hacia la solución óptima del problema, de forma que las soluciones de la población en la generación t se encuentren, en su conjunto, más “cerca” de la solución óptima al problema que las soluciones de la población en la generación $t - 1$.

Esta transformación se lleva a cabo, en un esquema básico de funcionamiento, por medio de la aplicación de los tres operadores de variación siguientes como reglas de transición probabilísticas:

1. Selección.
2. Cruce.
3. Mutación.

El operador de selección se encarga de modelizar el mecanismo de reproducción dentro del algoritmo evolutivo, es decir, asegura la supervivencia en mayor número de los individuos más adaptados (soluciones más adecuadas). Los operadores de cruce y mutación realizan la recombinación de los individuos seleccionados para generar otros individuos. El cruce combina información de individuos diferentes, mientras que la mutación introduce una variación aleatoria en la información, necesaria para mantener la suficiente diversidad entre las soluciones que forman la población.

Operador de selección

Los *operadores de selección* de un algoritmo evolutivo pueden ser modelizados a través de dos algoritmos diferentes [Gey95]:

1. *Algoritmo de selección*

Si t es la generación actual, el algoritmo de selección obtiene la probabilidad de selección de cada individuo de la población en la generación anterior $t - 1$, es decir, asigna a cada individuo el número esperado de descendientes en la generación actual t .

2. Algoritmo de muestreo.

El algoritmo de muestreo produce copias de los individuos, desde la población en la generación $t - 1$ a la población en la generación actual t , de acuerdo a sus probabilidades de selección.

En el algoritmo genético que estamos describiendo se usa, como algoritmo de selección, la *selección proporcional* [Gey95], y como algoritmo de muestreo, el *muestreo estocástico con reemplazamiento* [DeJ75]. La combinación de estos dos algoritmos de selección y muestreo se denomina *selección por rueda de ruleta* en [Gol89], y *selección simple* en [Mic92].

La selección proporcional asigna a cada individuo i de la población en la generación $t - 1$, la siguiente probabilidad de selección:

$$p_i^{t-1} = \frac{f_i^{t-1}}{F^{t-1}}, \quad i = 1, \dots, N$$

donde F^{t-1} es la suma de la adecuación de todos los individuos de la población en la generación $t - 1$:

$$F^{t-1} = \sum_{i=1}^N f_i^{t-1}$$

El algoritmo de muestreo estocástico con reemplazamiento realiza los siguientes pasos:

- Calcular la probabilidad acumulada de cada individuo i de la población en la generación $t - 1$:

$$q_i^{t-1} = \sum_{j=1}^i p_j^{t-1}, \quad i = 1, \dots, N$$

- Repetir N veces:

- Generar un número real aleatorio $\alpha \in [0, 1]$,

- Si $\alpha \leq q_1$ entonces seleccionar el primer individuo de la población en la generación $t - 1$ y copiarlo en la población de la generación actual t ; en otro caso seleccionar y copiar el i -ésimo individuo ($2 \leq i \leq N$) tal que $q_{i-1}^{t-1} < \alpha \leq q_i^{t-1}$.

Este proceso de muestreo representa una rueda de ruleta con N marcas y separaciones entre ellas fijadas proporcionalmente de acuerdo a la adecuación de los individuos. En un algoritmo genético normalmente se usa el esquema de *sustitución generacional completa* [Gey95, Gol89], con el cual los N miembros de la población anterior son sustituidos por los N miembros de la nueva población, por lo que “tendremos que girar N veces la rueda de ruleta”. Los individuos con mayor probabilidad de selección (mejor adecuación) serán seleccionados un número mayor de veces, y tendrán mayor contribución en las siguientes generaciones.

Operador de cruce

Una vez formada la nueva población en la generación actual t , se aplica el primer operador de variación, el *operador de cruce* [Hol75]. Uno de los parámetros de entrada que usa el algoritmo genético es la probabilidad de cruce p_{cruce} que nos da el número esperado de individuos $p_{cruce} \cdot m$ a los cuales se les aplicará el operador de cruce en cada generación. La forma de seleccionar los individuos para el cruce se describe a continuación:

Para cada individuo i de la población en la generación t :

- Generar un número real aleatorio $\alpha \in [0, 1]$,
- Si $\alpha < p_{cruce}$ entonces seleccionar dicho individuo para el cruce.

Seguidamente, se realiza el cruce de los individuos seleccionados. Uno de los esquemas de cruce más usados es el *cruce aleatorio* [Gol89], el cual selecciona el cruce de forma aleatoria con igual probabilidad, y el *cruce simple* [Gol89].

Dados dos individuos i y j seleccionados para el cruce en la generación t :

$$\begin{aligned} \mathbf{b}_i^t &= (b_{i1}^t \dots b_{ipos}^t b_{i(pos+1)}^t \dots b_{in}^t) \\ \mathbf{b}_j^t &= (b_{j1}^t \dots b_{jpos}^t b_{j(pos+1)}^t \dots b_{jn}^t) \end{aligned}$$

se genera un número entero aleatorio $pos \in [1 \dots n - 1]$, el cual indica la posición del punto de cruce y los individuos anteriores son sustituidos por sus descendientes:

$$\mathbf{b}_i^t = (b_{i1}^t \dots b_{i_{pos}}^t b_{j_{(pos+1)}}^t \dots b_{jn}^t)$$

$$\mathbf{b}_j^t = (b_{j1}^t \dots b_{j_{pos}}^t b_{i_{(pos+1)}}^t \dots b_{in}^t)$$

los cuales contienen la información genética cruzada de los padres.

Operador de mutación

El siguiente operador de variación es el *operador de mutación* [Hol75]. En el algoritmo genético que estamos describiendo se usa la *mutación simple* [Gol89], que se aplica, después del operador de cruce, en cada cromosoma de la población en la generación actual t . En cada bit del cromosoma se lleva a cabo un cambio de 0 a 1 o viceversa con probabilidad $p_{mutación}$, parámetro éste de entrada que nos da el número esperado de bits mutados $p_{mutación} \cdot n \cdot m$ en cada generación. Se puede entonces proceder de la siguiente forma:

Para cada individuo i de la población en la generación t , y para cada bit del cromosoma de dicho individuo:

- Generar un número real aleatorio $\alpha \in [0, 1]$,
- Si $\alpha < p_{mutación}$ entonces mutar dicho bit.

Después de la aplicación de los operadores de variación (selección, cruce y mutación), la nueva población está preparada para su evaluación. El resto de la evolución, tal como muestra el esquema de la figura 1.2, es simplemente un proceso iterativo de selección-variación-evaluación en el transcurso de las generaciones hasta que se cumple la condición de parada, que suele ser alcanzar un número determinado de generaciones o el cumplimiento de algún predicado de éxito si es posible reconocer soluciones optimales o satisfactorias para el decisor.

1.2.5 Parámetros

Los parámetros de entrada más significativos que el algoritmo evolutivo utiliza son los siguientes:

- *Tamaño de la población*
- *Número de generaciones*
- *Probabilidad de cruce*
- *Probabilidad de mutación*

De Jong [DeJ75], completando los primeros estudios de Holland [Hol75], realiza un análisis para la correcta fijación de parámetros haciendo distinción entre situaciones *off-line* y situaciones *on-line*. Para el tamaño de la población, tales estudios apuntan a un buen rendimiento *off-line* para poblaciones grandes, ya que convergen finalmente mejor debido a una mayor diversidad, y a un buen rendimiento *on-line* para poblaciones pequeñas, ya que éstas tienen la habilidad de cambiar más rápidamente y ofrecer una convergencia inicial mejor. En [Gol89c] se establece una guía para la elección de posibles tamaños de población para representaciones binarias de longitud fija en función de la longitud de los cromosomas. Asimismo, en [Gol89c] se estima que el número esperado de generaciones hasta la convergencia es una función logarítmica del tamaño de la población. Para las probabilidades de cruce y mutación, los estudios realizados en el tema apuntan a probabilidades de cruce altas y probabilidades de mutación bajas. Los siguientes valores han sido considerados como aceptables para un buen rendimiento de los algoritmos genéticos para funciones de optimización [Sch89]:

- Tamaño de la población: 50 – 100.
- Probabilidad de cruce: 0.60.
- Probabilidad de mutación 0.001.

1.2.6 Fundamentos teóricos

Para entender por qué funcionan bien los algoritmos genéticos es necesario recurrir a todo el formalismo teórico que sustenta a los algoritmos genéticos [Ank91, Ber93, DeJ75, Gol85, Gol89, Gol89b, Gol91, Hol75, Hol88, Mic92]. Este estudio teórico se realiza sobre algoritmos genéticos por ser el tipo original de algoritmo evolutivo y los más estudiados, aunque hay algún estudio que extiende estos resultados, por ejemplo, Wright [Wri91] amplía estos fundamentos teóricos para algoritmos con codificación real. En este capítulo, de carácter introductorio, no pretendemos realizar un estudio exhaustivo y riguroso de tal formalismo, sino que nos limitamos a comentar brevemente algunas notas de carácter general sobre los fundamentos teóricos de los algoritmos genéticos.

En un algoritmo genético, la búsqueda de soluciones idóneas de un problema consiste en la búsqueda de determinadas cadenas binarias, de forma que el universo de todas las posibles cadenas puede ser concebido como un paisaje imaginario con cimas y valles que albergan a buenas y malas soluciones respectivamente. También podemos definir *regiones* en el espacio de soluciones fijándonos en las cadenas que posean unos o ceros en lugares determinados, y que usualmente son denominadas *esquemas* o *hiperplanos*. Un esquema es por tanto un patrón de similitud que describe un subconjunto de cadenas con similitudes en ciertas posiciones de éstas, y se define sobre el alfabeto ternario $\{0, 1, *\}$, donde el símbolo $*$ en una posición del esquema indica que el alelo en dicha posición no está determinado. Así por ejemplo, para cadenas de longitud $n = 5$, el esquema $(1 * * * *)$ representa todas las cadenas que comiencen por 1 en el espacio de posibilidades, y el esquema $(1 * 0 * 1)$ describe un subconjunto de cuatro miembros $\{(10001), (10011), (11001), (11011)\}$. De esta forma, el conjunto de cadenas que forma una población en el algoritmo genético sondea el espacio de soluciones en muchos esquemas a la vez. Por tanto, una de las claves de la buena conducta de los algoritmos genéticos reside en que cada cadena individual pertenece a todos los esquemas en los cuales aparece uno cualquiera de sus bits. Un algoritmo genético que manipula una población de un número determinado de cadenas está tomando muestras de un número de esquemas enormemente mayor. Holland [Hol75] estima que el número de esquemas procesados útilmente en un algoritmo genético que manipula una población de N individuos es del orden de N^3 . Este es un resultado importante al que Holland da el nombre de *paralelismo implícito*.

El algoritmo genético [Gol89, Hol75] explota los esquemas de más alto rendimiento del espacio de soluciones, ya que las sucesivas generaciones de selección y cruce generan un número creciente de cadenas pertenecientes a ellas. La mutación, que por sí sola no puede generar progresos en la búsqueda de una determinada solución, proporciona un mecanismo de exploración (nuevos esquemas) que impide el desarrollo de una población uniforme e incapaz de ulterior evolución. Ahora bien, el efecto de los operadores de cruce y mutación puede provocar también que determinadas cadenas abandonen el esquema de sus progenitores en el transcurso de las generaciones. La probabilidad de que una cadena particular abandone el esquema de sus progenitores depende del *orden* y de la *longitud* de dicho esquema [Gol89, Hol75, Mic92]. El orden $o(H)$ se define como el número de posiciones fijas del esquema H , y será útil para calcular la probabilidad de supervivencia del esquema H bajo el efecto de la mutación. La longitud $\delta(H)$ se define como la distancia entre la primera posición fija y la última posición fija en el esquema H , y será útil para calcular la probabilidad de supervivencia del esquema H bajo el efecto del cruce. Así pues, los esquemas constituidos con pocas y contiguas posiciones fijas que además tengan una adecuación por encima de la media, llamados *bloques constructivos*, tienen mayor probabilidad de salir intactos de cruces y mutaciones y propagarse a las generaciones futuras.

Aplicando únicamente el operador de selección, el número esperado de representantes del esquema H en la población en la generación $t + 1$ viene dado por la siguiente expresión [Gol89, Hol75]:

$$N(H, t + 1) = N(H, t) \frac{f(H, t)}{\bar{f}^t}$$

donde $N(H, t)$ es el número de representantes del esquema H en la población en la generación t ,

$f(H, t)$ es la adecuación media de los representantes del esquema H en la población en la generación t , y

\bar{f}^t es la adecuación media de la población en la generación t :

$$\bar{f}^t = \frac{\sum_{i=1}^N f_i^t}{N}$$

Si asumimos que un esquema H permanece por encima de la media en un $\epsilon\%$, es decir:

$$f(H, t) = \overline{f^t} + \epsilon \overline{f^t}$$

entonces:

$$N(H, t+1) = N(H, 0) (1 + \epsilon)^t$$

lo que significa que los esquemas por encima de la media reciben un incremento exponencial de sus representantes en las siguientes generaciones.

Si consideramos también el efecto de los operadores de cruce y mutación, tenemos que:

$$N(H, t+1) \geq N(H, t) \frac{f(H, t)}{\overline{f^t}} \left[1 - p_{cruce} \frac{\delta(H)}{n-1} - o(H) p_{mutación} \right]$$

Tal efecto no es significativo si el esquema es de longitud corta y de orden bajo, por lo que en tales condiciones se sigue produciendo un incremento exponencial de representantes del esquema en las siguientes generaciones.

Todo lo comentado anteriormente se resume a través de la *Hipótesis del Bloque Constructivo* y del *Teorema Fundamental de los Algoritmos Genéticos* o *Teorema del Esquema* [Gol89, Hol75, Mic92].

Hipótesis del Bloque Constructivo

Un algoritmo genético busca soluciones cerca del óptimo a través de la yuxtaposición de esquemas de longitud corta, orden bajo y adecuación alta, llamados bloques constructivos.

Teorema del Esquema

Los esquemas de longitud corta, orden bajo y adecuación por encima de la media reciben un incremento exponencial de sus representantes en subsecuentes generaciones de un algoritmo genético.

1.3 Computación Evolutiva: Mejoras y Extensiones

La teoría que sustenta a los algoritmos genéticos proporciona una explicación de por qué estos algoritmos convergen a soluciones óptimas; sin embargo, en aplicaciones prácticas, surgen inconvenientes que hacen que no siempre se cumplan estos resultados teóricos. Una de las principales causas que hacen que los algoritmos genéticos se vean incapacitados para encontrar soluciones óptimas en algunas circunstancias es la *convergencia prematura*. Tal problema, que aparece también en otras técnicas de optimización, consiste, como su nombre indica, en una convergencia demasiado rápida, posiblemente a un óptimo local, debido a la presencia de *superindividuos* en la población actual, los cuales son mucho mejores que la adecuación media de la población, por lo que acarrean un gran número de descendientes e impiden que otros individuos contribuyan con su descendencia en la siguiente generación, con la consecuente pérdida de información, lo que lleva consigo la formación de poblaciones altamente uniformes e incapaces de evolucionar. Ciertas implementaciones de algoritmos genéticos son más propensas a la convergencia prematura que otras, y los investigadores más conocidos en el tema aluden a los mecanismos de selección y muestreo, a la ruptura de esquemas debido al cruce, a la fijación de parámetros y a las características propias de la función.

La convergencia prematura, sin embargo, no es el único problema con el que se encuentran los algoritmos genéticos [Gol89]. Aunque los algoritmos genéticos están clasificados como algoritmos *débiles* [Gol89, Mic92], existe una gran variedad de problemas los cuales resultan difíciles o imposibles de resolver con el esquema simple de algoritmo genético visto anteriormente, como por ejemplo, problemas que contienen restricciones no triviales, problemas que requieren una precisión alta de los parámetros, problemas con función multimodal, con múltiples objetivos, etc.

Todo esto ha llevado en los últimos años al estudio de mejoras y extensiones sobre los algoritmos genéticos, formando un conjunto de algoritmos más amplios, los *algoritmos evolutivos*, que proporcionan un mejor rendimiento así como un mayor ámbito de aplicación, tales como representaciones alternativas, mecanismos de obtención de poblaciones iniciales, funciones de evaluación, esquemas de selección, muestreo y sustitución generacional, esquemas de apareamiento, nuevos operadores de cruce y mutación, operadores de reordenación, operadores diploides y de dominancia, formación de especies y nichos, técnicas basadas en el conocimiento específico

del problema, técnicas para el manejo de restricciones, optimización multiobjetivo, etc. [Gey95, Gol89, Mic92]. A continuación describimos brevemente algunas de las mejoras y extensiones más significativas que aparecen en la literatura. La optimización multiobjetivo y el manejo de restricciones por ser dos de los aspectos más relevantes en este trabajo se discuten en dos secciones aparte del siguiente capítulo.

1.3.1 Representación

La elección de la representación de las soluciones del problema adquiere una gran importancia en los algoritmos evolutivos. La representación elegida puede limitar directamente la forma en la que el sistema observa su ambiente, y repercute en gran medida en el diseño de otros componentes del sistema. Existe una gran variedad de alternativas de representación en los algoritmos evolutivos. Tal variedad aparece incluso considerando únicamente alfabetos binarios, encontrándonos con representaciones de longitud fija y de longitud variable, representaciones con codificación binaria y con codificación Gray [Hol71], representaciones haploides y diploides, etc. [Gol89].

El alfabeto binario facilita los análisis teóricos y permite el diseño de operadores de variación elegantes. Sin embargo, la representación binaria puede presentar inconvenientes cuando se aplica a problemas numéricos que requieren una alta precisión, o en presencia de restricciones no triviales [Mic92], y puede resultar difícil de aplicar y no natural en muchos problemas prácticos. Por otro lado, el paralelismo implícito no depende del uso de representaciones binarias [Ant89], por lo que se abre un camino hacia el uso de representaciones distintas. Estas representaciones pueden diferir de la representación con cadenas binarias tanto en la cardinalidad del alfabeto como en la estructura propiamente dicha. Se han considerado alfabetos de mayor cardinalidad, como pueden ser los enteros y los reales, que resultan especialmente útiles en problemas de optimización numérica, y una gran variedad de estructuras de datos diferentes, como por ejemplo matrices, permutaciones, listas, árboles, grafos, etc.

Muchos investigadores [Mic92] mantienen que una representación natural de las soluciones del problema junto con una familia de operadores de variación aplicables, puede ser bastante útil en la aproximación de soluciones para muchos problemas.

1.3.2 Población inicial

Usualmente, los individuos de la primera población de un algoritmo evolutivo son generados por medio de un *muestreo aleatorio uniforme* dentro del espacio de búsqueda de las soluciones, como se mostró anteriormente. Sin embargo, algunas excepciones están contempladas en la literatura:

Muestreo aleatorio no uniforme

Existen algunas codificaciones para las cuales encontrar un algoritmo rápido de muestreo aleatorio uniforme puede resultar bastante complicado, usándose entonces algoritmos de muestreo aleatorio no uniforme [Koza92].

Muestreo aleatorio con información híbrida

En ciertas aplicaciones prácticas puede resultar de gran importancia introducir en la población inicial buenas soluciones conocidas, obtenidas con algún otro método de resolución propio del problema (*Técnicas basadas en el conocimiento* [Dav91, Gol89]), con lo cual, combinado con el *elitismo* [DeJ75], se garantiza que el algoritmo evolutivo no obtendrá peores soluciones que el método propio. Sin embargo, esta técnica debe implementarse con cuidado, ya que si estas soluciones introducidas en la población están muy por encima de la adecuación media de la población, es bastante probable que ocurra la convergencia prematura. Como solución al problema, estas buenas soluciones pueden introducirse cuando hayan transcurrido algunas generaciones, o bien utilizar *selección por ranking* [Bak85, Whi89].

Inicialización parcialmente enumerativa

En este método, se generan inicialmente todos los posibles esquemas de un tamaño dado y en una primera fase se reduce esta población inicial, demasiado grande, al tamaño de población normal.

La motivación de este método es asegurar que todos los buenos esquemas necesarios para la solución están representados en la población inicial, por que al menos un representante de éstos es generado [Gol89b].

1.3.3 Función de adecuación

Hasta ahora se ha tomado como adecuación el propio valor de la función objetivo, $f_i^t = f(\mathbf{x}_i^t)$. A este valor se llama *adecuación bruta*. En algunos casos, sin embargo, conviene realizar alguna transformación de dicho valor, por ejemplo, realizar algún tipo de escalado o normalización del mismo. A continuación se discuten algunas de las técnicas más utilizadas para tratar la adecuación.

Adecuación estandarizada

Hemos visto que los problemas de maximización y minimización resultan equivalentes y por tanto suponemos sólo problemas de minimización. Sin embargo, si queremos resolver también problema de maximización Koza [Koza92] propone utilizar el reverso del valor de la función objetivo, de forma que siempre aseguremos que los valores menores de la función de adecuación son los mejores:

$$f_i'^t = \begin{cases} f_i^t, & \text{para problemas de minimización} \\ f_{max} - f_i^t, & \text{para problemas de maximización} \end{cases}$$

siendo f_{max} el valor máximo de la función objetivo f .

Escalado dinámico

En muchos problemas resulta conveniente que el valor de la mejor adecuación sea igual a cero, para conseguir esto Koza [Koza92] realizar un escalado dinámico de la forma:

$$f_i'^t = f_i^t - f_{min}^t$$

donde f_{min}^t es el valor mínimo de la función objetivo para todos los individuos de la población en la iteración t :

$$f_{min}^t = \min_{i=1,\dots,M} f_i^t$$

Otra forma de realizar el escalado dinámico es considerando todos los individuos en las últimas p generaciones, es decir:

$$f_i'^t = f_i^t - f_{min}^{t,p}$$

donde $f_{min}^{t,p}$ es el valor mínimo de la función objetivo para todos los individuos de la población en las últimas p iteraciones:

$$f_{min}^{t,p} = \min_{\substack{i = 1, \dots, N \\ j = t - p, \dots, t}} f_i^j$$

Ajuste de la adecuación

Koza [Koza92] realiza un ajuste de la adecuación de forma que el valor de ésta se encuentre siempre entre 0 y 1 con valores mayores para los mejores individuos. La ventaja de este método es que incrementa la diferencia entre individuos a medida que estos son mejores:

$$f_i'^t = \frac{1}{1 + f_i^t}$$

Normalización

Koza [Koza92] también propone normalizar la adecuación de forma que el valor se encuentre entre 0 y 1 asignando a los mejores individuos valores mayores, de forma que la suma de todas las adecuaciones normalizadas en una población sea igual a uno:

$$f_i'^t = \frac{f_i^t}{F^t}$$

donde F^t es la suma de todas las adecuaciones de la población en la generación t :

$$F^t = \sum_{i=1}^N f_i^t$$

Escalado lineal

Una práctica ampliamente aceptada para mantener unos niveles de competición apropiados durante el proceso de evolución consiste en el *escalado* de la adecuación bruta (valor de la función objetivo, f_i^t), con lo cual se previene el dominio de la población por parte de los superindividuos que pueden dar lugar a la convergencia prematura. Entre los mecanismos de escalado más importantes se incluyen el escalado lineal, el truncamiento sigma o el escalado de potencia [Gol89, Mic92].

Con el escalado lineal la adecuación bruta f_i^t de un cromosoma es escalada usando una ecuación lineal de la forma:

$$f_i'^t = af_i^t + b$$

donde los coeficientes a y b son normalmente seleccionados de forma que:

1. el valor de adecuación medio bruto sea igual al valor de adecuación medio escalado, y
2. el valor de la mejor adecuación sea un múltiplo (típicamente 2) del valor de adecuación medio.

El escalado lineal trabaja bastante bien excepto cuando aparecen valores de adecuación negativos, que deben ser tratados de forma específica.

Truncamiento sigma

Para tratar valores de adecuación negativos e incorporar información dependiente del problema Goldberg [Gol89] propone el truncamiento sigma, que calcula la nueva adecuación de la siguiente forma:

$$f_i'^t = f_i^t + (\overline{f^t} - c\sigma^t)$$

donde $\overline{f^t}$ es el valor medio de adecuación de la población en la generación t ,

σ^t es la desviación estándar de la adecuación la población en la generación t , y

c es elegido como un múltiplo razonable de σ^t (típicamente entre 1 y 3).

Los resultados negativos ($f_i'^t < 0$) son fijados a cero.

Escalado de potencia

Con este método, la adecuación escalada se toma como alguna potencia específica de la adecuación bruta:

$$f_i'^t = (f_i^t)^k$$

para algún k cercano a 1.

Gillies [Gil85] concluye en sus estudios que la elección de k es dependiente del problema, fijando $k = 1.005$ para una aplicación en una máquina de visión.

1.3.4 Esquemas de selección, muestreo y sustitución generacional

En el algoritmo genético descrito en el apartado 1.2 se ha utilizado la selección proporcional, el muestreo estocástico con sustitución y la sustitución generacional completa. Con la selección proporcional se tiene el inconveniente de que la presencia de individuos muy por encima de la adecuación media puede dar lugar a la convergencia prematura. Con el muestreo estocástico con sustitución puede ocurrir que un mismo individuo sea seleccionado un número excesivo de veces (incluso podría completar la nueva población), lo que daría lugar a poblaciones muy uniformes. El esquema de sustitución generacional completa presenta el inconveniente de que el mejor individuo de la población (quizás una solución óptima) puede desaparecer en la siguiente población. Todos estos inconvenientes han motivado el estudio de nuevos modelos de algoritmos evolutivos que han supuesto mejoras sustanciales en el rendimiento de éstos.

Los primeros y más reconocidos trabajos sobre variaciones del modelo de selección simple fueron realizados por DeJong [DeJ75]. Desde entonces, diversos autores han desarrollado diferentes métodos. Una extensa clasificación de los diferentes mecanismos de selección y muestreo puede encontrarse en Bäck y Hoffmeister [Bac91]. A continuación se discuten algunas de las técnicas más importantes propuestas en la literatura para realizar la selección, muestreo y sustitución generacional.

Modelo elitista

Este modelo de sustitución generacional, sugerido por DeJong [DeJ75], consiste en forzar al mejor individuo de la población en cada generación a existir en la población de la siguiente generación. Aunque este modelo puede incrementar el problema de la convergencia prematura, en la mayoría de los casos mejora el rendimiento del algoritmo evolutivo.

Modelo del valor esperado

Otro modelo introducido por DeJong [DeJ75] es el modelo del valor esperado. Con este modelo para cada individuo \mathbf{x}_i^t se introduce un contador c_i^t , fijado inicialmente

con el número esperado e_i de descendientes para dicho individuo:

$$c_i^0 = e_i = \frac{f_i^0}{\overline{f^0}}$$

siendo $\overline{f^0}$ el valor medio de la adecuación para los individuos en la generación 0.

El valor c_i^t se decrementa por 0.5 ó 1 cuando el individuo es seleccionado para el cruce o mutación respectivamente, de forma que cuando el contador de un individuo está por debajo de 0, deja de estar disponible para su selección.

Modelo del valor esperado elitista

DeJong [DeJ75] también propone utilizar una combinación los dos modelos anteriores.

Modelo del factor de multitud (crowding)

Este modelo es también sugerido por DeJong [DeJ75]. En este esquema de sustitución, un individuo nuevo sustituye a uno antiguo el cual es seleccionado de entre un subconjunto de CF miembros elegidos aleatoriamente de la población completa. En este subconjunto se selecciona para ser sustituido el individuo más parecido al nuevo, usando como medida un contador de semejanza bit-a-bit.

Muestreo determinista

Brindle [Bri81] considera algunas modificaciones adicionales a los modelos anteriores. En el modelo del muestreo determinista se calcula el número esperado e_i de descendientes para cada individuo de la forma:

$$e_i = \frac{f_i^t}{\overline{f^t}}$$

siendo $\overline{f^t}$ el valor medio de la adecuación para los individuos en la generación t .

Cada individuo produce muestras de acuerdo a la parte entera de su valor e_i . La población se ordena de acuerdo a las partes fraccionales de los valores e_i , y el resto de individuos necesarios para completar la población se extraen de la parte superior de la lista ordenada.

Muestreo estocástico del resto con sustitución

Este modelo es otra variación propuesta por Brindle [Bri81]. El método comienza de forma idéntica al muestreo determinista, es decir, calcula los valores e_i y se producen muestras de acuerdo a las partes enteras de dichos valores. Para completar el resto de la población, las partes fraccionales se usan para calcular los pesos de una rueda de ruleta como en el muestreo estocástico con sustitución, con lo cual, un individuo puede ser seleccionado cualquier número de veces.

Muestreo estocástico del resto sin sustitución

Este método, también propuesto por Brindle [Bri81], es idéntico al muestreo estocástico del resto con sustitución excepto que para completar el resto de la población se favorece a los individuos cuyos valores e_i tengan partes fraccionales pequeñas.

Torneo estocástico

Este es otro método sugerido por Brindle [Bri81]. Consiste en calcular las probabilidades de selección de acuerdo al modelo de selección proporcional y extraer pares sucesivos de individuos mediante un muestreo estocástico con sustitución. Después de extraer un par, el cromosoma con mayor adecuación se declara ganador y es insertado en la nueva población. Este proceso continúa hasta que se completa la nueva población.

Otra forma de realizar el torneo propuesta por Koza [Koza92] consiste en extraer un conjunto aleatorio de r individuos de la población y elegir como ganador del torneo al mejor individuo de dicho conjunto.

Muestreo estocástico universal

Baker [Bak89] realiza un estudio de las modificaciones anteriores y presenta una versión mejorada. La idea básica de este método es construir una rueda de ruleta con N punteros distribuidos equidistantemente (en lugar de un sólo puntero como ocurre con la rueda de ruleta convencional). De esta forma, con un simple ‘giro’ de la ruleta se seleccionan N ganadores a la vez.

Ranking lineal

Como alternativa al mecanismo de selección proporcional, Baker [Bak85] y, posteriormente Whitley [Whi89], estudian los métodos de *ranking*. Con estos métodos, los individuos son ordenados en una lista de acuerdo a sus adecuaciones, y la probabilidad de selección de un cromosoma se obtiene en base a su posición en la lista ordenada.

En el ranking lineal, la probabilidad de selección de un individuo \mathbf{x}_i^t puede obtenerse como:

$$p_i^t = \frac{1}{m} \left(a_{max}^t - (a_{max}^t - a_{min}^t) \frac{rank(\mathbf{x}_i^t) - 1}{N - 1} \right)$$

donde $rank(\mathbf{x}_i^t)$ es la posición en la lista ordenada del individuo \mathbf{x}_i^t , y

los coeficientes a_{min}^t y a_{max}^t representan el número esperado de descendientes del peor y mejor individuo de la población en la generación t respectivamente.

Ranking no lineal

Otra posibilidad para realizar el ranking [Mic92] es utilizar un parámetro q introducido por el usuario y definir la probabilidad de selección de la forma:

$$p_i^t = q(1 - q)^{rank(\mathbf{x}_i^t) - 1}$$

Selección de estado estable (steady-state)

Como mecanismo de sustitución generacional alternativo a la sustitución generacional completa, podemos destacar la *selección de estado estable (steady-state)* [Sys89]. Con esta técnica, se selecciona un número $n \in [1 \dots N]$ de individuos los cuales serán sustituidos por n nuevos. Para la elección de los individuos que son sustituidos suele utilizarse un ranking inverso, es decir, ordenando la lista de cromosomas de menor a mayor adecuación.

Esta técnica supone una familia de estrategias de sustitución generacional. La sustitución generacional completa pertenece a esta familia con $n = N$. El modelo del factor de multitud [DeJ75] es también un caso particular con las variantes comentadas anteriormente, así como el algoritmo evolutivo modificado de Michalewicz [Mic92].

1.3.5 Esquemas de apareamiento

En [Gol89] podemos encontrar varios esquemas de apareamiento los cuales fueron estudiados previamente por Hollstien [Hol71]:

Apareamiento aleatorio. Todos los cromosomas tienen la misma probabilidad de apareamiento con cualquier otro.

Endogamia. Los individuos emparentados son apareados intencionadamente.

Línea de reproducción. Un único individuo con adecuación alta se aparea con una población base y los hijos son seleccionados como padres.

Outbreeding. Los individuos con un fenotipo marcadamente diferente son seleccionados como padres.

Auto-fertilización. Los individuos se aparean consigo mismos.

Apareamiento de concordancia positiva. Se aparean los individuos similares.

Apareamiento de concordancia negativa. Se aparean los individuos diferentes.

1.3.6 Operadores de cruce

Se han descrito multitud de variantes para los operadores de variación básicos en algoritmos evolutivos. Aunque tales variantes se han llevado a cabo también en representaciones binarias, es en las representaciones de alta cardinalidad en donde se observa una mayor diversidad de técnicas. Además se ha desarrollado una amplia gama de micro-operadores (operadores que actúan en un nivel cromosómico) de naturaleza distinta a los operadores de variación básicos de cruce y mutación, y macro-operadores (operadores que actúan a nivel de población).

Los operadores de cruce son operadores de variación binarios (toman como entrada dos individuos) y generan uno o dos individuos de salida. Se han desarrollado muchas variantes de este tipo de operadores, sobre todo para representaciones no binarias. En este apartado discutimos algunos de los operadores de cruce propuestos en la literatura.

Cruce multi-punto [DeJ75]

Este operador es una generalización del cruce simple en la cual se consideran un número mayor de puntos de cruce. Un parámetro CP determina el número de puntos de cruces, los cuales son elegidos aleatoriamente.

Cruce uniforme [Sys89]

Cada gen en el descendiente se crea copiando el correspondiente gen de un padre u otro, utilizando para ello una máscara de cruce generada aleatoriamente. Donde hay un 1 en la máscara, el gen en el descendiente se toma del primer padre, y donde hay un 0 el gen se toma del segundo padre.

Dados dos individuos i y j seleccionados como padres en la generación t para la operación de cruce con cromosomas $\mathbf{b}_i^t = (b_{i1}^t \dots b_{in}^t)$ y $\mathbf{b}_j^t = (b_{j1}^t \dots b_{jn}^t)$ se generan dos descendientes con cromosomas $\mathbf{b}_i^t = (b_{i1}^t \dots b_{in}^t)$ y $\mathbf{b}_j^t = (b_{j1}^t \dots b_{jn}^t)$ donde:

$$\begin{aligned} b_{il}^t &= b_{il}^t \quad y \quad b_{jl}^t = b_{jl}^t, \quad si \quad \alpha_l < 0.5 \\ b_{il}^t &= b_{jl}^t \quad y \quad b_{il}^t = b_{il}^t, \quad en \quad otro \quad caso \end{aligned}$$

siendo α_l una variable aleatoria en $[0, 1]$,

para $l = 1, \dots, n$.

Cruce adaptativo [Sch87]

La elección de los posibles puntos de cruce está especificada por una lista codificada en el cromosoma, la cual es modificada por los operadores de cruce y mutación.

Cruce PMX, OX y CX [Gol89]

Usados cuando se utilizan permutaciones como representación. Con los operadores PMX (*cruce parcialmente apareado*) y OX (*cruce ordenado*) se definen dos puntos entre los cuales se efectúa en cruce. Primero, se intercambian las subcadenas contenidas entre los puntos marcados. Segundo, las copias repetidas de alelos en el exterior de las subcadenas son eliminadas haciendo intercambios punto-a-punto, en el operador PMX, o por medio de movimientos deslizantes para completar los huecos, en el operador OX. El operador CX (*cruce cíclico*) actúa de forma diferente a los dos anteriores. Comienza fijando el primer alelo de ambos padres. El siguiente alelo que se fija en el primer padre es el que se fijó anteriormente en el segundo padre, y en

la nueva posición fijada para el primer padre se fija también el alelo en el segundo padre. Este proceso continúa hasta que se completa un ciclo, es decir, cuando un alelo aparece por segunda vez, y entonces cada cadena se completa con los alelos de la otra.

A continuación se discuten algunos de los operadores de cruce desarrollados para representación con números reales.

Cruce plano [Rad91]

Dados dos individuos i y j seleccionados como padres en la generación t para la operación de cruce con cromosomas: $\mathbf{b}_i^t = (b_{i1}^t \dots b_{in}^t)$ y $\mathbf{b}_j^t = (b_{j1}^t \dots b_{jn}^t)$ se genera el descendiente con cromosoma $\mathbf{b}_k^t = (b_{k1}^t \dots b_{kn}^t)$ donde:

b_{kl}^t , se genera aleatoria y uniformemente en el intervalo $[min_{kl}^t, max_{kl}^t]$

siendo $min_{kl}^t = \min(b_{il}^t, b_{jl}^t)$,

$max_{kl}^t = \max(b_{il}^t, b_{jl}^t)$, y

para $l = 1, \dots, n$.

Cruce lineal [Wri91]

Denotando los padres i y j como puntos p_i y p_j , se generan tres puntos nuevos:

$$\frac{1}{2}p_i + \frac{1}{2}p_j, \quad \frac{3}{2}p_i - \frac{1}{2}p_j, \quad y \quad -\frac{1}{2}p_i + \frac{3}{2}p_j$$

de los cuales se seleccionan finalmente como descendientes los dos mejores.

Cruce aritmético uniforme [Mic92]

Dados dos individuos i y j seleccionados como padres en la generación t para la operación de cruce con cromosomas $\mathbf{b}_i^t = (b_{i1}^t \dots b_{in}^t)$ y $\mathbf{b}_j^t = (b_{j1}^t \dots b_{jn}^t)$ se generan dos descendientes con cromosomas $\mathbf{b}'_i^t = (b'_{i1}^t \dots b'_{in}^t)$ y $\mathbf{b}'_j^t = (b'_{j1}^t \dots b'_{jn}^t)$ donde:

$$b'^t_{il} = ab^t_{il} + (1 - a) b^t_{jl}$$

$$b'^t_{jl} = ab^t_{jl} + (1 - a) b^t_{il}$$

para $l = 1, \dots, n$, y a constante.

Cruce aritmético no uniforme [Mic92]

Igual que el cruce aritmético uniforme excepto que a es una variable cuyo valor depende de la edad de la población.

Cruce BLX- α [Esh93]

Dados dos individuos i y j con cromosomas $\mathbf{b}_i^t = (b_{i1}^t \dots b_{in}^t)$ y $\mathbf{b}_j^t = (b_{j1}^t \dots b_{jn}^t)$ seleccionados como padres para la operación de cruce, se genera el descendiente con cromosoma $\mathbf{b}_k^t = (b_{k1}^t \dots b_{kn}^t)$, donde b_{kl}^t se genera aleatoria y uniformemente en el intervalo:

$$[\min_{kl}^t - \alpha I_{kl}^t, \max_{kl}^t + \alpha I_{kl}^t]$$

siendo $\min_{kl}^t = \min(b_{il}^t, b_{jl}^t)$,

$\max_{kl}^t = \max(b_{il}^t, b_{jl}^t)$, y

$I_{kl}^t = \max_{kl}^t - \min_{kl}^t$,

para $l = 1, \dots, n$.

El cruce plano es un caso particular del cruce BLX- α para $\alpha = 0.0$.

Cruce discreto [Muh93]

Dados dos individuos i y j con cromosomas $\mathbf{b}_i^t = (b_{i1}^t \dots b_{in}^t)$ y $\mathbf{b}_j^t = (b_{j1}^t \dots b_{jn}^t)$ seleccionados como padres para la operación de cruce, se genera el descendiente con cromosoma $\mathbf{b}_k^t = (b_{k1}^t \dots b_{kn}^t)$, donde:

b_{kl}^t se genera aleatoria y uniformemente en el conjunto $\{b_{il}^t, b_{jl}^t\}$,

para $l = 1, \dots, n$.

Cruce intermedio extendido [Muh93]

Dados dos individuos i y j con cromosomas $\mathbf{b}_i^t = (b_{i1}^t \dots b_{in}^t)$ y $\mathbf{b}_j^t = (b_{j1}^t \dots b_{jn}^t)$ seleccionados como padres para la operación de cruce, se genera el descendiente con cromosoma $\mathbf{b}_k^t = (b_{k1}^t \dots b_{kn}^t)$, donde:

$$b_{kl}^t = b_{il}^t + \alpha_l (b_{jl}^t - b_{il}^t)$$

con α_l generado aleatoria y uniformemente en el intervalo $[-0.25, 1.25]$,

para $l = 1, \dots, n$.

Cruce lineal extendido [Muh93]

Dados dos individuos i y j con cromosomas $\mathbf{b}_i^t = (b_{i1}^t \dots b_{in}^t)$ y $\mathbf{b}_j^t = (b_{j1}^t \dots b_{jn}^t)$ seleccionados como padres para la operación de cruce, se genera el descendiente con cromosoma $\mathbf{b}_k^t = (b_{k1}^t \dots b_{kn}^t)$, donde:

$$b_{kl}^t = b_{il}^t + \alpha_k (b_{jl}^t - b_{il}^t)$$

con α_k generado aleatoria y uniformemente en el intervalo $[-0.25, 1.25]$,

para $l = 1, \dots, n$.

1.3.7 Operadores de mutación

Los operadores de mutación son operadores de variación unarios, es decir, toman un individuo como entrada y generan otro individuo produciendo un cambio en el primero. Se han propuesto múltiples variantes para realizar la mutación, sobre todo con representaciones no binarias, en este apartado se describen algunos de los métodos propuestos en la literatura.

Mutación por intercambio [Sys91a]

Se intercambian dos elementos elegidos aleatoriamente de una permutación.

Mutación uniforme para representación real [Mic92, Rad91]

El operador de mutación uniforme comentado anteriormente para representaciones binarias puede ser definido también para representaciones reales cambiando el gen a mutar por un valor generado aleatoria y uniformemente en el dominio de la variable.

Dado el individuo i en la generación t con cromosoma $\mathbf{b}_i^t = (b_{i1}^t \dots b_{in}^t)$ y supuesto que el gen r -ésimo, b_{ir}^t , ha sido seleccionado para la mutación, el resultado es el descendiente con cromosoma $\mathbf{b}_i^t = (b_{i1}^t \dots b_{ir}^t \dots b_{in}^t)$ donde b_{ir}^t es un valor aleatorio en el dominio de la variable r -ésima $[l_r, u_r]$.

Mutación no uniforme para representación real [Mic92]

Para codificación real, este es uno de los operadores que permiten realizar un ajuste fino de los parámetros en las últimas iteraciones del algoritmo.

Dado el individuo i en la generación t con cromosoma $\mathbf{b}_i^t = (b_{i1}^t \dots b_{in}^t)$ y supuesto que el gen r -ésimo, b_{ir}^t , ha sido seleccionado para la mutación, el resultado es el descendiente con cromosoma $\mathbf{b}'_i^t = (b_{i1}^t \dots b'_{ir}{}^t \dots b_{in}^t)$ donde $b'_{ir}{}^t$ es, para representaciones reales:

$$b'_{ir}{}^t = \begin{cases} b_{ir}^t + \Delta(t, u_r - b_{ir}^t), & \text{si } \alpha = 0 \\ b_{ir}^t - \Delta(t, b_{ir}^t - l_r), & \text{si } \alpha = 1 \end{cases}$$

siendo α un valor booleano aleatorio,

u_r y l_r los límites superior e inferior del dominio de la variable r -ésima, y

$\Delta(t, y)$ una función que devuelve un valor en el rango $[0, y]$ de forma que la probabilidad de que éste sea cercano a cero aumente conforme el algoritmo avanza; en [Mic92] se toma:

$$\Delta(t, y) = y \left(1 - \beta^{(1 - \frac{t}{T})^b} \right)$$

donde β es un número aleatorio en el intervalo $[0, 1]$,

T es el número máximo de generaciones, y

b determina el grado de no uniformidad o dependencia con respecto a la generación actual t (en [Mic92] se toma $b = 2$).

Mutación no uniforme para representación binaria [Mic92]

La mutación no uniforme ha sido definida también para representaciones binarias en [Mic92]. En este caso el valor de $b'_{ir}{}^t$ se define de la forma:

$$b'_{ir}{}^t = \text{mutar}(b_{ir}^t, \nabla(t, k))$$

donde k es el número de bits por cada elemento de un cromosoma (en [Mic92] se toma $k = 30$),

$\text{mutar}(b_{ir}^t, pos)$ significa: mutar el valor de la posición pos en el elemento r -ésimo del individuo i en la iteración t , y

$\nabla(t, k)$ se define de la forma:

$$\nabla(t, k) = \begin{cases} \lfloor \Delta(t, k) \rfloor, & \text{si } \alpha = 0 \\ \lceil \Delta(t, k) \rceil, & \text{si } \alpha = 1 \end{cases}$$

siendo α un valor booleano aleatorio, y

b el parámetro de Δ ajustado de forma adecuada ([Mic92] toma el valor $b = 1.5$).

Mutación por desplazamiento [Dav91]

Se aumentan o disminuyen los genes en una cantidad aleatoria cuyo valor máximo lo define el usuario.

Dado el individuo i en la generación t con cromosoma $\mathbf{b}_i^t = (b_{i1}^t \dots b_{in}^t)$ y supuesto que el gen r -ésimo, b_{ir}^t , ha sido seleccionado para la mutación, el resultado es el descendiente con cromosoma $\mathbf{b}'_i^t = (b_{i1}^t \dots b'_{ir}^t \dots b_{in}^t)$ donde b'_{ir}^t es, para representaciones reales:

$$b'_{ir}^t = \begin{cases} b_{ir}^t + \beta, & \text{si } \alpha = 0 \\ b_{ir}^t - \beta, & \text{si } \alpha = 1 \end{cases}$$

donde α es un valor booleano aleatorio, y β un valor aleatorio entre 0 y β_r , siendo β_r un valor real positivo elegido por el usuario. Debe cumplirse que el valor final b'_{ir}^t se encuentre entre u_r y l_r , los límites superior e inferior respectivamente del dominio de la variable r -ésima, si no es así debe truncarse a este valor, es decir:

$$b'_{ir}^t = \begin{cases} \max \{b_{ir}^t + \beta, u_r\}, & \text{si } \alpha = 0 \\ \min \{b_{ir}^t - \beta, l_r\}, & \text{si } \alpha = 1 \end{cases}$$

1.3.8 Otros operadores

Además de los operadores de cruce y mutación, existen otros operadores de variación que actúan de forma diferente. [Gol89] realiza una discusión sobre los siguientes operadores:

Estructura diploide y dominación

En lugar de trabajar con una única estructura cromosómica (*haploide*), la estructura diploide utiliza un par de genes (*homólogos*) para codificar cada característica del fenotipo. Una estructura cromosómica diploide puede expresarse de la siguiente forma, suponiendo que las letras mayúsculas y minúsculas corresponden a los diferentes genes homólogos:

AbCDe

aBCde

En estructuras cromosómicas diploides, dos genes homólogos se decodificarán como dos valores diferentes del fenotipo. El operador de dominación decide qué genes se deben elegir para expresarse en el fenotipo, para ello se asigna a los genes la carac-

terística de *dominantes* o de *recesivos*, y se le da prioridad al gen dominante sobre el recesivo. La estructura cromosómica diploide anterior se decodifica de la siguiente forma, suponiendo que las letras mayúsculas corresponden a genes dominantes y las minúsculas a genes recesivos:

$$\begin{array}{l} \text{AbCDe} \\ \text{aBCde} \end{array} \rightarrow \text{ABCDe}$$

Para cada característica, tenemos dos tipos de combinaciones de genes homólogos:

- *Homocigóticos*. Los dos genes homólogos son iguales. En este caso no hay conflicto para decodificar el gen, puesto que son iguales. En el ejemplo anterior son homocigóticos $\text{CC} \rightarrow \text{C}$ y $\text{ee} \rightarrow \text{e}$.
- *Heterocigóticos*. Los dos genes homólogos sí son diferentes. En este caso se decodifica el gen dominante. En el caso anterior son heterocigóticos $\text{Aa} \rightarrow \text{A}$, $\text{bB} \rightarrow \text{B}$ y $\text{Dd} \rightarrow \text{D}$.

Se puede notar que los genes dominantes siempre se decodificarán, mientras que los recesivos sólo se decodificarán cuando se encuentren en una estructura homocigótica, acompañados de otro gen recesivo.

La dominación de un gen puede cambiar a lo largo de la evolución, y genes dominantes pueden pasar a ser recesivos y viceversa, lo que permite una adaptación más rápida ante posibles cambios en el entorno.

La estructura diploide y la dominación forman un mecanismo especialmente apropiado cuando un entorno cambia regularmente entre estados.

Inversión y otros operadores de reordenación

Los operadores de reordenación no modifican el valor del cromosoma, sino su estructura. El operador de reordenación más importante es el operador de inversión. Dadas dos posiciones elegidas aleatoriamente en un cromosoma, el operador de inversión crea un nuevo cromosoma en el cual los genes comprendidos entre esas dos posiciones quedan invertidos con respecto al cromosoma original. Es decir, dado un individuo:

$$\mathbf{b} = (b_1 \dots b_{pos_1} \dots b_{pos_2} \dots b_n)$$

se generan dos números enteros aleatorios:

$$pos_1, pos_2 \in [1 \dots, n], \quad pos_1 < pos_2$$

y el individuo anterior es sustituido por:

$$\mathbf{b}' = (b_1 \dots b_{pos_2} \dots b_{pos_1} \dots b_n)$$

El objetivo de este operador es realizar una reordenación de los genes. En la representación binaria simple el valor de un gen está asociado a su posición dentro del cromosoma, por lo que cambiar la posición modifica también su valor; sin embargo, este operador no pretende cambiar el valor del gen, sino solamente su posición, por tanto para utilizar este operador es necesario hacerlo sobre una representación donde el valor del gen sea independiente de su posición.

Otros operadores que también realizan una reordenación son los operadores PMX, OX y CX ya vistos en el apartado anterior.

Los operadores de reordenación aparecen con el objetivo de propagar de manera más eficiente los bloques constructivos en un algoritmo evolutivo. Los operadores anteriores tratan de optimizar el valor del cromosoma; los operadores de reordenación, sin embargo, tratan de optimizar la propia estructura del cromosoma.

Diferenciación sexual

Se puede determinar el sexo de un individuo añadiendo un nuevo elemento a la estructura cromosómica. Pueden existir dos o más sexos dentro de una especie.

La diferenciación sexual divide una especie en dos o más grupos cooperativos, cada uno de los cuales se especializa en algún aspecto concreto, de forma que se cubre un rango de necesidades para la supervivencia mayor que con una simple población competidora.

Formación de especies y nichos

Una especie es una subpoblación estable de individuos que actúan en un particular subdominio de una función o nicho. Intuitivamente podemos ver una especie como una clase de organismos con características comunes, y un nicho como una regla del

entorno o tarea del organismo. En [Gol89] se observa la formación de especies y nichos como macro-operadores que actúan a nivel de la población.

La formación de especies y nichos es particularmente adecuada para obtener una diversidad apropiada en problemas con función objetivo multimodal, en los cuales se quiere localizar los picos, y en problemas de optimización con múltiples objetivos. Se han descrito numerosas técnicas para la formación de especies y nichos [Deb89, Gol87, Gol89], tanto de forma implícita, como el modelo del factor de multitud (crowding) comentado anteriormente, como de forma explícita. A continuación comentamos algunas de estas técnicas.

Preselección. En este esquema de formación implícita de especies y nichos, un descendiente sustituye a su peor padre si la adecuación del descendiente es mejor. De esta forma se mantiene diversidad ya que los individuos tienden a sustituir individuos similares a sí mismos (a uno de sus padres).

Crowding. Este modelo, ya discutido en el apartado anterior, es una generalización del esquema de preselección, es por tanto otra técnica implícita de formación de especies y nichos.

Participación (sharing). Una forma explícita para inducir especies y nichos es definir una *función de participación* para determinar la vecindad y grado de participación de cada par de individuos en la población. La función de participación da una medida de participación entre dos individuos. Así, el valor de participación de dos individuos parecidos deberá de ser alto (cercano a 1), y el valor de participación de dos individuos distantes deberá de ser bajo (cercano a 0). Podemos establecer entonces la función de participación de dos individuos i, j como:

$$participación_{ij} = \begin{cases} 1 - \frac{d_{ij}}{\sigma_{participación}}, & \text{si } d_{ij} < \sigma_{participación} \\ 0, & \text{en otro caso} \end{cases}$$

donde $d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|$ es la distancia entre los fenotipos de los individuos i y j , y $\sigma_{participación}$ es la distancia máxima permitida entre los fenotipos de los individuos para que formen parte de un mismo nicho.

El valor de participación en la población de un individuo o *cuenta de nicho* es un valor de la participación de un individuo con respecto a todos los demás indi-

viduos de la población, normalmente calculada como un sumatorio de la función de participación. Un individuo aislado tendrá una cuenta de nicho muy baja, mientras que un individuo que tenga muchos individuos cercanos tendrá una cuenta de nicho muy alta. Para un individuo dado i , su grado de participación $nicho_i$ en la población se determina sumando los valores de la función de participación entre el individuo y todos los restantes individuos en la población, es decir:

$$nicho_i = \sum_{j=1}^N participación_{ij}$$

El valor de adecuación de un individuo queda modificado por su cuenta de nicho, de forma que se decrementa la adecuación de aquellos individuos que tengan muchos individuos cercanos, mientras que se potencia la adecuación de los individuo que se encuentren aislados. La adecuación final f' de los individuos se determina dividiendo la adecuación potencial por el grado de participación en la población:

$$f'_i = \frac{f_i}{nicho_i}$$

con lo cual se limita el crecimiento incontrolado de especies particulares en la población.

La preselección, el modelo de crowding y la formación de nicho son técnicas de gran interés para resolver problemas de optimización con múltiples objetivos, por lo que se discuten detalladamente en el siguiente capítulo.

1.3.9 Técnicas basadas en el conocimiento

Los algoritmos evolutivos en su forma más simple son algoritmos ciegos de búsqueda; utilizan únicamente la codificación y el valor de la función objetivo para determinar qué soluciones se mantienen en la siguiente generación. Esta indiferencia hacia la información específica del problema dota a los algoritmos evolutivos de una gran robustez y versatilidad, pero, sin embargo, el no usar todo el conocimiento disponible de un problema particular pone a los algoritmos evolutivos en desventaja competitiva con respecto a los métodos que sí usan tal información. Con esta motivación, en [Dav91, Gol89] se discuten varias formas de combinar información específica del problema con algoritmos evolutivos. A continuación se discuten algunas de estas técnicas.

Usar la codificación propia del problema

La utilización de una representación natural de las soluciones del problema permite representar el conocimiento del experto de igual forma a como éste lo observa. El esquema de representación juega un papel muy importante en los algoritmos evolutivos, puesto que puede limitar la forma en la que el sistema observa su entorno, lo que hace que en muchos problemas prácticos se requieran representaciones más potentes que la binaria, la que, a su vez, puede resultar de difícil manejo al tratarse de una representación no natural para el problema.

Hibridación

Los algoritmos evolutivos pueden ser combinados con las técnicas de búsqueda específicas del problema para formar un híbrido que explote la perspectiva global de los algoritmos evolutivos y la convergencia de las técnicas específicas del problema. Se han sugerido varias formas de hibridar un algoritmo evolutivo:

- Un primer método de hibridación, mostrado en la figura 1.3(a), consiste en incorporar soluciones obtenidas con las técnicas específicas del problema a la población inicial del algoritmo evolutivo, lo que, combinado con el elitismo, garantiza que no se obtengan soluciones peores a las obtenidas con las técnicas específicas.
- Otro método, sugerido en [Hol75], consiste en utilizar los algoritmos evolutivos como un preprocesador para llevar a cabo una búsqueda inicial, para después explotar el conocimiento específico del problema que guíe una búsqueda local. Esto puede conseguirse con un esquema de algoritmo evolutivo híbrido en batería, como ilustra la figura 1.3(b).

Adaptar los operadores de variación

El uso de una representación natural del problema implica tener que adaptar los operadores de variación a la nueva representación, intentando seguir una analogía con los operadores estudiados para representaciones binarias, o creando nuevos operadores de variación con conocimiento incorporado.

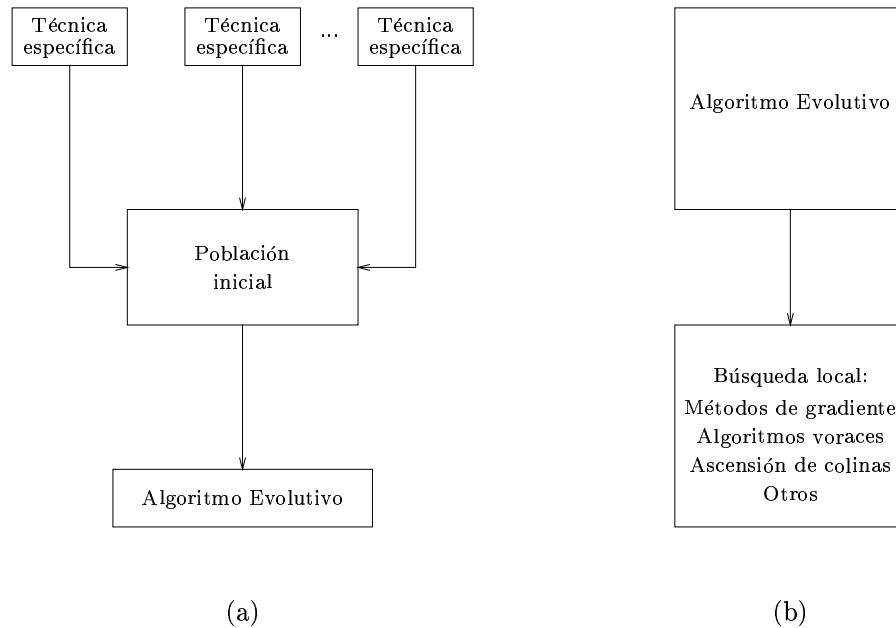


Figura 1.3: Dos esquemas de algoritmos evolutivos híbridos.

Operadores de variación con conocimiento incorporado

Otra forma de incorporar el conocimiento específico del problema para aumentar la velocidad de búsqueda de los algoritmos evolutivos es usar tal información para que los operadores de variación guíen el algoritmo más directamente hacia mejores soluciones, o dicho de otra forma, usar técnicas conocidas de búsqueda (como los *métodos de gradiente*, los *algoritmos voraces*, la *ascensión de colinas*, etc.) o variantes de éstas como operadores de variación en el algoritmo evolutivo.

Métodos de aproximación de la función de adecuación

En muchos problemas se puede utilizar conocimiento específico para construir modelos aproximados que hacen posible calcular aproximaciones más o menos precisas de la función objetivo con un coste computacional mucho menor que realizar el cálculo completo de la función objetivo. Esto puede resultar especialmente para aquellos problemas de optimización en los que la evaluación de la función resulta un proceso bastante costoso, con varios niveles de subrutinas, cálculos numéricos o simbólicos y

varias funciones de codificación y decodificación. Utilizando los modelos aproximados para reducir el número de evaluaciones completas se pueden realizar más evaluaciones en el mismo tiempo, mejorando el resultado obtenido.

1.4 Sumario

En este capítulo se ha realizado una revisión de las técnicas evolutivas más importantes. La robustez y el paralelismo implícito son dos características de los algoritmos evolutivos que los hacen especialmente idóneos para resolver algunos problemas. La versatilidad de diseño de los algoritmos evolutivos permite a éstos adaptarse para resolver problemas muy diferentes. Las técnicas para mantener la diversidad evitan la convergencia previa y hacen posible explorar todo el espacio de soluciones. Mediante el uso de poblaciones se mantienen múltiples soluciones simultáneamente, lo que resulta especialmente adecuado para resolver problemas multimodales y problemas multiobjetivo. El elitismo es otra técnica utilizada en los algoritmos evolutivos que asegura mantener siempre la mejor solución; como se verá en el siguiente capítulo, el elitismo resulta fundamental para resolver problemas multiobjetivo. Las técnicas evolutivas han encontrado un campo de aplicación muy importante dentro de la optimización multiobjetivo. El trabajo desarrollado en esta memoria trata de resolver problemas de este tipo y para ello utiliza técnicas evolutivas como método para encontrar soluciones. La discusión realizada en este capítulo ha permitido conocer las técnicas evolutivas más importantes para poder utilizar posteriormente aquellas que resulten más interesantes para desarrollar nuestro trabajo.

Capítulo 2

Optimización Multiobjetivo

Los problemas de *optimización* consisten en encontrar una o más soluciones que correspondan a los valores extremos de una o más funciones objetivo.

Cuando un problema de optimización tiene un único objetivo, se denomina *optimización simple*. Existen muchos algoritmos que resuelven problemas de optimización simple utilizando técnicas de búsqueda determinista basadas en gradiente o heurísticas. Otros algoritmos realizan una búsqueda estocástica, por ejemplo, los algoritmos evolutivos y el enfriamiento simulado. Estos últimos algoritmos, para ciertos problemas, encuentran soluciones óptimas globales de forma más fiable.

Cuando un problema de optimización tiene más de una función objetivo, entonces se llama *optimización multiobjetivo*. La mayoría de problemas de optimización del mundo real tienen múltiples objetivos, los cuales suelen entrar en conflicto; es decir, mejorar un objetivo significa empeorar otro. Por ejemplo, en la fabricación de un producto se intenta minimizar el coste y maximizar la calidad; evidentemente ambos objetivos son incompatibles entre ellos. Entre dos soluciones que optimicen dos objetivos diferentes, no se puede decidir qué solución es mejor; de hecho, ambas soluciones son óptimas. Esta es la diferencia fundamental entre problemas de optimización simple y multiobjetivo; en optimización simple existe una única solución óptima, mientras que en optimización multiobjetivo existen múltiples soluciones. El procedimiento ideal de optimización multiobjetivo es, por tanto, de la siguiente forma:

1. Encontrar múltiples soluciones óptimas dentro de un rango de valores objetivos.
2. Escoger una de las soluciones obtenidas utilizando un criterio de preferencia.

Los algoritmos tradicionales que resuelven problemas de optimización multiobjetivo transforman el problema de optimización multiobjetivo en un problema de optimización simple utilizando algún criterio de preferencias y encuentran, por tanto, una única solución de compromiso; es decir, una solución con valores aceptables en todos los objetivos según el criterio de preferencias que se ha establecido. Si deseamos obtener más de una solución es necesario ejecutar repetidamente el algoritmo modificando dicho criterio de preferencias.

La gran ventaja de utilizar algoritmos evolutivos en optimización multiobjetivo es que permiten obtener múltiples soluciones óptimas en una sola ejecución del algoritmo. Un algoritmo evolutivo es capaz de obtener múltiples soluciones óptimas en su población final. Una vez tenemos el conjunto de soluciones óptimas, podemos escoger aquella solución que más nos satisfaga según algún criterio de preferencias.

Otro punto importante en optimización es la definición de *restricciones*, es decir, imponer ciertas condiciones que una solución debe cumplir para poder ser aceptada. En la mayoría de problemas de optimización del mundo real, es inevitable que surjan una serie de restricciones. Los algoritmos evolutivos también han resultado eficaces para resolver problemas de optimización simple y multiobjetivo con restricciones.

La optimización multiobjetivo utilizando algoritmos evolutivos es, por tanto, un tema de investigación de gran actualidad, como lo demuestran la gran cantidad de publicaciones a las que ha dado lugar en los últimos años. Dentro de este campo, destacan los libros escritos por Deb [Deb01c] y por Coello [Coe02] como referencia fundamental. El trabajo realizado en esta memoria se encuadra dentro de la optimización multiobjetivo con restricciones mediante algoritmos evolutivos.

En este capítulo se realiza una revisión de las técnicas más importantes dentro de optimización multiobjetivo. En la sección 2.1 se muestran algunos de los conceptos fundamentales en optimización multiobjetivo; la sección 2.2 recoge algunos de los métodos tradicionales para resolver problemas de optimización multiobjetivo; en la sección 2.3 se muestran las técnicas evolutivas utilizadas para manejar restricciones en problemas de optimización simple; los métodos evolutivos para resolver problemas de optimización multiobjetivo se muestran en la sección 2.4 y la sección 2.5 expone técnicas evolutivas para tratar problemas de optimización multiobjetivo con restricciones. Por último, la sección 2.6 recoge un resumen del estudio realizado a lo largo de todo el capítulo.

2.1 Problemas de Optimización Multiobjetivo

2.1.1 Descripción del problema

La optimización multiobjetivo (también llamada optimización multicriterio o de vectores) puede ser definida como el problema de encontrar un vector de variables de decisión que satisfaga unas restricciones y optimice una función vector cuyos elementos representan las funciones objetivo. Estas funciones forman una descripción matemática de los criterios de rendimiento, los cuales usualmente están en conflicto unos con otros. Por tanto, el término *optimizar* significa encontrar una solución que obtenga, para todas las funciones objetivos, valores aceptables.

Consideramos el siguiente problema de optimización multiobjetivo:

$$\begin{aligned} & \text{Minimizar} && f_i(\mathbf{x}), && i = 1, \dots, n \\ & \text{sujeto a :} && g_j(\mathbf{x}) \leq 0, && j = 1, \dots, m \end{aligned} \quad (2.1)$$

donde $\mathbf{x} = (x_1, \dots, x_p)$ es un vector de parámetros reales $x_k \in \mathbb{R}$ pertenecientes a un dominio $[l_k, u_k]$, $k = 1, \dots, p$, y $f_i(\mathbf{x})$, $g_j(\mathbf{x})$, son funciones arbitrarias lineales o no lineales. Podemos notar que no nos estamos restringiendo solamente a problemas de minimización con restricciones del tipo menor o igual que cero, puesto que:

- Los problemas de maximización y minimización pueden resolverse de forma equivalente;
- Una restricción menor o igual $g_j(\mathbf{x}) \leq b_j$ puede ser reescrita como $g_j(\mathbf{x}) - b_j \leq 0$;
- Una restricción mayor o igual $g_j(\mathbf{x}) \geq b_j$ puede ser reescrita como $b_j - g_j(\mathbf{x}) \leq 0$;
- Una restricción de igualdad $g_j(\mathbf{x}) = b_j$ puede ser representada por dos restricciones $g_j(\mathbf{x}) - b_j \leq 0$ y $-g_j(\mathbf{x}) + b_j \leq 0$.

Se llama \mathcal{F} al *espacio de soluciones factibles* o *región factible*, formado por aquellas soluciones \mathbf{x} que satisfacen las restricciones $g_j(\mathbf{x}) \leq 0$ y \mathcal{S} es el *espacio completo de búsqueda* formado por todas las posibles soluciones \mathbf{x} . El espacio de búsqueda \mathcal{S} suele definirse como un rectángulo p -dimensional en \mathbb{R}^p . Es evidente que $\mathcal{F} \subseteq \mathcal{S}$.

Es posible incluir restricciones de igualdad, puesto que una restricción de la forma $g_j(\mathbf{x}) = 0$ es equivalente a incluir dos restricciones de la forma $g_j(\mathbf{x}) \leq 0$ y

$-g_j(\mathbf{x}) \leq 0$. En la práctica, una igualdad suele reemplazarse por dos desigualdades de la forma $g_j(\mathbf{x}) - \delta \leq 0$ y $-g_j(\mathbf{x}) - \delta \leq 0$ para algún valor pequeño $\delta > 0$. Asumiremos, por tanto, que el conjunto de restricciones consiste por tanto en m desigualdades.

Para una solución dada $\mathbf{x} \in \mathcal{F}$, las restricciones que satisfacen $g_j(\mathbf{x}) = 0$ se llaman *restricciones activas en \mathbf{x}* .

Los problemas de optimización multiobjetivo se caracterizan por tener que optimizar diversas medidas (los objetivos) que pueden ser independientes y que suelen estar en conflicto unas con otras. Los múltiples objetivos, por tanto, establecen en el espacio de búsqueda un orden parcial, no total. De hecho, encontrar el óptimo global de un problema de optimización multiobjetivo es un problema NP-completo. Las soluciones óptimas, donde se satisfacen todas las restricciones y las funciones objetivo alcanzan mínimos globales, pueden ni siquiera existir.

2.1.2 Concepto de óptimo Pareto

El concepto de *Pareto óptimo* o *Pareto optimalidad* se formula por Vilfredo Pareto [Par96] en el siglo XIX y constituye por sí mismo el origen de la investigación en optimización multiobjetivo.

Se dice que una solución $\mathbf{x} \in \mathcal{F}$ *domina* a otra solución $\mathbf{x}' \in \mathcal{F}$ si:

$$\begin{aligned} f_i(\mathbf{x}) &\leq f_i(\mathbf{x}') \quad \text{para todo } i = 1, \dots, n \\ \text{y } f_i(\mathbf{x}) &< f_i(\mathbf{x}') \quad \text{para algún } i = 1, \dots, n \end{aligned}$$

Una solución $\mathbf{x} \in \mathcal{F}$ es *Pareto óptima* si, para todo $\mathbf{x}' \in \mathcal{F}$:

$$\begin{aligned} \text{o bien } f_i(\mathbf{x}) &= f_i(\mathbf{x}') \quad \text{para todo } i = 1, \dots, n \\ \text{o bien } f_i(\mathbf{x}) &< f_i(\mathbf{x}') \quad \text{para algún } i = 1, \dots, n \end{aligned}$$

Es decir, una solución $\mathbf{x} \in \mathcal{F}$ es *Pareto óptima* en el si no existe ninguna otra solución $\mathbf{x}' \in \mathcal{F}$ que mejore a \mathbf{x} en algún criterio sin empeorarlo simultáneamente en otro criterio. Dicho de otra forma, una solución $\mathbf{x} \in \mathcal{F}$ es Pareto óptima si no existe ninguna otra solución $\mathbf{x}' \in \mathcal{F}$ que la domine.

Las soluciones Pareto óptimas también se denominan soluciones no inferiores, admisibles, eficientes o no dominadas.

El conjunto de soluciones para el problema (2.1) está compuesto por todos

aquellos elementos del espacio de solución para los cuales el correspondiente vector de objetivos no puede ser simultáneamente mejorado en todos sus componentes, es decir las soluciones *no dominadas*, *no inferiores*, o *Pareto-óptimos*.

2.1.3 Frente Pareto

Los óptimos Pareto se encuentran en el límite de la región factible, o en el lugar de los puntos tangentes a las funciones objetivo. A este conjunto de puntos se denomina *frente Pareto*. En general, no resulta sencillo encontrar expresiones analíticas de la línea o superficie que contiene el frente Pareto y normalmente se calculan los puntos Pareto y los valores en las funciones objetivo. Cuando tenemos una cantidad suficiente de puntos calculados, se puede decidir qué punto o puntos son los que nos interesan.

2.2 Optimización Multiobjetivo mediante Métodos Clásicos

Existen numerosos enfoques de resolución para problemas con múltiples objetivos. Los procedimientos existentes para el análisis de los problemas multiobjetivo pueden clasificarse en dos categorías generales. Estos métodos, o bien generan el conjunto de todas las soluciones no dominadas (solución completa), o bien construyen una solución de compromiso.

Un primer grupo de problemas multiobjetivo son los llamados *problemas multiobjetivo lineales*; este grupo está formado por aquellos problemas cuyas funciones objetivo $f_i(\mathbf{x})$ y restricciones $g_j(\mathbf{x})$ son lineales, tal como se indica en la siguiente ecuación:

$$\begin{aligned} \text{Minimizar} \quad & f_i(\mathbf{x}) = \sum_{k=1}^p c_{ik}x_k, \quad i = 1, \dots, n \\ \text{sujeto a :} \quad & g_j(\mathbf{x}) = \sum_{k=1}^p a_{jk}x_k = b_j, \quad j = 1, \dots, m \end{aligned}$$

Notar que cualquier problema multiobjetivo lineal puede ser escrito de la forma expresada en la ecuación anterior, puesto que si tenemos desigualdades en las restricciones, éstas pueden ser convertidas en igualdades añadiendo variables.

En un problema multiobjetivo lineal, el conjunto de soluciones factibles \mathcal{F} forma un poliedro; para resolver dicho problema se buscan, en primer lugar, las *soluciones no dominadas básicas* o *soluciones extremas* pertenecientes a \mathcal{F} , que son aquellos

puntos no dominados que forman los extremos del poliedro; el conjunto completo de soluciones no dominadas está formado por las caras no dominadas del poliedro.

Diversos autores han propuesto métodos para la resolución de problemas multiobjetivo lineales. Entre ellos se encuentra Ralph E. Steuer [Ste74] que desarrolla el algoritmo ADBASE para calcular todas las soluciones no dominadas básicas de un problema multiobjetivo lineal. Isermann [Ise77] propone otro algoritmo para calcular tanto las soluciones básicas como las caras no dominadas del poliedro de soluciones factibles en un problema multiobjetivo lineal. Zeleny [Zel76] desarrolla el *método multicriterio simplex*. Esta técnica genera todos los puntos no dominados extremos para un problema multiobjetivo lineal, el cual se debate más adelante. Entre las técnicas que utilizan el método anterior se encuentran la *descomposición multiparamétrica*, *programación fraccional* y *programación por metas*.

Cuando las funciones objetivo o las restricciones no son lineales, tenemos *problemas multiobjetivo no lineales* y las técnicas anteriores no pueden aplicarse. Para resolver este tipo de problemas se ha desarrollado la *programación de compromiso*. Zeleny [Zel73] introduce los primeros conceptos sobre programación de compromiso. La mayoría de estos métodos resuelven los problemas de optimización multiobjetivo reduciendo el vector de objetivos a uno sólo, con la ayuda de algún conocimiento del problema.

2.2.1 Método simplex multiobjetivo

El *método simplex multiobjetivo* desarrollado por Zeleny [Zel76] permite en problemas multiobjetivo lineales localizar todos los puntos extremos no dominados. A partir de estos puntos es posible identificar los segmentos o planos no dominados del conjunto de soluciones factibles \mathcal{F} .

Para resolver el problema de optimización multiobjetivo lineal mediante el método simplex multiobjetivo, en primer lugar es necesario crear la *tabla general del método simplex multiobjetivo* (tabla 2.2.1). Esta tabla es similar a la tabla convencional utilizada en el método simplex para resolución de problemas lineales con un único objetivo, con la diferencia de tener múltiples filas objetivo. Todas las variables x_1, \dots, x_p se dividen en dos grupos: m variables básicas x_1, \dots, x_m y $(p-m)$ variables no básicas x_{m+1}, \dots, x_p , cuyos valores por definición son igual a cero, pues al tener m restricciones y p variables, sólo m variables pueden ser positivas, mientras que las

Variables básicas actuales	Variables básicas			Variables no básicas			Valores de las variables básicas
	x_1	\dots	x_m	x_{m+1}	\dots	x_p	
x_1	1	\dots	0	$y_{1(m+1)}$	\dots	y_{1p}	x_i^0
\vdots	\vdots		\vdots	\vdots		\vdots	\vdots
x_m	0	\dots	1	$y_{m(m+1)}$	\dots	y_{mp}	x_m^0
Filas de objetivos	0	\dots	0	$z_{1(m+1)}$	\dots	z_{1p}	$f_1(\mathbf{x}^0)$
	\vdots		\vdots	\vdots		\vdots	\vdots
	0	\dots	0	$z_{n(m+1)}$	\dots	z_{np}	$f_n(\mathbf{x}^0)$

Tabla 2.1: Tabla general del método simplex multiobjetivo

restantes $(p-m)$ variables son igual a cero. Las soluciones básicas se identificarán mediante superíndices, que indicarán el orden en el cual serán generadas. Por ejemplo, $x^0 = (x_1^0, \dots, x_p^0)$, $x^1 = (x_1^1, \dots, x_p^1)$ y así sucesivamente.

Los valores y_{jk} con $j = 1, \dots, m$ y $k = 1, \dots, p$ pueden interpretarse como el ratio de substitución entre las variables x_j y x_k . Los valores a_{jk} con $j = 1, \dots, m$ y $k = 1, \dots, p$ indican el efecto del objetivo j -ésimo en la variable x_k . Dicho valor se calcula de la siguiente forma:

$$z_{jk} = \sum_{i=1}^m c_{ji} y_{ik} - c_{jk}$$

Realizando sucesivas iteraciones en la tabla del método simplex multiobjetivo se puede encontrar un conjunto de soluciones no dominadas, a partir de las cuales es posible identificar los segmentos o planos no dominados.

2.2.2 Descomposición multiparamétrica

La descomposición multiparamétrica es un procedimiento interactivo que soluciona en parte el problema de la determinación a priori de pesos. En lugar de optimizar las funciones objetivo $f_1(\mathbf{x}), \dots, f_m(\mathbf{x})$ como entradas independientes, se combinan utilizando una función de agregación multiparamétrica. En tal caso, el problema multiparamétrico lineal es el siguiente:

$$\text{Minimizar } f(\boldsymbol{\lambda}, \mathbf{x}) = \sum_{i=1}^m \lambda_i f_i(\mathbf{x}) = \boldsymbol{\lambda} f(\mathbf{x})$$

donde $\boldsymbol{\lambda} = (\lambda_1, \dots, \lambda_m)$ es un vector de parámetros de peso tales que $\lambda_i \geq 0$ para todo $i = 1, \dots, m$ y $\sum_{i=1}^m \lambda_i = 1$. Se llama Λ al conjunto de todos los vectores $\boldsymbol{\lambda}$ que cumplen las condiciones anteriores.

Se puede calcular el conjunto de soluciones no dominadas optimizando $f(\boldsymbol{\lambda}, \mathbf{x})$ para todo vector $\boldsymbol{\lambda} \in \Lambda$. Si minimizamos $f(\boldsymbol{\lambda}, \mathbf{x})$ sobre un poliedro convexo, cada punto no dominado extremo estará asociado a un vector $\boldsymbol{\lambda}$, de forma que $f(\boldsymbol{\lambda}, \mathbf{x})$ alcanza su mínimo precisamente en dicho punto. La ventaja de este método es que nos permite resolver el problema para todo $\boldsymbol{\lambda} \in \Lambda$ de una sola vez.

Para problemas lineales, se puede resolver la descomposición multiparamétrica utilizando el método simple multiobjetivo, tal como ha sido estudiado por Yu y Zeleny [Yu76].

2.2.3 Programación fraccional

En muchos problemas prácticos es normal encontrar funciones objetivo definidas como ratios, cuotas, proporciones o fracciones. Esto provoca que las funciones objetivo dejen de ser lineales. Este tipo de problemas pueden resolverse mediante *programación fraccional*. La programación fraccional multiobjetivo es desarrollada por Tigan [Tig75] y posteriormente ampliada por Kornbluth [Kor79].

2.2.4 Programación por metas

La *programación por metas* consiste en minimizar el conjunto de desviaciones de unas *metas preestablecidas*. En general, las funciones objetivo y restricciones lineales pueden ser escritas de la siguiente forma:

$$\begin{aligned} f_i(\mathbf{x}) &= \sum_{k=1}^p c_{ik} x_k + d_i^- - d_i^+ = b_i, \quad i = 1, \dots, n \\ g_j(\mathbf{x}) &= \sum_{k=1}^p a_{jk} x_k = e_j, \quad j = 1, \dots, m \end{aligned}$$

donde d_i^- indica las desviaciones negativas, d_i^+ indica las desviaciones positivas y b_i son las m metas de las funciones objetivo o valores de las restricciones.

Si se trata de una restricción, debe alcanzarse el valor e_j , si se trata de una meta, entonces debe conseguirse un valor lo más cercano posible, es decir, debe minimizarse el valor de las variables d_i^- y d_i^+ .

Existen tres aproximaciones básicas para este tipo de programación: metas preestablecidas, metas arquimedianas o no preestablecidas y múltiples metas. Estas tres aproximaciones se diferencian tan sólo en el manejo de las funciones objetivo.

Programación por Metas Preferentes

Minimizar los objetivos $f(d_i^-, d_i^+)$ en orden lexicográfico según la jerarquía determinada por los valores P_1, \dots, P_m , llamados *pesos preferentes* o *pesos de prioridad*. Los objetivos con mayor nivel de prioridad deben satisfacerse primero y sólo entonces pueden considerarse los objetivos de menor prioridad.

Cuando el problema multiobjetivo es lineal podemos utilizar el método simplex; resolver un problema por este método es más sencillo que por el método simplex multiobjetivo. Para ello se utiliza *tabla de objetivos preferentes* muy similar a la tabla general del método simplex multiobjetivo (tabla 2.2.1). La diferencia principal con el método simplex multiobjetivo es que en este caso no se consideran todas las filas de objetivos simultáneamente, sino que se considera cada vez una sola, de acuerdo con los niveles de prioridad o preferencia P_i . Esta aproximación ha sido la utilizada en los trabajos de Lee [Lee72] e Ignizio [Ign76]. Este último resuelve, además, problemas con funciones objetivo no lineales.

Los problemas de programación por metas preferentes pueden resolverse de forma más eficiente teniendo en cuenta el hecho de que estamos tratando con un conjunto de problemas lineales ordenados. Arthur y Ravindran [Art70] desarrollan un *algoritmo de partición* que consiste en resolver el conjunto de problemas lineales, tomando la solución del problema de mayor prioridad como solución inicial para el problema de menor prioridad.

Los pesos preferentes no sólo se aplican a las funciones objetivos, sino también a las restricciones. Esto nos permite considerar sólo aquellas restricciones y variables que son relevantes de forma inmediata al problema en un nivel dado de prioridad.

El principal problema de utilizar metas preferentes es que falla cuando se trata de identificar soluciones sin límites, es decir, cuando una función objetivo puede ser arbitrariamente grande.

Programación por metas arquimedianas o no preferentes

En esta aproximación se trata de minimizar la suma de las funciones f_i elevadas a una potencia p y ponderadas con un peso w_i tal como se expresa en la siguiente ecuación:

$$\text{Minimizar } \sum_{i=1}^n w_i [f_i(d_i^-, d_i^+)]^p$$

Esta suma nos da la medida de la distancia total a las metas establecidas. Se consideran todos los objetivos f_i al mismo tiempo, p puede tomar cualquier valor, típicamente 1 ó 2. Este enfoque es el utilizado en el trabajo de Charnes y Cooper [Char61].

Programación por múltiples metas

La programación por múltiples metas consiste en minimizar $[f_1(d_1^-, d_1^+), \dots, f_n(d_n^-, d_n^+)]$ en el sentido vectorial, es decir, identificar todas las soluciones no dominadas con respecto a las funciones objetivo $f_i(d_i^-, d_i^+)$ tal y como se realiza en programación multiobjetivo lineal. En este caso no existe ningún criterio de peso, ni se define ninguna función de agregación.

2.2.5 Programación de compromiso

La programación de compromiso es una metodología mucho más general que permite solucionar problemas multiobjetivo no lineales y está basada en el concepto de *distancia a una solución ideal*.

El concepto de *solución ideal* indica los valores extremos óptimos que pueden ser alcanzados por cada uno de los objetivos. Si $\mathcal{F} = \{\mathbf{x}^1, \dots, \mathbf{x}^K\}$ es el conjunto de soluciones factibles, entonces $y_i^k = f_i(\mathbf{x}^k)$ indica el valor de la solución k -ésima para el objetivo i -ésimo. La solución ideal $\mathbf{y}^* = (y_1^*, \dots, y_m^*)$ viene dada por:

$$y_i^* = \min_{k=1, \dots, K} y_i^k, \quad i = 1, \dots, m$$

Si tenemos m objetivos, la *distancia* de una solución dada $\mathbf{y} = (y_1, \dots, y_m)$ a una solución ideal $\mathbf{y}^* = (y_1^*, \dots, y_m^*)$ puede expresarse de forma generalizada como se indica a continuación:

$$d_p = \left(\sum_{i=1}^m \lambda_i^p (y_i^* - y_i)^p \right)^{1/p}$$

donde $\lambda_i > 0$, para $i = 1, \dots, m$, y $\sum_{i=1}^m \lambda_i = 1$ son los pesos que indican el nivel de contribución de cada objetivo a la distancia total; p es un valor que puede variar entre 1 y ∞ , cuando $p = 1$ indica la distancia más larga entre dos puntos en el sentido geométrico, si $p = 2$ es la distancia más corta, o en línea recta; y para $p = \infty$ la expresión anterior es $d_\infty = \max_{i=1, \dots, m} \{\lambda_i (y_i^* - y_i)\}$. Puede observarse que en este caso la mayor de las desviaciones es la que determina el valor de la distancia. En general, cuanto mayor es el valor p , más importancia tienen las desviaciones mayores para formar la distancia total. El caso contrario lo tenemos con valores de p negativos, donde las desviaciones menores tienen mayor relevancia hasta llegar al caso extremo $d_{-\infty} = \min_{i=1, \dots, m} \{\lambda_i (y_i^* - y_i)\}$.

Zeleny [Zel73] introduce el concepto de *solución de compromiso* como toda solución \mathbf{x} que minimiza el valor d_p para algún vector de pesos $(\lambda_1, \dots, \lambda_m)$ y para algún valor p . Se puede demostrar que toda solución de compromiso es no dominada en el conjunto de soluciones factibles.

Una forma de ponderar cada objetivo es tomar $\lambda_i = 1/y_i^*$. Con esto conseguimos una medida de *distancia relativa* que puede expresarse de forma general tal como indica la siguiente ecuación:

$$d_p = \left[\sum_{i=1}^m \left(\frac{y_i^* - y_i}{y_i^*} \right)^p \right]^{1/p}$$

Otra medida utilizada es la *distancia compuesta* mostrada en la siguiente ecuación:

$$d_p(k) = \left[\sum_{i=1}^m (1 - d_i^k)^p \right]^{1/p}$$

donde d_i^k es un valor entre 0 y 1 que indica el grado de compatibilidad para el objetivo i -ésimo entre la solución k -ésima \mathbf{y}^k y la solución ideal \mathbf{y}^* . Es decir, d_i^k es una medida de la satisfacción que obtiene la solución k -ésima con respecto al objetivo i -ésimo. Este valor puede calcularse de la siguiente forma:

$$d_i^k = \frac{y_i^k - y_{i*}}{y_i^* - y_{i*}}$$

El valor y_{i*} indica el máximo alcanzable para el objetivo i -ésimo. La *solución anti-ideal* $\mathbf{y}_* = (y_{1*}, \dots, y_{m*})$ representa el caso contrario al de la solución ideal y viene dada por:

$$y_{i*} = \max_{k=1, \dots, K} y_i^k, \quad i = 1, \dots, m$$

Es posible identificar las soluciones dominadas maximizando la distancia a la solución anti-ideal, tal como indica la siguiente ecuación:

$$d'_p(k) = \left[\sum_{i=1}^m (d_i^k)^p \right]^{1/p}$$

Se puede demostrar que al maximizar $d'_p(k)$ o minimizar $d_p(k)$ podemos obtener las mismas soluciones para ciertos valores de p .

Yager [Yag78] introduce el concepto de objetivos *compensatorios* y *competitivos*. Dos o más objetivos son compensatorios si un valor bajo en uno de ellos puede ser compensado por un valor alto en otro. Dos o más objetivos son competitivos si cada uno de ellos debe ser optimizado por sí mismo y aparentemente se encuentran en conflicto directo. Yager estudia las propiedades de $d_p(k)$ y $d'_p(k)$ y demuestra que los casos extremos entre objetivos compensatorios y competitivos son simplemente casos especiales de $d_p(k)$ y $d'_p(k)$. Cuando $p = 1$ nos encontramos con objetivos completamente compensados, mientras que $p = \infty$ indica lo contrario. De igual forma, los objetivos competitivos están relacionados con minimizar $d_p(k)$ mientras que los objetivos no competitivos tienen relación con maximizar $d'_p(k)$. En la realidad los objetivos nunca son puramente competitivos, ni puramente compensatorios por lo que es preferible utilizar $d_p(k)$ y $d'_p(k)$ para todo p entre 1 y ∞ .

Si se desea ponderar los objetivos, dando mayor importancia a unos y menos a otros, esto puede realizarse estableciendo un vector de pesos $\alpha = (\alpha_1, \dots, \alpha_m)$. En este caso, la medida de distancia ponderada $d_p(k, \alpha)$ para el caso competitivo será de la siguiente forma:

$$d_p(k, \alpha) = \left\{ \sum_{i=1}^n [1 - (d_i^k)^{\alpha_i}] \right\}^{1/p}$$

y para el caso compensatorio:

$$d'_p(k, \alpha) = \left\{ \sum_{i=1}^n (d_i^k)^{1/\alpha_i} \right\}^{1/p}$$

2.3 Técnicas Evolutivas para Manejo de Restricciones

En este apartado se trata de resolver un problema de optimización general no lineal, $f(\mathbf{x})$ sujeto a una serie de restricciones $g_j(\mathbf{x}) \leq 0$ en general también no lineales. Consideramos el siguiente problema:

$$\begin{aligned} & \text{Minimizar} && f(\mathbf{x}) \\ & \text{sujeto a :} && g_j(\mathbf{x}) \leq 0, \quad j = 1, \dots, m \end{aligned}$$

donde $\mathbf{x} = (x_1, \dots, x_p)$ es un vector de parámetros reales $x_k \in \Re$ pertenecientes a un dominio $[l_k, u_k]$, $k = 1, \dots, p$, y $f(\mathbf{x})$, $g_j(\mathbf{x})$ son funciones arbitrarias lineales o no lineales.

Dentro de un algoritmo evolutivo, se llaman *individuos factibles* a aquellos individuos que satisfacen las restricciones e *individuos no factibles* a aquellos individuos que no las satisfacen.

Centrando el estudio en problemas de optimización numérica no lineal, las técnicas evolutivas para manejo de restricciones pueden clasificarse dentro de las siguientes:

- Funciones de penalización.
- Representación y operadores de variación específicos.
- Separación de objetivos y restricciones.
- Métodos híbridos.
- Otros métodos.

2.3.1 Funciones de penalización

La aproximación más común en computación evolutiva para manejar restricciones es el uso de penalizaciones. Este método se basa en definir el valor de adecuación de un individuo $F(\mathbf{x})$ extendiendo el dominio de la función objetivo $f(\mathbf{x})$ de la siguiente forma:

$$F(\mathbf{x}) = f(\mathbf{x}) + Q(\mathbf{x})$$

donde $Q(\mathbf{x})$ representa, para individuos no factibles, o bien una penalización, o bien un coste para reparar dicho individuo (es decir, el coste para hacer que dicho individuo pase a ser factible). Si el individuo es factible, entonces $Q(\mathbf{x}) = 0$.

La función $Q(\mathbf{x})$ puede definirse de múltiples formas. Goldberg [Gol89] propone la siguiente función:

$$Q(\mathbf{x}) = c \sum_{j=1}^m \Omega(\phi_j(\mathbf{x}))$$

donde c es un coeficiente de penalización definido por el usuario, $\Omega(y) = y^2$ y $\phi_j(\mathbf{x}) = H(g_j(\mathbf{x}))g_j(\mathbf{x})$, siendo $H : \Re \rightarrow \{0, 1\}$ la función de Heavyside:

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

Es decir, la función $\phi_j(\mathbf{x})$ se define como:

$$\phi_j(\mathbf{x}) = \begin{cases} 0, & \text{si } g_j(\mathbf{x}) \leq 0 \\ g_j(\mathbf{x}), & \text{si } g_j(\mathbf{x}) > 0 \end{cases}, \quad j = 1, \dots, m$$

Hoffmeister y Sprave [Hof96] proponen utilizar:

$$Q(\mathbf{x}) = \sqrt{\sum_{j=0}^m \Omega(\phi_j(\mathbf{x}))}$$

Idealmente, la función de penalización Q debería tener un valor mínimo cuando se acerca al límite debajo del cual las soluciones son óptimas; esto se llama *regla de la penalización mínima* [Smi93b]. Aunque esta regla es conceptualmente sencilla, su aplicación en la práctica es muy difícil debido a que en la mayoría de problemas no se conoce cual es el límite entre la región factible y la no factible.

Por otra parte, la relación entre individuos no factibles y la región factible, juega un papel importante para penalizar dicho individuo [Richa89], aunque no resulta claro cómo aprovechar esta relación para guiar la búsqueda en la dirección deseada. Existen básicamente tres formas de definir la relación entre un individuo no factible y la región factible [Das97]:

1. Se penaliza a un individuo simplemente por ser no factible; es decir, no se utiliza ninguna información sobre lo cerca o lejos que se encuentra de la región factible.

2. Se mide la cantidad de no factibilidad, es decir, se mide lo lejos que dicho individuo se encuentra de la región factible y se utiliza esta medida para determinar la penalización.
3. Se mide el esfuerzo de reparar un individuo, es decir, lo que costaría hacer que dicho individuo pasara a ser factible.

Diversos autores han planteado heurísticas para el diseño de funciones de penalización. Posiblemente el más conocido de estos estudios es el realizado por Richardson et al. [Richa89] del cual se extraen las siguientes directrices para establecer una buena función de penalización:

1. Las penalizaciones que dependen de la distancia a la región factible tienen mejor rendimiento que aquellas que se basan sólo en el número de restricciones que no se satisfacen.
2. En problemas con pocas restricciones y pocas soluciones factibles, las penalizaciones basadas en el número de restricciones no satisfechas probablemente no obtendrán ninguna solución factible.
3. Se pueden obtener funciones de penalización adecuadas teniendo en cuenta dos cantidades: el *coste máximo de cumplimiento* y el *coste esperado de cumplimiento*. El *coste de cumplimiento* es el coste de hacer que una solución no factible pase a ser factible.
4. La penalización debe acercarse al coste esperado de cumplimiento, pero no debería ser menor que dicho coste. Cuanto más precisa es una penalización, mejor será la solución encontrada. Cuando una penalización subestima el coste de cumplimiento, entonces la búsqueda puede fallar y no encontrar una solución.

A partir de estas directrices, diversos autores han desarrollado técnicas adecuadas para construir funciones de penalización. A continuación se analizan las más importantes. Sin embargo, estas directrices resultan difíciles de seguir en algunos casos. Por ejemplo, el coste esperado de cumplimiento en algunos problemas tiene que ser estimado utilizando métodos alternativos como realizar un escalado relativo de la distancia entre múltiples restricciones o estimar el grado de no satisfacción de las restricciones, etc. [Smi97]. Tampoco resulta claro cómo combinar los dos valores

indicados por Richardson et al. [Richa89] y cómo diseñar una función de adecuación que utilice penalizaciones precisas.

Penalizaciones estáticas

Homaifar et al. [Hom94] proponen un método en el cual el usuario define diversos niveles de violación y se elige un coeficiente de penalización para cada uno de dichos niveles, de forma que el coeficiente de penalización sea alto en los niveles de violación altos y disminuya en los niveles de violación menores. Un individuo se evalúa por tanto usando la siguiente función:

$$F(\mathbf{x}) = f(\mathbf{x}) + \sum_{j=1}^m R_{jk} \phi_j^2(\mathbf{x})$$

donde R_{jk} es el coeficiente de penalización para la restricción j -ésima en el nivel k -ésimo, con $k = 1, \dots, l$, siendo l el número de niveles de violación definidos por el usuario. La idea de esta técnica es evaluar por separado cada una de las restricciones del individuo definiendo un conjunto de coeficientes diferente para cada una. El mayor inconveniente de esta aproximación es el elevado número de parámetros necesarios, los cuales en muchos problemas no resultan fáciles de establecer.

Penalizaciones dinámicas

En esta aproximación, propuesta por Joines y Houck [Joi94], las penalizaciones son dinámicas, es decir, cambian en el tiempo. Un individuo en la generación t se evalúa utilizando:

$$F(\mathbf{x}) = f(\mathbf{x}) + (C + t)^\alpha \sum_{j=1}^m |\phi_j(\mathbf{x})|^\beta$$

donde C , α y β son constantes definidas por el usuario (β normalmente toma los valores 1 ó 2). La penalización aumenta al aumentar el número de generaciones. El problema de este método es que, al igual que en la anterior aproximación, resulta difícil encontrar buenos coeficientes; en ésta resulta difícil encontrar buenas funciones de penalización; por ejemplo, en el trabajo realizado por Joines y Houck, los parámetros son dependientes del problema y en la mayoría de ocasiones producen convergencia prematura. Por otro lado, esta técnica normalmente converge a una solución no factible o a una solución factible, pero alejada del óptimo global.

Penalizaciones con enfriamiento simulado

Michalewicz y Attia [Mic94] desarrollan un método basado en la idea del enfriamiento simulado (GENOCOP II); los coeficientes de penalización cambian cada vez que el algoritmo se encuentra atrapado en un óptimo local y la penalización aumenta con las generaciones, de forma que los individuos no factibles son altamente penalizados en las últimas generaciones.

La población evoluciona teniendo en cuenta la siguiente función de adecuación para los individuos:

$$F(\mathbf{x}, t) = f(\mathbf{x}) + \frac{1}{2\tau(t)} \sum_{j=1}^m \phi_j^2(\mathbf{x})$$

donde $\tau(t)$ es el factor de enfriamiento en la generación t . En cada generación el valor $\tau(t)$ disminuye y la nueva población se crea utilizando la mejor solución encontrada en la generación anterior como punto de partida. El proceso termina cuando se alcanza un valor predefinido de temperatura final τ_f .

Otra propuesta similar es la realizada por Carlson et al. [Car] en la que se define la función de adecuación de la siguiente forma:

$$F(\mathbf{x}, t) = f(\mathbf{x}) \prod_{j=1}^m e^{-M_j/\tau(t)}$$

donde M_j mide el grado de violación de la restricción j -ésima y el coeficiente de enfriamiento $\tau(t)$ es:

$$\tau(t) = \frac{1}{\sqrt{t}}$$

Por último, Joines y Houck [Joi94] también experimentan con funciones de penalización basadas en enfriamiento simulado, en concreto, utilizan la siguiente función de adecuación:

$$F(\mathbf{x}) = f(\mathbf{x}) + e^{(C+t)^\alpha \sum_{j=1}^m |\phi_j(\mathbf{x})|^\beta}$$

con los valores $C = 0.05$, $\alpha = \beta = 1$.

Penalizaciones adaptativas

Bean y Hadj-Alouane [Bea92, Had92] desarrollan un método para adaptar las penalizaciones utilizando un sistema de realimentación. Cada individuo se evalúa de la forma:

$$F(\mathbf{x}, t) = f(\mathbf{x}) + \lambda(t) \sum_{j=1}^m \phi_j^2(\mathbf{x})$$

donde el valor $\lambda(t)$ se actualiza en cada generación mediante la siguiente ecuación:

$$\lambda(t+1) = \begin{cases} (1/\beta_1)\lambda(t), & \text{si } \mathbf{b}^i \in \mathcal{F} \quad \text{para todo } t-k+1 \leq i \leq t \\ \beta_2\lambda(t), & \text{si } \mathbf{b}^i \in \mathcal{S} - \mathcal{F} \quad \text{para todo } t-k+1 \leq i \leq t \\ \lambda(t), & \text{en otro caso} \end{cases}$$

donde \mathbf{b}^i representa el mejor individuo en la generación i -ésima; $\beta_1, \beta_2 > 1$ y $\beta_1 \neq \beta_2$. Es decir, la penalización decrece si todos los mejores individuos en las últimas k iteraciones son factibles, o se incrementa en el caso contrario; en otro caso el valor de penalización se mantiene igual.

Smith y Tate [Smi93b] proponen otro método, posteriormente redefinido por Coit y Smith [Coi96a] y por Coit et al. [Coi96b] en el que el valor de la penalización se adapta según la adecuación de la mejor solución encontrada hasta el momento, es decir:

$$F(\mathbf{x}, t) = f(\mathbf{x}) + [F_{\text{factible}}(t) - F_{\text{total}}(t)] \sum_{j=1}^m \left(\frac{\phi_j(\mathbf{x})}{q_j(t)} \right)^k$$

donde $F_{\text{factible}}(t)$ es el mejor valor factible obtenido hasta el momento (hasta la generación t) para la función objetivo, $F_{\text{total}}(t)$ es el mejor valor sin penalizaciones obtenido hasta el momento (hasta la generación t) para la función objetivo, k es una constante que ajusta la penalización (Coit y Smith [Coi96a] sugieren un valor $k = 2$) y $q_j(t)$ es el valor en la iteración t de disparo cercano a la factibilidad para la restricción j , que se define como la distancia a la región factible a la cual el usuario considera que la búsqueda está razonablemente cerca de dicha región. El valor $q_j(t)$ es dinámico y cambia durante la búsqueda de forma que la penalización se incrementa conforme avanzan las iteraciones. Es posible definir este valor de la forma:

$$q_j(t) = \frac{q_j^0}{1 + \beta_j t}, \quad j = 1, \dots, m$$

donde q_j^0 es el límite superior de $q_j(t)$ y β_j es una constante definida para la restricción j -ésima que asegura que se busca en toda la región factible entre q_j^0 y cero.

Yokota et al. [Yok95] proponen una variante del método de Smith y Coit en la que se utiliza una función de adecuación multiplicativa:

$$F(\mathbf{x}, t) = f(\mathbf{x}) \times P(\mathbf{x}, t)$$

donde $P(\mathbf{x})$ se define como:

$$P(\mathbf{x}, t) = 1 - \frac{1}{m} \sum_{j=1}^m \left(\frac{\Delta b_j(\mathbf{x}, t)}{b_j(t)} \right)^k$$

$$\Delta b_j(\mathbf{x}, t) = \max\{0, \phi_j(\mathbf{x}) - b_j(t)\}$$

En este caso, $\Delta b_j(\mathbf{x}, t)$ hace referencia a la violación de la restricción j -ésima en la iteración t . Los valores $b_j(t)$ pueden cambiar de forma adaptativa a lo largo de las generaciones.

Gen y Cheng [Gen96] redefinen esta aproximación introduciendo penalizaciones más severas para las soluciones no factibles. En esta nueva versión se redefine $P(\mathbf{x})$ como:

$$P(\mathbf{x}, t) = 1 - \frac{1}{m} \sum_{j=1}^m \left(\frac{\Delta b_j(\mathbf{x}, t)}{\Delta b_j^{max}(t)} \right)^k$$

$$\Delta b_j^{max}(t) = \max\{\epsilon, \Delta b_j(\mathbf{x}, t); \mathbf{x} \in \mathcal{P}(t)\}$$

donde $\mathcal{P}(t)$ hace referencia al conjunto de individuos que forman la población en la iteración t , $\Delta b_j^{max}(t)$ es la violación máxima de la restricción j -ésima en toda la población y ϵ un número real positivo pequeño que se utiliza para evitar la división por cero. Con esta técnica se mantiene la diversidad en la población, evitando al mismo tiempo sobrepenalizar soluciones no factibles, las cuales formarán la mayor parte de la población en las primeras generaciones.

Penalizaciones auto-adaptativas

Coello [Coe00a] propone el uso de una función de penalización de la forma:

$$F(\mathbf{x}) = f(\mathbf{x}) - \left(w_1 \sum_{j=1}^m \phi_j(\mathbf{x}) + w_2 \sum_{j=1}^m \alpha_j(\mathbf{x}) \right)$$

donde w_1 y w_2 son factores de penalización (considerados enteros en la aproximación de Coello), el término $\sum_{j=1}^m \phi_j(\mathbf{x})$ es la suma de todas las violaciones de las restricciones y $\sum_{j=1}^m \alpha_j(\mathbf{x})$ es un factor entero que indica el número de restricciones

violadas, siendo:

$$\alpha_j(\mathbf{x}) = \begin{cases} 1, & \text{si } g_j(\mathbf{x}) > 0 \\ 0, & \text{en otro caso} \end{cases}, \quad j = 1, \dots, m$$

Algoritmos genéticos segregados

Le Riche et al. [Riche95] diseñan un algoritmo genético segregado en el que se utilizan dos parámetros de penalización por cada restricción; con estos dos valores se puede llegar a un equilibrio entre penalizaciones moderadas y severas manteniendo dos subpoblaciones de individuos en lugar de una sola.

El problema de esta aproximación es nuevamente cómo elegir los parámetros de penalización para cada subpoblación, pues resulta difícil producir valores genéricos para dichos parámetros que puedan servir para cualquier problema.

Penalizaciones mortales

La técnica de penalizaciones mortales consiste en rechazar aquellos individuos no factibles; ésta es probablemente la forma más simple y eficiente de manejar las restricciones. Una aproximación interesante que utiliza este método es la realizada por Kuri [Kuri98] en la que define la adecuación de un individuo utilizando:

$$F(\mathbf{x}) = \begin{cases} f(\mathbf{x}), & \text{si } \mathbf{x} \in \mathcal{F} \\ K \left(1 - \frac{s}{m}\right), & \text{si } \mathbf{x} \in \mathcal{S} - \mathcal{F} \end{cases}$$

donde s es el número de restricciones satisfechas, m el número de restricciones totales y K una constante de valor muy alto (Kuri utiliza un valor $K = 1 \times 10^9$). En esta aproximación, cuando un individuo no es factible no se calcula su función objetivo y su adecuación depende solamente del número de restricciones satisfechas, independientemente de lo lejos que se encuentre de la región factible.

La técnica de penalizaciones mortales es simple y eficiente, pero se limita a problemas en los cuales la mayor parte del espacio de búsqueda es factible y además es convexo.

2.3.2 Representación y operadores de variación específicos

En algunas ocasiones la opción elegida es desarrollar esquemas de representación y operadores específicos para resolver un cierto problema. Normalmente estas representaciones y operadores se diseñan de forma que todos los individuos del problema sean factibles. Esta aproximación es la más adecuada para problemas en los que resulta particularmente difícil encontrar al menos una solución factible. Sin embargo, estos métodos no son fácilmente generalizables, pues el tipo de representación y los operadores son dependientes del problema concreto que se pretende resolver. A continuación se describen algunos de los algoritmos más representativos que utilizan esta técnica.

Operadores de variación específicos

GENOCOP (GEnetic algorithm for Numerical Optimization for CONstrained Problems), desarrollado por Michalewicz [Mic92], trata restricciones lineales eliminando igualdades y diseñando operadores de variación específicos que garantizan que todas las soluciones se encuentran dentro del espacio de búsqueda factible. GENOCOP asume una población inicial de individuos factibles, por lo que, para un problema dado, se ha de tener un método para generar en un tiempo razonable dicha población; por otra parte, GENOCOP maneja sólo restricciones lineales, por lo que su aplicación se restringe a espacios de búsqueda convexos.

Consistencia con las restricciones

Kowalczyk [Kow97] propone el uso de consistencias con las restricciones. Este método se basa en utilizar sólo individuos consistentes con las restricciones, es decir, con soluciones factibles, de forma que se reduzca el espacio de búsqueda. En este caso, Kowalczyk utiliza una codificación con números reales junto con operadores específicos al problema y un procedimiento de inicialización propio que incorpora el concepto de consistencia con las restricciones.

Localizar el borde de la región factible

La principal idea de esta técnica es realizar la búsqueda en el área cercana al borde de la región factible. Puesto que en muchos problemas de optimización no lineal al

menos alguna de las restricciones es activa en el óptimo global, está perfectamente justificado centrar la búsqueda en el límite entre las regiones factibles y no factibles.

La idea es propuesta originalmente por Glover [Glo77] como una técnica de investigación operativa llamada *estrategia de oscilación* y ha sido utilizada en problemas combinatorios y de optimización no lineal [Glo95]. Normalmente se utilizan penalizaciones adaptativas u otros métodos similares (por ejemplo, gradientes) para cruzar el límite de factibilidad en uno u otro sentido relajando o estrechando algún factor que determine la dirección del movimiento.

Los dos componentes básicos de este método son:

1. Un procedimiento de inicialización que genere individuos factibles.
2. Operadores de variación que exploren la región factible.

Decodificadores

En este método, un individuo *da instrucciones* sobre cómo construir una solución factible. Cada decodificador impone una relación T entre una solución factible y una solución decodificada. Sin embargo, cuando se utilizan decodificadores, es importante que se cumplan ciertas condiciones:

1. Para cada solución factible \mathbf{s} , existe una solución decodificada \mathbf{d} .
2. Cada solución decodificada \mathbf{d} corresponde a una solución factible \mathbf{s} .
3. Todas las soluciones factibles deberían ser representadas por el mismo número de soluciones decodificadas \mathbf{d} .

Además de estas soluciones, también resulta conveniente que se cumpla:

4. La transformación T es computacionalmente eficiente.
5. Existen características de localidad, es decir, pequeños cambios en la solución decodificada producen pequeños cambios en la solución real.

Koziel y Michalewicz [Koz98, Koz99] proponen una aplicación homomorfa entre un hipercubo de n dimensiones y un espacio de búsqueda factible (convexo o no).

Este método se basa en transformar el problema original en otro (topológicamente equivalente) que sea más sencillo de optimizar.

El método propuesto por Koziel y Michalewicz tiene muchas ventajas, pero también tiene algunos inconvenientes; utiliza un parámetro extra v que debe establecerse de forma empírica realizando una serie de ejecuciones y además es un método poco eficiente computacionalmente.

Algoritmos de reparo

En algunos problemas resulta relativamente fácil *reparar* un individuo no factible, es decir, hacer que dicho individuo pase a ser factible.

Entre las aplicaciones que utilizan este método podemos nombrar el sistema GENOCOP III [Mic95] que incorpora algoritmos de reparo al original GENOCOP haciendo posible el manejo de restricciones no lineales.

Una de las cuestiones abiertas en esta técnica es si los individuos reparados deben volver a introducirse en la población o no. Algunos autores como Liepins et al. [Lie90, Lie91] han decidido no reemplazar nunca las soluciones reparadas, mientras otros autores como Nakano [Nak91] toman la opción contraria, es decir, devolver siempre a la población las soluciones reparadas. En general, la solución reparada puede devolverse a la población sólo en ciertos casos (usando por ejemplo una probabilidad). Orvosh y Davis [Orv93, Orv94] establecen que los mejores resultados aplicados a problemas de optimización combinatoria se obtienen cuando un 5% de los individuos reparados son devueltos a la población. Michalewicz et al. [Mic96a], sin embargo, sostienen que reemplazar un 15% es la mejor opción para problemas de optimización numérica con restricciones no lineales.

No existe una heurística estándar para diseñar algoritmos de reparo; puede utilizarse un algoritmo de reparo voraz, reparo aleatorio o cualquier otra heurística que se adapte al problema que deseamos resolver. El buen funcionamiento de este método depende fundamentalmente de encontrar una heurística de reparo adecuada, la cual es dependiente del problema concreto que se desea resolver y debe, por tanto, diseñarse un algoritmo de reparo específico para cada problema particular. Para aquellos problemas en los cuales resulta sencillo reparar una solución, esta técnica puede ser una buena elección; sin embargo, esto no resulta siempre posible y por tanto este método no siempre puede aplicarse con éxito.

2.3.3 Separación de objetivos y restricciones

Existen diversos métodos que manejan los objetivos y las restricciones de forma separada. En este apartado revisamos aquellos más representativos.

Co-evolución

Paredis [Pare94] propone una técnica basada en un modelo co-evolutivo en el que hay dos poblaciones: la primera contiene las restricciones que deben ser satisfechas (de hecho no es una población en el sentido general del término, puesto que su contenido se mantiene inalterado en el tiempo) y la segunda población contiene soluciones potenciales (posiblemente no factibles) del problema. Usando una analogía con el modelo depredador-presa, la presión de selección en los miembros de una población depende de la adecuación de los miembros de la otra población.

Un individuo con una alta adecuación en la segunda población representa una solución que satisface muchas restricciones, mientras que un individuo con alta adecuación en la primera población representa una restricción que es violada por muchas soluciones.

Las soluciones y las restricciones tienen *encuentros* en los cuales se evalúan los individuos de ambas poblaciones. Cada individuo mantiene un registro de dichos encuentros y su adecuación se calcula de acuerdo a la suma de los últimos n encuentros (Paredis utiliza $n = 25$). La idea de este método consiste en incrementar la adecuación de aquellas restricciones más difíciles de satisfacer de forma que la búsqueda se centre en dichas restricciones. Por otra parte, la importancia de una restricción cambia de forma dinámica.

Este método resulta similar a las funciones de penalización auto-adaptativas en las que la importancia de una restricción cambia en el tiempo de acuerdo a su dificultad. Paredis obtuvo muy buenos resultados con este método y de forma eficiente, pues no es necesario evaluar todas las restricciones al mismo tiempo. Un problema de este método es que puede pararse si todas las restricciones son igualmente difíciles de resolver.

Superioridad de soluciones factibles

Deb [Deb00a] propone un método interesante para evaluar los individuos utilizando:

$$F(\mathbf{x}) = \begin{cases} f(\mathbf{x}), & \text{si } \mathbf{x} \in \mathcal{F} \\ f_{\text{peor}} + \sum_{j=1}^m g_j(\mathbf{x}), & \text{si } \mathbf{x} \in \mathcal{S} - \mathcal{F} \end{cases}$$

donde f_{peor} es el valor de la función objetivo de la peor solución factible; si no existe ninguna solución factible, $f_{\text{peor}} = 0$.

Usando una selección de torneo binario, Deb utiliza las siguientes reglas para comparar dos individuos:

1. Una solución factible siempre es mejor que otra no factible.
2. Entre dos soluciones factibles, es mejor la que tenga mejor valor objetivo.
3. Entre dos soluciones no factibles, es mejor la que provoca una menor violación de las restricciones.

Esta técnica tiene problemas para mantener la diversidad en la población y hace falta utilizar métodos de formación de nichos.

Richardson et al. [Richa89] sugieren una regla heurística propuesta por Powell y Skolnick [Pow93] para procesar soluciones no factibles: la adecuación de las soluciones factibles debe estar en el intervalo $[-\infty, 1]$ y la adecuación de las soluciones no factibles en el intervalo $[1, +\infty]$. Los individuos se evalúan, por tanto, mediante la siguiente ecuación:

$$F(\mathbf{x}) = \begin{cases} f(\mathbf{x}), & \text{si } \mathbf{x} \in \mathcal{F} \\ 1 + r \sum_{j=1}^m g_j(\mathbf{x}), & \text{si } \mathbf{x} \in \mathcal{S} - \mathcal{F} \end{cases}$$

donde $f(\mathbf{x})$ está escalada en el intervalo $[-\infty, 1]$, $g_j(\mathbf{x})$ está escalada en el intervalo $[1, +\infty]$ y r es una constante. Es decir, la adecuación de un individuo factible siempre es mejor que la adecuación de un individuo no factible.

Este método funciona bien en general, sin embargo, en aquellos problemas donde la región factible es muy pequeña en relación con el espacio total de búsqueda, es decir, aquellos problemas con restricciones muy difíciles de satisfacer, el método puede fallar a menos que se introduzca en la población inicial una solución factible.

Jiménez y Verdegay [Jim99a] proponen el uso de una aproximación min-max para manejar restricciones. Su método se basa en aplicar un conjunto de reglas simples para decidir el proceso de selección:

1. Si los dos individuos son factibles, se selecciona en función del valor mínimo de la función objetivo.
2. Si uno de los dos individuos es factible y el otro no, se selecciona el individuo factible.
3. Si ninguno de los individuos es factible, entonces la selección se realiza en base a la violación máxima de las restricciones: $g_{max} = \max \{g_j(\mathbf{x}), j = 1, \dots, m\}$. Se selecciona el individuo con el menor máximo.

Este método primero satisface las restricciones y luego busca soluciones en la región factible. El inconveniente es que puede quedar atrapado en una región factible aislada. Sin embargo, este método es una buena alternativa cuando se trata de encontrar un punto factible dentro de un espacio de búsqueda altamente restringido.

Memoria de comportamiento

Esta técnica es propuesta originalmente por Garis [Gar90] para tratar problemas de optimización sin restricciones. Schoenauer y Xanthakis [Scho93] proponen extenderla para tratar problemas de optimización con restricciones. La idea básica de este método consiste en manejar las restricciones siguiendo un orden concreto. El algoritmo es el siguiente:

1. Empezar con una población de individuos aleatorios.
2. Establecer $j \leftarrow 1$ (j es el contador de restricciones)
3. Evolucionar la población para minimizar la violación en la restricción j -ésima hasta que un porcentaje dado de la población (llamado umbral de cambio, Φ) sea factible para dicha restricción. En ese momento se evalúan los individuos de la forma:

$$F(\mathbf{x}) = M - g_1(\mathbf{x})$$

donde M es un número positivo suficientemente grande que se ajusta dinámicamente en cada generación.

4. Actualizar $j \leftarrow j + 1$

5. Evolucionar la población minimizando la restricción j -ésima,

$$F(\mathbf{x}) = M - g_j(\mathbf{x})$$

Durante esta fase, las soluciones que no satisfacen ninguna de las $(j - 1)$ -ésimas restricciones son eliminados de la población. El criterio de parada es de nuevo la satisfacción de la restricción j -ésima por un porcentaje Φ de la población.

6. Si $j < m$ volver al paso 4, en otro caso ($j = m$) optimizar la función objetivo f rechazando los individuos no factibles.

La idea de esta técnica es satisfacer secuencialmente (una por una) las restricciones. El orden que se utiliza influye en el resultado obtenido. Por otra parte Schoenauer y Xanthakis recomiendan el uso de una función de participación para mantener la diversidad de la población. Este método es computacionalmente muy costoso y no se justifica su uso cuando las restricciones no son difíciles de satisfacer. Sin embargo, este método resulta especialmente adecuado para aquellos problemas en los cuales las restricciones tienen una naturaleza jerárquica, como es el problema de generar datos para testeo de software usado por Schoenauer y Xanthakis.

Técnicas de optimización multiobjetivo

La idea principal es redefinir el problema como un problema de optimización multiobjetivo en el cual tenemos $m + 1$ objetivos, donde m es el número de restricciones. Entonces podemos aplicar cualquier técnica de optimización multiobjetivo (revisadas en la sección 2.4) para solucionar el problema de optimización multiobjetivo $(f(\mathbf{x}), g_1(\mathbf{x}), \dots, g_m(\mathbf{x}))$ donde $g_1(\mathbf{x}), \dots, g_m(\mathbf{x})$ son las restricciones originales $g_j(\mathbf{x})$. Una solución ideal \mathbf{x} debe cumplir $g_j(\mathbf{x}) \leq 0, j = 1, \dots, m$, y $f(\mathbf{x}) \leq f(\mathbf{y})$ para todo \mathbf{y} que cumpla $g_j(\mathbf{y}) \leq 0, j = 1, \dots, m$.

Surry et al. [Sur95] proponen utilizar la ordenación Pareto [Fon93] y VEGA [Sch85] (ver sección 2.4) para manejar restricciones utilizando esta técnica. En su aproximación, llamada *COMOGA* (*constrained MOGA*) la población se ordena en base a las violaciones de las restricciones (contando el número de individuos dominados por cada solución). Entonces, se selecciona una parte de la población basándose en dicha ordenación y el resto basándose en el coste real (adecuación) de los individuos.

Parmee y Purchase [Parm94] implementan una versión de VEGA que maneja las restricciones como objetivos para localizar una región factible dentro de un espacio de

búsqueda altamente restringido. Sin embargo, una vez se localiza la región factible, no se utiliza VEGA, sino que se utilizan unos operadores especiales para mantener los individuos dentro de la zona factible.

Camponogara y Talukdar [Cam97] proponen redefinir el problema de forma que se tengan en cuenta dos objetivos; el primero optimiza la función objetivo original y el segundo optimiza una composición de las restricciones. El problema queda por tanto redefinido de la forma $(f(\mathbf{x}), \Phi(\mathbf{x}))$ donde:

$$\Phi(\mathbf{x}) = \sum_{j=1}^m \max(0, g_j(\mathbf{x}))$$

Fonseca y Fleming [Fon98] proponen una ampliación de su algoritmo MOGA (ver sección 2.4.3) para tratar restricciones, basándose en metas y prioridades. Las restricciones son tratadas como objetivos de alta prioridad hasta que se alcanza su meta, es decir, se cumple la restricción.

La preferencia entre dos soluciones \mathbf{x} y \mathbf{x}' se define de la siguiente forma:

- Si \mathbf{x} alcanza todas las metas en los términos de mayor prioridad (satisface todas las restricciones) y \mathbf{x}' no las alcanza, se prefiere la solución \mathbf{x} .
- Si ambas soluciones alcanzan todas las metas en los términos de mayor prioridad (satisfacen todas las restricciones), se comparan utilizando el concepto Pareto aplicado a los términos de menor prioridad (los objetivos).
- Si en ninguna solución se alcanzan todas las metas en los términos de mayor prioridad (ninguna solución satisface las restricciones), se comparan utilizando la función objetivo.

Coello [Coe00a] propone utilizar una técnica de optimización multiobjetivo similar a VEGA, de forma que se manejen las restricciones como objetivos. En cada generación, la población se divide en $m + 1$ subpoblaciones (m es el número de restricciones y una población más para la función objetivo). En este esquema, una subpoblación utiliza la función objetivo (sin restricciones) como función de adecuación; las otras subpoblaciones utilizarán el valor de cada una de las restricciones; así, para

la subpoblación asociada a la restricción j -ésima, tendremos:

$$F(\mathbf{x}) = \begin{cases} g_j(\mathbf{x}), & \text{si } g_j(\mathbf{x}) \leq 0 \\ -v, & \text{si } g_j(\mathbf{x}) > 0 \text{ y } v > 0 \\ f(\mathbf{x}), & \text{si } v = 0 \end{cases}$$

donde v es el número de restricciones que no son satisfechas.

Jiménez et al. [Jim02] proponen transformar el problema de la forma:

$$(f'(\mathbf{x}), g'_1(\mathbf{x}), \dots, g'_m(\mathbf{x}))$$

siendo:

$$\begin{aligned} f'(\mathbf{x}) &= \begin{cases} f(\mathbf{x}), & \text{si } \mathbf{x} \in \mathcal{F} \\ M, & \text{si } \mathbf{x} \in \mathcal{S} - \mathcal{F} \end{cases} \\ g'_j(\mathbf{x}) &= \begin{cases} g_j(\mathbf{x}), & \text{si } \mathbf{x} \in \mathcal{F} \\ m, & \text{si } \mathbf{x} \in \mathcal{S} - \mathcal{F} \end{cases} \\ j &= 1, \dots, m \end{aligned}$$

donde M debe ser mayor que $f(\mathbf{x})$ para todo \mathbf{x} y $m \leq 0$ (Jiménez et al. proponen utilizar $M = +\infty$ y $m = 0$).

El enfoque propuesto es una generalización para contemplar problemas multi-objetivo basados en metas y prioridades por lo que se retomará en la sección 2.5.

2.3.4 Métodos híbridos

Dentro de este apartado se encuentran aquellos métodos que utilizan alguna técnica adicional (normalmente algún método de optimización numérica) para manejar las restricciones en un algoritmo evolutivo.

Multiplicadores de Lagrange

Adeli y Cheng [Ade94] proponen un algoritmo evolutivo híbrido que integra el método de la función de penalización con el método primero-doble. Esta técnica se basa en la minimización secuencial del método de Lagrange, el cual utiliza una función de adecuación de la forma:

$$F(\mathbf{x}) = f(\mathbf{x}) + \frac{1}{2} \sum_{j=1}^m \gamma_j \{\max[0, g_j(\mathbf{x}) + \mu_j]\}^2$$

donde γ_j y μ_j , $j = 1, \dots, m$ son parámetros asociados a la restricción j -ésima. Adeli y Cheng definen μ_j en función de la máxima violación registrada hasta el momento para la restricción asociada y escalan este valor utilizando un parámetro $\beta > 1$ definido por el usuario. El valor de γ_j se incrementa utilizando también el parámetro β .

Kim y Myung [Kim97, Myu98] proponen combinar el algoritmo evolutivo con una función de Lagrange extendida de forma que garantice la creación de soluciones factibles durante el proceso de búsqueda. Este método es una extensión del sistema *Evolian* [Myu95, Myu97] que utiliza programación evolutiva con un procedimiento de optimización multi-fase en el cual se escalan las restricciones. Durante la primera fase del algoritmo, el objetivo es minimizar:

$$F(\mathbf{x}) = f(\mathbf{x}) + \frac{s}{2} \left(\sum_{j=1}^m g_j^2(\mathbf{x}) \right)$$

donde s es una constante. Una vez que finaliza esta fase (es decir, una vez la violación de las restricciones ha descendido tanto como se desea) empieza la segunda fase, en la cual se aplica el algoritmo de optimización de Maa y Shanblatt [Maa92]. Esta segunda fase utiliza multiplicadores de Lagrange para ajustar la función de penalización de una forma muy parecida a la propuesta por Adeli y Cheng. En esta fase, se itera hasta que el sistema:

$$\mathbf{x}' = -\nabla f(\mathbf{x}) - \sum_{j=1}^m \nabla g_j(\mathbf{x}) (sg_j(\mathbf{x}) + \lambda_j)$$

está en equilibrio, siendo λ_j los multiplicadores de Lagrange asociados con la restricción j -ésima, que se actualizan según la ecuación:

$$\lambda_j = \epsilon sg_j(\mathbf{x})$$

siendo ϵ una constante positiva pequeña.

Evolución aleatoria

Belur [Bel97] propone una técnica híbrida llamada *Optimización con restricciones mediante Evolución Aleatoria* (*Constrained Optimization by Random Evolution, CORE*). La idea básica de este método es utilizar una búsqueda evolutiva aleatoria combinada con una técnica de programación matemática para optimización sin restricciones

(Belur utiliza el método simplex de Nelder y Mead [Nel65], aunque podría utilizarse cualquier otra técnica similar). Cuando se encuentra una solución no factible, se minimiza la siguiente función:

$$F(\mathbf{x}) = \sum_{j=1}^m g_j(\mathbf{x})$$

Lógica difusa

Van Le [Le95] propone utilizar una combinación de lógica difusa y programación evolutiva para manejar restricciones. Este método se basa en reemplazar las restricciones de la forma $g_j(\mathbf{x}) \leq 0$ por un conjunto de restricciones difusas C_1, \dots, C_m de la forma:

$$\mu_{C_j}(\mathbf{x}) = \mu_{\sigma(b_j, \epsilon_j)}(g_j(\mathbf{x})), \quad j = 1, \dots, m$$

donde ϵ_j es un valor real positivo que representa la violación tolerada para la restricción j -ésima y:

$$\mu_{\sigma(a, x)}(x) = \begin{cases} 1, & \text{si } x \leq a \\ \frac{e^{-p\left(\frac{x-a}{s}\right)^2} - e^{-p}}{1 - e^{-p}}, & \text{si } a < x \leq a + s \\ 0, & \text{si } x > a + s \end{cases}$$

El motivo de utilizar estos conjuntos difusos es permitir un mayor grado de tolerancia si $g_j(\mathbf{x})$ es mayor que b_j pero se acerca bastante a dicho valor. La función de adecuación se redefine como:

$$F(\mathbf{x}) = f(\mathbf{x}) \times \min(\mu_{C_1}(\mathbf{x}), \dots, \mu_{C_m}(\mathbf{x}))$$

2.3.5 Otros métodos

En este apartado revisamos aquellos métodos que no encajan dentro de ninguna de las categorías anteriores y, por tanto, se consideran de forma separada.

Sistemas inmunes

Forrest y Perelson [For91] y Smith et al. [Smi92, Smi93a] estudian cómo utilizar un modelo computacional del sistema inmunológico en el cual una población de *anticuerpos* evoluciona para cubrir una conjunto de *antígenos*. En esta propuesta, se proponen

cadenas binarias para codificar tanto anticuerpos como antígenos y se dice que un anticuerpo corresponde con un antígeno si sus cadenas de bits son complementarias.

Smith et al. [Smi92, Smi93a] proponen este método como sistema para mantener la diversidad en problemas de optimización multimodal, sin embargo, Hajela y Lee [Haj95, Haj96a] lo extienden para el manejo de restricciones.

La técnica de Hajela y Lee se basa en separar los individuos factibles de una población (antígenos) de aquellos no factibles (anticuerpos). Utilizando una simple función de correspondencia que calcula la similaridad entre un antígeno y un anticuerpo, se realiza una co-evolución de la población de anticuerpos hasta que son suficientemente parecidos a los antígenos; en ese momento se mezclan las dos poblaciones y evolucionan según un algoritmo genético estándar, puesto que en este momento todos los individuos son factibles. Si los operadores genéticos crean individuos no factibles, estos son simplemente eliminados de la población, de forma que los individuos de la población se mantengan siempre factibles.

Hajela y Yoo [Haj96b] realizan una implementación sencilla de este método, llamada *estrategias de expresión*. En este caso, los individuos factibles y no factibles se combinan de forma que se intercambia información.

Algoritmos culturales

Algunos investigadores sociales han sugerido que la cultura podría ser codificada simbólicamente y transmitida mediante los mecanismos de herencia. Utilizando esta idea, Reynolds [Rey94] desarrolla un modelo computacional en el cual la evolución cultural se plantea como un proceso de herencia a dos niveles: micro y macro-evolutivo.

En el nivel micro-evolutivo, los individuos se describen en función de sus *pautas de comportamiento* (que pueden ser socialmente aceptables o no). Estas pautas de comportamiento son transmitidas de generación en generación utilizando diversos operadores motivados socialmente. En el nivel macro-evolutivo, los individuos son capaces de generar *mapas*, es decir, descripciones generalizadas de sus experiencias. Los mapas individuales pueden ser mezclados para formar mapas de grupo, utilizando un conjunto de operadores genéricos o específicos del problema. Ambos niveles de evolución comparten un enlace de comunicación.

Reynolds [Rey94] propone utilizar algoritmos genéticos para modelizar el proceso micro-evolutivo y Version Spaces [Mit78] para modelizar el proceso macro-evolutivo.

La idea básica de este método consiste en conservar las creencias que son socialmente aceptadas y descartar las creencias no aceptadas. Las creencias aceptadas pueden ser vistas como restricciones que dirigen la población en el nivel micro-evolutivo. Por tanto, las restricciones pueden influenciar directamente el proceso de búsqueda.

Chung y Reynolds [Chu96] utilizan un algoritmo híbrido que mezcla la programación cultural y GENOCOP, en el que se incorporan las restricciones del problema. Se considera que un individuo es *acceptable* si satisface todas las restricciones del problema. En caso contrario, se ajusta un *espacio de creencias* en el que evolucionan los individuos hasta ser aceptables. Puesto que GENOCOP asume un espacio de búsqueda convexo, es relativamente sencillo diseñar operadores que realicen la búsqueda hacia el límite entre las regiones factible y no factible.

Uno de los mayores problemas de este método es extender los operadores culturales para manejar restricciones no lineales, puesto que encontrar el límite entre las regiones factible y no factible de un problema resulta mucho más complicado.

Colonias de hormigas

Esta técnica es propuesta por Dorigo et al. [Col91, Dor89, Dor96, Dor97] y consiste en una meta-heurística para problemas difíciles de optimización combinatoria, tales como el problema del viajante de comercio. El algoritmo principal es un sistema multi-agente¹ donde se dan interacciones de bajo nivel entre los agentes (también llamados *hormigas*) resultando en un comportamiento complejo del conjunto. La idea se inspira en las colonias de hormigas reales, las cuales depositan en el suelo una sustancia química llamada *feromona*. Esta sustancia influye en el comportamiento de las hormigas de forma que tienden a seguir los caminos donde hay una mayor cantidad de feromona.

Bilchev y Parmee [Bil95, Bil96] proponen utilizar este método para manejo de restricciones; en primer lugar es necesario establecer un punto común de origen llamado *nido*, a partir del cual se realiza la búsqueda en un número finito de direcciones, a continuación se consideran tres niveles de abstracción:

1. Los agentes individuales de búsqueda; el nivel más bajo, en el cual puede uti-

¹El concepto de *agente* puede encontrarse en [Woo95].

lizarse cualquier técnica de búsqueda local.

2. La cooperación entre agentes; el nivel medio, que consiste en realizar una búsqueda conjunta en una cierta dirección.
3. La meta-cooperación entre agentes; el nivel más alto, que determina la cooperación entre diferentes direcciones de búsqueda.

El objetivo del algoritmo es buscar *fuentes de comida* que se modelizan utilizando las restricciones; una fuente de comida es aceptable si se satisfacen todas las restricciones y no es aceptable si se viola alguna de las restricciones. Al principio de la ejecución las restricciones se hacen más flexibles y conforme la ejecución avanza, se van endureciendo más las restricciones, de forma que una fuente de comida que en un momento dado era factible, puede convertirse más adelante en no factible.

Este método ha tenido muy buenos resultados con espacios de búsqueda con restricciones muy severas, sin embargo es necesario definir una serie de parámetros para que funcione de forma adecuada: en primer lugar hace falta utilizar un procedimiento aparte para localizar el *nido* (Bilchev y Parmee sugieren utilizar un algoritmo genético con nichos), lo cual implica un esfuerzo computacional extra. En segundo lugar, hay que definir el valor R o radio de búsqueda que define la porción de búsqueda que será explorada por las hormigas. En tercer lugar se debe evitar que las hormigas recorran el mismo camino más de una vez. Por último, también es necesario encontrar un equilibrio entre búsqueda local y global.

2.3.6 Conclusiones

En esta sección se han revisado las técnicas evolutivas más importantes para manejar restricciones en problemas de optimización simple. Estas técnicas pueden dividirse en diferentes grupos:

- Funciones de penalización

Las técnicas basadas en funciones de penalización han sido tradicionalmente las más utilizadas para manejo de restricciones. Su ventaja es que son métodos relativamente sencillos y computacionalmente eficientes. Sin embargo, estas técnicas dependen de la definición de ciertos parámetros, dependientes del problema concreto, para funcionar de forma correcta y establecer estos parámetros

no siempre resulta sencillo. Por otra parte, en problemas donde las restricciones son fuertes, estas técnicas pueden tener problemas para hallar soluciones factibles.

- Representación y operadores de variación específicos

Estos métodos se basan en definir esquemas de representación y operadores especiales para un determinado problema de forma que todos los individuos sean factibles. Estas técnicas son especialmente adecuadas en problemas donde resulte difícil encontrar soluciones factibles, sin embargo, estas técnicas son dependientes del problema y para muchos problemas puede ser complicado definir los esquemas de representación y operadores.

- Separación de objetivos y restricciones

Existen diversos métodos que utilizan separación de objetivos y restricciones; la gran ventaja de estos métodos es que, a diferencia de los anteriores, no son dependientes del problema. Por otra parte, las técnicas evolutivas son muy adecuadas para aplicar estos métodos y también para su posterior extensión a problemas de optimización multiobjetivo. Estas ventajas han hecho que estos métodos hayan obtenido un gran interés en la investigación actual.

- Métodos híbridos

Dentro de la actualidad de investigación, las técnicas de hibridación se encuentran en un punto de gran importancia. Aplicadas al problema concreto del manejo de restricciones, normalmente se utiliza algún método numérico para manejar las restricciones dentro de un algoritmo evolutivo. Estas técnicas han conseguido buenos resultados en problemas con restricciones fuertes, sin embargo, en muchos casos el coste computacional resulta excesivo y en otros casos son métodos complicados de llevar a la práctica.

- Otros métodos

Existen otros métodos que no corresponden con ninguno de los tipos anteriores. En algunos problemas, se han obtenido buenos resultados con estas técnicas, sin embargo, en muchas ocasiones estas técnicas resultan complejas o necesitan de ciertos parámetros dependientes del problema.

De todos estos métodos, para nuestro trabajo resultan más adecuados los que realizan separación de objetivos y restricciones por sus ventajas. En primer lugar, son métodos no dependientes del problema. Por otra parte, los algoritmos evolutivos se adaptan de forma muy adecuada a estas técnicas y su extensión para problemas multiobjetivo es directa. Por estas ventajas, como punto de partida de nuestra propuesta se ha tomado este tipo de métodos para manejar las restricciones.

2.4 Técnicas Evolutivas para Optimización Multiobjetivo

En esta sección estudiaremos técnicas evolutivas para resolver problemas multiobjetivo sin restricciones, es decir, problemas de la forma:

$$\text{Minimizar } f_i(\mathbf{x}), \quad i = 1, \dots, n \quad (2.2)$$

donde $\mathbf{x} = (x_1, \dots, x_p)$ es un vector de parámetros reales (o variables de decisión) $x_k \in \mathfrak{R}$ pertenecientes a un dominio $[l_k, u_k]$, $k = 1, \dots, p$, y $f_i(\mathbf{x})$, $i = 1, \dots, n$ son funciones arbitrarias lineales o no lineales. Podemos notar que no nos estamos restringiendo a problemas de minimización, puesto que los problemas de minimización y maximización pueden ser resueltos de forma equivalente.

En la sección 2.2 ya se han estudiado los métodos clásicos para resolver problemas de optimización multiobjetivo. Existen técnicas apropiadas para resolver problemas multiobjetivo lineales; pero, cuando las funciones objetivo son no lineales deben utilizarse otros métodos diferentes conocidos como *programación de compromiso*. Sin embargo, como ya se ha visto, la mayoría de estos métodos resuelven los problemas de optimización multiobjetivo reduciendo el vector de objetivos a uno sólo con la ayuda de algún conocimiento del problema. De esta forma, aunque se garantice encontrar soluciones no dominadas, se obtiene una única solución y el decisor debe conocer la prioridad de cada objetivo antes de construir la función objetivo individual. El inconveniente de estos enfoques es, por tanto, que ante situaciones diferentes del problema se requieren nuevas ejecuciones para la obtención de nuevas soluciones, ya que con cada ejecución obtenemos únicamente una solución que representa el mejor compromiso entre los objetivos para un ambiente de decisión concreto.

Los algoritmos evolutivos, al trabajar con una población compuesta por un conjunto de individuos, poseen un paralelismo que permite explorar múltiples soluciones de forma simultánea, obteniendo así en una sola ejecución múltiples soluciones no dominadas, resultando por tanto un método muy adecuado para resolución de problemas de optimización multiobjetivo.

El primer trabajo donde se sugiere utilizar un algoritmo evolutivo para resolver un problema multiobjetivo lo realiza Rosenberg a finales de los 60 [Ros67]; desde entonces esta nueva área de investigación (actualmente llamada optimización evolutiva

multiobjetivo) ha tenido cada vez una mayor relevancia y la efectividad de los Algoritmos Evolutivos para la resolución de problemas de optimización multiobjetivo ha sido ampliamente reconocida durante los últimos años. Prueba de ello es la inclusión cada vez más frecuente de sesiones especiales y workshops de optimización evolutiva multiobjetivo en el marco de Congresos de reconocido prestigio internacional, así como la aparición reciente de la Primera Conferencia Internacional sobre Optimización Evolutiva Multi-Criterio [Zit01a], celebrada en Zurich en Marzo de 2001.

Existen diferentes técnicas evolutivas para resolver problemas multiobjetivo. Una posible clasificación es la siguiente:

- Articulación de preferencias *a priori*.
- Articulación de preferencias *a posteriori*.

Otro aspecto importante son los problemas test [Deb01c]. En el siguiente capítulo se realizará un estudio de los problemas test actuales de mayor relevancia para optimización multiobjetivo.

Articulación de preferencias a priori

En estas técnicas, se deben tomar decisiones de preferencia antes de ejecutar el algoritmo; normalmente estas decisiones se realizan en forma de pesos, prioridades o preferencias asignadas a cada uno de los objetivos de forma que se convierte el problema multiobjetivo en un problema con un único objetivo. El resultado de estas técnicas es una sola solución no dominada que cumple las preferencias establecidas. Estas técnicas suelen ser rápidas y eficientes, el problema es que necesitamos saber a priori cuales son nuestras preferencias y si estas cambian en un futuro, es necesario volver a ejecutar el algoritmo para obtener nuevas soluciones de acuerdo con las nuevas preferencias. Entre las técnicas que se encuentran englobadas en este conjunto tenemos:

- Método de la Suma Ponderada
- Método de la Multiplicación
- Programación por Metas

- Consecución de Metas
- Aproximación Min-max con Pesos
- Orden Lexicográfico

Articulación de preferencias a posteriori

En estas técnicas el uso de algoritmos evolutivos alcanza su plena realización, puesto que la optimización se realiza sin tener en cuenta preferencias, evolucionando una población donde todos los objetivos se tienen en cuenta, de forma que el algoritmo obtiene al final un conjunto de soluciones no dominadas, entre las cuales el decisor puede tomar aquella que más le interese. La ventaja de estas técnicas es que no es necesario conocer de antemano las preferencias y si éstas cambian, no es necesario volver a ejecutar el algoritmo, sino simplemente se puede elegir otra solución del conjunto no dominado. Por otra parte, estas técnicas son las que realmente aprovechan la ventaja del paralelismo de los algoritmos evolutivos para resolver problemas multiobjetivo. Podemos distinguir las siguientes:

- Métodos no basados en el concepto Pareto
 - Algoritmo Genético sin Generaciones
 - Método de las restricciones ϵ
 - Algoritmo Genético de Vector Evaluado (VEGA)
 - Algoritmo Genético Basado en Pesos (WBGA)
 - Uso de la Teoría de Juegos
 - Uso del Género para identificar objetivos
 - Estrategia de Evolución Presa-Depredador
 - Algoritmo Genético con Participación Distribuida
 - Método del Aprendizaje por Refuerzo Distribuido
 - Sistema Neuro-Evolutivo Multiobjetivo (MONESSY)
- Métodos no elitistas basados en el concepto Pareto
 - Algoritmo de Ordenación Pareto

- Algoritmo Genético con Múltiples Objetivos (MOGA)
- Algoritmo Genético de Ordenación No Dominada (NSGA)
- Algoritmo Genético Pareto con Nichos (NPGA)
- Métodos elitistas basados en el concepto Pareto
 - Algoritmo Genético de Ordenación No Dominada Elitista (NSGA-II)
 - Algoritmo Genético Pareto basado en la Distancia (DPGA)
 - Algoritmo Evolutivo con Fuerzas Pareto (SPEA)
 - Algoritmo Genético Termodinámico (TDGA)
 - Estrategia de Evolución con Archivo Pareto (PAES)
 - Algoritmo Genético con Selección basada en Particiones Pareto (PESA)
 - Algoritmo Genético Desordenado Multiobjetivo (MOMGA)
 - Micro-Algoritmo Genético Multiobjetivo ($M\mu$ GA)
 - Algoritmo Evolutivo Multiobjetivo Elitista con Participación Co-evolutiva (ERMOCS)

2.4.1 Métodos con articulación de preferencias *a priori*

En este apartado se describen algunos de los algoritmos evolutivos que utilizan métodos con articulación de preferencias *a priori*.

Método de la Suma Ponderada

Este método consiste en sumar todas las funciones objetivo, utilizando para cada una un *coeficiente de peso* diferente. Con esto, se transforma el problema de optimización multiobjetivo en un problema de optimización con un único objetivo, de la forma:

$$\text{Minimizar } \sum_{i=1}^n w_i f_i(\mathbf{x})$$

donde $w_i \geq 0, i = 1, \dots, n$ son los coeficientes de peso que representan la importancia relativa de cada uno de los objetivos. Normalmente se asume que $\sum_{i=1}^n w_i = 1$.

Puesto que el resultado de resolver la ecuación anterior puede ser muy distinto según el conjunto de valores de los coeficientes de peso que se tomen, normalmente es necesario resolver el problema para diferentes conjuntos de valores y aun en este caso, el diseñador se encuentra con el problema de decidir qué conjuntos toma, lo cual no resulta trivial. Por otra parte, los valores w_i no reflejan proporcionalmente la importancia relativa de cada uno de los objetivos, pues para que esto fuera así, todas las funciones deberían expresarse en unidades del mismo valor numérico. Para esto, deberíamos transformar la ecuación anterior de la siguiente forma:

$$\text{Minimizar} \quad \sum_{i=1}^n w_i f_i(\mathbf{x}) c_i$$

donde $c_i, i = 1, \dots, n$ son coeficientes de multiplicación constantes que realizan el escalado adecuado de los objetivos.

Este método es la primera técnica desarrollada para obtener soluciones no inferiores en problemas de optimización multiobjetivo y ha sido utilizado por diversos autores (Syswerda y Palmucci [Sys91b], Jakob et al. [Jak92], Jones et al [Jon93], Wilson y Macleod [Wil93], Liu et al [Liu98], Yang y Gen [Yan94]). La principal ventaja de este método es su eficiencia (en términos computacionales) y puede ser utilizado para generar soluciones no dominadas que se utilicen como soluciones iniciales para otras técnicas. Su principal problema es la dificultad para determinar los pesos apropiados cuando no hay suficiente información acerca del problema y no es capaz de generar soluciones Pareto óptimas adecuadas cuando el espacio de búsqueda es no convexo, lo cual supone un serio problema en la mayoría de aplicaciones del mundo real.

Método de la Multiplicación

Las técnicas multiplicativas son métodos de agregación escalar donde los objetivos de un individuo se combinan mediante la multiplicación. La forma general de este método puede representarse matemáticamente de la siguiente forma:

$$\text{Minimizar} \quad \prod_{i=1}^n f_i(\mathbf{x})$$

Esta técnica puede utilizarse junto con una selección proporcional, de torneo, o basada en rango. Entre los autores que han utilizado esta técnica se encuentran

Horn y Nafpliotis [Hor93] que utilizan la multiplicación de objetivos junto con nichos y Wallace et al. [Wall96] que utilizan el logaritmo de la multiplicación de todos los objetivos.

Programación por Metas

Charnes y Cooper [Char61] e Ijiri [Iji65] desarrollan el método de programación por metas para modelos lineales. En este método, el decisor tiene que asignar valores *destino* o *metas* que se desean alcanzar para cada uno de los objetivos. La función objetivo deberá por tanto minimizar las desviaciones de cada objetivo con respecto a su valor destino. La forma más sencilla de este método puede describirse de la siguiente forma [Duc84]:

$$\text{Minimizar} \quad \sum_{i=1}^n |f_i(\mathbf{x}) - T_i|$$

donde T_i es el valor destino o meta fijada por el decisor para la función objetivo i -ésima $f_i(\mathbf{x})$.

Una formulación más general de este método consiste en utilizar pesos para ponderar cada objetivo. Dicha formulación ha sido llamada *programación por metas generalizada*:

$$\text{Minimizar} \quad \sum_{i=1}^n w_i \|f_i(\mathbf{x}) - T_i\|_{\alpha}^p$$

donde w_i es el peso para el objetivo i -ésimo, p es un parámetro entero y α es el valor de la norma aplicada para calcular la distancia. En el caso más común $\alpha = 2$ y se aplica la 2-norma, o distancia Euclídea.

Esta técnica resulta computacionalmente eficiente, en caso de que conozcamos las metas a las cuales deseamos llegar. Sin embargo, es necesario decidir qué valores asignar a dichas metas, lo cual puede resultar complicado si no se tiene un conocimiento a priori del problema. Esta técnica puede ser útil cuando se pueda realizar una aproximación por tramos lineales de las funciones objetivos, pero en los casos no lineales, pueden resultar más eficientes otros métodos.

Consecución de Metas

Este método requiere definir un vector de pesos (w_1, \dots, w_n) relacionados con la consecución de las metas, así como el vector de metas (b_1, \dots, b_n) para las funciones objetivo. El problema por tanto queda planteado de la siguiente forma:

$$\begin{aligned} & \text{Minimizar } \alpha \\ & \text{sujeeto a : } b_i + \alpha w_i \geq f_i(\mathbf{x}), \quad i = 1, \dots, n \end{aligned}$$

donde α es una variable escalar sin restricciones y los pesos (w_1, \dots, w_n) están normalizados de forma que $\sum_{i=1}^n |w_i| = 1$.

Si un peso w_i es igual a 0, esto significa que el límite máximo para la función objetivo $f_i(\mathbf{x})$ es igual a b_i . El conjunto de soluciones no dominadas para un problema puede ser generado variando los pesos w_i [Che94], incluso para problemas no convexos. El valor α obtenido indica si las metas se han conseguido o no. Un valor de α negativo implica que las metas han sido conseguidas. En otro caso, si $\alpha > 0$, significa que no se han alcanzado las metas.

Aproximación Min-Max con Pesos

La idea de establecer el *óptimo min-max* y aplicarlo a problemas de optimización multiobjetivo se toma, en primer lugar, de la teoría de juegos para tratar con múltiples objetivos en conflicto. La aproximación min-max para modelos lineales es propuesta por Jutler [Jut67] y Solich [Sol69] y posteriormente ampliada por Osyczka [Osy78, Osy81, Osy85], Rao [Rao87] y Tseng y Lu [Tse90].

El óptimo min-max compara las desviaciones relativas de los óptimos alcanzables por cada objetivo por separado. Para el objetivo i -ésimo, la desviación relativa puede ser calculada como:

$$z'_i(\mathbf{x}) = \frac{|f_i(\mathbf{x}) - f_i^0|}{|f_i^0|}$$

donde f_i^0 es el valor óptimo alcanzable para el objetivo i -ésimo. Otra forma de calcular la desviación relativa es:

$$z''_i(\mathbf{x}) = \frac{|f_i(\mathbf{x}) - f_i^0|}{|f_i(\mathbf{x})|}$$

La primera de las anteriores ecuaciones define los incrementos relativos de la función objetivo i -ésima y la segunda define los decrementos relativos.

Sea $\mathbf{z}(\mathbf{x}) = (z_1(\mathbf{x}), \dots, z_n(\mathbf{x}))$ el vector de incrementos relativos. Los componentes del vector $\mathbf{z}(\mathbf{x})$ pueden ser calculados de la siguiente forma:

$$z_i(\mathbf{x}) = \max \{z'_i(\mathbf{x}), z''_i(\mathbf{x})\}, \quad i = 1, \dots, n$$

Se dice que un punto \mathbf{x}^* es *óptimo min-max* si se cumple el siguiente conjunto de ecuaciones:

$$v_1(\mathbf{x}^*) = \min_{\mathbf{x} \in X} \max_{i \in I} \{z_i(\mathbf{x})\}$$

y entonces $I_1 = \{i_1\}$, donde i_1 es el índice para el cual el valor $z_{i_1}(\mathbf{x})$ es maximal en la ecuación anterior. Si existe un conjunto de soluciones X_1 que satisface la ecuación v_1 , entonces:

$$v_2(\mathbf{x}^*) = \min_{\mathbf{x} \in X_1} \max_{i \in I, i \notin I_1} \{z_i(\mathbf{x})\}$$

siendo ahora $I_2 = \{i_1, i_2\}$ e i_2 el índice para el cual el valor de $z_{i_2}(\mathbf{x})$ es maximal en la ecuación v_2 . En general tendremos:

$$v_k(\mathbf{x}^*) = \min_{\mathbf{x} \in X_{k-1}} \max_{i \in I, i \notin I_{k-1}} \{z_i(\mathbf{x})\}$$

donde $v_1(\mathbf{x}^*), \dots, v_k(\mathbf{x}^*)$ es el conjunto de valores óptimos de las desviaciones fraccionales ordenadas de forma decreciente. El punto \mathbf{x}^* es la mejor solución de compromiso considerando todos los objetivos simultáneamente y de igual importancia.

Orden Lexicográfico

En este método se ordenan los objetivos en orden según la importancia o prioridad asignada a priori a cada uno de los objetivos. La solución óptima se obtiene minimizando las funciones objetivo, una por una, empezando por la de mayor prioridad y siguiendo en orden de prioridad.

Supongamos que los subíndices de las funciones objetivo indican la importancia de dicha función, de forma que $f_1(\mathbf{x})$ y $f_n(\mathbf{x})$ indican las funciones objetivo de mayor y menor prioridad respectivamente. En este caso, el primer problema se formula como:

$$\text{Minimizar } f_1(\mathbf{x})$$

encontrándose la solución \mathbf{x}_1 . El segundo problema, entonces, será:

$$\begin{aligned} &\text{Minimizar } f_2(\mathbf{x}) \\ &\text{sujeto a : } f_1(\mathbf{x}) = f_1(\mathbf{x}_1) \end{aligned}$$

y se obtiene la solución \mathbf{x}_2 . En general, el problema i -ésimo se plantea de la forma:

$$\begin{aligned} & \text{Minimizar} && f_i(\mathbf{x}) \\ & \text{sujeito a :} && f_k(\mathbf{x}) = f_k(\mathbf{x}_k) \quad k = 1, \dots, i-1 \end{aligned}$$

para encontrar la solución i -ésima \mathbf{x}_i . Este procedimiento se repite hasta que se han considerado los n objetivos. La solución n -ésima \mathbf{x}_n obtenida al final del procedimiento se toma como la solución deseada del problema.

Fourman [Fou85] utiliza un esquema de selección basado en un orden lexicográfico. En una primera versión de su algoritmo, las prioridades son asignadas a los objetivos por el usuario. En una segunda versión, se selecciona de forma aleatoria un objetivo en cada ejecución. Kursawe [Kurs91] formula una estrategia de evolución multiobjetivo basada en orden lexicográfico. La selección se realiza seleccionando de forma aleatoria cada uno de los objetivos según un vector de probabilidades.

La ventaja de esta aproximación está en su simplicidad y en algunos casos puede identificar soluciones no dominadas sobre un espacio de búsqueda cóncavo, lo cual es una ventaja respecto a VEGA, aunque esto depende de la distribución de la población y del tipo de problema. Sin embargo, en esta aproximación se tiende a favorecer más unos objetivos que otros, lo cual llevará a que la población converja a una parte limitada del frente Pareto en lugar de distribuirse por todo el frente.

2.4.2 Métodos *a posteriori* no basados en el concepto Pareto

Como ya se ha dicho, los algoritmos evolutivos multiobjetivo alcanzan su realización plena cuando se utilizan métodos *a posteriori*. Dentro de este apartado se describen algunos de los algoritmos evolutivos que utilizan métodos con articulación de preferencias *a posteriori*, pero sin basarse en el concepto de óptimo Pareto.

Algoritmo Genético sin Generaciones

Valenzuela-Rendón y Uresti-Charre [Val97] proponen un algoritmo genético que utiliza selección sin generaciones, es decir, simplemente se reemplaza el peor individuo de la población y a continuación se actualizan los valores de adecuación del resto de la población. El cálculo del valor de adecuación se hace de forma incremental. El algoritmo de Valenzuela y Uresti transforma el problema con n objetivos en otro problema con sólo dos objetivos:

1. Minimizar la *cuenta de dominación*; la media ponderada del número de individuos que han dominado a este individuo hasta el momento. Este valor se actualiza en cada generación de la siguiente forma:

$$d(\mathbf{x}, t+1) = d(\mathbf{x}, t) - k_d d(\mathbf{x}, t) + D(\mathbf{x}, t)$$

donde $d(\mathbf{x}, t)$ es el valor de la cuenta de dominación en la iteración t , $k_d = 0$ en el experimento realizado por Valenzuela y $D(\mathbf{x}, t)$ tiene valor 1 si \mathbf{x} es dominado en la iteración t y 0 en otro caso.

2. Minimizar la *cuenta de nicho*; la media ponderada del número de individuos que se encuentran próximos a éste, de acuerdo a una cierta función de participación. Este valor también puede calcularse incrementalmente mediante la siguiente ecuación:

$$w(\mathbf{x}, t+1) = w(\mathbf{x}, t) - k_w w(\mathbf{x}, t) + nicho(\mathbf{x})$$

donde $w(\mathbf{x}, t)$ es el valor de la cuenta de dominación en la iteración t , $k_w = 1/P$ en el experimento de Valenzuela (P es el número de individuos tomados para realizar la selección por torneo) y $nicho(\mathbf{x}, t)$ se refiere al valor de nicho del individuo \mathbf{x} en la iteración t calculado como en (2.5) tomando $d_{jk} = |\mathbf{x}^j - \mathbf{x}^k|$.

A partir de las ecuaciones anteriores, el problema se transforma en un problema de optimización con un único objetivo tomando una combinación lineal de los dos objetivos anteriores de la forma:

$$F(\mathbf{x}, t) = c_d d(\mathbf{x}, t) + c_w w(\mathbf{x}, t)$$

siendo c_d y c_w constantes elegidas arbitrariamente que expresan el compromiso entre los dos objetivos.

Método de las restricciones ϵ

Este método es sugerido por Ritzel y Wayland [Rit94] como un método sencillo de resolver problemas de optimización multiobjetivo utilizando algoritmos genéticos. Se basa en la minimización de una sola de las funciones objetivo (la principal o más importante) y se consideran las otras funciones objetivo como restricciones para algún nivel permitido ϵ_i . Variando los niveles ϵ_i se puede obtener todo el conjunto Pareto óptimo. El método puede formularse de la siguiente forma:

1. Encontrar el mínimo de la r -ésima función objetivo, es decir:

$$\begin{aligned} & \text{Minimizar } f_r(\mathbf{x}) \\ & \text{sujeto a : } f_i(\mathbf{x}) \leq \epsilon_i, \quad i = 1, \dots, n \quad i \neq r \end{aligned}$$

2. Repetir el paso anterior para diferentes valores de ϵ_i hasta encontrar una solución satisfactoria.

Para obtener valores adecuados de ϵ_i , se efectúan normalmente optimizaciones individuales para cada una de las funciones objetivo en turno. Para cada función objetivo $f_i(\mathbf{x})$ hay un *vector de diseño óptimo* \mathbf{x}_i^* para el cual $f_i(\mathbf{x}_i^*)$ es un mínimo. Sea $f_i(\mathbf{x}_i^*)$ el límite inferior de ϵ_i :

$$\epsilon_i \geq f_i(\mathbf{x}_i^*) \quad i = 1, \dots, n \quad i \neq r$$

y $f_i(\mathbf{x}_r^*)$ el límite superior de ϵ_i :

$$\epsilon_i \leq f_i(\mathbf{x}_r^*) \quad i = 1, \dots, n \quad i \neq r$$

Cuando los límites ϵ_i son demasiado pequeños, no hay solución y al menos uno de ellos debe relajarse.

Loughlin y Ranjithan [Lou97] proponen una aproximación a este método que llaman *NCGA* (*neighborhood constrained GA*) en la que, en lugar de utilizar diferentes vectores ϵ en diferentes ejecuciones, a cada individuo de la población se le asigna un vector ϵ diferente. Los vectores ϵ son inicializados dentro de unos niveles de restricción máximos y mínimos. Para mantener la diversidad en la población, la selección se realiza entre individuos cercanos; es necesario, por tanto establecer un parámetro de distancia para decidir si dos individuos son cercanos; también es necesario establecer los valores máximos y mínimos para cada restricción.

Algoritmo Genético de Vector Evaluado (VEGA)

David Shaffer [Sch85] desarrolla una extensión para el programa GENESIS de Grefenstette [Gre84] en la que se incluyen funciones multiobjetivo. El método utilizado por Shaffer consiste en utilizar una extensión del Algoritmo Genético Simple (Simple Genetic Algorithm, SGA) que llama *Algoritmo Genético de Vector Evaluado* (*Vector*

Evaluated Genetic Algorithm, VEGA). La diferencia con el primero se encuentra sólo en la forma de realizar la selección: para un problema con n objetivos, se crean n subpoblaciones de tamaño N/n (suponiendo que N es el tamaño de la población total). En cada una de estas subpoblaciones se realiza la selección de individuos de forma independiente atendiendo a su objetivo correspondiente para obtener una nueva población total de tamaño N , en la cual se aplicarán en la forma usual los operadores genéticos del GA (cruce y mutación).

El esquema de VEGA es el siguiente:

1. Dividir de forma aleatoria la población P_t de tamaño N en n subpoblaciones P_t^1, \dots, P_t^n de tamaño N/n cada una.
2. Calcular la adecuación de un individuo \mathbf{x} que se encuentra en la subpoblación P_t^i como el valor de su función objetivo i -ésima, $f_i(\mathbf{x})$.
3. Aplicar un operador de selección proporcional en cada subpoblación según el valor de adecuación calculado anteriormente y crear n subpoblaciones Q_t^1, \dots, Q_t^n .
4. Unir las n subpoblaciones Q_t^1, \dots, Q_t^n para formar una nueva población Q_t :

$$Q_t = \cup_{i=1}^n Q_t^i.$$
5. Aplicar los operadores de cruce y mutación en la población Q_t para crear la nueva población P_{t+1} .

Éste es el primer esquema práctico de Algoritmo Evolutivo que incorpora múltiples objetivos, sin embargo tiene algunos problemas. En concreto, este método produce un problema conocido en genética como *especiación*², es decir, evolución de diferentes *especies* dentro de la población global, cada una de las cuales se especializa en mejorar uno de los objetivos, ignorando el resto. Este problema surge porque con esta técnica se seleccionan individuos que son buenos en alguna de las funciones objetivo, sin tener en cuenta las otras. Con esto, se ignoran soluciones que son buenas a través de todos los criterios; individuos que, sin ser los mejores en ningún objetivo concreto, son moderadamente buenos en todos los criterios. Esto resulta contrario a nuestro

²Traducción del término inglés *speciation*.

objetivo que es, precisamente, encontrar soluciones de compromiso. Shaffer sugiere algunas heurísticas para tratar de solucionar este problema; utilizar una heurística de selección dando preferencia a aquellos individuos con soluciones de compromiso, o bien dando preferencia al cruce entre individuos de diferentes especies.

La principal ventaja de VEGA es su simplicidad; sin embargo, con esta técnica no se pueden encontrar soluciones Pareto óptimas cuando el espacio de búsqueda es no convexo y no puede encontrar soluciones que se encuentren en una región cóncava.

Ritzel y Wayland [Rit94] utilizan una variación de VEGA en la que incorporan un parámetro para controlar el ratio de selección.

Cvetković et al. [Cve98] proponen diversas técnicas para solucionar los problemas de VEGA; esperar un cierto número de generaciones antes de mezclar las poblaciones, o no realizar la mezcla de poblaciones y en su lugar realizar copias o migraciones de una cierta cantidad de individuos desde una subpoblación hasta otra.

Tamaki et al. [Tam95] desarrollan una técnica de elitismo en la cual los individuos no dominados de cada generación se mantienen en la siguiente generación. Esta aproximación realmente es una mezcla entre VEGA y selección Pareto (ver sección 2.4.3). En una versión posterior de este algoritmo, llamado *estrategia de Reserva Pareto*, Tamaki et al. [Tam96] también utilizan una función de participación de la adecuación entre individuos no dominados para mantener la diversidad en la población.

Algoritmo Genético Basado en Pesos

Hajela y Lin [Haj92] proponen un *Algoritmo Genético Basado en Pesos* (*Weight-Based Genetic Algorithm, WBGA*); incluyen los pesos de cada objetivo dentro del cromosoma. Es decir, para cada solución \mathbf{x} se asocia un vector de pesos diferente $\mathbf{w} = (w_1, \dots, w_n)$, tales que $\sum_{i=1}^n w_i = 1$. La función de adecuación es de la forma:

$$F(\mathbf{x}) = \sum_{i=1}^n w_i f'_i(\mathbf{x}) \quad (2.3)$$

donde w_i es el peso para el objetivo i -ésimo asociado a la solución \mathbf{x} y $f'_i(\mathbf{x})$ es el valor normalizado de la función objetivo i -ésima para la solución \mathbf{x} :

$$f'_i(\mathbf{x}) = \frac{f_i(\mathbf{x}) - f_i^{\min}}{f_i^{\max} - f_i^{\min}}$$

siendo f_i^{min} y f_i^{max} los valores mínimo y máximo respectivamente para la función objetivo i -ésima dentro de la población.

En este esquema, lo importante es mantener la diversidad de pesos en la población, para ello Hajela y Lin utilizan una *función de participación*³ de la adecuación de la forma:

$$participación(d_{jk}) = \begin{cases} 1 - \left(\frac{d_{jk}}{\sigma_{participación}} \right)^\alpha, & \text{si } d_{jk} < \sigma_{participación} \\ 0, & \text{en otro caso} \end{cases} \quad (2.4)$$

con $\alpha = 1$ y $\sigma_{participación}$ es el *factor de participación* entre 0.01 y 0.1 El valor d_{jk} hace referencia a la distancia entre los pesos de la solución j -ésima \mathbf{x}^j y la solución k -ésima \mathbf{x}^k :

$$d_{jk} = |\mathbf{w}^j - \mathbf{w}^k|$$

donde \mathbf{w}^j y \mathbf{w}^k son los vectores de pesos asociados a las soluciones \mathbf{x}^j y \mathbf{x}^k .

Para una solución \mathbf{x}^j se define el valor *cuenta de nicho* como:

$$nicho(\mathbf{x}^j) = \sum_{k=1}^N participación(d_{jk}) \quad (2.5)$$

donde N es el número de soluciones en la población.

La adecuación de una solución \mathbf{x} queda, por tanto, modificada de la forma:

$$F(\mathbf{x}) = \frac{F(\mathbf{x})}{nicho(\mathbf{x})}$$

La ventaja de este método es su simplicidad y eficiencia al no requerir comprobar la no dominación y el uso de una función de participación consigue una buena diversidad de la población evitando una convergencia prematura; sin embargo, no resulta fácil determinar el factor de participación, incrementándose el número de parámetros requeridos por el algoritmo genético.

Otra propuesta de Hajela y Lin [Haj92] para mantener la diversidad es utilizar un esquema similar al VEGA para mantener la diversidad entre los vectores de pesos:

1. Crear k vectores de pesos $\mathbf{w}^1, \dots, \mathbf{w}^k$.
2. Repetir para $i = 1$ hasta k .

³Traducción del término inglés *sharing*.

- (a) Evaluar los individuos de la población P_t con el vector de peso w^i según la ecuación (2.3).
 - (b) Seleccionar los N/k mejores individuos para crear la subpoblación P_t^i .
3. Realizar la selección, cruce y mutación en las subpoblaciones P_t^1, \dots, P_t^k para crear nuevas subpoblaciones Q_t^1, \dots, Q_t^k .
 4. Combinar las subpoblaciones Q_t^1, \dots, Q_t^k para formar una nueva población P_{t+1} :

$$P_{t+1} = \cup_{i=1}^k Q_t^i.$$

La ventaja de esta aproximación es que no requiere determinar el factor de participación.

Coello [Coe96, Coe97] propone dos variaciones del método utilizado por Hajela y Lin. En la primera aproximación, se debe establecer un conjunto de pesos predefinidos que se utiliza para expandir múltiples pequeñas subpoblaciones que evolucionan de forma separada, pero concurrente, convergiendo cada una a un punto Pareto. Esta técnica resulta eficiente y relativamente simple, sin embargo no siempre resulta fácil establecer un conjunto de pesos apropiado para encontrar el frente Pareto.

En su segunda aproximación, Coello propone el uso de un *vector ideal local*, el cual se calcula en cada generación y la selección se realiza por torneo min-max. Es decir, un individuo se considera el ganador de un torneo si su desviación máxima con respecto al vector ideal es la menor de entre todos sus competidores. Este método también incorpora una función de participación para evitar la convergencia prematura. Esta aproximación, al no utilizar pesos, resulta mucho más eficiente, aunque depende en este caso del valor $\sigma_{participación}$ para obtener buenos resultados.

Murata e Ishibuchi [Mur95] proponen un *Algoritmo Genético de Pesos Aleatorios* (*Random Weighted Genetic Algorithm, RWGA*) similar al anterior WBGA, con la diferencia que a cada individuo se le asocia un vector de pesos aleatorios. En el algoritmo RWGA, la diversidad entre soluciones no dominadas se mantiene de dos formas: (i) utilizando un vector de pesos aleatorio para evaluar cada solución, y (ii) utilizando un operador de intercambio que reemplaza soluciones de la población por otras soluciones de un conjunto externo. Por otra parte, RWGA mantiene el elitismo, ya que el conjunto de soluciones no dominadas que se obtiene en cada generación se mantiene separado de la población actual.

Uso de la Teoría de Juegos

Si tenemos un problema con n objetivos, podemos utilizar la teoría de juegos asumiendo un jugador por cada objetivo. Cada jugador intentará minimizar uno de los objetivos. Suponiendo un *juego no cooperativo* finalmente se llegará a un punto donde se producirá la convergencia entre los distintos jugadores; a este punto se le denomina *solución de equilibrio Nash* [Nash50] y representa una condición de equilibrio estable, en el sentido de que ningún jugador puede desviarse unilateralmente de dicho punto para mejorar su propio criterio.

Périaux et al. [Per97] proponen un algoritmo genético basado en esta aproximación. Utilizan dos jugadores sin cooperación representados por dos subpoblaciones del algoritmo genético. En cada iteración se aplican los operadores genéticos de forma independiente en las subpoblaciones para minimizar cada objetivo por separado y después el mejor individuo de cada población migra a la otra población. Este proceso se repite hasta que se alcanza el equilibrio Nash.

La principal ventaja de este método es su eficiencia computacional, aunque en el trabajo de Périaux no era posible generar más de una solución no dominada. Sin embargo, es posible extender esta aproximación a n jugadores (siendo n el número de objetivos) y obtener múltiples puntos de equilibrio Nash con los cuales se puede encontrar el frente Pareto. En este caso, utilizar el juego cooperativo resultaría mejor que utilizar el juego no cooperativo [Rao87].

Uso del Género para identificar objetivos

Robin Alleson [Alle92] utiliza una técnica poblacional basada en VEGA la cual requiere diferenciación entre géneros masculino y femenino de forma que sólo se permite el cruce entre individuos de diferente género. La población se inicializa con individuos a los cuales se asigna el género de forma aleatoria, con igual probabilidad para cada género, de forma que se espera un número similar entre individuos masculinos y femeninos. Sin embargo, este equilibrio no se mantiene después de aplicar los operadores genéticos. En cada generación, el peor individuo de cada género se elimina siendo reemplazado por otro individuo escogido aleatoriamente del mismo género. Alleson usa estrategias evolutivas para implementar una especie de atractores sexuales que modifican la forma en que se efectúa el cruce. La idea es modelizar la atracción sexual que algunos individuos tienen hacia otros en la naturaleza.

Lis y Eiben [Lis96] también incorporan el género en su algoritmo genético, aunque de una forma más general. En este caso, el número de géneros no está limitado a dos, sino que hay tantos géneros como objetivos. Otra diferencia de este algoritmo consiste en el operador de cruce, que permite cruzar varios padres (más de dos) para obtener un solo hijo, en lugar del cruce tradicional en el que dos padres generan dos hijos. La idea consiste en seleccionar un padre de cada género para la obtención de un hijo. Por tanto, si tenemos n objetivos, tenemos n géneros y se cruzarán n padres para obtener un hijo. Por último se mantiene una lista de individuos no dominados, que se actualiza en cada generación eliminando los individuos que dejan de ser no dominados y añadiendo los nuevos individuos no dominados. Al final del algoritmo, en esta lista se encuentran las soluciones Pareto óptimas.

Estrategia de Evolución Presa-Depredador

Laumanns et al. [Lau98] sugieren una estrategia de evolución multiobjetivo que no usa la dominación para asignar un valor de adecuación, sino que utiliza el concepto de presa-depredador. Las presas representan un conjunto de soluciones; los depredadores se asocian con una función objetivo en concreto de forma que haya por lo menos un depredador por cada función objetivo; por tanto si tenemos una función con n objetivos, el número de depredadores deberá ser mayor o igual que n . Cada depredador busca en sus inmediaciones alguna presa y captura aquella que tiene el peor valor en la función objetivo asociada a dicho depredador. Cuando se captura una presa, se obtiene una nueva solución mutando y cruzando alguna otra presa que se encuentre cercana. Es decir, en este esquema tenemos funcionando múltiples depredadores simultáneos, cada uno de ellos favoreciendo cada uno de los objetivos; eventualmente se encontrarán soluciones buenas para todos los objetivos.

La mayor ventaja de este esquema es su simplicidad, sin embargo no utiliza ningún método para mantener la diversidad de las soluciones. Por otra parte, es necesario definir la relación entre presa y depredador de forma adecuada; si hay demasiadas presas y pocos depredadores, se enfatizan los óptimos de objetivos individuales, sin encontrar soluciones con valores intermedios en todos los objetivos; pero si hay demasiados depredadores y pocas presas, ocurre lo contrario.

Deb [Deb01a] propone una modificación del esquema anterior introduciendo un conjunto de pesos, de forma que a cada depredador se asocia un vector de pesos

diferentes. Cada depredador evalúa sus presas con respecto a la suma ponderada de los objetivos. Puesto que cada depredador enfatiza más las soluciones correspondientes a su vector de pesos, este algoritmo es capaz de mantener la diversidad en las soluciones.

Algoritmo Genético con Participación Distribuida

Hiroyasgu et al. [Hir99] sugieren un algoritmo genético con participación distribuida. Se utiliza el modelo de islas para mantener la diversidad. La población se divide en un número de subpoblaciones (o islas) y los operadores genéticos se aplican de forma independiente en cada isla. Se permiten migraciones ocasionales de soluciones de unas islas a otras. Cuando el tamaño del conjunto no dominado es mayor que un número dado, aplicando una función de participación con un valor $\sigma_{participación}$ adecuado, entonces estas soluciones se dividen nuevamente en más islas y continúa el proceso.

Método del Aprendizaje por Refuerzo Distribuido

Mariano y Morales [Mar00] sugieren un método de aprendizaje por refuerzo distribuido, en el cual una familia de agentes son asignados a las diferentes funciones objetivo. Cada agente propone una solución que optimiza su función objetivo. Todas las soluciones se combinan y se identifica el conjunto de soluciones no dominadas. Cada solución no dominada es recompensada. El mecanismo de recompensa proporciona al algoritmo una dirección para moverse hacia la región Pareto-óptima.

Sistema Neuro-Evolutivo Multiobjetivo (MONESSY)

Köppen et al. [Kop97] proponen un sistema neuro-evolutivo (*Neural Evolutional Strategy SYstem, NESSY*) para optimización con un único objetivo; este sistema tiene una red neuronal con tres capas; en la primera capa (de solución), cada neurona representa un vector de variables de decisión; en la segunda capa (de generación), cada neurona representa una sola variable de decisión. Por tanto, si hay p variables de decisión, en esta capa tenemos p neuronas. Por último, la tercera capa (de salida) representa el objetivo a optimizar. Köppen et al. sugieren también una ampliación de NESSY para tratar múltiples objetivos (MONESSY) utilizando múltiples neuronas en la capa de salida. La adecuación de las neuronas se realiza eligiendo una función objetivo aleatoria.

2.4.3 Métodos no elitistas basados en el concepto Pareto

Los algoritmos revisados en este apartado realizan también una selección de preferencias a posteriori de igual forma que los del apartado anterior, pero a diferencia de los anteriores, estos algoritmos se basan en el concepto Pareto para comparar diferentes soluciones y hacer la selección de individuos.

Algoritmo de Ordenación Pareto

Goldberg [Gol89] es el primero en introducir la asignación de adecuaciones basada en la no dominación para resolver los problemas que presenta el algoritmo VEGA de Shaffer. Con este método, la población es ordenada en base a la no dominación, a través de la obtención de *frentes Pareto optimales*. Primero se identifican y marcan todos los individuos no dominados en la población, asignando a todos la posición 1 (primer frente Pareto optimal). Estos individuos son ignorados temporalmente del proceso y se identifica el siguiente conjunto de individuos no dominados, asignando a todos la posición 2 (segundo frente Pareto optimal). El proceso continúa hasta que toda la población está clasificada. Las probabilidades de selección pueden ser entonces asignadas de acuerdo a las posiciones de los individuos. Goldberg sugiere que, para mantener una diversidad apropiada en la población, este método debe ser usado en combinación con alguna técnica de formación de especies y nichos.

La mayor desventaja de la ordenación Pareto es que no existen algoritmos eficientes para comprobar la no dominación en un conjunto de soluciones factibles. La eficiencia de estos algoritmos disminuye drásticamente cuando aumenta el tamaño de la población o el número de objetivos. Por otra parte, la necesidad de utilizar una función de participación para la formación de nichos, requiere estimar el valor de $\sigma_{participación}$ lo cual no resulta fácil y en general, la eficiencia del algoritmo depende en gran medida de dicho valor. A pesar de estos inconvenientes, la ordenación Pareto es la forma más apropiada para generar soluciones que cubran el frente Pareto completamente en una sola ejecución del algoritmo genético, siendo además independiente de la forma o continuidad del frente Pareto.

Algoritmo Genético con Múltiples Objetivos (MOGA)

Fonseca y Fleming [Fon93] proponen un algoritmo genético con múltiples objetivos (*Multiple Objective Genetic Algorithm, MOGA*) en el cual el rango de un cierto individuo corresponde al número de individuos en la población actual que lo dominan. El rango de un individuo puede calcularse de la forma:

$$rango(\mathbf{x}) = 1 + p(\mathbf{x})$$

donde $p(\mathbf{x})$ es el número de individuos de la población que dominan a \mathbf{x} .

Todos los individuos no dominados tienen rango 1, mientras que los individuos dominados se penalizan de acuerdo a la densidad de población de la región correspondiente.

La función de adecuación se asigna de la siguiente forma:

1. Ordenar la población según la función *rango*.
2. Asignar la adecuación a los individuos interpolando desde el mejor (rango 1) hasta el peor (rango $k \leq N$) de acuerdo a alguna función normalmente lineal, aunque podría ser una función no lineal.
3. Calcular la media de la adecuación de individuos con el mismo rango, de forma que todos ellos sean muestreados con la misma frecuencia. Este procedimiento mantiene la adecuación de la población constante.

Tal como Goldberg y Deb [Gol91] señalan, este tipo de asignación en bloque de la adecuación produce una gran presión de selección, que puede llevar a una convergencia prematura. Para evitar esto, Fonseca y Fleming utilizan un método de formación de nichos para distribuir la población por todo el frente Pareto óptimo. La función de participación se define de la forma usual tal como se indica en la ecuación (2.4) con $\alpha = 1$. La distancia entre individuos se calcula como la distancia euclídea normalizada en el espacio de las funciones objetivo.

$$d_{jk} = \sqrt{\sum_{i=1}^n \left(\frac{f_i(\mathbf{x}^k) - f_i(\mathbf{x}^j)}{f_i^{max} - f_i^{min}} \right)^2}$$

donde f_i^{max} y f_i^{min} son los valores máximos y mínimos respectivamente para la función objetivo i -ésima. El contador de nicho se define sólo entre individuos con igual valor

de rango Pareto:

$$nicho(\mathbf{x}^j) = \sum_{k=1}^{\mu(j)} participación(d_{jk})$$

siendo $\mu(j)$ el número de individuos cuyo rango es igual a \mathbf{x}^j .

Algoritmo Genético de Ordenación No Dominada (NSGA)

Srinivas y Deb [Sri94] implementan la idea de Goldberg con algunas variaciones en su Algoritmo Genético de Ordenación No Dominada (*Non-dominated Sorting Genetic Algorithm, NSGA*). Este algoritmo se basa en mantener múltiples niveles de clasificación de los individuos. Es decir, los individuos se clasifican por *frentes no dominados*. Antes de realizar la selección, se ordena la población en base a la no dominación: todos los individuos no dominados se clasifican en una categoría con un *valor de adecuación ficticio* proporcional al tamaño de la población, para que todos los individuos tengan la misma probabilidad de reproducirse. Seguidamente, y con objeto de mantener la diversidad, se calcula la función de participación para estos individuos a partir de su valor de adecuación ficticio. Se pasa entonces al siguiente nivel de individuos no dominados. El proceso continúa hasta que todos los individuos de la población son clasificados.

Para este método se utiliza una selección proporcional estocástica. Puesto que los individuos del primer nivel siempre tendrán el máximo valor de adecuación, con este esquema de selección siempre conseguirán más copias que el resto de la población. Esto permite buscar regiones no dominadas y converger rápidamente a tales regiones. Se utiliza una función de participación para mantener la dispersión entre los individuos de cada frente no dominado. La distancia de participación entre dos individuos se calcula como la distancia euclídea normalizada en el espacio de decisión, a diferencia de MOGA donde se calcula en el espacio objetivo, es decir:

$$d_{jk} = \sqrt{\sum_{i=1}^p \left(\frac{x_i^k - x_i^j}{x_i^{max} - x_i^{min}} \right)^2}$$

donde x_i^{max} y x_i^{min} son los valores máximos y mínimos respectivamente para el parámetro de decisión i -ésimo.

El valor de la función de participación se calcula utilizando la ecuación (2.4) con $\alpha = 2$, teniendo en cuenta solamente los individuos pertenecientes al mismo frente

Pareto.

La eficiencia de NSGA reside en la forma en que múltiples objetivos se reducen a un solo valor de adecuación ficticio utilizando un procedimiento de ordenación no dominado.

Leung et al. [Leu98] realizan una modificación de NSGA introduciendo un criterio de aceptación basado en el enfriamiento simulado. En esta aproximación se realiza una ordenación no dominada en un algoritmo genético de enfriamiento (*Non-dominated Sorting in Annealing Genetic Algorithm, NSAGA*) utilizando el concepto de reducción de la temperatura del enfriamiento simulado. La población de hijos es aceptada cuando la probabilidad de crear tal población y, de aceptarla, (con una temperatura dada) es la apropiada.

Algoritmo Genético Pareto con Nichos (NPGA)

Horn y Nafpliotis [Hor93] proponen un esquema de selección de torneo basado en la dominación Pareto (*Niched Pareto Genetic Algorithm, NPGA*). La diferencia de este método con respecto a los anteriores está en su operador de selección; en lugar de utilizar una selección proporcional, NPGA utiliza una selección de torneo. El torneo entre dos individuos i y j se realiza de la siguiente forma:

1. Seleccionar una subpoblación aleatoria P_t de tamaño $N_t < N$ (típicamente $N_t = 10$).
2. Calcular α_i como el número de individuos de P_t que dominan al individuo i y α_j como el número de individuos de P_t que dominan a j .
3. Si $\alpha_i = 0$ y $\alpha_j > 0$ elegir i .
4. Si $\alpha_i > 0$ y $\alpha_j = 0$ elegir j .
5. En otro caso, calcular el valor para la cuenta de nicho de i y de j y elegir el individuo con menor cuenta de nicho.

Horn y Nafpliotis definen la función de adecuación en el dominio de los objetivos y se sugiere utilizar para la distancia una métrica que combine tanto los dominios de los objetivos como los dominios de las variables de decisión, construyendo así lo que llaman una *función de participación anidada*.

La mayor ventaja de este método reside en que al no aplicar la selección Pareto a toda la población, sino sólo a una parte, resulta un algoritmo muy eficiente; sin embargo, además de establecer el factor de participación, este algoritmo requiere que se elija un valor adecuado para el tamaño del torneo, lo cual complica todavía más su uso en la práctica.

Erickson et al. [Eri01] proponen una versión mejorada del algoritmo original NPGA. En NPGA2 se utiliza un método de selección por torneo de igual forma que en NPGA, pero a diferencia de éste, se utiliza el grado de dominación de un individuo como su rango, es decir, el torneo se realiza de la siguiente forma:

1. Seleccionar un grupo de k competidores.
2. Para cada competidor i , calcular su rango r_i como el número de individuos de la población que le dominan.
3. Si hay un solo competidor i con el valor de rango r_i más bajo, entonces elegir i .
4. En otro caso, si tenemos varios competidores con el menor valor de rango, calcular la cuenta de nicho de cada uno de dichos competidores y elegir aquel con menor cuenta de nicho. El cálculo de la cuenta de nicho se realiza de igual forma que en el algoritmo NPGA original.

La ventaja de esta modificación del algoritmo es que ahora la selección se realiza de forma determinista, con lo cual se ha eliminado gran parte del ruido que se produce en el algoritmo NPGA original.

2.4.4 Métodos elitistas basados en el concepto Pareto

La introducción de elitismo dentro de los algoritmos evolutivos multiobjetivo da lugar a una nueva generación de algoritmos. Estos algoritmos evolutivos realizan una articulación de preferencias a posteriori y se basan en el concepto Pareto. En este apartado se discuten los algoritmos evolutivos multiobjetivo con elitismo de mayor relevancia.

Algoritmo Genético de Ordenación No Dominada Elitista (NSGA-II)

Deb et al. [Deb00b] proponen un algoritmo genético de ordenación no dominada con elitismo. NSGA-II utiliza una estrategia elitista junto con un mecanismo explícito de diversidad. En realidad, este algoritmo se diferencia en muchos aspectos del original NSGA, aunque los autores han decidido mantener el nombre NSGA-II para referenciar su origen.

En NSGA-II, a partir de la población P_t se crea una nueva población de descendientes Q_t . Estas dos poblaciones se mezclan para formar una nueva población R_t de tamaño $2N$ (siendo N el tamaño de la población original P_t). Los individuos de la población R_t se ordenan por frentes no dominados y se obtiene la nueva población P_{t+1} aplicando a R_t una selección por torneo junto con un mecanismo de nichos para mantener la diversidad.

El esquema de NSGA-II es el siguiente:

1. Crear una nueva población Q_t de N individuos a partir de la población P_t (de N individuos) aplicando los operadores evolutivos de cruce y mutación,
2. Combinar las poblaciones P_t y Q_t para crear una nueva población $R_t = P_t \cup Q_t$.
3. Ordenar los individuos de R_t por frentes no dominados y asignar a cada individuo un rango r_i según el frente en el que se ordena.
4. Crear una nueva población P_{t+1} seleccionando N individuos de la población R_t utilizando el operador de selección con nichos.

La *selección por torneo con nichos* compara dos soluciones y devuelve la ganadora del torneo asumiendo que cada solución i tiene dos atributos:

1. Un rango de no dominación r_i dentro de la población.
2. Una distancia de nicho local d_i , que es una medida del espacio de búsqueda alrededor de la solución i que no está ocupado por ninguna otra solución de la población.

Basándose en estos dos atributos, una solución i se prefiere a una solución j si ocurre alguna de las siguientes condiciones:

1. Si la solución i tiene mejor rango que j , es decir, si $r_i < r_j$.
2. Si las soluciones i y j tienen el mismo rango, pero la solución i tiene mejor distancia de nicho que la solución j , es decir, $r_i = r_j$ y $d_i > d_j$.

La primera condición asegura que la solución escogida se encuentra en un frente no dominado mejor. La segunda condición resuelve el conflicto en caso de que ambas soluciones se encuentren en el mismo frente, decidiéndose por aquella solución que tenga una mejor distancia de nicho, asegurando por tanto la diversidad. La distancia de nicho d_i puede calcularse de muchas formas. En NSGA-II, dicha métrica se calcula de la siguiente forma:

- Sea \mathcal{F}_i el conjunto de individuos de la población que pertenecen al mismo frente Pareto que el individuo i .
- Para cada función objetivo $j = 1, \dots, n$:
 - Ordenar los individuos de \mathcal{F}_i según el valor de la función objetivo f_j en una lista l_j^i .
 - f_j^{i+1} es el valor de la función j para el individuo en la lista l_j^i siguiente al individuo i .
 - f_j^{i-1} es el valor de la función j para el individuo en la lista l_j^i anterior al individuo i .

El valor de la distancia d_i se calcula de la forma:

$$d_i = \sum_{j=1}^n \frac{f_j^{i+1} - f_j^{i-1}}{f_j^{max} - f_j^{min}}$$

donde f_j^{max} y f_j^{min} son los valores máximos y mínimos para la función objetivo j -ésima en toda la población. Esta métrica denota el valor normalizado de la mitad del perímetro del hipercubo definido por las soluciones más cercanas situadas como vértices.

Deb y Goel [Deb01b] modifican el algoritmo NSGA-II para incluir una técnica de *elitismo controlado*; en esta técnica se restringe el número de individuos elitistas, haciendo que se mantenga la diversidad no sólo entre soluciones pertenecientes al mismo frente no dominado, sino también entre soluciones pertenecientes a frentes

distintos. En concreto, el número de individuos de cada frente, se restringe de forma adaptativa utilizando una distribución geométrica de la forma:

$$n_i = rn_{i-1}$$

donde n_i es el máximo número de individuos permitidos en el frente i -ésimo y $r < 1$ es la tasa de reducción. Es importante establecer un valor adecuado para el parámetro r para mantener un equilibrio correcto entre exploración y explotación del espacio de búsqueda. En general, el valor óptimo de r depende del problema y resulta difícil de establecer de forma teórica.

La técnica de elitismo controlado permite disminuir la presión de selección que produce el algoritmo NSGA-II debido a las soluciones elitistas, consiguiéndose una población más dispersa. Deb y Goel indican también que su técnica de elitismo controlado es genérica y puede implementarse con cualquier otro algoritmo multiobjetivo elitista.

Algoritmo Genético Pareto basado en la Distancia (DPGA)

Osyczka y Kundu [Osy95] proponen utilizar un *algoritmo genético Pareto basado en la distancia* (*Distance-based Pareto Genetic Algorithm, DPGA*). La idea de este método es utilizar el *teorema del contacto* [Lin76] para determinar las distancias relativas de una solución dada al conjunto Pareto; en este algoritmo se mantiene separada una población de individuos elitistas $E = (\mathbf{e}^1, \dots, \mathbf{e}^K)$ y la adecuación de un individuo en la población viene determinada de forma directa por el valor de la distancia de dicho individuo al conjunto de individuos elitista. La distancia de una solución \mathbf{x} al cada individuo \mathbf{e}^k perteneciente al conjunto E se calcula de la forma:

$$d^k(\mathbf{x}) = \sqrt{\sum_{i=1}^n \left(\frac{f_i(\mathbf{e}^k) - f_i(\mathbf{x})}{f_i(\mathbf{e}^k)} \right)^2}$$

Se halla la distancia mínima d^k para el individuo \mathbf{x} :

$$d^{min}(\mathbf{x}) = \min_{k=1}^K d^k(\mathbf{x})$$

siendo k^* el índice para el cual $d^{k^*}(\mathbf{x}) = d^{min}(\mathbf{x})$

Si \mathbf{x} es no dominado, su adecuación se calcula de la forma:

$$F(\mathbf{x}) = F(\mathbf{e}^{k^*}) + d^{min}(\mathbf{x})$$

y en ese caso, el conjunto E se actualiza añadiendo la nueva solución \mathbf{x} y eliminando aquellas soluciones \mathbf{e}^k dominadas por \mathbf{x} .

Si \mathbf{x} es dominado, su adecuación se calcula como:

$$F(\mathbf{x}) = \max\{0, F(\mathbf{e}^{k^*}) - d^{\min}(\mathbf{x})\}$$

Por último, se calcula la máxima adecuación entre todas las soluciones elitistas:

$$F_{\max} = \max_{k=1}^K F(\mathbf{e}^k)$$

y se asigna a todas las soluciones elitistas una adecuación igual a F_{\max} .

Este método es, en cierta forma, parecido a la aproximación Min-Max descrita previamente, con la diferencia de que en este caso no se requieren pesos para cada objetivo, ni hace falta una función de participación para mantener la diversidad en la población.

La principal ventaja de este método es su eficiencia y relativa simplicidad; además, no necesita una función de participación explícita. Sin embargo, requiere definir valores de penalización para las restricciones y otro valor llamado *distancia de inicio* que se utiliza para escalar las cantidades entre las diferentes soluciones. Si alguno de estos valores no se escoge de forma adecuada, el algoritmo salta entre las regiones factibles y no factibles, perdiendo consecuentemente porciones del frente Pareto.

Algoritmo Evolutivo con Fuerzas Pareto (SPEA)

Zitzler y Thiele [Zit98] proponen un algoritmo evolutivo elitista que llaman de fuerzas Pareto (*Strength Pareto Evolutionary Algorithm: SPEA*). Este algoritmo introduce elitismo manteniendo una población elitista separada $E = (\mathbf{e}^1, \dots, \mathbf{e}^K)$. Para crear la población E se realiza lo siguiente:

1. Obtener todas las soluciones no dominadas de P .
2. Si el tamaño de E es mayor que un tamaño máximo N_E , entonces eliminar de E aquellas soluciones con peor valor de dispersión mediante un procedimiento de clustering.

Además de mantener el elitismo, la población E se utiliza para calcular la adecuación, de forma que la población evolucione hacia el frente Pareto verdadero más eficientemente. A cada solución de la población elitista \mathbf{e} se le asigna un valor de adecuación llamado *fuerza* (*Strength*) que se calcula como:

$$F(\mathbf{e}) = S(\mathbf{e}) = \frac{n(\mathbf{e})}{N + 1}$$

siendo $n(\mathbf{e})$ el número de soluciones de P dominadas por \mathbf{e} .

La adecuación de un individuo \mathbf{x} perteneciente a la población se calcula como:

$$F(\mathbf{x}) = 1 + \sum_{\mathbf{e} \in E(\mathbf{x})} S(\mathbf{e})$$

donde $E(\mathbf{x})$ es el conjunto de soluciones $\mathbf{e} \in E$ que dominan a \mathbf{x} .

La selección se realiza mediante torneo aplicado a la unión de las poblaciones $P \cup E$ para crear una nueva población P .

Una ventaja de este algoritmo es que resulta fácil calcular la adecuación, que se hace de forma similar a como se hace en MOGA. Además, el clustering asegura encontrar las soluciones con una mayor diversidad; sin embargo, SPEA introduce un nuevo parámetro (N_E) que hay que establecer de forma adecuada para que el algoritmo funcione correctamente; si N_E es demasiado grande, se pierde la diversidad, pero si es demasiado pequeño se pierde el efecto de elitismo.

Zitzler et al. [Zit01b] realizan una mejora de SPEA a la que denominan *SPEA2*; las principales diferencias entre SPEA2 y su antecesor son:

- Se utiliza un esquema de asignación de adecuación mejorado, donde se tiene en cuenta para cada individuo cuantos individuos domina y por cuantos individuos es dominado.
- Se incorpora una técnica de estimación de densidad según el vecino más próximo, que permite una guía más precisa en el proceso de búsqueda.
- Un nuevo método de truncamiento del archivo permite mantener las soluciones que se encuentran en los límites.

Los autores afirman que SPEA2 ha mostrado resultados de rendimiento, convergencia y diversidad mejores que el algoritmo original SPEA.

Algoritmo Genético Termodinámico (TDGA)

Kita et al. [Kit96] proponen una función de adecuación basada en el concepto termodinámico de mínima energía que llaman *algoritmo genético termodinámico* (*Thermodynamic Genetic Algorithm: TDGA*); en este método, la adecuación de un individuo se calcula como el valor de la energía:

$$F(\mathbf{x}, t) = E(\mathbf{x}) - H(\mathbf{x})T(t)$$

donde $E(\mathbf{x})$ es el valor medio de energía de la solución \mathbf{x} , que se calcula como el rango de no dominación. El término $H(\mathbf{x})$ define la entropía de \mathbf{x} y es una medida de la diversidad. Por último, $T(t)$ es el factor de enfriamiento en la generación t .

Estrategia de Evolución con Archivo Pareto (PAES)

Knowles y Corne [Kno00] proponen utilizar una estrategia de evolución con archivo Pareto (*Pareto-Archive Evolution Strategy: PAES*). El esquema PAES utiliza una estrategia de evolución (1 + 1)-ES, es decir, se utiliza el operador de mutación (no existe el cruce) sobre un solo padre para producir un hijo. Para realizar la comparación entre dos individuos, PAES utiliza el concepto Pareto y mantiene una población de individuos no dominados (el archivo). El esquema es el siguiente:

1. Seleccionar un individuo aleatorio de la población: \mathbf{p} .
2. Aplicar la mutación a \mathbf{p} para obtener un hijo: \mathbf{c} .
3. Si \mathbf{p} domina a \mathbf{c} , entonces \mathbf{c} es rechazado.
4. Si \mathbf{c} domina a \mathbf{p} , entonces \mathbf{c} sustituye a \mathbf{p} .
5. Si \mathbf{p} y \mathbf{c} son no dominados respectivamente, entonces:
 - (a) Si existe algún otro individuo \mathbf{x} dentro de la población que domina a \mathbf{c} , entonces rechaza a \mathbf{c} .
 - (b) Si \mathbf{c} domina a algún individuo \mathbf{x} dentro de la población, entonces \mathbf{x} es sustituido por \mathbf{c} .
 - (c) En otro caso, \mathbf{c} sustituye a su padre \mathbf{p} si su valor de diversidad es menor.

Para calcular el valor de diversidad se divide cada objetivo en 2^d divisiones iguales, donde d es un parámetro definido por el usuario; el espacio objetivo queda por tanto dividido en $(2^d)^n$ hipercubos n -dimensionales de igual tamaño. El valor de diversidad para una solución dada se calcula como el número de individuos que se encuentran ubicados en el mismo hipercubo de dicha solución.

Con objeto de dar una perspectiva global al esquema PAES, Knowles y Corne introducen el concepto de *estrategia de evolución con múltiples miembros*, es decir, un algoritmo $(1 + \lambda)$ -PAES, donde un padre es mutado λ veces, creando cada vez un hijo. La mejor solución entre el padre y los λ hijos será la que pase a la siguiente generación.

Por último, también se propone un $(\mu + \lambda)$ -PAES, donde μ padres generan λ hijos; en este caso, tanto los padres como los hijos compiten para pasar a la siguiente generación.

Algoritmo Genético con Selección basada en Particiones Pareto (PESA)

Corne et al. [Corn00] proponen un algoritmo genético con estrategia de selección basada en Particiones Pareto (*Pareto Envelope-based Selection Algorithm: PESA*). Este algoritmo tiene muchas similitudes con PAES; calcula el valor de diversidad de un individuo dividiendo el espacio de búsqueda en hipercubos de igual forma que PAES; sin embargo, la diferencia fundamental es que PESA es una técnica basada en población, mientras que PAES es un método de búsqueda local; esto hace que sean algoritmos fundamentalmente distintos. PESA mantiene una población P de individuos y otra población externa (elitista) E .

El esquema básico de PESA es el siguiente:

1. Generar y evaluar la población inicial P e inicializar E al conjunto vacío.
2. Incorporar las soluciones no dominadas de P a E .
3. Si se alcanza el criterio de parada, devolver E como resultado; en otro caso, generar una nueva P repitiendo lo siguiente:
 - Seleccionar dos padres de E , generar un hijo mediante cruce y mutación y otro hijo mediante sólo mutación de uno de los dos padres.
4. Volver al paso 2.

Posteriormente, Corne et al. [Corn01] realizan una nueva versión de PESA a la que denominan *PESA-II* que utiliza *selección basada en la región*; en este tipo de selección, no se calcula la adecuación de un individuo aislado, sino que se calcula la adecuación de una cierta región (o hipercubo); una vez seleccionado un hipercubo, el individuo concreto dentro de dicho hipercubo se elige de forma aleatoria. Este tipo de selección favorece la diversidad, ya que aumenta la probabilidad de seleccionar individuos que se encuentran más aislados.

Algoritmo Genético Desordenado Multiobjetivo (MOMGA)

Veldhuizen [Vel99] propone una aproximación completamente distinta para optimización multiobjetivo. Modifica el algoritmo genético desordenado para un solo objetivo de Goldberg et al. [Gol89b] para encontrar múltiples soluciones Pareto en optimización multiobjetivo.

El algoritmo genético desordenado para un único objetivo (*messy Genetic Algorithm*, *mGA*) se basa en resolver de forma secuencial las dos tareas de identificar los bloques constructivos de un problema (soluciones parciales correspondientes a la solución óptima) y combinar dichos bloques. Cada individuo tiene cromosomas con información de los genes (posiciones) y de los alelos (valor del bloque constructivo en dicha posición). En una primera fase, se identifican los alelos y en una segunda fase se combinan entre ellos para formar la solución completa. En ambas fases se utiliza una técnica de selección por torneo, añadiendo en la primera fase una técnica de nicho para obtener una buena diversidad entre los bloques constructivos.

El algoritmo desordenado multiobjetivo (*Multi-Objective Messy Genetic Algorithm*, *MOMGA*) trata con múltiples objetivos, por tanto la evaluación y comparación de los cromosomas resulta más complicada. Cada cromosoma tiene ahora n valores diferentes para las funciones objetivo. Se realiza una comparación en base a la dominación Pareto. Las dos fases se realizan de forma similar a mGA, aunque para mantener la diversidad se utiliza la técnica del torneo con nichos utilizada en NPGA. Por otra parte, en la segunda fase se mantiene una población externa de soluciones no dominadas que asegura el elitismo.

Veldhuizen también describe una versión concurrente de MOMGA (*CMOMGA*) proponiendo aplicaciones paralelas de MOMGA utilizando diferentes individuos iniciales aleatorios y realizando intercambios ocasionales entre individuos pertenecientes

a MOMGAs diferentes para mejorar la diversidad. Cuando todos los MOMGAs terminan, se combinan todas las poblaciones externas de soluciones no dominadas en una sola que contiene los individuos no dominados globales.

Zydallis et al. [Zyd01] realizan una ampliación que llaman MOMGA-II en la cual se realiza una inicialización de la población de forma que todos bloques constructivos se encuentren en la población inicial. Se realiza además un filtrado de los bloques de forma que se reduce el número es éstos manteniéndose los mejores bloques encontrados hasta el momento.

Micro-Algoritmo Genético Multiobjetivo ($M\mu GA$)

Krishnakumar [Kri89] propone un micro-Algoritmo Genético con una población pequeña (cuatro o cinco individuos) para optimización de un único objetivo. Coello y Toscano [Coe00b] amplían esta idea para optimización multiobjetivo, proponiendo un *micro-Algoritmo Genético multiobjetivo* (*multi-objective micro-Genetic Algorithm*, $M\mu GA$) con dos poblaciones. La población del algoritmo genético (con cuatro individuos) evoluciona de forma similar a la del micro-Algoritmo Genético para un único objetivo, mientras que la segunda población mantiene un registro con los individuos no dominados. Esta población se actualiza con las nuevas soluciones de forma similar a como se hace en el archivo de PAES.

Algoritmo Evolutivo Multiobjetivo Elitista con Participación Co-evolutiva (ERMOCS)

Neeff et al. [Nee99] proponen utilizar un *algoritmo evolutivo multiobjetivo con recombinación elitista y participación co-evolutiva* (*Elitist Recombinative Multi-Objective with Co-evolutionary Sharing*, *ERMOCS*). Para mantener la diversidad, este algoritmo se basa en el concepto de participación co-evolutiva de Goldberg y Wang [Gol98] (*Coevolutionary Shared Niching*, *CSN*). El elitismo se mantiene utilizando un esquema de preselección, de forma que un hijo reemplaza a su padre sólo si es mejor. En el modelo de co-evolución existen dos poblaciones, la población de clientes y la población de vendedores, que interaccionan entre ellas de la forma descrita en el método CSN, aunque en este caso se utiliza un operador adicional para enfatizar las soluciones no dominadas. Cada vendedor es comparado con un conjunto aleatorio de clientes; si algún cliente domina al vendedor y éste se encuentra al menos a una distancia mínima

d_{min} de otro vendedor, entonces el cliente reemplaza a dicho vendedor.

2.4.5 Conclusiones

En esta sección se han revisado las técnicas evolutivas fundamentales utilizadas para resolver problemas multiobjetivo. Estas técnicas se clasifican en dos tipos:

- Técnicas con articulación de preferencias *a priori*

En este tipo de técnicas se definen las preferencias antes de ejecutar el algoritmo, en muchos casos reduciendo el problema multiobjetivo a un problema con un único objetivo. Estas técnicas han sido muy utilizadas tradicionalmente, pero su desventaja es que obtienen una única solución; si se desean más soluciones, es necesario realizar múltiples ejecuciones del algoritmo.

- Técnicas con articulación de preferencias *a posteriori*

En estas técnicas, los algoritmos evolutivos encuentran su máxima realización, pues permiten encontrar en una sola ejecución múltiples soluciones del problema, decidiendo después qué solución es la que nos interesa según nuestras preferencias. Podemos decir que estos algoritmos evolutivos son los propiamente multiobjetivo. Dentro de estas técnicas podemos encontrar diversos métodos:

- Métodos no basados en el concepto Pareto

Estos métodos no utilizan el concepto Pareto. Existen diversos algoritmos que encuentran múltiples soluciones sin utilizar el concepto Pareto, en algunos casos con buenos resultados, sin embargo, en muchos casos, estos algoritmos no son capaces de obtener soluciones con suficiente diversidad. Podemos citar el algoritmo VEGA como representante de estos métodos y algoritmo pionero en el campo de la optimización multiobjetivo.

- Métodos no elitistas basados en el concepto Pareto

La utilización del concepto Pareto para comparar soluciones ha dado lugar a la primera generación de algoritmos evolutivos multiobjetivo. Los algoritmos MOGA, NSGA y NPGA son representantes de estos métodos.

- Métodos elitistas basados en el concepto Pareto

La introducción de elitismo dentro de los algoritmos evolutivos multiobjetivo, ha supuesto la creación de una segunda generación de algoritmos. Este es el campo de investigación actual sobre algoritmos evolutivos multiobjetivo y donde mejores resultados se están obteniendo. NSGA-II, DPGA, PESA, PAES o MOMGA son algunos de los algoritmos representativos de esta generación.

En nuestro trabajo vamos a realizar una selección de preferencias a posteriori, es decir, estamos interesados en obtener un conjunto de soluciones y realizar la decisión después de la ejecución del algoritmo. Dentro de este tipo de algoritmos, vamos a utilizar el concepto Pareto y también elitismo, encuadrando, por tanto, nuestro trabajo dentro del campo de los algoritmos evolutivos multiobjetivo de última generación.

2.5 Técnicas Evolutivas para Optimización Multiobjetivo con Restricciones

Los problemas de optimización multiobjetivo con restricciones se describen tal como se muestra en la siguiente ecuación:

$$\begin{aligned} & \text{Minimizar} && f_i(\mathbf{x}), && i = 1, \dots, n \\ & \text{sujeto a :} && g_j(\mathbf{x}) \leq 0, && j = 1, \dots, m \end{aligned}$$

donde $\mathbf{x} = (x_1, \dots, x_p)$ es un vector de parámetros reales $x_k \in \mathfrak{R}$ pertenecientes a un dominio $[l_k, u_k]$, $k = 1, \dots, p$, y $f_i(\mathbf{x})$, $g_j(\mathbf{x})$, son funciones arbitrarias lineales o no lineales.

El trabajo desarrollado en esta memoria pretende resolver problemas de optimización multiobjetivo con restricciones, por ello, en esta sección se discuten diferentes técnicas evolutivas para resolver este tipo de problemas.

2.5.1 Ignorar las soluciones no factibles

Una forma simple de tratar las restricciones consiste en ignorar cualquier solución que no satisfaga todas las restricciones [Coe99]. Esta aproximación resulta muy sencilla de implementar, pero en muchos problemas encontrar una única solución factible puede ser muy difícil. En tales casos, esta aproximación no llegará ni siquiera a encontrar una sola solución factible. Por este motivo, esta aproximación puede utilizarse solamente cuando las restricciones no son demasiado fuertes.

2.5.2 Funciones de penalización

Esta es una estrategia muy extendida que ya estudiamos en la sección 2.3, pero extendida al caso multiobjetivo. Consiste en minimizar la suma ponderada del valor normalizado de las restricciones. La violación de cada restricción se calcula como:

$$w_j(\mathbf{x}) = \begin{cases} |\bar{g}_j(\mathbf{x})|, & \text{si } \bar{g}_j(\mathbf{x}) < 0 \\ 0, & \text{en otro caso} \end{cases}$$

donde $\bar{g}_j(\mathbf{x})$ es el valor normalizado de la restricción j -ésima:

$$\bar{g}_j(\mathbf{x}) = \frac{g_j(\mathbf{x})}{\sum_{i=1}^m g_i(\mathbf{x})}$$

Todas las violaciones se suman para obtener la violación global:

$$\Omega(\mathbf{x}) = \sum_{j=1}^m w_j(\mathbf{x})$$

El valor de la función objetivo n -ésima queda modificado de la forma:

$$F_n(\mathbf{x}) = f_n(\mathbf{x}) + R_n \Omega(\mathbf{x})$$

siendo R_n un *parámetro de penalización* para el objetivo n -ésimo.

Uno de los problemas de este método es elegir el valor de los parámetros de penalización. Se pueden utilizar las estrategias de actualización dinámicas desarrolladas para algoritmos con un solo objetivo [Mic92, Mic96b]. Sin embargo, la mayoría de estudios en optimización multiobjetivo utilizan valores estáticos de R_n escogidos cuidadosamente [Sri94, Deb99]

2.5.3 Separar las soluciones factibles de las no factibles

Los métodos de este tipo se basan en tratar de forma diferenciada las soluciones factibles y las no factibles, en algunos casos utilizando frentes Pareto diferentes, en otros casos definiendo la función de adecuación de forma diferenciada, o realizando la selección de soluciones de uno u otro tipo.

Selección por Ranking y Crowding

Jiménez y Verdegay [Jim98] proponen una técnica evolutiva para problemas de optimización con restricciones. Utilizan un algoritmo evolutivo basado en dominación Pareto con el siguiente procedimiento de ordenación:

1. Los individuos factibles se ordenan en K frentes Pareto ($K \geq 0$).
2. Si existen individuos no factibles, estos forman un frente adicional y se ordenan de acuerdo a la función:

$$U(\mathbf{x}) = \max_{j=1, \dots, m} \{g_j(\mathbf{x})\} \quad (2.6)$$

Las probabilidades de selección se asignan de acuerdo al rango de cada individuo dentro de la población. Para mantener diversidad en la población se utiliza una selección por ranking no lineal [Mic92] y formación implícita de nichos mediante el modelo de factor de crowding [Gol89].

Torneo estocástico y Niching

Jiménez, Verdegay y Gómez-Skarmeta [Jim99b] proponen un procedimiento sistemático para el manejo de restricciones en optimización multiobjetivo. En este método se realiza un estudio diferenciado entre soluciones factibles y no factibles y se utilizan nichos para mantener la diversidad entre las soluciones Pareto óptimas. La selección se realiza mediante un torneo entre dos individuos. Pueden darse tres casos distintos:

1. Ambas soluciones son factibles.

En este caso no se tienen en cuenta las restricciones y se sigue un procedimiento de selección similar a NPGA, aunque el cálculo de la cuenta de nicho se realiza como en NSGA.

2. Una solución es factible y la otra no.

Se elige la solución factible.

3. Ambas soluciones son no factibles.

De igual forma que en el primer caso, se realiza una selección mediante un torneo entre las dos soluciones eligiendo un conjunto aleatorio de n soluciones no factibles. Como criterio de comparación se utiliza la función $U(\mathbf{x})$ descrita en la ecuación (2.6).

Torneo binario con Restricciones

Deb [Deb01c] implementa un método de penalizaciones sin parámetros para satisfacción de restricciones en problemas de un sólo objetivo. Este método puede extenderse también a optimización multiobjetivo. El método utiliza torneo binario; se toman dos soluciones aleatorias de la población y se elige aquella solución que es “mejor”. En este contexto multiobjetivo con restricciones se dice que una solución \mathbf{x} es “mejor” que otra solución \mathbf{y} , si \mathbf{x} *domina con restricciones* a \mathbf{y} , es decir, si:

1. \mathbf{x} es factible e \mathbf{y} no lo es.
2. \mathbf{x} e \mathbf{y} son ambas no factibles, y \mathbf{x} tiene una menor violación de las restricciones.
3. \mathbf{x} e \mathbf{y} son ambas factibles y \mathbf{x} domina a \mathbf{y} .

Deb incluye esta definición en su algoritmo NSGA-II para manejo de restricciones. Esta definición de dominación con restricciones es similar a la sugerida por otros autores [Jim99b, Fon98, Dre98, Bin97], la diferencia está en la forma de tratar las soluciones no factibles. En el método propuesto por Deb, dos soluciones no factibles se comparan según el valor de restricción global, mientras que Jiménez y Verdegay [Jim99b] utilizan el valor de la función $U(\mathbf{x})$ según la ecuación (2.6); Fonseca y Fleming [Fon98] clasifican soluciones que violan diferentes restricciones en un mismo frente de dominación con restricciones. Dreschser [Dre98], en cambio, propone un orden más detallado. Binh y Korn [Bin97] sugieren la siguiente medida de la violación:

$$C(\mathbf{x}) = \left(\sum_{j=1}^m [c_j(\mathbf{x})]^p \right)^{1/p}$$

donde $p > 0$ y $c_j(\mathbf{x}) = |\min(0, g_j(\mathbf{x}))|$, $j = 1, \dots, m$.

Método de Ray-Tai-Seow's

Ray et al. [Ray01] sugieren una técnica de manejo de restricciones más detallada; las violaciones no se suman, sino que se realiza una comprobación de no dominación de las restricciones. Se establecen tres rangos de no dominación diferentes dentro de la población:

1. El primer rango se establece utilizando los valores de las n funciones objetivo y el resultado se guarda en un vector de n dimensiones R_{obj} .
2. Para el segundo rango se utilizan los valores de violación de las m restricciones, sin utilizar información de las funciones objetivo y se crea un vector de m dimensiones R_{con} .
3. El tercer rango se crea utilizando la combinación de valores objetivo y restricciones, un total de $n + m$ valores. Esto produce el vector R_{com} de dimensión $n + m$.

Una vez se establecen estos rangos se utiliza el siguiente algoritmo para manejar las restricciones:

1. Utilizando R_{com} , seleccionar todas las soluciones factibles de rango 1 para crear la población P' . Si P' tiene menos de N elementos, entonces seleccionar el resto de elementos (hasta completar los N) utilizando el segundo paso.

2. Elegir una solución A utilizando R_{obj} para dar preferencia. Para seleccionar su pareja, elegir dos soluciones B y C utilizando R_{con} y entre estas dos soluciones elegir la ganadora de la siguiente forma:

- (a) Si B y C son ambas factibles, elegir la que tenga mejor rango en R_{obj} . Si ambas tienen el mismo rango, elegir aquella con mejor dispersión (de acuerdo a alguna medida).
- (b) Si B y C son ambas no factibles, elegir la de mejor en R_{con} . Si ambos rangos son iguales, utilizar una métrica de satisfacción e restricciones que mide el número de restricciones que se B o C satisfacen junto con A y elegir aquella solución que tenga el menor número de restricciones satisfechas comunes con A .
- (c) Si B es factible y C no, entonces elegir B ; en el caso contrario, elegir C .

Utilizar las soluciones A y la solución ganadora entre B y C como padres para crear descendencia y copiar tanto los padres como sus descendientes en la población P' .

2.5.4 Definir el problema utilizando metas y prioridades

Las técnicas de este tipo definen el problema de forma que las restricciones se tratan como objetivos de mayor prioridad y la meta a conseguir es la satisfacción de restricciones. Una vez definido el problema de esta forma, es necesario utilizar un algoritmo que resuelva problemas multiobjetivo con prioridades y metas, o bien transformar de alguna forma el problema.

Ampliación de MOGA

Fonseca y Fleming [Fon98] proponen una ampliación de su algoritmo MOGA para tratar restricciones en problemas multiobjetivo. La ampliación que realizan de MOGA consiste en definir un sistema de preferencia entre soluciones, basándose en metas y prioridades. Así, las restricciones son tratadas como objetivos de alta prioridad hasta que se alcanza su meta, es decir, se cumple la restricción. Los objetivos tienen menor prioridad y no tienen meta.

Dado un problema con n objetivos y m restricciones, se establece un *vector de preferencias* $(g_{11}, \dots, g_{1n}), (g_{21}, \dots, g_{2m})$. A los objetivos se les asigna menor prioridad y a las restricciones mayor prioridad. Fonseca y Fleming proponen asignar el vector $(-\infty, \dots, -\infty), (g_{21}, \dots, g_{2m})$

La preferencia entre dos soluciones \mathbf{x} y \mathbf{x}' se define de la siguiente forma:

1. Si \mathbf{x} alcanza todas las metas en los términos de mayor prioridad (satisface todas las restricciones) y \mathbf{x}' no las alcanza, se prefiere la solución \mathbf{x} .
2. Si ambas soluciones alcanzan todas las metas en los términos de mayor prioridad (satisfacen todas las restricciones), se comparan utilizando el concepto Pareto aplicado a los términos de menor prioridad (los objetivos).
3. Si en ninguna solución se alcanzan todas las metas en los términos de mayor prioridad (ninguna solución satisface las restricciones), se comparan utilizando el concepto Pareto aplicado a los términos de menor prioridad que no alcancen la meta (es decir, las restricciones no satisfechas).

Extensiones de Algoritmos Evolutivos Multiobjetivo

Jiménez et al. [Jim02] utilizan un algoritmo multiobjetivo basado en el concepto Pareto con un esquema de preselección como técnica implícita de diversidad y elitismo para resolver problemas de optimización con metas y prioridades. Las metas indican los niveles que se desean alcanzar en cada objetivo y las prioridades son valores enteros que determinan en qué orden deben ser optimizados los objetivos. Se considera la siguiente formulación:

$$\text{Minimizar } (f_1(\mathbf{x}), \dots, f_p(\mathbf{x})) \quad (2.7)$$

con metas $\mathbf{u} = (u_1, \dots, u_p)$ y prioridades $\mathbf{p} = (p_1, \dots, p_p)$.

Utilizando metas y prioridades se pueden resolver una gran variedad de problemas de optimización:

- *Optimización de un vector.* Todos los objetivos tienen igual prioridad y no hay metas.
- *Optimización con restricciones.* Las restricciones tienen mayor prioridad y metas; los objetivos tienen menor prioridad y no tienen metas.

- *Satisfacción de restricciones.* Todas las restricciones tienen igual prioridad y metas; no hay objetivos.
- *Programación por metas.* Los objetivos tienen asignadas metas que pueden ser alcanzadas de forma simultánea o secuencial.

Para resolver problemas de la forma (2.7) se considera el siguiente vector objetivo:

$$(f'_1(\mathbf{x}), \dots, f'_p(\mathbf{x}))$$

donde las funciones $f'_i(\mathbf{x})$ se definen de la forma:

$$f'_i(\mathbf{x}) = \begin{cases} m, & \text{si } h(i) \text{ y } f_i(\mathbf{x}) \leq u_i \\ f_i(\mathbf{x}), & \text{si } h(i) \\ M, & \text{en otro caso} \end{cases}$$

donde m es un número suficientemente pequeño ($m \leq u_i, \forall i = 1, \dots, p$), M es un número suficientemente grande y $h : \{1, \dots, p\} \rightarrow \{0, 1\}$ es una función booleana definida de la siguiente forma:

$$h(i) = \begin{cases} 1, & \text{si } f_j(\mathbf{x}) \leq u_j, \forall j : p_j > p_i \\ 0, & \text{en otro caso} \end{cases}$$

Cada una de las nuevas funciones objetivo f'_i toma el mejor valor posible m cuando todos los objetivos con menor prioridad y él mismo son alcanzados. En otros casos, si todos los objetivos con menor prioridad son alcanzados, entonces la nueva función f'_i toma el valor original f_i y, por último, si algún objetivo con menor prioridad no es alcanzado, entonces f'_i toma el peor de los posibles valores, M .

Un problema de optimización con n objetivos y m restricciones puede transformarse según el esquema anterior de la siguiente forma:

$$(f'_1(\mathbf{x}), \dots, f'_n(\mathbf{x}), g'_1(\mathbf{x}), \dots, g'_m(\mathbf{x}))$$

siendo:

$$f'_i(\mathbf{x}) = \begin{cases} f_i(\mathbf{x}), & \text{si } g_j(\mathbf{x}) \leq 0 \forall j = 1, \dots, m \\ M, & \text{en otro caso} \end{cases}$$

$$g'_j(\mathbf{x}) = \begin{cases} m, & \text{si } g_j(\mathbf{x}) \leq 0 \\ g_j(\mathbf{x}), & \text{en otro caso} \end{cases}$$

donde M debe ser mayor que $f(\mathbf{x})$ para todo \mathbf{x} y $m \leq 0$.

2.5.5 Conclusiones

En esta sección se han recogido algunas de las técnicas más importantes para manejo de restricciones en problemas de optimización multiobjetivo. Podemos distinguir los siguientes métodos:

- Ignorar las soluciones no factibles.

Es el método más sencillo, pero con restricciones fuertes puede no llegar a encontrar una sola solución factible. Coello [Coe99] propone utilizar este método para problemas de optimización multiobjetivo.

- Funciones de penalización.

Consiste en establecer funciones de penalización para las restricciones. El método es similar al utilizado en optimización simple descrito en la sección 2.3, pero extendida al caso multiobjetivo. El problema de este método es establecer los parámetros de forma adecuada.

- Separar las soluciones factibles de las no factibles.

Las soluciones factibles y no factibles pueden separarse para formar diferentes frentes o rangos, entre los algoritmos que utilizan métodos de este tipo tenemos el de Jiménez y Verdegay [Jim98] y Ray et al. [Ray01].

Otra opción es definir la comparación entre soluciones de forma diferenciada según sean factibles o no factibles. Jiménez, Verdegay y Gómez-Skarmeta [Jim99b] utilizan esta técnica junto con torneo estocástico y formación de nichos y Deb [Deb01c] utiliza esta técnica para incluir el manejo de restricciones en el algoritmo NSGA-II.

- Definir el problema utilizando metas y prioridades.

Puede definirse el problema de forma que las restricciones sean objetivos de mayor prioridad con metas; Fonseca y Fleming [Fon98] proponen una ampliación de MOGA con manejo de restricciones utilizando este método.

Jiménez et al. [Jim02] proponen un método para transformar un problema multiobjetivo con restricciones en un problema multiobjetivo sin restricciones utilizando metas y prioridades.

2.6 Sumario

Este capítulo comienza realizando un breve estudio de algunas de las técnicas clásicas utilizadas en optimización multiobjetivo. Dichas técnicas resultan muy adecuadas para problemas de optimización multiobjetivo lineales, pero cuando tratan funciones no lineales estas técnicas no pueden encontrar todas las soluciones deseadas. Las técnicas evolutivas aparecen entonces como una opción adecuada para tratar este tipo de problemas, puesto que son capaces de obtener en una sola ejecución múltiples soluciones del problema.

La versatilidad de diseño de los algoritmos evolutivos nos permite seleccionar aquellas técnicas que resultan más adecuadas para resolver problemas de optimización multiobjetivo con restricciones. Prestaremos una atención especial a los métodos que mantienen la diversidad en las soluciones; estos métodos, además de evitar la convergencia prematura y permitir una exploración completa del espacio de soluciones, en optimización multiobjetivo permiten mantener la diversidad entre las soluciones no dominadas, de forma que se obtengan soluciones dispersas por todo el frente Pareto. El uso de poblaciones tiene una especial importancia también en optimización multiobjetivo, pues permite obtener, en una sola ejecución del algoritmo, múltiples soluciones no dominadas en la población final. El elitismo también adquiere una especial importancia en optimización multiobjetivo. Por otra parte, también deseamos manejar restricciones, por lo que se utilizan técnicas de manejo de restricciones. Nuestro trabajo realiza, por tanto, el diseño y evaluación de algoritmos evolutivos multiobjetivo basados en el concepto de óptimo Pareto con elitismo, es decir, algoritmos evolutivos multiobjetivo de última generación.

Las técnicas anteriores van a ser utilizadas de diversas formas para resolver problemas de optimización multiobjetivo generales y problemas de modelización difusa. En ambos casos se utiliza el concepto de óptimo Pareto y un esquema de sustitución generacional como sistema de diversidad implícito. Para modelización difusa se añade un esquema de formación de nichos con el fin de obtener mayor diversidad. Por último, el diseño y estudio de problemas test para optimización multiobjetivo es un tema de gran interés en la investigación actual, por lo que en el siguiente capítulo se estudian algunos de los problemas test más destacados y se utilizan para comprobar el funcionamiento de los algoritmos propuestos.

Capítulo 3

Nuevos Algoritmos Evolutivos para Optimización Multiobjetivo

En este capítulo se experimenta con diversos aspectos de computación evolutiva para problemas de optimización multiobjetivo y se proponen nuevas técnicas evolutivas dentro del campo multiobjetivo.

El capítulo comienza proponiendo en la sección 3.1 un algoritmo evolutivo para resolver problemas de optimización multiobjetivo sin restricciones tal como se plantea en la ecuación (2.2), que son el tipo de problema más sencillo dentro del campo de optimización multiobjetivo. Para este algoritmo, se propone una variante del esquema de preselección como sistema sencillo y eficiente para mantener la diversidad, puesto que es una técnica de formación de nichos implícita y, por otra parte, mantiene el elitismo. Se utiliza este algoritmo también para resolver problemas de optimización multiobjetivo con restricciones utilizando la transformación descrita por Jiménez et al. [Jim02]. Este algoritmo es, por tanto, una primera aproximación válida para problemas de optimización multiobjetivo sencillos y sirve como base para plantear posteriormente nuevos algoritmos más complejos.

En la sección 3.2 se proponen tres nuevos algoritmos evolutivos (Algoritmo I, Algoritmo II y ENORA) para resolver problemas de optimización multiobjetivo con restricciones de la forma planteada en la ecuación (2.1). Para desarrollar estos algoritmos se toma como base el algoritmo desarrollado en la sección 3.1; sin embargo, estos nuevos algoritmos se diferencian del anterior en dos puntos fundamentales: (i) utilizan técnicas específicas para manejar las restricciones, e (ii) implementan dife-

rentes esquemas para mejorar la diversidad. La primera característica permite que el algoritmo pueda resolver problemas multiobjetivo con restricciones sin necesidad de realizar una transformación de los mismos. El segundo punto permite mantener la diversidad en aquellos problemas donde esto resulta más difícil.

El Algoritmo I utiliza el esquema de preselección simple propuesto para el algoritmo de la sección 3.1 y añade una técnica de manejo de restricciones. Este algoritmo funciona de forma adecuada en muchos problemas multiobjetivo; sin embargo, en otros problemas no consigue mantener la suficiente diversidad en las soluciones, por lo que en estos casos es necesario añadir una técnica explícita para mantener la diversidad. Para solucionar este problema, se proponen dos nuevos algoritmos que utilizan distintas técnicas de diversidad explícita. El Algoritmo II utiliza una métrica de diversidad como un objetivo adicional. Otra aproximación para mantener la diversidad es la que se propone en el último algoritmo, ENORA, que realiza una partición del espacio de búsqueda por tramos radiales, forzando a que las nuevas soluciones obtenidas mantengan la diversidad. Por otra parte, todas estas técnicas, además de mejorar la diversidad en problemas donde esto resulta difícil, mantienen el elitismo, que es otro punto importante dentro de optimización multiobjetivo.

Los algoritmos que se plantean en este capítulo forman, por tanto, un banco de pruebas donde se estudian aspectos de sustitución generacional (esquema de preselección) y aspectos de diversidad (uso de métricas de diversidad y formación explícita de nichos).

3.1 Algoritmo Evolutivo para Problemas de Optimización Multiobjetivo sin Restricciones

En esta sección se describe un nuevo algoritmo evolutivo multiobjetivo basado en dominación Pareto para resolver problemas de optimización multiobjetivo sin restricciones de la forma planteada en la ecuación (2.2) y se utiliza este algoritmo para resolver problemas de optimización basados en metas y prioridades (2.7) utilizando el método de transformación propuesto por Jiménez et al. [Jim02].

3.1.1 Estructura del algoritmo evolutivo

La aproximación propuesta es un *algoritmo evolutivo multiobjetivo basado en dominación Pareto* que resuelve problemas de optimización multiobjetivo sin restricciones y está diseñado para encontrar, en una sola ejecución, múltiples soluciones no dominadas de acuerdo con la estrategia de decisión Pareto, puesto que minimiza todas las funciones objetivo simultáneamente.

El algoritmo presenta una estructura general clásica. Se utiliza una representación en punto flotante y la población inicial se obtiene de forma aleatoria y uniforme entre los dominios de las variables. Posteriormente se entra en un proceso iterativo donde en cada iteración se genera una nueva población mediante selección, muestreo, variación y sustitución generacional.

3.1.2 Representación de las soluciones

Representación codificada mediante números reales.

La codificación de una solución $\mathbf{x} = (x_1, \dots, x_p)$ se realiza mediante un individuo I formado por p números reales $I = \{x_1, \dots, x_p\}$.

3.1.3 Inicialización de los individuos

La población inicial de N individuos se genera aleatoriamente con una distribución uniforme dentro de los límites del espacio de búsqueda.

Cruce	$pCross$	Cruce uniforme	$pUniformCross$
		Cruce aritmético	—
Mutación	$pMutate$	Mutación uniforme	$pUniformMutate$
		Mutación aritmética	$pNotUniformMutate$
		Mutación mínima	-

Tabla 3.1: Operadores de variación y sus parámetros de probabilidad asociados.

Un individuo $I = \{x_1, \dots, x_p\}$ se genera inicialmente de la siguiente forma:

$$x_i = r(u_i + l_i) + l_i, \quad i = 1, \dots, p$$

siendo r un valor real aleatorio entre 0 y 1, y

$[l_i, u_i]$ el dominio de la variable i -ésima.

3.1.4 Operadores de variación

Teniendo en cuenta que el algoritmo evolutivo utiliza una representación en punto flotante, los operadores de variación actúan entonces sobre cadenas de números reales. Después de un proceso de experimentación con distintos operadores de variación para optimización de parámetros reales [Jim98, Jim99a], finalmente se utilizan dos tipos de cruce, el *cruce uniforme* y el *cruce aritmético*, y tres tipos de mutación, la *mutación uniforme*, *mutación no uniforme* y *mutación mínima*. Los cuatro primeros son operadores bien conocidos en la literatura [Mic96b] y han sido ya vistos en el primer capítulo. El último de los operadores, la mutación mínima, produce un cambio mínimo en el descendiente con respecto al padre, resultando especialmente apropiada para un ajuste fino de parámetros reales.

El cruce se realiza con una probabilidad $pCross$ y el tipo de cruce que se realiza se decide con una probabilidad $pUniformCross$. Por otra parte, para cada uno de los valores x_i se realiza la mutación de dicho valor con probabilidad $pMutate$ y el tipo de mutación se decide con probabilidades $pUniformMutate$ y $pNotUniformMutate$. El resumen de operadores de variación y las probabilidades asociadas se muestra en la tabla 3.1.

Cruce uniforme

Para realizar el *cruce uniforme* dados dos individuos $I_1 = \{x_1^1, \dots, x_p^1\}$ e $I_2 = \{x_1^2, \dots, x_p^2\}$ obtenemos otros dos individuos $I_3 = \{x_1^3, \dots, x_p^3\}$ e $I_4 = \{x_1^4, \dots, x_p^4\}$ donde:

$$\begin{aligned} x_i^3 &= x_i^1 & y & \quad x_i^4 = x_i^2, & \text{ si } r < 0.5 \\ x_i^3 &= x_i^2 & y & \quad x_i^4 = x_i^1, & \text{ si } r \geq 0.5 \end{aligned} \quad , \quad i = 1, \dots, p$$

siendo r un valor real aleatorio entre 0 y 1.

Cruce aritmético

El otro tipo de cruce utilizado es el *cruce aritmético* que opera de la siguiente forma: dados dos individuos $I_1 = \{x_1^1, \dots, x_p^1\}$ e $I_2 = \{x_1^2, \dots, x_p^2\}$ se obtienen otros dos individuos $I_3 = \{x_1^3, \dots, x_p^3\}$ e $I_4 = \{x_1^4, \dots, x_p^4\}$ donde:

$$\begin{aligned} x_i^3 &= x_i^1 d + x_i^2 (1 - d) \\ x_i^4 &= x_i^1 (1 - d) + x_i^2 d \end{aligned} \quad , \quad i = 1, \dots, p$$

siendo d un valor real aleatorio entre 0 y 1.

Mutación uniforme

La *mutación uniforme* genera un individuo aleatorio. Dado un individuo $I_1 = \{x_1^1, \dots, x_p^1\}$ se genera otro individuo $I_2 = \{x_1^2, \dots, x_p^2\}$ donde:

$$x_i^2 = d(u_i + l_i) + l_i \quad , \quad i = 1, \dots, p$$

siendo d un valor real aleatorio entre 0 y 1, y

$[l_i, u_i]$ el dominio de la variable i -ésima.

Mutación no uniforme

La *mutación no uniforme* depende del número de la iteración actual produciendo un ajuste local fino en las últimas generaciones del algoritmo. Dado un individuo $I_1 = \{x_1^1, \dots, x_p^1\}$ se genera otro individuo $I_2 = \{x_1^2, \dots, x_p^2\}$ donde:

$$x_i^2 = \begin{cases} x_i^1 + (u_i - x_i^1)d, & \text{ si } r < 0.5 \\ x_i^1 - (x_i^1 - l_i)d, & \text{ si } r \geq 0.5 \end{cases} \quad , \quad i = 1, \dots, p$$

siendo r un valor real aleatorio entre 0 y 1, y

d un valor real que se calcula de la siguiente forma:

$$d = 1 - l^{(1-\frac{t}{T})^c}$$

siendo t el número de iteración actual,

T el número de iteraciones totales,

l un valor real aleatorio entre 0 y 1, y

c un parámetro del algoritmo que determina el grado de no uniformidad.

Mutación mínima

La *mutación mínima* consiste en realizar variaciones muy pequeñas en los valores, de forma que se realiza una búsqueda de individuos cercanos al original. Este tipo de mutación ha dado muy buenos resultados al trabajar con valores reales donde es necesario realizar un ajuste muy fino de los mismos [Jim99b]. Esta mutación se realiza de igual forma que la mutación no uniforme, con la diferencia que d se calcula como $1 - l^{(1E-5)}$ [Jim99b]. Dado un individuo $I_1 = \{x_1^1, \dots, x_p^1\}$ se genera otro individuo $I_2 = \{x_1^2, \dots, x_p^2\}$ donde:

$$x_i^2 = \begin{cases} x_i^1 + (u_i - x_i^1)d, & \text{si } r < 0.5 \\ x_i^1 - (x_i^1 - l_i)d, & \text{si } r \geq 0.5 \end{cases}, \quad i = 1, \dots, p$$

siendo r un valor real aleatorio entre 0 y 1, y

d un valor real que se calcula de la siguiente forma:

$$d = 1 - l^{1E-5}$$

siendo l un valor real aleatorio entre 0 y 1.

3.1.5 Diversidad

En los algoritmos evolutivos mantener una población diversa es uno de los principios básicos para evitar la convergencia prematura a óptimos locales. Cuando se trata de optimizar múltiples objetivos, el mantener la diversidad en la población se vuelve un punto fundamental ya que no se optimiza un solo punto, sino todo un conjunto de puntos, el *frente Pareto óptimo*. El algoritmo proporciona un buen resultado cuando obtiene un conjunto de puntos distribuidos uniformemente por todo el frente Pareto óptimo. Por tanto, para obtener una buena distribución de puntos Pareto, es necesario que toda la población mantenga una buena diversidad a la vez que sus puntos se acercan al frente Pareto.

Existen diversas técnicas que mantienen la diversidad en la población. Dichas técnicas pueden ser explícitas o implícitas. Entre las técnicas explícitas para mantener la diversidad se encuentra la formación de nichos, pero esta técnica resulta computacionalmente costosa al tener que calcular la cuenta de nicho para cada individuo y, además, es necesario establecer la distancia de nicho como parámetro dependiente del problema para que dicha técnica funcione de forma adecuada. Por otra parte, el esquema de preselección es una técnica de formación de nichos implícita, ya que un descendiente reemplaza a un individuo similar a si mismo (uno de sus padres) y, puesto que las poblaciones iniciales están distribuidas uniformemente en el espacio de búsqueda, con ello se garantiza mantener un cierto grado de diversidad en la población sin un coste computacional excesivo y sin necesidad de establecer parámetros adicionales dependientes del problema.

Otra ventaja de este esquema de selección es el elitismo implícito, ya que, siempre que se reemplaza un individuo, lo es por otro mejor, de forma que el mejor individuo de la población es reemplazado sólo por un individuo mejor, siendo un individuo *I mejor* que otro individuo *J* si *I* domina a *J* (es en este punto donde interviene el concepto Pareto). El mejor individuo de un grupo de individuos es un individuo *I* que no sea dominado por ningún otro individuo del grupo. Si existen varios individuos no dominados en un grupo, el mejor es cualquiera de dichos individuos.

Por todo esto, para nuestro trabajo la preselección resulta la técnica de selección adecuada y ha sido, por tanto, la técnica utilizada como base de los algoritmos propuestos. En concreto, se utiliza la siguiente *variante del esquema de preselección* detallada en la figura 3.1.

- Elegir dos individuos aleatoriamente de la población, $padre_1$ y $padre_2$.
- Crear un individuo $mejor_1$, inicialmente $mejor_1 \leftarrow padre_1$.
- Crear un individuo $mejor_2$, inicialmente $mejor_2 \leftarrow padre_2$.
- Repetir un número $nChildren$ de veces
 - Cruzar $padre_1$ y $padre_2$ para obtener una pareja de descendientes, $hijo_1$ e $hijo_2$.
 - Mutar $hijo_1$ e $hijo_2$.
 - Si $hijo_1$ es mejor que $mejor_1$, entonces $mejor_1 \leftarrow hijo_1$.
 - Si $hijo_2$ es mejor que $mejor_2$, entonces $mejor_2 \leftarrow hijo_2$.
- Si $mejor_1$ es mejor que $padre_1$, entonces $padre_1 \leftarrow mejor_1$.
- Si $mejor_2$ es mejor que $padre_2$, entonces $padre_2 \leftarrow mejor_2$.

Figura 3.1: Esquema de preselección simple.

3.1.6 Experimentos y resultados

Se han considerado los siguientes problemas, resumidos en la tabla 3.5, para comprobar el funcionamiento del algoritmo multiobjetivo propuesto:

1. Optimización con restricciones (un solo objetivo).

Problemas test $G1, \dots, G11$ de Koziel y Michalewicz [Koz99, Mic96b]. Estos problemas vienen detallados en las tablas 3.8, 3.9, 3.10 y 3.11.

2. Optimización multiobjetivo sin restricciones.

Problemas test $F1$ y $F2$ de Srinivas [Sri94]. Estos problemas están detallados en la tabla 3.12.

3. Optimización multiobjetivo con restricciones.

Problema test $F3$ de Srinivas [Sri94] y problema test de Kita et al. [Kit96] detallados en las tablas 3.12 y 3.2 respectivamente.

Para resolver problemas de optimización con restricciones (sean multiobjetivo o no) se realiza la transformación descrita por Jiménez et al. [Jim02]. Como ejemplo, estudiamos el problema de optimización multiobjetivo con restricciones propuesto por Kita et al. [Kit96] detallado en la tabla 3.2.

$$\text{KITA : } \left\{ \begin{array}{l} \text{Minimizar } f_1(\mathbf{x}) = x_1^2 - x_2 \\ \quad \quad \quad f_2(\mathbf{x}) = -\frac{1}{2}x_1 - x_2 - 1 \\ \text{sujeto a :} \\ \quad \quad \quad g_1(\mathbf{x}) = \frac{1}{6}x_1 + x_2 - \frac{13}{2} \leq 0 \\ \quad \quad \quad g_2(\mathbf{x}) = \frac{1}{2}x_1 + x_2 - \frac{15}{2} \leq 0 \\ \quad \quad \quad g_3(\mathbf{x}) = 6x_1 + x_2 - 30 \leq 0 \\ \quad \quad \quad x_1, x_2 \geq 0 \end{array} \right.$$

Tabla 3.2: Problema test de Kita et al. [Kit96].

Este problema tiene dos funciones objetivo y tres restricciones; por tanto puede verse como un problema de optimización multiobjetivo basado en metas y prioridades con cinco funciones objetivo:

$$\mathbf{f}'(\mathbf{x}) = (f'_1(\mathbf{x}), f'_2(\mathbf{x}), f'_3(\mathbf{x}), f'_4(\mathbf{x}), f'_5(\mathbf{x}))$$

con niveles de metas:

$$\mathbf{u} = (-\infty, -\infty, 0, 0, 0)$$

y prioridades:

$$\mathbf{p} = (1, 1, 2, 2, 2)$$

De acuerdo con la formulación descrita por Jiménez et al. [Jim02] definimos las nuevas funciones objetivo de la siguiente forma:

$$f'_1(\mathbf{x}) = \begin{cases} f_1(\mathbf{x}), & \text{si } g_1(\mathbf{x}) \leq 0 \text{ y } g_2(\mathbf{x}) \leq 0 \text{ y } g_3(\mathbf{x}) \leq 0 \\ M, & \text{en otro caso} \end{cases}$$

$$f'_2(\mathbf{x}) = \begin{cases} f_2(\mathbf{x}), & \text{si } g_1(\mathbf{x}) \leq 0 \text{ y } g_2(\mathbf{x}) \leq 0 \text{ y } g_3(\mathbf{x}) \leq 0 \\ M, & \text{en otro caso} \end{cases}$$

$$f'_3(\mathbf{x}) = \begin{cases} -\infty, & \text{si } g_1(\mathbf{x}) \leq 0 \\ g_1(\mathbf{x}), & \text{en otro caso} \end{cases}$$

$$f'_4(\mathbf{x}) = \begin{cases} -\infty, & \text{si } g_2(\mathbf{x}) \leq 0 \\ g_2(\mathbf{x}), & \text{en otro caso} \end{cases}$$

$$f'_5(\mathbf{x}) = \begin{cases} -\infty, & \text{si } g_3(\mathbf{x}) \leq 0 \\ g_3(\mathbf{x}), & \text{en otro caso} \end{cases}$$

Ahora podemos utilizar el algoritmo evolutivo propuesto para resolver el problema de optimización multiobjetivo:

$$\mathbf{f}'(\mathbf{x}) = (f'_1(\mathbf{x}), f'_2(\mathbf{x}), f'_3(\mathbf{x}), f'_4(\mathbf{x}), f'_5(\mathbf{x}))$$

Tamaño de la población	$N = 200$
Probabilidad de cruce	$pCross = 0.9$
Probabilidad de mutación	$pMutate = 0.2$
Probabilidad de cruce uniforme	$pUniformCross = 0.6$
Probabilidad de mutación uniforme	$pUniformMutate = 0.1$
Probabilidad de mutación no uniforme	$pNotUniformMutate = 0.4$
Parámetro c para la mutación no uniforme	$c = 2.0$
Número de hijos para la preselección	$nChildren = 10$

Tabla 3.3: Parámetros en la ejecución del algoritmo para los problemas $G1, \dots, G11$.

Para la ejecución de los problemas $G1, \dots, G11$ se utilizan los parámetros mostrados en la tabla 3.3 y para los problemas $F1, \dots, F3$ y Kita et al. se utilizan los parámetros de la tabla 3.4. El número de iteraciones depende de cada problema. Para los problemas $G1, \dots, G11$ se han realizado 200.000 iteraciones y para el resto de problemas se han realizado 1.000.000 iteraciones. Nótese que al no seguir un esquema de sustitución generacional completo, el número de evaluaciones requeridas para cada iteración no depende del tamaño de la población, sino que es fijo, determinado, en este caso, por el número de hijos utilizado para la preselección, con lo que el número de evaluaciones totales realizadas es menor en este caso. En la tabla 3.5 se realiza un resumen de las características de cada uno de los problemas test utilizado.

La tabla 3.6 resume los resultados obtenidos para los problemas test de optimización con restricciones. Para cada problema test se muestran los valores mejor, peor y medio obtenidos en 10 ejecuciones del algoritmo. Se comparan los resultados con los obtenidos por otros algoritmos evolutivos en [Koz99, Mic96b]. Los resultados mostrados por Koziel y Michalewicz en [Koz99] se obtienen mediante un método de decodificación que utiliza una aplicación homomorfa entre un hipercubo de n dimensiones y un espacio de búsqueda factible. La tabla 3.7 recoge un resumen de los métodos empleados para obtener los resultados mostrados por Michalewicz y Schoenauer en [Mic96b]. El valor mejor obtenido por nuestro algoritmo es, en todos los casos, mejor que el mejor resultado obtenido mediante otras técnicas específicas de

Tamaño de la población	$N = 200$
Probabilidad de cruce	$pCross = 0.9$
Probabilidad de mutación	$pMutate = 0.2$
Probabilidad de cruce uniforme	$pUniformCross = 0.5$
Probabilidad de mutación uniforme	$pUniformMutate = 1/3$
Probabilidad de mutación no uniforme	$pNotUniformMutate = 2/3$
Parámetro c para la mutación no uniforme	$c = 2.0$
Número de hijos para la preselección	$nChildren = 10$

Tabla 3.4: Parámetros en la ejecución del algoritmo para los problemas $F1, \dots, F3$ y Kita et al.

cada problema, como son las utilizadas en [Koz99, Mic96b].

Las figuras 3.2, 3.3, 3.4 y 3.6 muestran gráficamente los valores en el dominio de la función objetivo de las soluciones no dominadas obtenidas para los problemas test $F1$, $F2$, $F3$ y *Kita et al.* respectivamente.

Las figuras 3.5 y 3.7 también muestran los valores en el espacio de búsqueda para los problemas $F3$ y *Kita et al.* respectivamente.

Se puede apreciar la efectividad del esquema de preselección para mantener la diversidad en la población. El esquema de preselección no puede realizar un control preciso del tamaño de nicho, tal como hacen las técnicas de formación de nicho explícitas que utilizan funciones de participación de la adecuación, pero permite realizar una selección suficientemente diversa y se obtienen resultados satisfactorios con menor coste computacional.

Por otra parte, es importante notar que la utilización del concepto Pareto junto con la transformación propuesta por [Jim02] resulta un método adecuado para el manejo de restricciones, puesto que las restricciones son, en realidad, objetivos prioritarios. Además, este método no sólo sirve para manejar restricciones en problemas con un solo objetivo, sino que puede utilizarse de igual forma en problemas con múltiples objetivos y restricciones, tales como los problemas $F3$ de Srinivas y el problema de Kita et al.

<i>Problema</i>	<i>Tipo de f</i>	p	n	m_1	m_2	m_3
<i>G1</i>	<i>Cuadrática</i>	13	1	9	0	0
<i>G2</i>	<i>No lineal</i>	20	1	0	0	2
<i>G3</i>	<i>Polinomial</i>	10	1	0	1	0
<i>G4</i>	<i>Cuadrática</i>	5	1	0	0	6
<i>G5</i>	<i>Cúbica</i>	4	1	2	3	0
<i>G6</i>	<i>Cúbica</i>	2	1	0	0	2
<i>G7</i>	<i>Cuadrática</i>	10	1	3	0	5
<i>G8</i>	<i>No lineal</i>	2	1	0	0	2
<i>G9</i>	<i>Polinomial</i>	7	1	0	0	4
<i>G10</i>	<i>Lineal</i>	8	1	3	0	3
<i>G11</i>	<i>Cuadrática</i>	2	1	1	0	1
<i>F1</i>	<i>Cuadrática</i>	1	2	0	0	0
<i>F2</i>	<i>Lineal, Cuadrática</i>	1	2	0	0	0
<i>F3</i>	<i>Cuadrática</i>	2	2	0	0	2
<i>Kita et al.</i>	<i>Lineal, Cuadrática</i>	2	2	3	0	0

Tabla 3.5: Resumen de los problemas test. Los parámetros p y n representan el número de variables y el número de funciones objetivo respectivamente; m_1 , m_2 y m_3 representan el número de desigualdades lineales, ecuaciones no lineales y desigualdades no lineales respectivamente.

<i>Problema</i>	$f_{\text{óptimo}}$	f_{peor}	f_{mejor}	f_{media}	f_{Koz}	f_{Mic}
<i>G1</i>	-15.0000	-15.0000	-15.0000	-15.0000	-14.7864	-15.0000
<i>G2</i>	0.803619	0.803203	0.800588	0.794836	0.799530	0.803553
<i>G3</i>	1.0000	1.0000	1.0000	1.0000	0.9997	0.9999
<i>G4</i>	-30665.5	-30651.9	-30665.2	-30658.7	-30664.5	-30005.7
<i>G5</i>	4221.96	5058.55	4234.50	4484.74	—	5126.67
<i>G6</i>	-6961.8	-6959.0	-6961.8	-6961.2	-6952.1	-6955.0
<i>G7</i>	24.306	26.048	24.370	24.875	24.620	24.690
<i>G8</i>	0.095825	0.095825	0.095825	0.095825	0.095825	0.095825
<i>G9</i>	680.630	681.18	680.64	680.87	680.910	680.642
<i>G10</i>	7049.33	7501.63	7133.38	7292.78	7147.90	7286.65
<i>G11</i>	0.75	0.75	0.75	0.75	0.75	0.75

Tabla 3.6: Resultados de la simulación para los problemas test de optimización con restricciones $G1, \dots, G11$ de Koziel y Michalewicz [Koz99, Mic96b]. $f_{\text{óptimo}}$ es el valor óptimo para cada problema; f_{peor} , f_{mejor} y f_{media} son los valores peor, mejor y medio obtenidos en 10 ejecuciones del algoritmo; f_{Koz} y f_{Mic} son los valores mejores obtenidos por otros algoritmos evolutivos en [Koz99, Mic96b].

Problema	Método Utilizado	Referencia
Métodos que preservan la factibilidad de las soluciones		
G1	Operadores genéticos específicos, GENOCOP	[Mic92]
G2, G3	Localizar el borde de la región factible	[Glo77]
Métodos basados en penalizaciones		
G4	Penalizaciones estáticas	[Hom94]
G5	Penalizaciones dinámicas	[Joi94]
G6	Penalizaciones con enfriamiento simulado, GENOCOP II	[Mic94]
G7	Penalizaciones mortales	[Mic95b]
Métodos basados en buscar soluciones factibles		
G8	Memoria de comportamiento	[Scho93]
G9	Superioridad de los puntos factibles	[Pow93]
G10	Reparar individuos no factibles, GENOCOP III	[Mic95]
Métodos híbridos		
G11	Multiplicadores de Lagrange	[Kim97, Myu98]

Tabla 3.7: Resumen de los métodos empleados para obtener los resultados mostrados por Michalewicz y Schoenauer en [Mic96b].

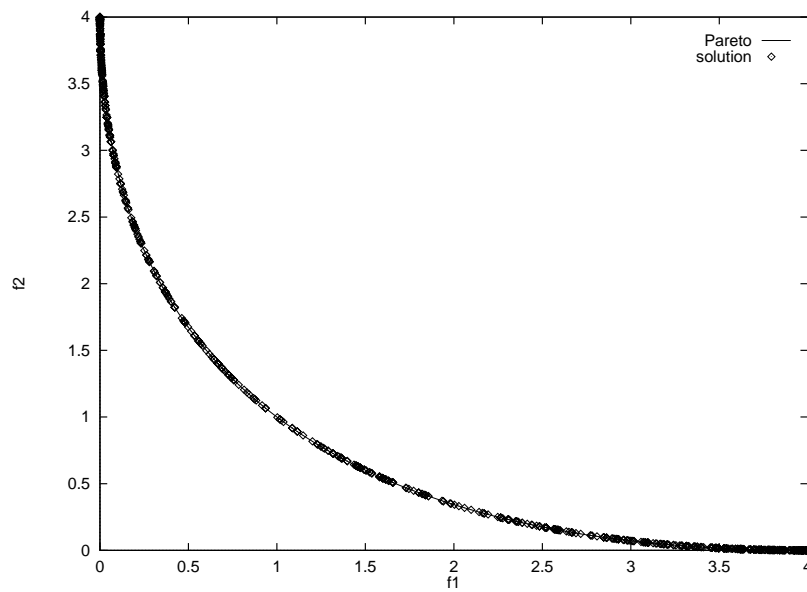


Figura 3.2: Soluciones no dominadas en el dominio de las funciones objetivo para el problema test $F1$ de Srinivas [Sri94].

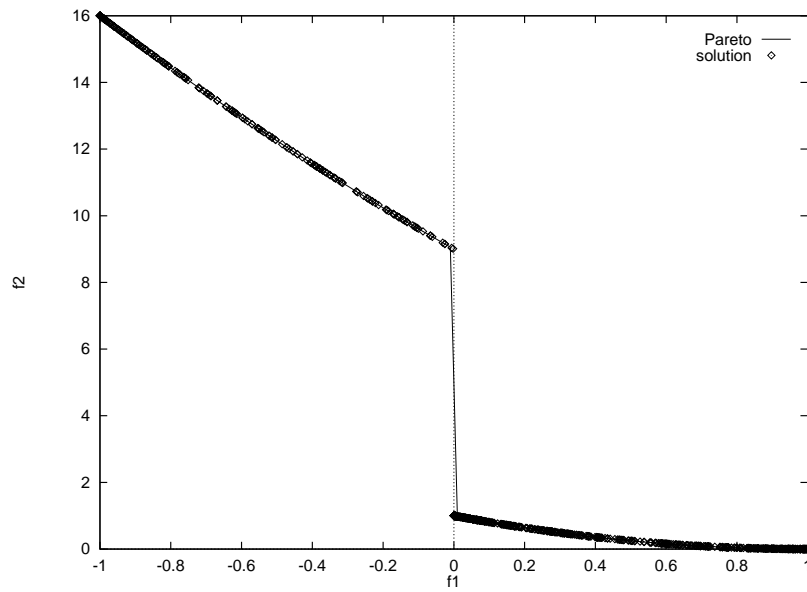


Figura 3.3: Soluciones no dominadas en el dominio de las funciones objetivo para el problema test $F2$ de Srinivas [Sri94].

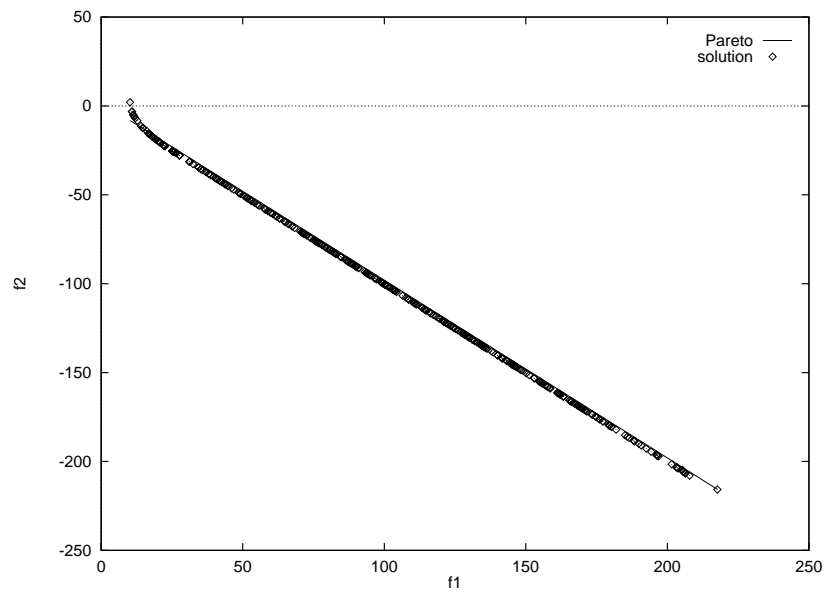


Figura 3.4: Soluciones no dominadas en el dominio de las funciones objetivo para el problema test $F3$ de Srinivas [Sri94].

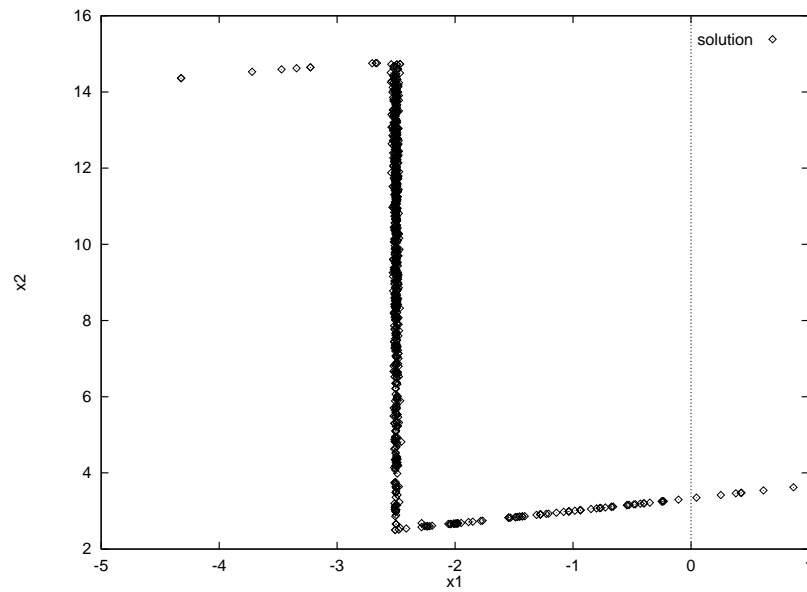


Figura 3.5: Soluciones no dominadas en el dominio del espacio de búsqueda para el problema test $F3$ de Srinivas [Sri94].

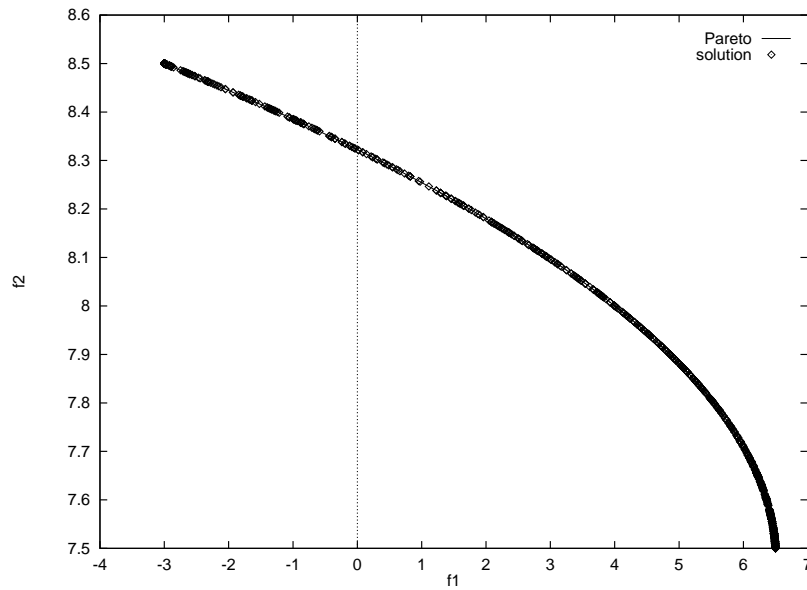


Figura 3.6: Soluciones no dominadas en el dominio de las funciones objetivo para el problema test de Kita et al. [Kit96].

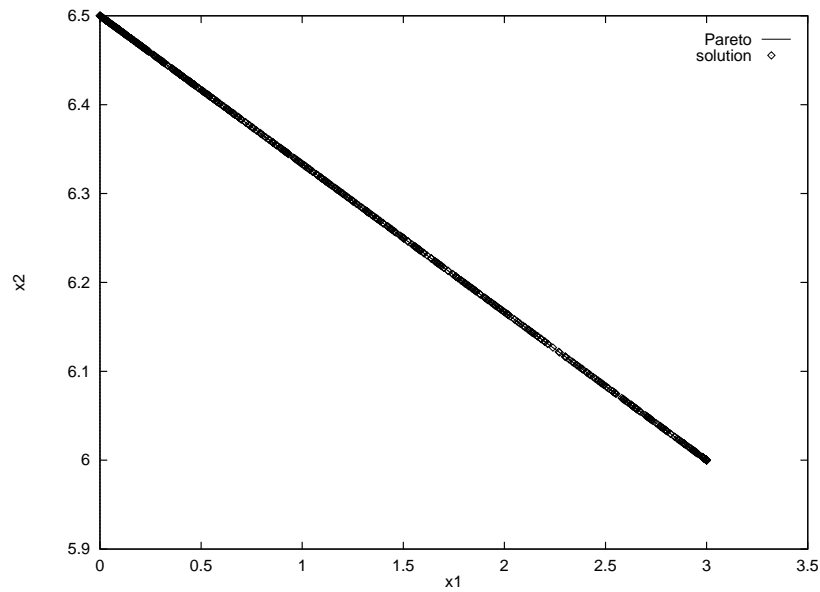


Figura 3.7: Soluciones no dominadas en el dominio del espacio de búsqueda para el problema test de Kita et al. [Kit96].

$G1 :$	$\left\{ \begin{array}{l} \text{Minimizar} \quad f(\mathbf{x}) = 5 \sum_{i=1}^4 x_i - 5 \sum_{i=1}^4 x_i^2 - \sum_{i=1}^{13} x_i \\ \\ \text{sujeto a :} \\ g_1(\mathbf{x}) \equiv 2x_1 + 2x_2 + x_{10} + x_{11} \leq 10 \quad 0 \leq x_i \leq 1, \ i = 1, \dots, 9 \\ g_2(\mathbf{x}) \equiv -8x_1 + x_{10} \leq 0 \quad 0 \leq x_i \leq 100, \ i = 10, \dots, 12 \\ g_3(\mathbf{x}) \equiv -2x_4 - x_5 + x_{10} \leq 0 \quad 0 \leq x_{13} \leq 1 \\ g_4(\mathbf{x}) \equiv 2x_1 + 2x_3 + x_{10} + x_{12} \leq 10 \\ g_5(\mathbf{x}) \equiv -8x_2 + x_{11} \leq 0 \\ g_6(\mathbf{x}) \equiv -2x_6 - x_7 + x_{11} \leq 0 \\ g_7(\mathbf{x}) \equiv 2x_2 + 2x_3 + x_{11} + x_{12} \leq 10 \\ g_8(\mathbf{x}) \equiv -8x_3 + x_{12} \leq 0 \\ g_9(\mathbf{x}) \equiv -2x_8 - x_9 + x_{12} \leq 0 \end{array} \right.$
$G2 :$	$\left\{ \begin{array}{l} \text{Maximizar} \quad f(\mathbf{x}) = \left \frac{\sum_{i=1}^p \cos^4(x_i) - 2 \prod_{i=1}^p \cos^2(x_i)}{\sqrt{\sum_{i=1}^p i x_i^2}} \right \\ \\ \text{sujeto a :} \\ g_1(\mathbf{x}) \equiv \prod_{i=1}^p x_i \geq 0.75 \quad 0 \leq x_i \leq 10, \ i = 1, \dots, p \\ g_2(\mathbf{x}) \equiv \sum_{i=1}^p x_i \leq 7.5p \quad p = 20 \end{array} \right.$
$G3 :$	$\left\{ \begin{array}{l} \text{Maximizar} \quad f(\mathbf{x}) = (\sqrt{p})^p \prod_{i=1}^p x_i \\ \\ \text{sujeto a :} \\ g_1(\mathbf{x}) \equiv \sum_{i=1}^p x_i^2 = 1 \quad 0 \leq x_i \leq 1, \ i = 1, \dots, p \\ p = 10 \end{array} \right.$

Tabla 3.8: Problemas test $G1$, $G2$ y $G3$ de Koziel y Michalewicz [Koz99, Mic96b].

$G4 :$	$\left\{ \begin{array}{l} \text{Minimizar } f(\mathbf{x}) = 5.3578547x_3^2 + 0.8356891x_1x_5 + 37.293239x_1 - 40792.141 \\ \text{sujeto a :} \\ g_1(\mathbf{x}) \equiv 0 \leq 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 - 0.0022053x_3x_5 \leq 92 \\ g_2(\mathbf{x}) \equiv 90 \leq 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 + 0.0021813x_3^2 \leq 110 \\ g_3(\mathbf{x}) \equiv 20 \leq 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4 \leq 25 \\ \\ 78 \leq x_1 \leq 102 \\ 33 \leq x_2 \leq 45 \\ 27 \leq x_i \leq 45, \ i = 3, \dots, 5 \end{array} \right.$
$G5 :$	$\left\{ \begin{array}{l} \text{Minimizar } f(\mathbf{x}) = 3x_1 + 0.000001x_1^3 + 2x_2 + 0.000002/3x_2^3 \\ \text{sujeto a :} \\ g_1(\mathbf{x}) \equiv x_4 - x_3 + 0.55 \geq 0 \\ g_2(\mathbf{x}) \equiv x_3 - x_4 + 0.55 \geq 0 \\ g_3(\mathbf{x}) \equiv 1000 \sin(-x_3 - 0.25) + 1000 \sin(-x_4 - 0.25) + 894.8 - x_1 = 0 \\ g_4(\mathbf{x}) \equiv 1000 \sin(x_3 - 0.25) + 1000 \sin(x_3 - x_4 - 0.25) + 894.8 - x_2 = 0 \\ g_5(\mathbf{x}) \equiv 1000 \sin(x_4 - 0.25) + 1000 \sin(x_4 - x_3 - 0.25) + 1294.8 = 0 \\ \\ 0 \leq x_1, x_2 \leq 1200 \\ -0.55 \leq x_3, x_4 \leq 0.55 \end{array} \right.$
$G6 :$	$\left\{ \begin{array}{l} \text{Minimizar } f(\mathbf{x}) = (x_1 - 10)^3 + (x_2 - 20)^3 \\ \text{sujeto a :} \\ g_1(\mathbf{x}) \equiv (x_1 - 5)^2 + (x_2 - 5)^2 - 100 \geq 0 \quad 13 \leq x_1 \leq 100 \\ g_2(\mathbf{x}) \equiv -(x_1 - 6)^2 - (x_2 - 5)^2 + 82.81 \geq 0 \quad 0 \leq x_2 \leq 100 \end{array} \right.$

Tabla 3.9: Problemas test $G4$, $G5$ y $G6$ de Koziel y Michalewicz [Koz99, Mic96b].

$G7 :$	$\left\{ \begin{array}{l} \text{Minimizar} \quad f(\mathbf{x}) = x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 + \\ \quad + 4(x_4 - 5)^2 + (x_5 - 3)^2 + 2(x_6 - 1)^2 + 5x_7^2 + \\ \quad + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45 \\ \\ \text{sujeto a :} \\ \\ g_1(\mathbf{x}) \equiv 105 - 4x_1 - 5x_2 + 3x_7 - 9x_8 \geq 0 \\ g_2(\mathbf{x}) \equiv -3(x_1 - 2)^2 - 4(x_2 - 3)^2 - 2x_3^2 + 7x_4 + 120 \geq 0 \\ g_3(\mathbf{x}) \equiv -10x_1 + 8x_2 + 17x_7 - 2x_8 \geq 0 \\ g_4(\mathbf{x}) \equiv -x_1^2 - 2(x_2 - 2)^2 + 2x_1x_2 - 14x_5 + 6x_6 \geq 0 \\ g_5(\mathbf{x}) \equiv 8x_1 - 2x_2 - 5x_9 + 2x_{10} + 12 \geq 0 \\ g_6(\mathbf{x}) \equiv -5x_1^2 - 8x_2 - (x_3 - 6)^2 + 2x_4 + 40 \geq 0 \\ g_7(\mathbf{x}) \equiv 3x_1 - 6x_2 - 12(x_9 - 8)^2 + 7x_{10} \geq 0 \\ g_8(\mathbf{x}) \equiv -0.5(x_1 - 8)^2 - 2(x_2 - 4)^2 - 3x_5^2 + x_6 + 30 \geq 0 \\ \\ -10.0 \leq x_i \leq 10.0, \quad i = 1, \dots, 10 \end{array} \right.$
$G8 :$	$\left\{ \begin{array}{l} \text{Maximizar} \quad f(\mathbf{x}) = \frac{\sin^3(2\pi x_1) \sin(2\pi x_2)}{x_1^3(x_1 + x_2)} \\ \\ \text{sujeto a :} \\ \\ g_1(\mathbf{x}) \equiv x_1^2 - x_2 + 1 \leq 0 \quad \quad \quad 0 \leq x_1, x_2 \leq 10 \\ g_2(\mathbf{x}) \equiv 1 - x_1 + (x_2 - 4)^2 \leq 0 \end{array} \right.$
$G9 :$	$\left\{ \begin{array}{l} \text{Minimizar} \quad f(\mathbf{x}) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + \\ \quad + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7 \\ \\ \text{sujeto a :} \\ \\ g_1(\mathbf{x}) \equiv 127 - 2x_1^2 - 3x_2^4 - x_3 - 4x_4^2 - 5x_5 \geq 0 \\ g_2(\mathbf{x}) \equiv 282 - 7x_1 - 3x_2 - 10x_3^2 - x_4 + x_5 \geq 0 \\ g_3(\mathbf{x}) \equiv 196 - 23x_1 - x_2^2 - 6x_6^2 + 8x_7 \geq 0 \\ g_4(\mathbf{x}) \equiv -4x_1^2 - x_2^2 + 3x_1x_2 - 2x_3^2 - 5x_6 + 11x_7 \geq 0 \\ \\ -10.0 \leq x_i \leq 10.0, \quad i = 1, \dots, 7 \end{array} \right.$

Tabla 3.10: Problemas test $G7$, $G8$ y $G9$ de Koziel y Michalewicz [Koz99, Mic96b].

$G10 :$	$\left\{ \begin{array}{l} \text{Minimizar } f(\mathbf{x}) = x_1 + x_2 + x_3 \\ \\ \text{sujeto a :} \\ \\ g_1(\mathbf{x}) \equiv 1 - 0.0025(x_4 + x_6) \geq 0 \\ g_2(\mathbf{x}) \equiv 1 - 0.0025(x_5 + x_7 - x_4) \geq 0 \\ g_3(\mathbf{x}) \equiv 1 - 0.01(x_8 - x_5) \geq 0 \\ g_4(\mathbf{x}) \equiv x_1x_6 - 833.33252x_4 - 100x_1 + 83333.333 \geq 0 \\ g_5(\mathbf{x}) \equiv x_2x_7 - 1250x_5 - x_2x_4 + 1250x_4 \geq 0 \\ g_6(\mathbf{x}) \equiv x_3x_8 - 1250000 - x_3x_5 + 2500x_5 \geq 0 \\ \\ 100 \leq x_1 \leq 10000 \\ 1000 \leq x_2, x_3 \leq 10000 \\ 10 \leq x_i \leq 1000, \ i = 4, \dots, 8 \end{array} \right.$
$G11 :$	$\left\{ \begin{array}{l} \text{Minimizar } f(\mathbf{x}) = x_1^2 + (x_2 - 1)^2 \\ \\ \text{sujeto a :} \\ \\ g_1(\mathbf{x}) \equiv x_2 - x_1^2 = 0 \qquad -1 \leq x_1, x_2 \leq 1 \end{array} \right.$

Tabla 3.11: Problemas test $G10$ y $G11$ de Koziel y Michalewicz [Koz99, Mic96b].

$F1 :$	$\left\{ \begin{array}{l} \text{Minimizar} \quad f_1(\mathbf{x}) = x_1^2 \\ \quad \quad \quad f_2(\mathbf{x}) = (x_1 - 2)^2 \\ \\ -100 \leq x_1 \leq 100. \end{array} \right.$
$F2 :$	$\left\{ \begin{array}{l} \text{Minimizar} \\ \\ f_1(\mathbf{x}) = \begin{cases} -x_1, & \text{si } x_1 \leq 1 \\ -2 + x_1, & \text{si } 1 < x_1 \leq 3 \\ 4 - x_1, & \text{si } 3 < x_1 \leq 4 \\ -4 + x_1, & \text{si } x_1 > 4 \end{cases} \\ f_2(\mathbf{x}) = (x_1 - 5)^2 \\ \\ -10 \leq x_1 \leq 10. \end{array} \right.$
$F3 :$	$\left\{ \begin{array}{l} \text{Minimizar} \quad f_1(\mathbf{x}) = (x_1 - 2)^2 + (x_2 - 1)^2 + 2 \\ \quad \quad \quad f_2(\mathbf{x}) = 9x_1 - (x_x - 1)^2 \\ \text{sujeto a :} \\ \\ g_1(\mathbf{x}) \equiv x_1^2 + x_2^2 - 225 \leq 0 \\ g_2(\mathbf{x}) \equiv x_1 - 3x_2 + 10 \leq 0 \\ \\ -20 \leq x_1, x_2 \leq 20. \end{array} \right.$

Tabla 3.12: Problemas test $F1$, $F2$ y $F3$ de Srinivas [Sri94].

3.2 Algoritmos Evolutivos para Problemas de Optimización Multiobjetivo con Restricciones

En esta sección se proponen tres nuevos algoritmos evolutivos multiobjetivo basados en dominación Pareto para resolver problemas de optimización multiobjetivo con restricciones de la forma planteada en la ecuación (2.1). Estos algoritmos presentan una estructura general clásica similar a la del algoritmo evolutivo para problemas de optimización multiobjetivo sin restricciones presentado en la sección 3.1: basados en el concepto Pareto, representación en punto flotante, población inicial aleatoria y operadores de variación sobre números reales. La diferencia fundamental de los algoritmos presentados en esta sección respecto al algoritmo presentado en la sección anterior se encuentra en dos puntos: (i) manejo de restricciones, e (ii) implementación de diferentes técnicas de diversidad.

En primer lugar, los algoritmos presentados en esta sección incorporan manejo de restricciones, es decir, utilizan técnicas específicas para realizar el manejo de restricciones, por lo que pueden resolver directamente problemas de optimización multiobjetivo con restricciones de la forma descrita en la ecuación (2.1) sin necesidad de realizar ninguna transformación de los problemas.

Por otra parte, en los algoritmos evolutivos mantener una población diversa es uno de los principios básicos para evitar la convergencia previa a óptimos locales. Cuando se trata de optimizar múltiples objetivos, el mantener la diversidad en la población se vuelve un punto fundamental ya que tratamos de optimizar, no un solo punto, sino todo un conjunto de puntos llamado *frente Pareto óptimo* formado por todos aquellos puntos que no pueden ser dominados. Se trata de obtener soluciones lo más cercanas posibles al frente Pareto óptimo y también lo más dispersas a lo largo de dicho frente. Por tanto, para obtener una buena distribución de puntos Pareto es necesario que toda la población mantenga una buena diversidad, a la vez que sus puntos se acercan al frente Pareto. Adicionalmente, la introducción de restricciones puede hacer más difícil mantener la diversidad de la población al acercarse al frente Pareto. Con el objetivo de diseñar algoritmos que obtenga soluciones distribuidos uniformemente por todo el frente Pareto óptimo, incluso con la presencia de restricciones que puedan representar dificultades añadidas, se han propuesto diferentes técnicas para manejar la diversidad: (i) preselección simple, (ii) utilizar una métrica de diversidad,

y (iii) realizar una partición del espacio de búsqueda en tramos radiales. El primer algoritmo propuesto en esta sección utiliza la preselección simple, que es la técnica de diversidad más sencilla y sirve como base para los demás algoritmos; el segundo algoritmo utiliza una métrica de diversidad junto con el resto de objetivos, lo cual es una técnica de formación explícita de nichos y el último algoritmo, ENORA (Evolutionary NON dominated sorting with RADial slots) realiza una partición del espacio de búsqueda en tramos radiales, que es una técnica *ad hoc* mediante la cual se realiza la búsqueda sólo de soluciones diversas.

3.2.1 Algoritmo I. Algoritmo evolutivo con preselección simple

Este algoritmo utiliza la preselección simple descrita en la figura 3.1, que es la misma técnica utilizada por el algoritmo presentado en la sección 3.1. La diferencia fundamental de este algoritmo respecto al algoritmo presentado en la sección 3.1 se encuentra en el manejo de restricciones, ya que este algoritmo incorpora una técnica específica para manejo de restricciones. La técnica utilizada para el manejo de restricciones presenta como ventaja frente a otras técnicas la independencia con respecto al problema y la integración con la evaluación de las funciones objetivos.

Para el manejo de restricciones, las poblaciones generadas por el algoritmo están constituidas por individuos tanto factibles como no factibles. Los individuos factibles evolucionan hacia la optimalidad guiados por el concepto Pareto de optimización multiobjetivo, mientras que los individuos no factibles evolucionan hacia la factibilidad guiados por una función de adecuación basada en la formulación *min-max*, como se verá más adelante. De esta forma, el algoritmo resultante tiene una escasa dependencia con el problema a optimizar, ya que es la propia heurística evolutiva la que se usa para la satisfacción de restricciones, a diferencia de las técnicas de reparo, decodificación o penalización que suelen ser fuertemente dependientes del problema.

El manejo de las restricciones se realiza considerando lo siguiente:

1. Una población está formada por individuos factibles y no factibles.
2. Los individuos factibles evolucionan hacia la optimalidad guiados por una *función de adecuación de optimalidad*.

3. Los individuos no factibles evolucionan hacia la factibilidad guiados por una *función de adecuación de factibilidad*.
4. Los individuos factibles tienen mejor valor de *adecuación* que los individuos no factibles.

Para calcular las funciones de evaluación de optimalidad y de factibilidad, es necesario realizar las siguientes definiciones:

\mathbf{f}^j : Vector de valores de las funciones objetivo para el individuo j .

$$\mathbf{f}^j = (f_1^j, \dots, f_n^j)$$

\mathbf{g}^j : Vector de valores de las restricciones para el individuo j .

$$\mathbf{g}^j = (g_1^j, \dots, g_m^j)$$

Individuo factible: Se dice que un individuo j es *factible* si cumple todas las restricciones.

$$\forall i = 1, \dots, m, g_i^j \leq 0$$

Individuo no factible: Se dice que un individuo j no es *factible* si no cumple todas las restricciones.

$$\exists i = 1, \dots, m / g_i^j > 0$$

g_{max}^j : Función de evaluación de factibilidad para un individuo j .

$$g_{max}^j = \max_{i=1, \dots, m} \{g_i^j\}$$

La comparación entre dos individuos se realiza mediante la función *mejor* que se establece de la forma descrita en la figura 3.8.

Tanto la función de adecuación de optimalidad como la de factibilidad deben ser minimizadas. Existe una similitud entre la función de adecuación de factibilidad y el método de formulación min-max utilizado en optimización multiobjetivo [Cha83]. Este método se ha usado para minimizar las desviaciones relativas a su

- Un individuo factible es *mejor* que otro individuo no factible.
- Un individuo no factible es *mejor* que otro individuo no factible si la función g_{max} del primero es menor que la del segundo.
- Un individuo factible es *mejor* que otro individuo factible si el primero domina al segundo.

Figura 3.8: Función *mejor* para comparar individuos factibles y no factibles.

óptimo individual, y puede obtener la mejor solución compromiso cuando se optimizan objetivos con igual prioridad. Puesto que restricciones y objetivos se pueden tratar de forma similar, y se asume que todas las restricciones tienen igual prioridad, la formulación *min-max* resulta apropiada para satisfacción de restricciones, siendo además una técnica independiente del problema.

3.2.2 Algoritmo II. Algoritmo evolutivo con métricas de diversidad

En problemas con cierto grado de dificultad el esquema de preselección simple no garantiza un grado suficiente de dispersión para cubrir todo el frente Pareto. Para solucionar esto, se propone una técnica de diversidad explícita que consiste en modificar la definición de *mejor*. En el esquema anterior, un individuo es mejor que otro si su evaluación, según la función mejor descrita en la figura 3.8, resulta mejor que la del otro individuo. En el nuevo esquema, tal como se describe en la figura 3.9, además de la evaluación, se tiene en cuenta un valor de diversidad. Este esquema garantiza que al reemplazar un individuo por otro la diversidad, al menos, no empeora. Nótese que esto es utilizar el concepto Pareto para diversidad, sin embargo este método no es equivalente a considerar la diversidad como un objetivo más, sino que la diversidad es un objetivo y los demás objetivos del problema forman conjuntamente un segundo objetivo, es decir, tiene tanta importancia mantener la diversidad como avanzar hacia el frente Pareto.

Un individuo I_1 será *mejor*₂ que otro individuo I_2 si se cumple alguna de las siguientes condiciones:

1. I_1 es mejor que I_2 y la diversidad de I_1 no es peor que la de I_2
2. I_1 no es peor que I_2 y la diversidad de I_1 es mejor que la de I_2

Figura 3.9: Función *mejor*₂ con diversidad explícita.

Existen diversas formas de medir la diversidad de un individuo, algunas como medir la dispersión o calcular nichos resultan muy costosas computacionalmente al tener que tener en cuenta todos los individuos de la población. Necesitamos una medida de diversidad que resulte sencilla y poco costosa. Para ello se define una técnica que clasifica los individuos según su localización espacial. Para calcular la diversidad de un individuo según esta técnica es necesario, en primer lugar, realizar las siguientes definiciones:

N : Número de individuos de la población.

\mathbf{f}_{max} : Vector de valores máximos de las funciones objetivo.

$$\mathbf{f}_{max} = (f_1^{max}, \dots, f_n^{max}), \text{ donde } f_i^{max} = \max_{j=1, \dots, N} \{f_i^j\}$$

\mathbf{f}_{min} : Vector de valores mínimos de las funciones objetivo.

$$\mathbf{f}_{min} = (f_1^{min}, \dots, f_n^{min}), \text{ donde } f_i^{min} = \min_{j=1, \dots, N} \{f_i^j\}$$

\mathbf{f}_{med} : Vector de valores medianos de las funciones objetivo.

$$\mathbf{f}_{med} = (f_1^{med}, \dots, f_n^{med}), \text{ donde } f_i^{med} = \frac{f_i^{max} + f_i^{min}}{2}$$

\mathbf{h}_j : Vector de valores normalizados de las funciones objetivo para el individuo j .

$$\mathbf{h}_j = (h_1^j, \dots, h_n^j), \text{ donde } h_i^j = \frac{f_i^{max} - f_i^j}{f_i^{max} - f_i^{min}}$$

$distancia(j, k)$: Distancia entre los individuos j y k .

$$distancia(j, k) = \sqrt{\sum_{i=1}^n (h_i^j - h_i^k)^2}$$

$dmed(j)$: Distancia del individuo j a los valores medianos de las funciones objetivo.

$$dmed(j) = \sqrt{\sum_{i=1}^n (h_i^j - h_i^{med})^2}$$

d_{max} : Distancia máxima.

$$d_{max} = \frac{1}{2} \sqrt{\sum_{i=1}^n (h_i^{max} - h_i^{min})^2}$$

D : Número de tramos totales en los que se divide el espacio objetivo. En nuestro caso, el número de tramos totales D es igual al número de individuos N , de forma que idealmente al final de la ejecución del algoritmo se obtendría un individuo por cada tramo.

$$D = N$$

d_{min} : Distancia mínima que debe existir entre dos individuos para que se cuenten en la medida de diversidad. En nuestro caso, dicha distancia es igual a la anchura de cada uno de los tramos.

$$d_{min} = \frac{d_{max}}{D}$$

El cálculo del valor de diversidad para un individuo se hace de la siguiente forma:

1. El tramo t al cual pertenece un individuo j se calcula de la siguiente forma:

$$t = \lfloor dmed(j) \frac{D}{d_{max}} \rfloor$$

2. Por cada tramo tenemos una lista de individuos, de forma que cuando añadimos un nuevo individuo, éste se añade a la lista de su tramo correspondiente y cuando eliminamos un individuo, éste se elimina de la lista de su tramo correspondiente.

3. El valor de diversidad para un individuo es el número de individuos de su tramo y de los tramos adyacentes tales que su distancia al individuo sea menor que la distancia mínima d_{min} .

Con esta técnica se consigue un grado mayor de diversidad puesto que, además de la diversidad implícita en el esquema de preselección, se utiliza una técnica de diversidad explícita. Por otra parte, este método computacionalmente no es muy costoso ya que para calcular la diversidad de un individuo sólo tenemos que contar los individuos de su tramo o tramos adyacentes, no todos los individuos de la población.

3.2.3 ENORA. Algoritmo evolutivo con ordenación no dominada por tramos radiales

El último algoritmo propuesto, ENORA (Evolutionary NON dominated sorting with RAdial slots), utiliza una técnica de diversidad *ad hoc* que consiste en dividir el espacio de búsqueda en tramos radiales de forma que se busque sólo entre soluciones dispersas. El esquema que sigue ENORA es el siguiente:

1. Seleccionar dos individuos aleatorios.
2. Obtener dos descendientes mediante cruce y mutación de los padres.
3. Insertar los descendientes en la población.

En este esquema, el mecanismo de diversidad se encuentra incluido dentro de la inserción de los descendientes. Antes de pasar a describir detalladamente el mecanismo de inserción de los descendientes en la nueva población, definimos los siguientes términos:

F : Conjunto de individuos factibles de la población.

El individuo j pertenece a F si j es un individuo factible.

\mathbf{f}_{max} : Vector de valores máximos de las funciones objetivo para individuos factibles.

$$\mathbf{f}_{max} = (f_1^{max}, \dots, f_n^{max}), \text{ donde } f_i^{max} = \max_{j \in F} \{f_i^j\}$$

\mathbf{f}_{min} : Vector de valores mínimos de las funciones objetivo para individuos factibles.

$$\mathbf{f}_{min} = (f_1^{min}, \dots, f_n^{min}), \text{ donde } f_i^{min} = \min_{j \in F} \{f_i^j\}$$

\mathbf{h}_j : Vector de valores normalizados de las funciones objetivo para el individuo j .

$$\mathbf{h}_j = (h_1^j, \dots, h_n^j), \text{ donde } h_i^j = \frac{f_i^{max} - f_i^j}{f_i^{max} - f_i^{min}}$$

C_{in} : Conjunto de individuos j de la población tales que:

$$\forall i = 1, \dots, n, f_i^{min} \leq f_i^j \leq f_i^{max}$$

C_{out} : Conjunto de individuos j de la población que no pertenecen a C_{in} .

D : Número de listas con individuos que pertenecen a C_{in} .

$$D = d^{n-1}, \text{ donde } d = \left\lfloor N^{\frac{1}{n-1}} \right\rfloor$$

$L_{in}^1, \dots, L_{in}^D$: Listas que contienen los individuos pertenecientes a C_{in} .

L_{out} : Lista que contiene los individuos pertenecientes a C_{out} .

En este esquema, el punto fundamental para mantener la diversidad es la inserción de los hijos. Los individuos que pertenecen a C_{in} se distribuyen dentro de D listas $L_{in}^1, \dots, L_{in}^D$, y tenemos una lista adicional L_{out} para los individuos que pertenecen a C_{out} . Dentro de cada lista los individuos se ordenan de acuerdo a la función *mejor*, descrita en la figura 3.8.

Hay que notar también que no siempre se realiza de forma efectiva la inserción de los nuevos individuos, sino que esto ocurre únicamente en los casos en los que el nuevo individuo sea mejor que el individuo sustituido y la diversidad no empeore, o que el nuevo individuo no sea peor que el individuo sustituido y la diversidad mejore. Por tanto, la técnica permite de forma simultánea la optimización y preservación o mejora de la diversidad. También es una técnica elitista, puesto que los mejores individuos nunca son reemplazados por otros peores.

La inserción de un individuo j se realiza de la siguiente forma:

1. Si el individuo j pertenece a C_{out} :
 - Insertar el individuo j en la posición de la lista L_{out} que le corresponda, de acuerdo a la función *mejor*, y

- eliminar el último individuo de L_{out} .
2. Si el individuo j pertenece a C_{in} :
- Calcular el tramo t que corresponda al individuo j , e
 - insertar el individuo j en la lista L_{in}^t .

El tramo t correspondiente a un individuo j se calcula tal como se expresa en la siguiente ecuación:

$$t = \sum_{i=0}^{n-1} d^i \lfloor d \frac{\alpha_i}{\pi/2} \rfloor, \text{ con } \alpha_i = \begin{cases} \frac{\pi}{2}, & \text{si } h_i^j = 0 \\ \arctan(\frac{h_{i+1}^j}{h_i^j}), & \text{si } h_i^j \neq 0 \end{cases}, \quad i = 1, \dots, n$$

La inserción del individuo j en la lista L_{in}^t se hace de la siguiente forma:

- (a) Si L_{in}^t es vacía:
- Buscar la lista L_{in}^r que contenga un mayor número de individuos.
 - Si el número de individuos de L_{in}^r es mayor que 1, entonces eliminar el último elemento de L_{in}^r e insertar el individuo en L_{in}^t .
- (b) Si L_{in}^t no es vacía:
- Insertar el individuo en la posición de la lista que le corresponda, de acuerdo a la función *mejor*, y
 - eliminar el último individuo de L_{in}^t .

Esta técnica realiza una partición del espacio de búsqueda en tramos radiales tal como muestra la figura 3.10 para el caso de 2 objetivos. En el caso general de n objetivos cada uno de los tramos sería un hipercono de dimensión n y en el caso particular de un solo objetivo, el espacio de búsqueda tendría una sola dimensión por lo que no se realiza ninguna partición en este caso, sino que existe una sola lista L_{in}^0 y $t = 0$ en cualquier caso.

Es importante notar que ésta es una técnica *ad hoc* para mantener la diversidad que particiona el espacio de búsqueda en tramos radiales, de forma que fuerza a que se distribuyan los puntos en el espacio, garantizando con ello la diversidad. Es, por tanto, una técnica específica para problemas de optimización multiobjetivo.

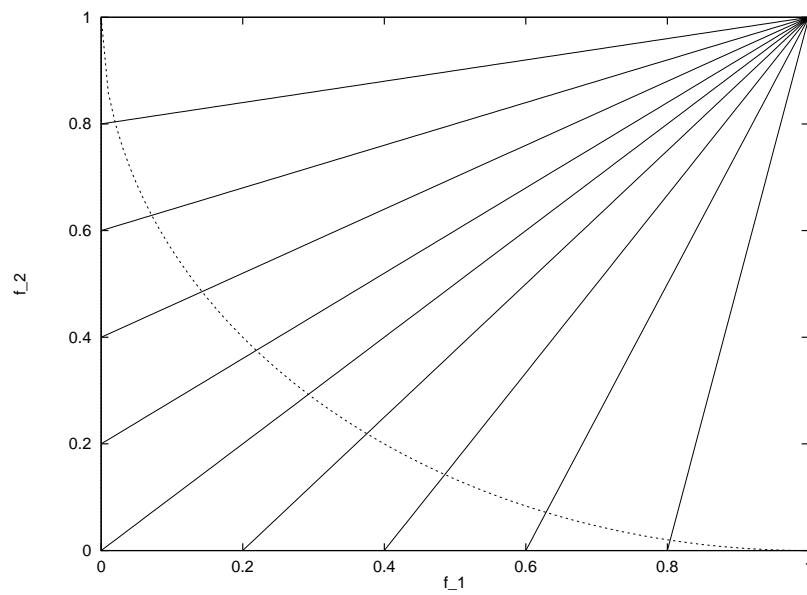


Figura 3.10: Partición del espacio de búsqueda en tramos radiales para 2 objetivos.

3.2.4 Problemas test

Para validar el comportamiento de los métodos propuestos se realizan ejecuciones sobre un conjunto significativo de problemas test importantes para optimización multiobjetivo. Deb et al. [Deb01a] recogen algunos problemas test clásicos para optimización multiobjetivo y proponen nuevos problemas. Utilizaremos los problemas descritos por Deb et al. para comprobar el funcionamiento de nuestros algoritmos.

Problemas test clásicos recogidos por Deb et al.

Los problemas test clásicos recogidos por Deb et al. son el problema propuesto por Srinivas y Deb [Sri94], el problema propuesto por Tanaka et al. [Tan95], y el problema propuesto por Osyczka y Kundy [Osy95]. A continuación se realiza una descripción de estos problemas.

Problema SRN. Problema propuesto por Srinivas y Deb [Sri94] detallado en la tabla 3.13. En este problema, las soluciones Pareto óptimas corresponden a los valores $x_1 = -2.5$ y $x_2 \in [-14.79, 2.50]$. La única dificultad que introducen las restricciones en este problema es que eliminan parte del conjunto Pareto óptimo sin restricciones.

$$SRN : \left\{ \begin{array}{l} \text{Minimizar} \quad f_1(\mathbf{x}) = 2 + (x_1 - 2)^2 + (x_2 - 1)^2 \\ \quad \quad \quad f_2(\mathbf{x}) = 9x_1 - (x_2 - 1)^2 \\ \text{suje}to \ a : \\ \quad \quad \quad g_1(\mathbf{x}) \equiv x_1^2 + x_2^2 \leq 225 \\ \quad \quad \quad g_2(\mathbf{x}) \equiv x_1 - 3x_2 + 10 \leq 0 \\ \\ \quad \quad \quad -20 \leq x_1, x_2 \leq 20 \end{array} \right.$$

Tabla 3.13: Problema test de Srinivas y Deb [Sri94].

Problema TNK. Problema propuesto por Tanaka et al. [Tan95] detallado en la tabla 3.14. En este problema, el espacio de decisión de las variables y el espacio objetivo son el mismo. El espacio de decisión de la variable sin restricciones consiste en todas las soluciones en el cuadrado $0 \leq x_1, x_2 \leq \pi$, por tanto la única solución Pareto óptima sin restricciones es $x_1 = x_2 = 0$. Sin embargo, la primera restricción hace que esta solución sea no factible. La solución Pareto óptima con restricciones se encuentra en el límite marcado por la primera restricción. Debido a que dicha restricción es periódica y también debe satisfacerse la segunda restricción, no todas las soluciones de dicho límite son Pareto óptimas, sino que forman un conjunto disconexo. Puesto que las soluciones se encuentran en una superficie no lineal, esto puede causar dificultades para encontrar soluciones dispersas por todo el frente Pareto óptimo.

$$TNK : \left\{ \begin{array}{ll} \text{Minimizar} & f_1(\mathbf{x}) = x_1 \\ & f_2(\mathbf{x}) = x_2 \\ \text{sujeto a :} & \\ & g_1(\mathbf{x}) \equiv x_1^2 + x_2^2 - 1 \geq 0.1 \cos \left(16 \arctan \frac{x_1}{x_2} \right) \\ & g_2(\mathbf{x}) \equiv (x_1 - \frac{1}{2})^2 + (x_2 - \frac{1}{2})^2 \leq \frac{1}{2} \\ & 0 \leq x_1, x_2 \leq \pi \end{array} \right.$$

Tabla 3.14: Problema test de Tanaka et al. [Tan95].

Nuevos problemas test propuestos por Deb et al.

Deb et al. plantean un conjunto de nuevos problemas test cuya mayor novedad consiste en ser problemas cuya complejidad puede ser controlada mediante diversos parámetros del problema, de forma que ajustando dichos parámetros se puede controlar tanto el grado como el tipo de dificultad. En concreto, los problemas test propuestos, han sido diseñados para provocar dos tipos de dificultad:

- Dificultad cerca del frente Pareto óptimo, y
- Dificultad en todo el espacio de búsqueda.

Dificultad cerca del frente Pareto óptimo

En primer lugar, Deb et al. proponen un conjunto de problemas para crear dificultades cerca del frente Pareto óptimo. A continuación se realiza una descripción de estos problemas.

Problema CTP1. Este es el primer problema test, mostrado en la tabla 3.16. En este problema las restricciones sólo afectan cerca del frente Pareto óptimo, de forma que parte del frente Pareto óptimo vendrá definido por las restricciones.

$$CTP1 : \left\{ \begin{array}{l} \text{Minimizar} \quad f_1(\mathbf{x}) = x_1 \\ \quad \quad \quad f_2(\mathbf{x}) = g(\mathbf{x}) \exp[-f_1(\mathbf{x})/g(\mathbf{x})] \\ \text{sujeto a :} \\ \quad \quad \quad g_j(\mathbf{x}) \equiv f_2(\mathbf{x}) - a_j \exp[-b_j f_1(\mathbf{x})] \geq 0, \quad j = 1, \dots, J \\ \\ \quad \quad \quad 0 \leq x_1 \leq 1 \\ \quad \quad \quad -5 \leq x_2, x_3, x_4 \leq 5 \end{array} \right.$$

Tabla 3.16: Problema test CTP1 de Deb et al. [Deb01a].

La función $g(\mathbf{x})$ puede ser una variable compleja multi-variable. Deb et al. proponen utilizar la siguiente función:

$$g(\mathbf{x}) = 31 + \sum_{i=1}^3 (x_i^2 - 10 \cos(4\pi x_i)) \quad (3.1)$$

El problema tiene J restricciones. Los parámetros (a_j, b_j) , $j = 1, \dots, J$ deben escogerse de forma que al menos alguna porción de la región Pareto óptima sin restricciones sea no factible. Deb et al. proponen el siguiente procedimiento para calcular (a_j, b_j) :

1. Establecer $j = 0$, $a_j = b_j = 1$, $\Delta = \frac{1}{J+1}$, y $\alpha = \Delta$.
2. Calcular $\beta = a_j \exp(-b_j \alpha)$,

$$a_{j+1} = \frac{a_j + \beta}{2}, \quad b_{j+1} = -\frac{1}{\alpha} \ln \left(\frac{\beta}{a_{j+1}} \right)$$
3. Incrementar $\alpha = \alpha + \Delta$, $j = j + 1$.
4. Si $j < J$ volver al paso 2, en otro caso terminar.

Para $J = 2$ restricciones, se tienen los siguientes valores:

$$\begin{aligned} a_1 &= 0.858 & b_1 &= 0.541 \\ a_2 &= 0.728 & b_2 &= 0.295 \end{aligned}$$

Otra forma de introducir dificultades cerca del frente Pareto consiste en poner restricciones de forma que el frente Pareto sea factible en algunos tramos y no factible en otros. En el caso extremo, el frente Pareto puede ser un conjunto de puntos discretos. Tal función puede ser descrita matemáticamente según se expresa en la tabla 3.17. Como función $g(\mathbf{x})$ se utiliza la misma que para el problema CTP1, expresada en la ecuación (3.1). Este problema puede utilizarse como un generador de problemas test, donde la complejidad de los diferentes problemas vendrá dada por los parámetros elegidos en cada caso. Los parámetros para los problemas CTP2 a CTP7 se muestran en la tabla 3.18.

$$CTP2 - CTP7 : \left\{ \begin{array}{l} \text{Minimizar} \quad f_1(\mathbf{x}) = x_1 \\ \quad \quad \quad f_2(\mathbf{x}) = g(\mathbf{x}) \left(1 - \sqrt{\frac{f_1(\mathbf{x})}{g(\mathbf{x})}} \right) \\ \text{sueto a :} \\ \quad \quad \quad c(\mathbf{x}) \equiv \cos(\theta) (f_2(\mathbf{x}) - e) - \sin(\theta) f_1(\mathbf{x}) \geq \\ \quad \quad \quad a |\sin \{ b\pi [\sin(\theta) (f_2(\mathbf{x}) - e) + \cos(\theta) f_1(\mathbf{x})]^c \}|^d \\ \\ \quad \quad \quad 0 \leq x_1 \leq 1 \\ \quad \quad \quad -5 \leq x_2, x_3, x_4 \leq 5 \end{array} \right.$$

Tabla 3.17: Problemas test CTP2-CTP7 de Deb et al. [Deb01a].

<i>Problema</i>	θ	a	b	c	d	e
<i>CTP2</i>	-0.2π	0.2	10	1	6	1
<i>CTP3</i>	-0.2π	0.1	10	1	0.5	1
<i>CTP4</i>	-0.2π	0.75	10	1	0.5	1
<i>CTP5</i>	-0.2π	0.75	10	2	0.5	1
<i>CTP6</i>	0.1π	40	0.5	1	2	-2
<i>CTP7</i>	-0.05π	40	5	1	6	0

Tabla 3.18: Parámetros para los problemas test CTP2-CTP7.

Problema CTP2. En este problema el frente Pareto se vuelve no factible por tramos. El número de regiones factibles del frente Pareto puede ser controlada mediante el parámetro b . Un algoritmo debería ser capaz de encontrar puntos en todas las regiones factibles del frente Pareto.

Problema CTP3. El problema anterior puede complicarse utilizando un valor menor del parámetro d de forma que en cada región factible del frente Pareto exista un solo punto. En el problema CTP3 la mayor parte del espacio de búsqueda es continuo, pero cerca de la región Pareto óptima las regiones factibles se vuelven discontinuas y finalmente cada región llega a una única solución factible Pareto óptima.

Problema CTP4. Incrementando el valor del parámetro a , la transición de la región factible de continua a discontinua se hace más alejada del frente Pareto óptimo, de forma que un algoritmo tiene que pasar a través de un largo y estrecho túnel factible para buscar la única solución Pareto óptima al final de dicho túnel. Este problema resulta mucho más difícil de solucionar comparado con los anteriores.

Problema CTP5. En los problemas anteriores las regiones factibles estaban igualmente distribuidas en el espacio objetivo. Puede conseguirse una distribución no uniforme de las regiones utilizando $c \neq 1$. Para este problema se utiliza $c = 2$. Puesto que $c > 1$ tendremos más soluciones con valores mayores de f_1 . Si se utilizara $c < 1$ tendríamos el efecto opuesto.

Dificultad en todo el espacio de búsqueda

Los problemas anteriores ofrecen dificultad cerca del frente Pareto. Sin embargo, las dificultades pueden estar presentes en todo el espacio de búsqueda. Para conseguirlo, puede utilizarse el mismo generador de problemas variando los parámetros.

Problema CTP6. En este problema el espacio objetivo tiene una serie de zonas no factibles de diferentes anchuras paralelos al frente Pareto óptimo. Un algoritmo tiene que superar dichas zonas no factibles antes de llegar a la región que contiene el frente Pareto. Más aún, en este caso, la región Pareto óptima no es factible, por tanto el frente Pareto óptimo se encuentra por entero definido por una parte de los límites de las restricciones. Se puede incrementar la dificultad haciendo más amplias las zonas no factibles utilizando un menor valor de d .

Problema CTP7. Por último, las zonas no factibles del espacio de búsqueda pueden ser perpendiculares al frente Pareto óptimo. En el problema CTP7 se obtiene parte de la región Pareto óptima factible y parte no factible. La dificultad en este caso puede aumentarse de nuevo haciendo más amplias las zonas no factibles utilizando un valor menor de d .

3.2.5 Experimentos y resultados

Para validar el funcionamiento de nuestros algoritmos y comparar los resultados con los de otros algoritmos, se realizan simulaciones con los problemas test anteriormente descritos. A continuación se describen las simulaciones efectuadas y los resultados obtenidos con las mismas.

Resolución de problemas de optimización con restricciones

Para probar la resolución de problemas de optimización con restricciones se ejecuta 10 veces el Algoritmo I sobre cada uno de problemas test de optimización con restricciones $G1, \dots, G11$ de Koziel y Michalewicz [Koz99, Mic96b] utilizando los parámetros detallados en la tabla 3.19. Los resultados obtenidos se muestran en la tabla 3.20.

En este caso no se ha ejecutado el Algoritmo II ni ENORA, puesto que la ventaja de estos algoritmos se encuentra precisamente en las técnicas de diversidad utilizadas, lo cual se demuestra con problemas multiobjetivo; sin embargo, los problemas $G1, \dots, G11$ tienen un sólo objetivo, por lo que no es necesario utilizar dichas técnicas. Por tanto, se reserva la ejecución de estos algoritmos para problemas multiobjetivo donde pueda demostrarse su eficacia.

Tamaño de la población	$N = 200$
Probabilidad de cruce	$pCross = 0.9$
Probabilidad de mutación	$pMutate = 0.2$
Probabilidad de cruce uniforme	$pUniformCross = 0.6$
Probabilidad de mutación uniforme	$pUniformMutate = 0.1$
Probabilidad de mutación no uniforme	$pNotUniformMutate = 0.4$
Parámetro c para la mutación no uniforme	$c = 2.0$
Número de hijos para la preselección	$nChildren = 10$

Tabla 3.19: Parámetros en la ejecución del algoritmo.

<i>Problema</i>	$f_{\text{óptimo}}$	f_{peor}	f_{mejor}	f_{media}	f_{Koz}	f_{Mic}
<i>G1</i>	-15.0000	-15.0000	-15.0000	-15.0000	-14.7864	-15.0000
<i>G2</i>	0.803619	0.792567	0.803556	0.800588	0.799530	0.803553
<i>G3</i>	1.0000	1.0000	1.0000	1.0000	0.9997	0.9999
<i>G4</i>	-30665.5	-30665.5	-30665.5	-30665.5	-30664.5	-30005.7
<i>G5</i>	4221.96	5528.65	4256.78	4615.80	—	5126.67
<i>G6</i>	-6961.8	-6961.8	-6961.8	-6961.8	-6952.1	-6955.0
<i>G7</i>	24.306	25.015	24.360	24.632	24.620	24.690
<i>G8</i>	0.095825	0.095825	0.095825	0.095825	0.095825	0.095825
<i>G9</i>	680.630	680.680	680.640	680.650	680.910	680.642
<i>G10</i>	7049.33	7313.35	7051.86	7131.59	7147.90	7286.65
<i>G11</i>	0.75	0.75	0.75	0.75	0.75	0.75

Tabla 3.20: Resultados de la simulación para los problemas test de optimización con restricciones $G1, \dots, G11$ de Koziel y Michalewicz [Koz99, Mic96b]. $f_{\text{óptimo}}$ es el valor óptimo para cada problema; f_{peor} , f_{mejor} y f_{media} son los valores peor, mejor y medio obtenidos en 10 ejecuciones del algoritmo; f_{Koz} y f_{Mic} son los valores mejores obtenidos por otros algoritmos evolutivos en [Koz99, Mic96b].

Estudio de la influencia del número de hijos

Con objeto de estudiar la influencia del número de hijos en la preselección se han realizado ejecuciones del problema *G2* variando el parámetro *nChildren* y estableciendo un número constante de evaluaciones, *nEvaluaciones*, por lo que el número de iteraciones *T* es inversamente proporcional al número de hijos según muestra la siguiente ecuación:

$$T = \frac{nEvaluaciones - N}{2 \times nChildren}$$

Se han realizado ejecuciones variando *nChildren* entre 1 y 10 y estableciendo el número de evaluaciones a 20000 y a 200000. El resto de parámetros son los mismos establecidos en la tabla 3.19. Los resultados obtenidos se muestran en las tablas 3.21 y 3.22 y en las figuras 3.11 y 3.12. Se puede observar que, cuando el número de evaluaciones es pequeño, se obtienen mejores resultados cuanto menor es el número de hijos, mientras que al aumentar el número de iteraciones, resulta más conveniente, en término medio, aumentar el número de hijos. Es decir, el número de hijos más adecuado depende del número de evaluaciones que se efectúen. Por otra parte, se observa que nuestro algoritmo no obtiene buenos resultados cuando se limita el número de evaluaciones. Sin embargo, tal como se muestra en la tabla 3.20, si se deja evolucionar durante un número suficiente de evaluaciones, es capaz de encontrar resultados mejores que las de otros algoritmos que utilizan técnicas específicas [Koz99, Mic96b].

$nChildren$	f_{peor}	f_{mejor}	f_{media}
1	0.702102	0.748440	0.721439
2	0.663346	0.736410	0.699065
3	0.657423	0.717895	0.685849
4	0.633961	0.734074	0.676725
5	0.592751	0.675889	0.642479
6	0.617926	0.695129	0.653602
7	0.595595	0.661723	0.626175
8	0.593677	0.665244	0.626665
9	0.588121	0.639910	0.612252
10	0.520074	0.615208	0.590400

Tabla 3.21: Valores obtenidos con 20000 evaluaciones para el problema $G2$ variando el número de hijos en la preselección.

$nChildren$	f_{peor}	f_{mejor}	f_{media}
1	0.792384	0.803238	0.800021
2	0.797624	0.803194	0.801148
3	0.794222	0.803295	0.801818
4	0.797808	0.803228	0.802015
5	0.792127	0.803219	0.801160
6	0.797739	0.803254	0.801608
7	0.802782	0.803322	0.803114
8	0.791695	0.803309	0.800321
9	0.797669	0.803073	0.802186
10	0.797673	0.803148	0.801365

Tabla 3.22: Valores obtenidos con 200000 evaluaciones para el problema $G2$ variando el número de hijos en la preselección.

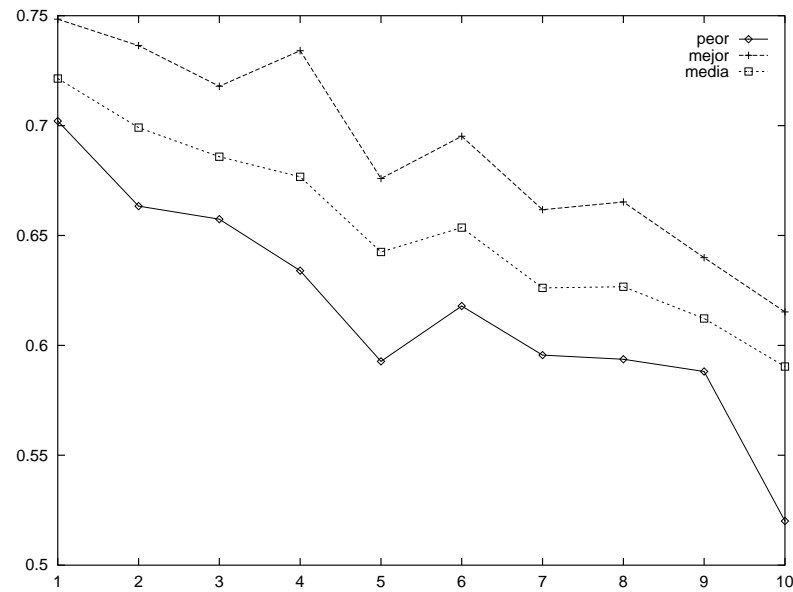


Figura 3.11: Valores obtenidos con 20000 evaluaciones para el problema $G2$ variando el número de hijos en la preselección.

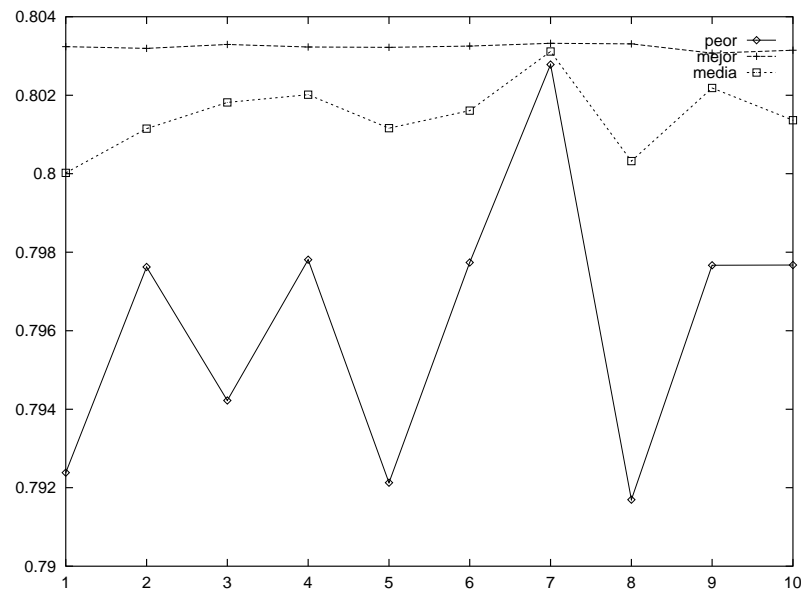


Figura 3.12: Valores obtenidos con 200000 evaluaciones para el problema $G2$ variando el número de hijos en la preselección.

Comportamiento con un número limitado de evaluaciones

Otro aspecto importante consiste en probar el funcionamiento de nuestro algoritmo cuando el número de evaluaciones de la función objetivo se encuentra limitado. Un marco idóneo para realizar esta comprobación lo encontramos en el concurso de computación evolutiva del congreso MAEB'03¹, pues las bases de este concurso establecen que el número máximo de evaluaciones es de 20000 para $p = 20$ y de 50000 para $p = 50$, es decir el número de evaluaciones está limitado a un valor pequeño. Para participar en el concurso se ha tomado como base el Algoritmo I. Puesto que el número de evaluaciones es pequeño, se establece el valor de $nChildren$ como 1. El resto de parámetros se ajustan específicamente para el problema $G2$, utilizando los valores mostrados en la tabla 3.23. Por otra parte, para mejorar los resultados obtenidos, se incluyen un par de heurísticas específicas del problema $G2$, por lo que se ejecuta el Algoritmo I con las siguientes modificaciones:

- Se reemplaza el cruce aritmético por el cruce geométrico descrito por Michalewicz y Schoenauer [Mic96b], que opera de la forma descrita a continuación: dados dos individuos $I_1 = \{x_1^1, \dots, x_p^1\}$ e $I_2 = \{x_1^2, \dots, x_p^2\}$, se obtienen otros dos individuos $I_3 = \{x_1^3, \dots, x_p^3\}$ e $I_4 = \{x_1^4, \dots, x_p^4\}$ donde:

$$x_i^3 = (x_i^1)^d (x_i^2)^{1-d}, \quad x_i^4 = (x_i^1)^{1-d} (x_i^2)^d, \quad i = 1, \dots, p$$

siendo d un valor aleatorio entre 0 y 1.

- Se añade un algoritmo de reparo *ad hoc* que realiza la siguiente modificación en un individuo:

$$x_i = \frac{0.75}{\prod_{j=1}^{i-1} x_j \prod_{j=i+1}^p x_j} + \epsilon$$

donde i es un número aleatorio entre 1 y p y $\epsilon > 0$ es un valor cercano a cero, en nuestro caso se establece $\epsilon = 1E - 10$. Este algoritmo de reparo se efectúa al inicializar cada individuo y después de obtener nuevos individuos mediante cruce y mutación, lo que permite que el algoritmo realice la búsqueda en el límite de la región factible, que es precisamente donde se encuentran los valores óptimos, obteniendo valores cercanos al óptimo con un número pequeño de evaluaciones.

¹Segundo Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados.

Tamaño de la población	$N = 200$
Probabilidad de cruce	$pCross = 1.0$
Probabilidad de mutación	$pMutate = 0.01$
Probabilidad de cruce uniforme	$pUniformCross = 0.8$
Probabilidad de mutación uniforme	$pUniformMutate = 0.1$
Probabilidad de mutación no uniforme	$pNotUniformMutate = 0.9$
Parámetro c para la mutación no uniforme	$c = 2.0$
Número de hijos para la preselección	$nChildren = 1$

Tabla 3.23: Parámetros en la ejecución del algoritmo para el problema $G2$.

El Algoritmo I modificado para $G2$ encuentra, para $p = 20$, un óptimo igual a 0.803595, y para $p = 50$ un óptimo igual a 0.834363, que son los valores con los que se ha participado en el concurso de computación evolutiva del MAEB'03.

Comparación de las diferentes técnicas de diversidad

Para comparar los resultados obtenidos se ejecuta un número de 5 veces cada uno de nuestros algoritmos evolutivos sobre el problema OSY propuesto por Osyczka y Kundy [Osy95] detallado en la tabla 3.15 y nos quedamos con los resultados de la mejor ejecución. En todos los casos, la inspección visual de las soluciones no dominadas obtenidas resulta tan clara que no se ha considerado utilizar medidas de dispersión como las propuestas por Deb [Deb01c], sino que simplemente se ha realizado la comparación de forma visual.

En todas las ejecuciones se utilizan los parámetros de la tabla 3.24.

Los resultados obtenidos utilizando el Algoritmo I, que utiliza preselección simple, se muestran en la figura 3.13. Se puede observar que esta técnica no garantiza un grado suficiente de dispersión para cubrir todo el frente Pareto. Para resolver esto se plantean las técnicas de diversidad explícita.

La figura 3.14 muestra los resultados obtenidos con el Algoritmo II, que utiliza métricas de diversidad. Con esta técnica de diversidad se ha mejorado sustancialmente

Tamaño de la población	$N = 100$
Probabilidad de cruce	$pCross = 0.6$
Probabilidad de mutación	$pMutate = 0.6$
Probabilidad de cruce uniforme	$pUniformCross = 0.5$
Probabilidad de mutación uniforme	$pUniformMutate = 0.2$
Probabilidad de mutación no uniforme	$pNotUniformMutate = 0.5$
Parámetro c para la mutación no uniforme	$c = 2.0$
Número de hijos para la preselección	$nChildren = 20$

Tabla 3.24: Parámetros en la ejecución del algoritmo.

el resultado obtenido inicialmente con la preselección simple. Sin embargo, todavía quedan zonas del frente Pareto óptimo poco cubiertas, por lo que la diversidad aún puede mejorarse.

El último algoritmo, ENORA, que realiza la partición del espacio de búsqueda en tramos radiales, es el que mejores resultados obtiene en cuanto diversidad, como pueden verse en la figura 3.15, distribuyendo las soluciones obtenidas por todo el frente Pareto óptimo.

Podemos observar que el Algoritmo I no obtiene una diversidad adecuada cuando el problema presenta cierto grado de dificultad. Por tanto, para el resto de problemas test se realizarán ejecuciones solamente utilizando los otros dos algoritmos que implementan técnicas de diversidad explícita:

- *Algoritmo II*. Utilización de métricas de diversidad.
- *ENORA*. Partición del espacio de búsqueda en tramos radiales.

Como puede verse en las figuras 3.16, 3.17, 3.18 y 3.19, en los problemas test SRN y TNK los resultados obtenidos en todos los casos son buenos en cuanto a dispersión y cercanía al frente Pareto óptimo; no se aprecian grandes diferencias en cuanto a los algoritmos utilizados, ya que ambos algoritmos consiguen buenos resultados.

La figura 3.20 muestra que en el problema CTP1 el Algoritmo II no consigue encontrar soluciones dispersas por todo el frente Pareto óptimo y tiene problemas

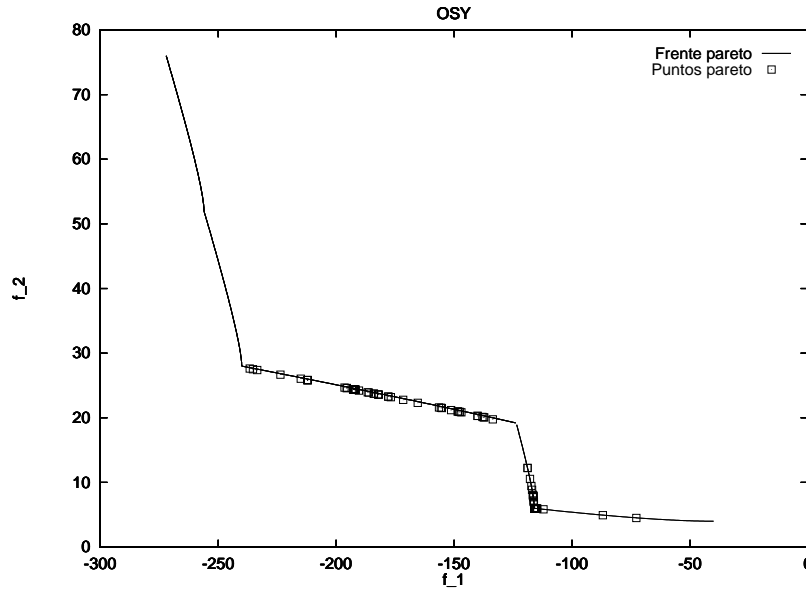


Figura 3.13: Soluciones no dominadas obtenidas con el Algoritmo I para el problema test OSY.

para encontrar soluciones que se encuentren en las zonas del frente Pareto óptimo definidas por las restricciones; sin embargo, tal como se observa en la figura 3.21, ENORA consigue soluciones dispersas por todo el frente Pareto óptimo.

En las figuras 3.22 y 3.23 puede observarse que en el problema CTP2 nuevamente el Algoritmo II no consigue encontrar soluciones dispersas por todo el frente Pareto óptimo, mientras que ENORA sí las encuentra.

Para el problema CTP3 ambos algoritmos obtienen buenos resultados, como muestran las figuras 3.24 y 3.25.

En la figura 3.26 se observa que para el problema CTP4, el Algoritmo II obtiene buena diversidad, pero no consigue que las soluciones se acerquen suficientemente al frente Pareto óptimo; en la figura 3.27 puede verse cómo ENORA de nuevo obtiene soluciones distribuidas en todo el frente Pareto óptimo.

En el problema CTP5, tal como muestran las figuras 3.28 y 3.29, los resultados de ambos algoritmos son buenos, aunque el Algoritmo II no consigue encontrar soluciones en todos los tramos del frente Pareto óptimo, mientras que ENORA sí consigue soluciones en todos los tramos del frente Pareto óptimo.

En las figuras 3.30 y 3.31 para el problema CTP6 puede observarse cómo ambas técnicas consiguen soluciones en el frente Pareto óptimo, pero el Algoritmo II no consigue distribuir las soluciones por todo el frente, mientras que ENORA sí consigue una buena distribución.

Por último, en las figuras 3.32 y 3.33 se observa que ambos algoritmos obtienen buenos resultados para CTP7, aunque una vez más ENORA obtiene una dispersión mejor que el Algoritmo II.

Se puede concluir por tanto que la técnica utilizada por ENORA es la más adecuada para conseguir una buena dispersión, es decir, la partición del espacio de búsqueda en tramos radiales. Por tanto, son los resultados obtenidos con ENORA los que utilizamos para comparar con otros algoritmos.

Comparación con otros algoritmos

En este apartado vamos a comparar los resultados obtenidos con ENORA con los resultados obtenidos por otros algoritmos, en concreto, comparamos con los resultados presentados por Deb et al. para los algoritmos NSGA-II y Ray et al. Nuevamente la inspección visual resulta suficientemente clara para realizar la comparación, por lo que no ha sido necesario utilizar medidas de diversidad.

La figura 3.34 muestra que para el problema test OSY, los resultados obtenidos por Deb et al. con el algoritmo de Ray et al. no consiguen una buena diversidad, mientras que NSGA-II sí consigue cubrir todo el frente Pareto, aunque en algunos tramos la dispersión es menor que en otros. Tal como se observa en la figura 3.15 ENORA consigue resultados con buena dispersión en todos los tramos no dominados.

Como puede verse en la figura 3.21, en el problema CTP1, ENORA es capaz de encontrar puntos Pareto distribuidos uniformemente por todo el frente Pareto. Deb et al. no muestran resultados para este problema.

Para el problema CTP2, según los resultados mostrados por Deb et al. en la figura 3.35, NSGA-II encuentra puntos en todas las regiones factibles del frente Pareto óptimo, pero el algoritmo propuesto por Ray et al. no es capaz de converger adecuadamente a todas las regiones. Como muestra la figura 3.23, ENORA es capaz de converger adecuadamente y encontrar Pareto en todas las regiones factibles del frente Pareto.

En el problema CTP3, tal como se observa en la figura 3.35, el algoritmo de Ray

et al. ni siquiera puede converger cerca de las soluciones Pareto óptimas, mientras que NSGA-II sí es capaz de encontrar las soluciones Pareto óptima en cada región. Tal como puede verse en la figura 3.25 ENORA también es capaz en este caso de converger adecuadamente a los puntos Pareto óptimos en cada región.

El problema CTP4 causa dificultades tanto al algoritmo de Ray et al. como a NSGA-II cuyos resultados quedaron bastante lejos de las soluciones Pareto óptimas, como puede verse en la figura 3.36. Sin embargo, en este problema ENORA es capaz de encontrar dichas soluciones como se muestra en la figura 3.27.

Como puede observarse en la figura 3.36, la no uniformidad planteada por CTP5 no resulta ser una gran dificultad para NSGA-II que encuentra los puntos Pareto en este caso, sin embargo el algoritmo de Ray et al. vuelve a tener dificultades para converger. ENORA, de nuevo, muestra un buen comportamiento en este caso, según puede verse en la figura 3.29.

Como muestra la figura 3.37, en el problema CTP6, tanto el algoritmo de Ray et al. como NSGA-II convergen adecuadamente al frente Pareto, si bien NSGA-II mantiene una mejor diversidad de soluciones. Tal como puede verse en las figuras 3.30 y 3.31 también ENORA es capaz de converger adecuadamente en este problema y mantener una buena diversidad de soluciones.

Por ultimo, como muestra la figura 3.37, el problema CTP7 es el que causa mayores dificultades tanto al algoritmo de Ray et al. como a NSGA-II. Ninguno de dichos algoritmos encuentra soluciones cerca del frente Pareto óptimo. El algoritmo de Ray et al. mantiene una mejor diversidad de las soluciones, sin embargo NSGA-II es capaz de llegar más cerca del frente Pareto. Este es un problema donde un algoritmo debe ser capaz de mantener una buena diversidad de soluciones desde el principio y, además, ser capaz de converger adecuadamente. También en este caso, tal y como vemos en la figura 3.33, ENORA es capaz de converger a soluciones óptimas en todas las regiones factibles del frente Pareto.

Se puede concluir que el algoritmo ENORA es capaz de obtener soluciones no dominadas dispersas por todo el frente Pareto en problemas donde esto resulta difícil, incluso, para algoritmos evolutivos multiobjetivo de última generación como NSGA-II, cuyo buen funcionamiento ha sido ampliamente comprobado en la literatura [Deb01c].

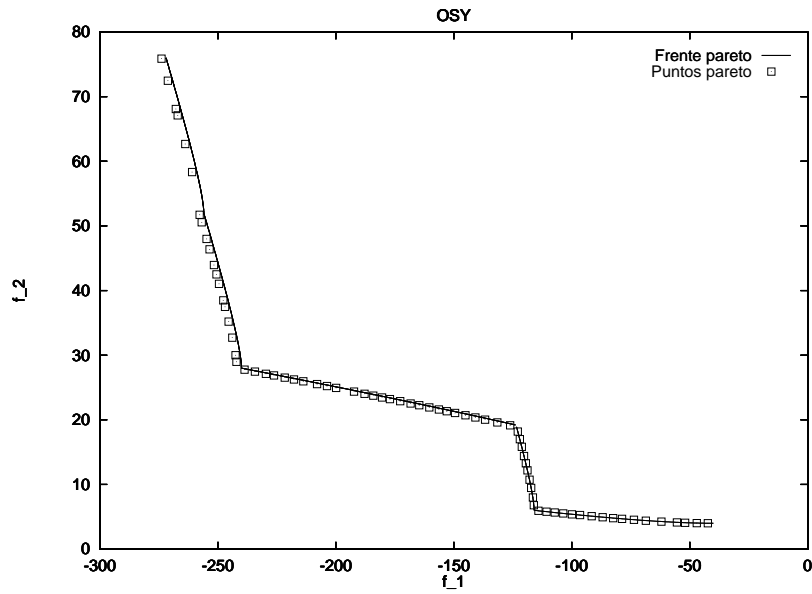


Figura 3.14: Soluciones no dominadas obtenidas con el Algoritmo II para el problema OSY.

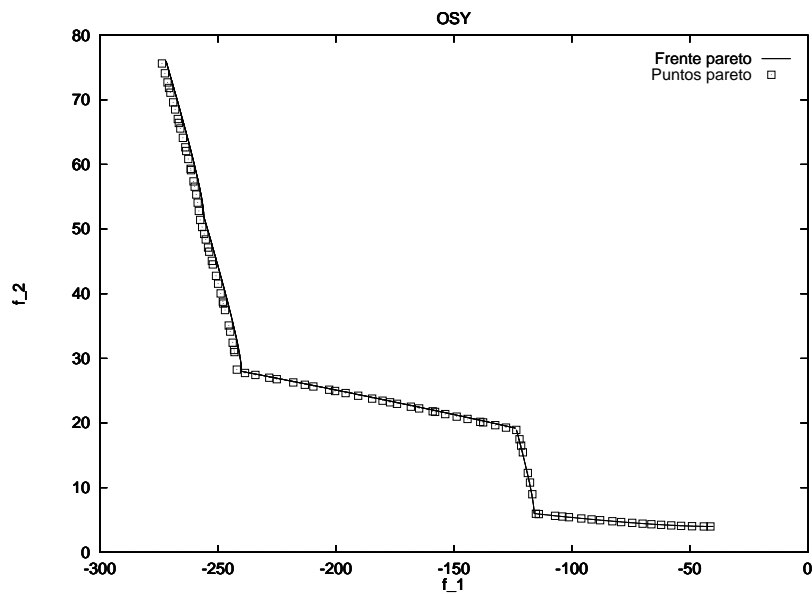


Figura 3.15: Soluciones no dominadas obtenidas con ENORA para el problema OSY.

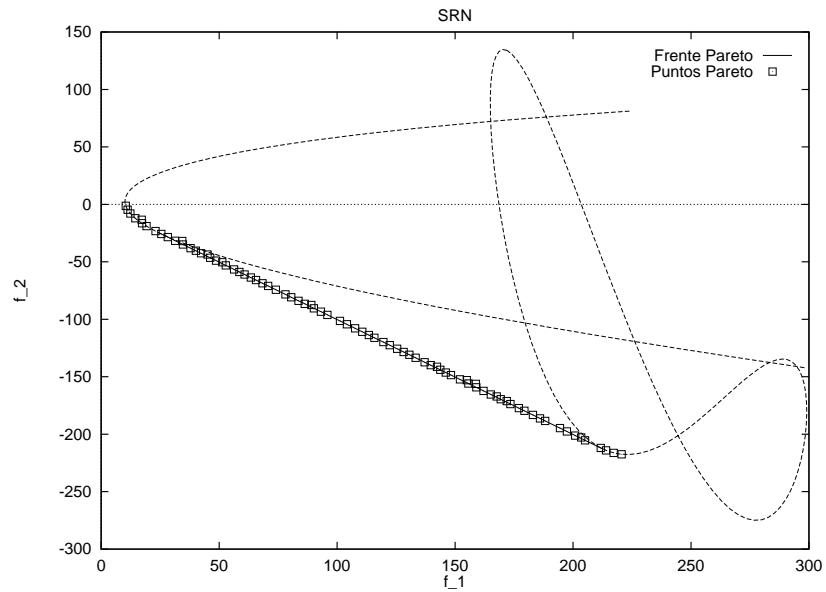


Figura 3.16: Soluciones no dominadas obtenidas con el Algoritmo II para el problema SRN.

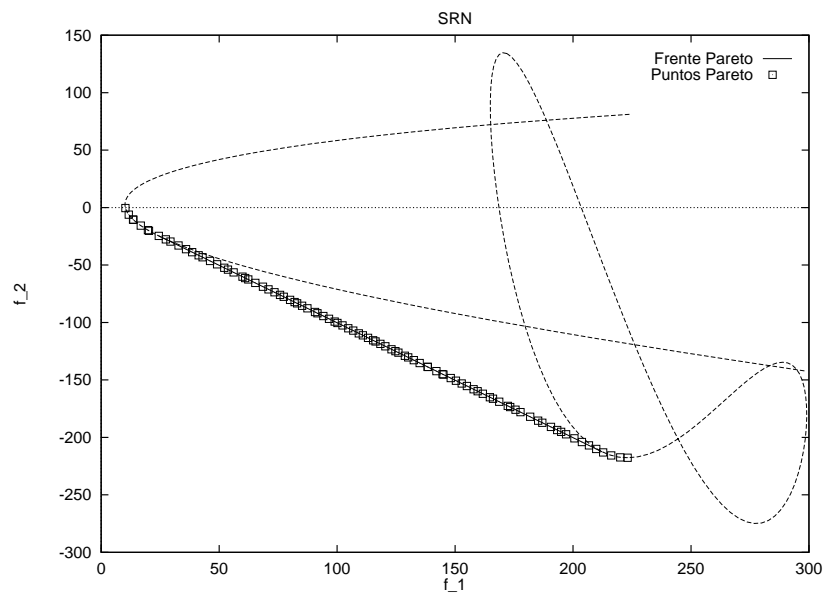


Figura 3.17: Soluciones no dominadas obtenidas con ENORA para el problema SRN.

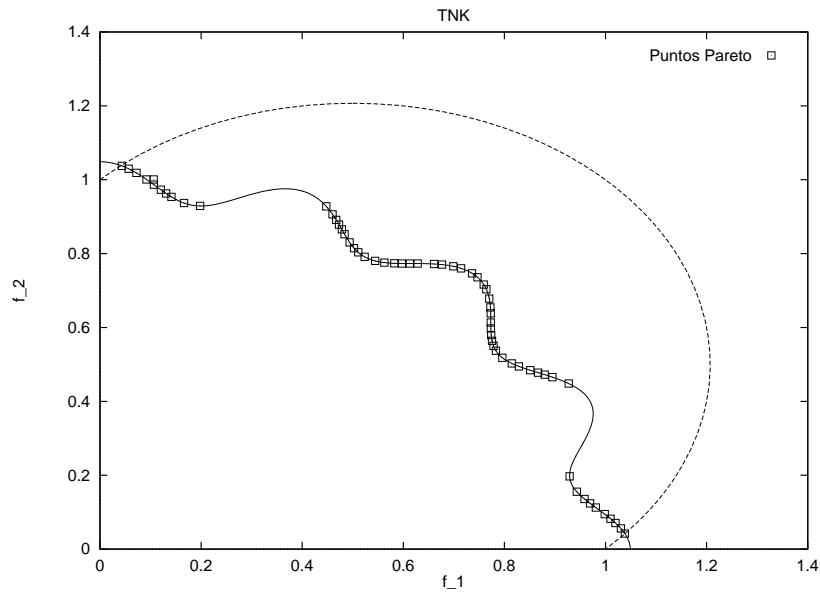


Figura 3.18: Soluciones no dominadas obtenidas con el Algoritmo II para el problema TNK.

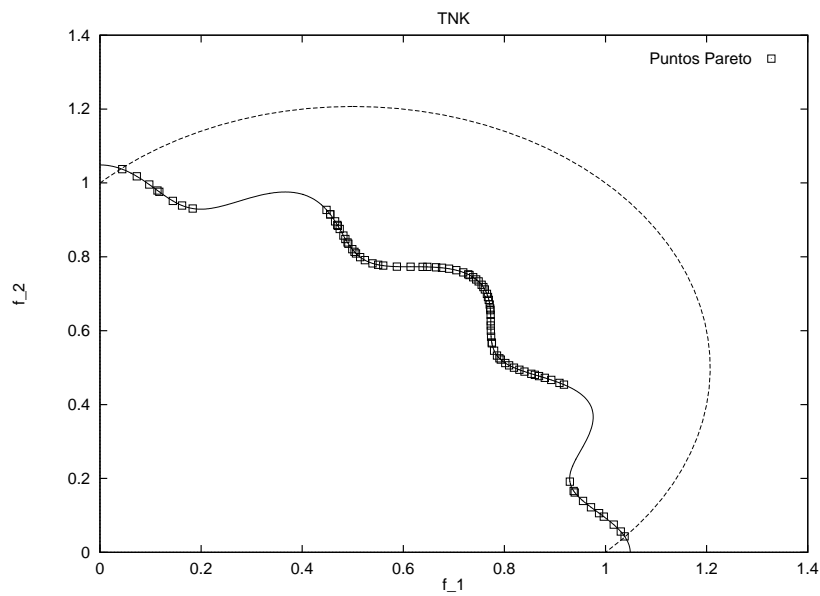


Figura 3.19: Soluciones no dominadas obtenidas con ENORA para el problema TNK.

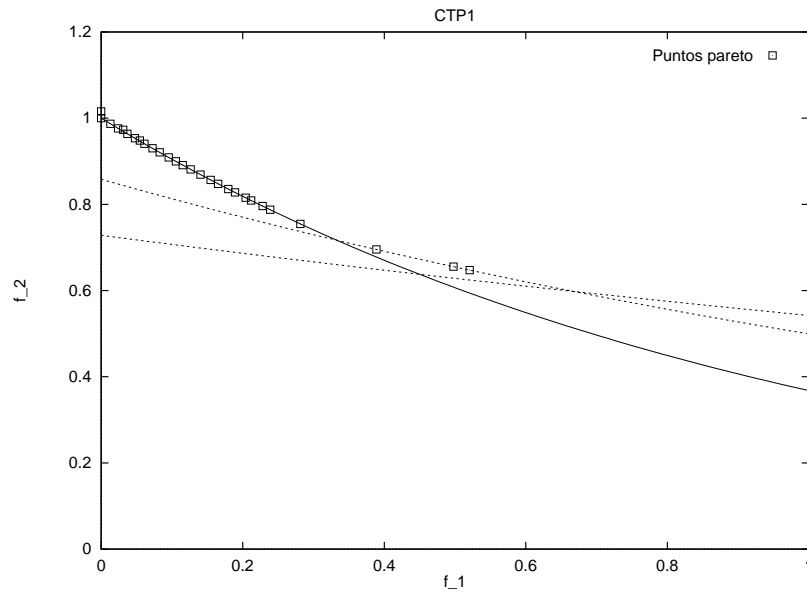


Figura 3.20: Soluciones no dominadas obtenidas con el Algoritmo II para el problema CTP1.

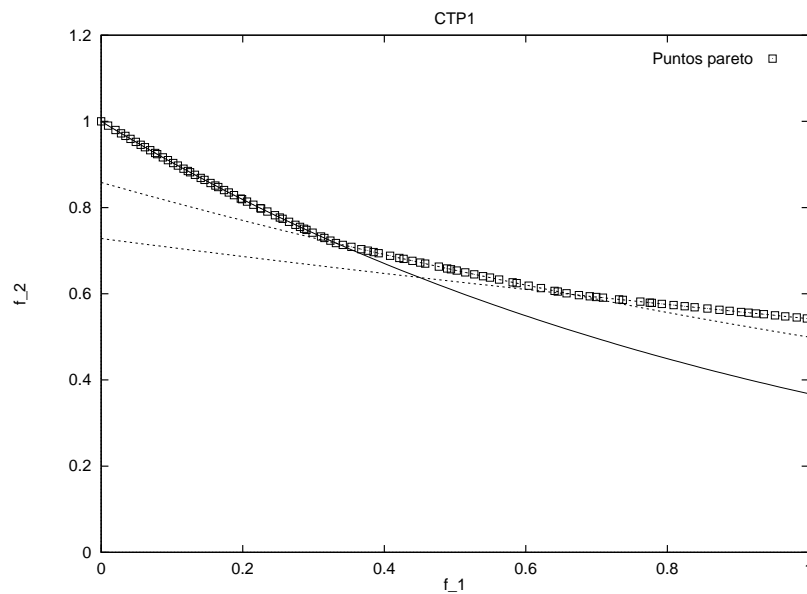


Figura 3.21: Soluciones no dominadas obtenidas con ENORA para el problema CTP1.

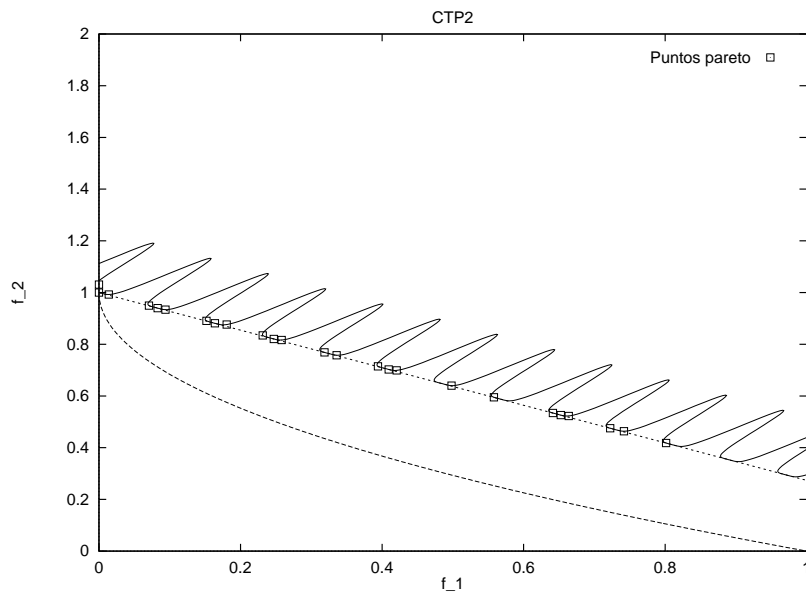


Figura 3.22: Soluciones no dominadas obtenidas con el Algoritmo II para el problema CTP2.

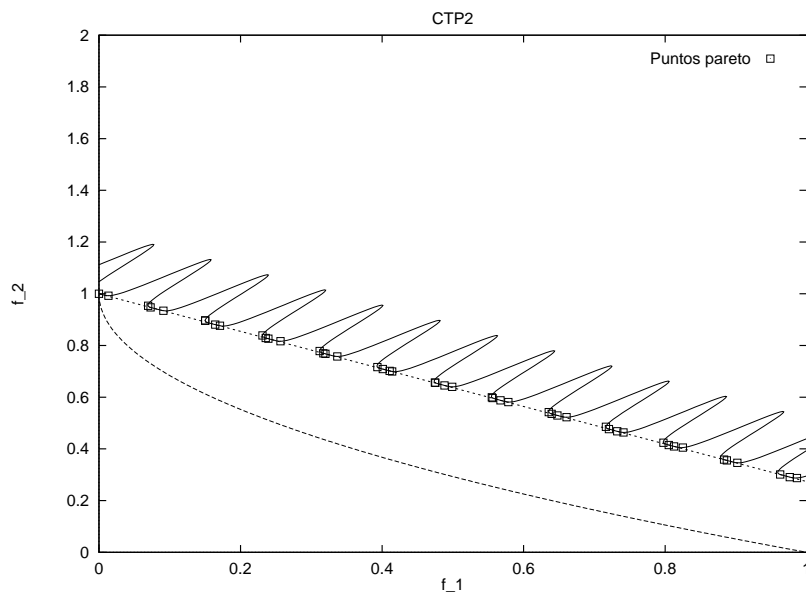


Figura 3.23: Soluciones no dominadas obtenidas con ENORA para el problema CTP2.

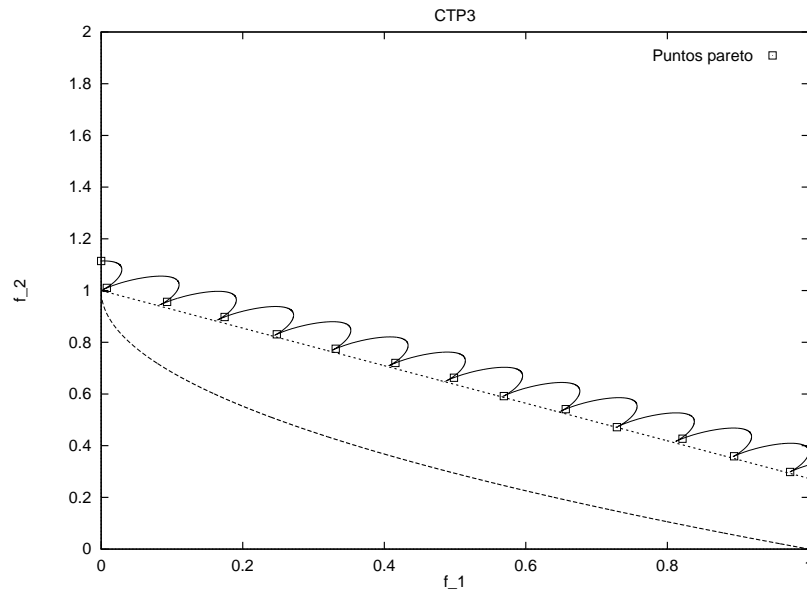


Figura 3.24: Soluciones no dominadas obtenidas con el Algoritmo II para el problema CTP3.

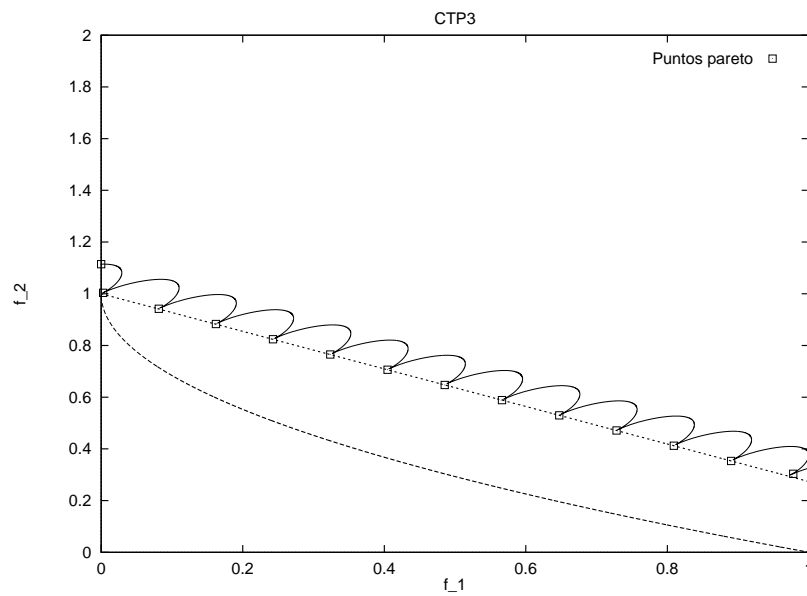


Figura 3.25: Soluciones no dominadas obtenidas con ENORA para el problema CTP3.

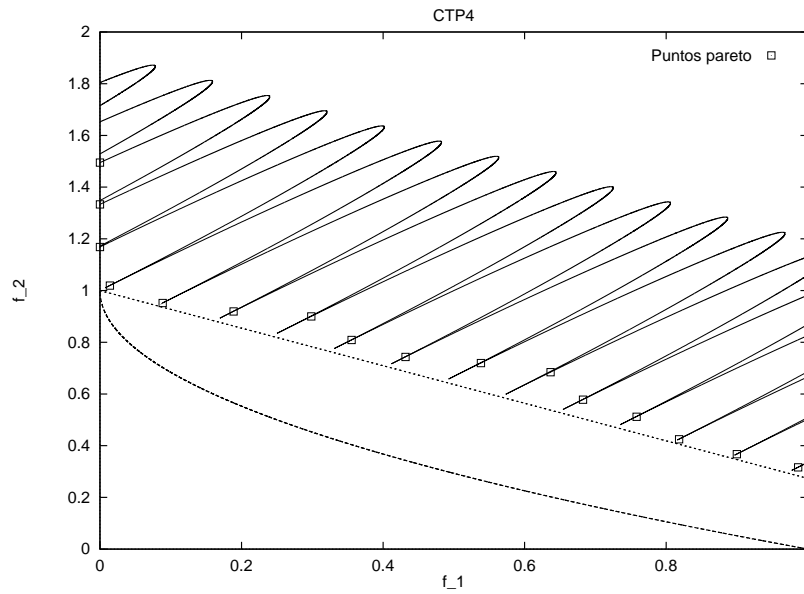


Figura 3.26: Soluciones no dominadas obtenidas con el Algoritmo II para el problema CTP4.

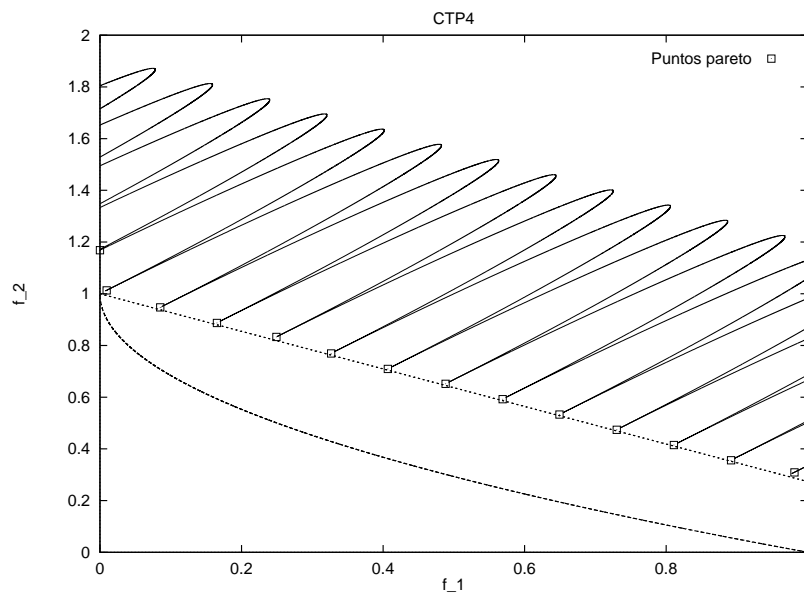


Figura 3.27: Soluciones no dominadas obtenidas con ENORA para el problema CTP4.

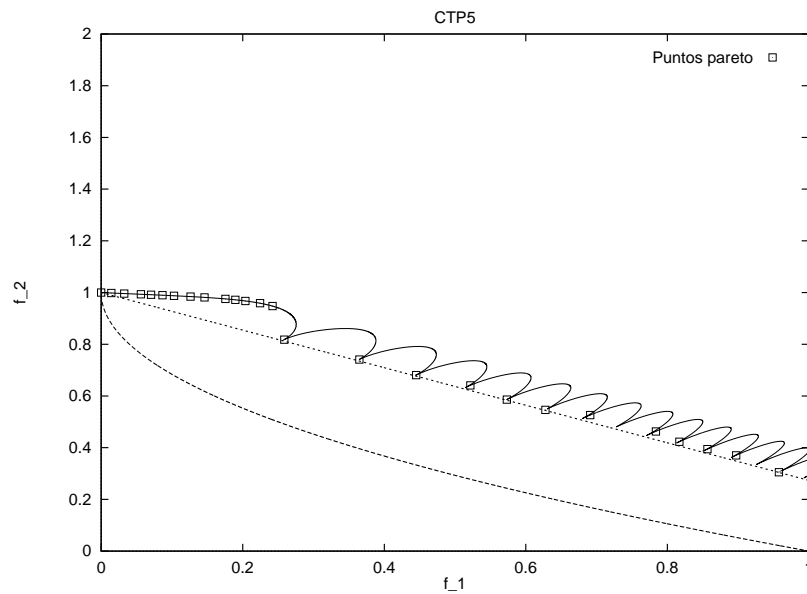


Figura 3.28: Soluciones no dominadas obtenidas con el Algoritmo II para el problema CTP5.

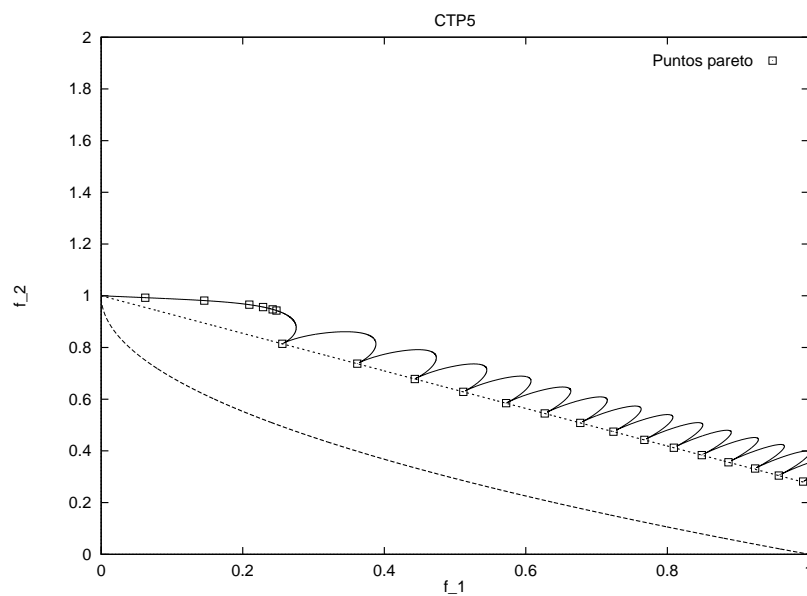


Figura 3.29: Soluciones no dominadas obtenidas con ENORA para el problema CTP5.

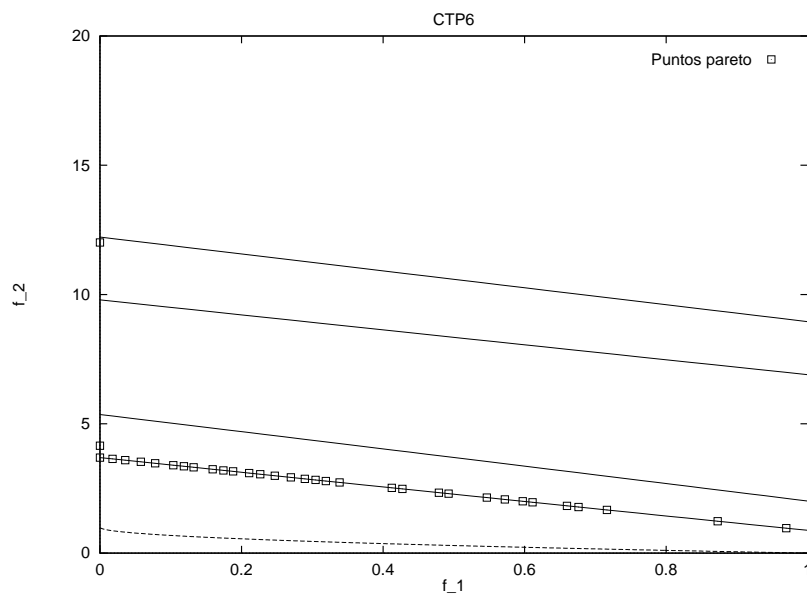


Figura 3.30: Soluciones no dominadas obtenidas con el Algoritmo II para el problema CTP6.

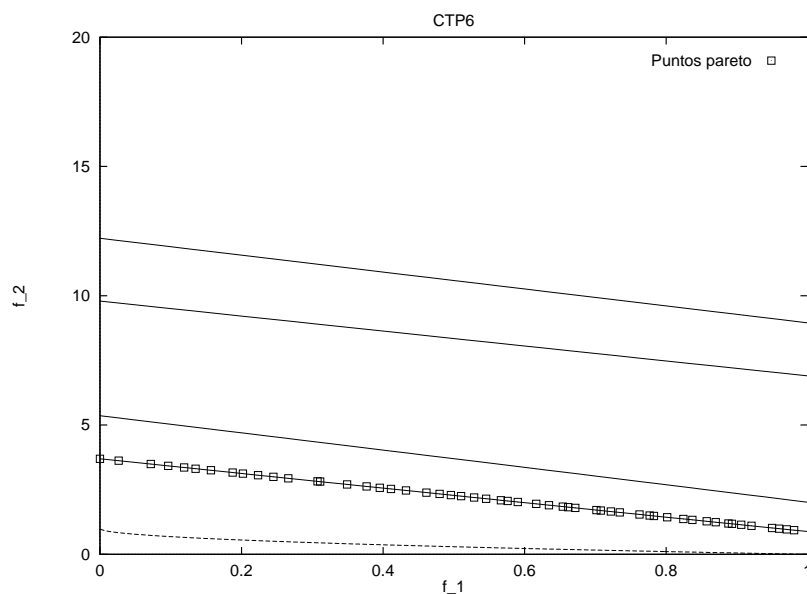


Figura 3.31: Soluciones no dominadas obtenidas con ENORA para el problema CTP6.

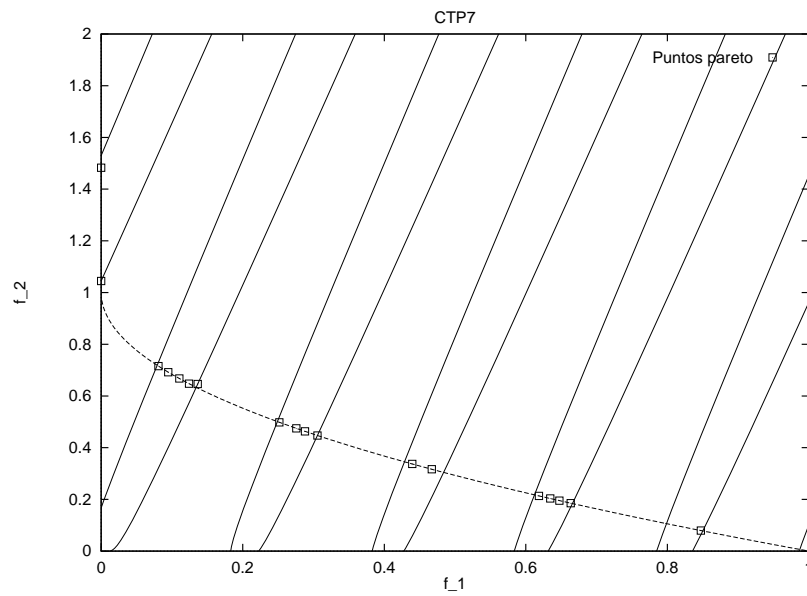


Figura 3.32: Soluciones no dominadas obtenidas con el Algoritmo II para el problema CTP7.

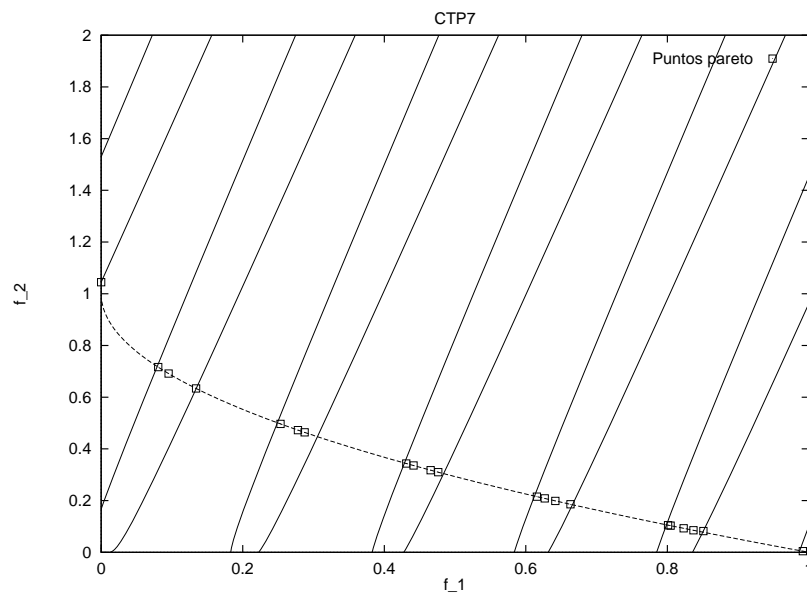


Figura 3.33: Soluciones no dominadas obtenidas con ENORA para el problema CTP7.

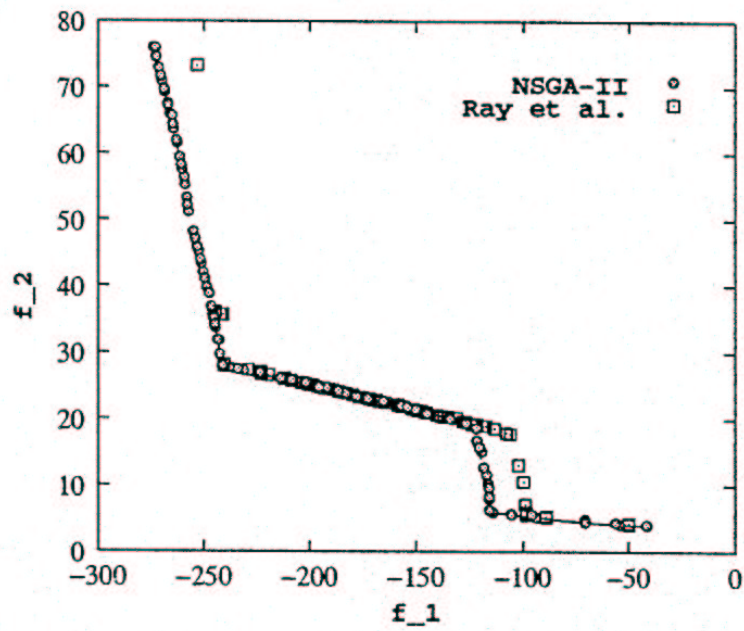


Figura 3.34: Soluciones no dominadas obtenidas con el algoritmo de Ray et al. y con NSGA-II para el problema OSY.

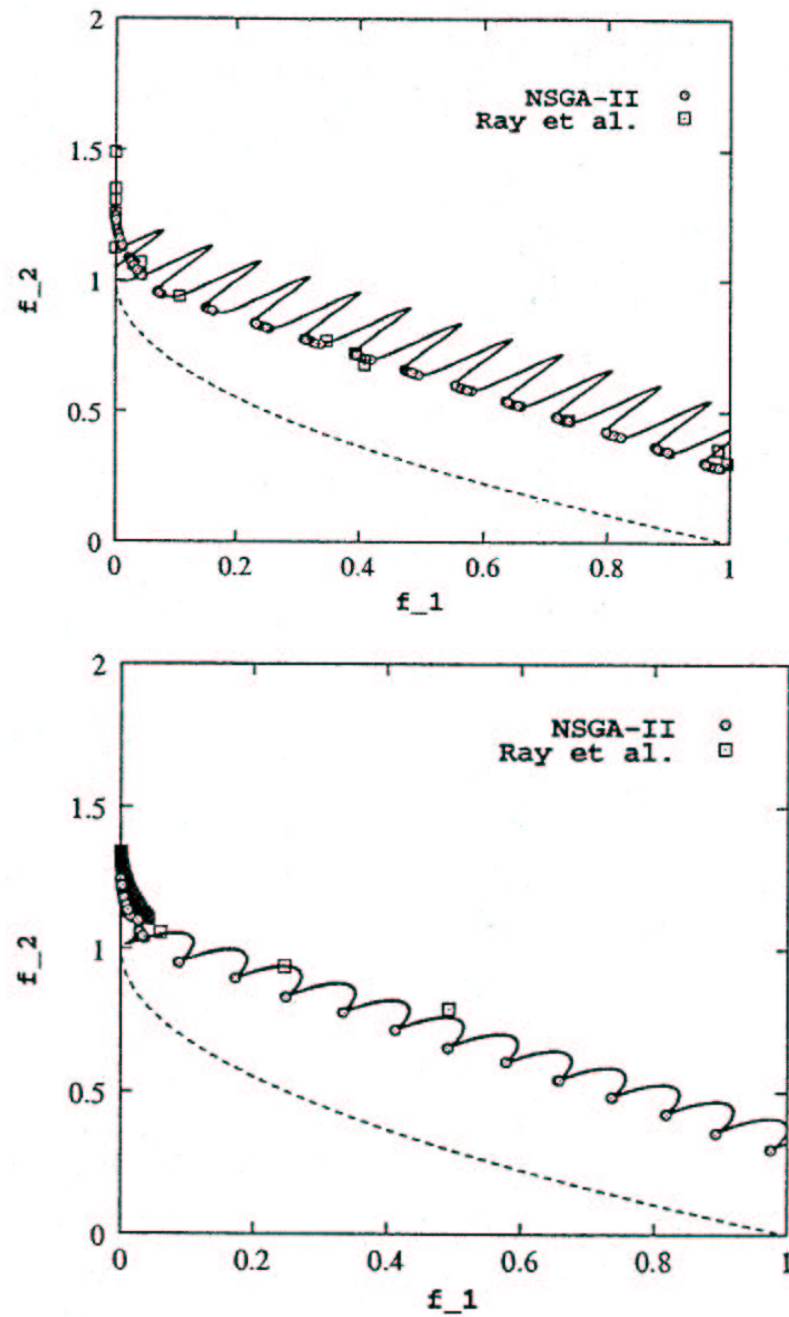


Figura 3.35: Soluciones no dominadas obtenidas con el algoritmo de Ray et al. y con NSGA-II para el problema CTP2 (arriba) y para el problema CTP3 (abajo).

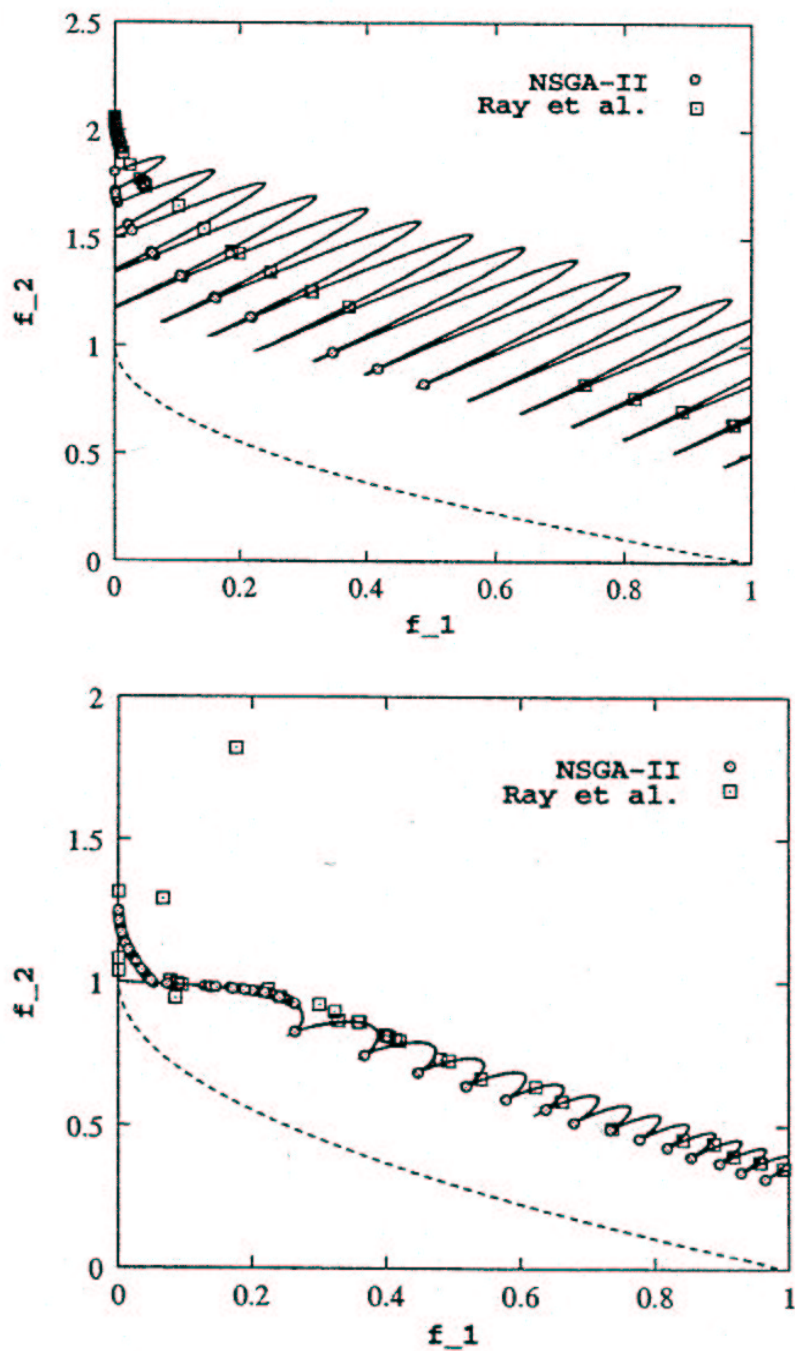


Figura 3.36: Soluciones no dominadas obtenidas con el algoritmo de Ray et al. y con NSGA-II para el problema CTP4 (arriba) y para el problema CTP5 (abajo).

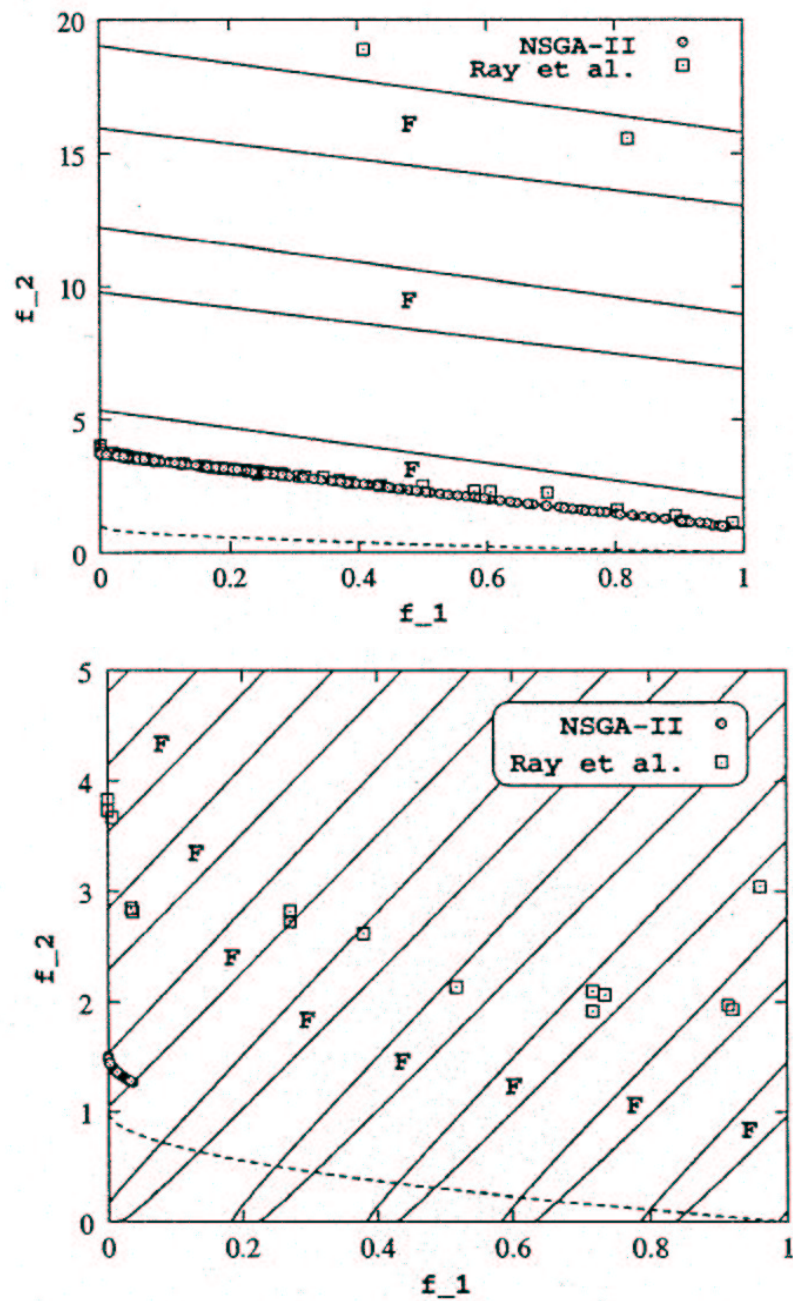


Figura 3.37: Soluciones no dominadas obtenidas con el algoritmo de Ray et al. y con NSGA-II para el problema CTP6 (arriba) y para el problema CTP7 (abajo).

3.3 Sumario

En este capítulo se han planteado nuevos algoritmos evolutivos multiobjetivo basados en el concepto de dominación Pareto.

En una primera sección se plantea un algoritmo evolutivo para resolver problemas de optimización multiobjetivo sin restricciones. Se ha utilizado este algoritmo también para resolver problemas multiobjetivo con restricciones, utilizando la transformación propuesta por Jiménez et al. [Jim02]. Este algoritmo es el más sencillo de los propuestos y sirve como base para iniciar la investigación.

En la segunda sección se plantean tres nuevos algoritmos evolutivos para resolver problemas de optimización multiobjetivo con restricciones, utilizando técnicas propias para manejo de restricciones. Cada uno de estos algoritmos utiliza diferentes técnicas para mantener la diversidad, desde la técnica más sencilla utilizada por el primer algoritmo, hasta la más desarrollada que utiliza el último algoritmo propuesto. El Algoritmo I utiliza preselección simple, que es la técnica más sencilla y sirve como punto de partida. El algoritmo II utiliza métricas de diversidad. El último algoritmo, ENORA, realiza una partición del espacio de búsqueda en tramos radiales, que es una técnica *ad hoc* para mantener la diversidad. Se prueban cada uno de estos algoritmos para diversos problemas test. El Algoritmo I no consigue obtener buenos resultados en cuanto a diversidad, sin embargo el Algoritmo II mejora la diversidad y por último ENORA obtienen los resultados más dispersos. De los experimentos realizados se puede concluir que ENORA es, por tanto, un algoritmo evolutivo multiobjetivo capaz de obtener soluciones no dominadas dispersas por todo el frente Pareto.

Capítulo 4

Modelización Difusa

La teoría de conjuntos difusos tiene su origen a mediados de los 60, desarrollada por el profesor Lotfi A. Zadeh [Zad65] en la universidad de California. Existe un gran número de investigadores que desde un principio han reconocido el potencial de los conjuntos difusos para modelizar y resolver numerosos problemas del mundo real. Sin embargo, también han existido otros investigadores que en los primeros años han cuestionado la validez de esta técnica. La situación cambia desde mediados de los 80, cuando ocurre el, así llamado, “boom difuso”, principalmente en Japón, pero también en Corea y en Europa. Básicamente, el cambio consiste en el lanzamiento al mercado de aplicaciones de control basadas en lógica difusa desarrolladas por Mamdani, Sugeno, Takagi y otros autores [Tak85, Sug88].

La modelización difusa, es decir, la descripción mediante modelos difusos de sistemas del mundo real, se convierte por tanto en una técnica fundamental, tanto para aplicaciones científicas, como de ingeniería. Existen diversas técnicas para realizar modelos difusos. Tradicionalmente se han utilizado métodos numéricos basados en gradiente, clustering, etc. También se pueden realizar modelos difusos utilizando técnicas evolutivas, que es la aproximación utilizada en este trabajo.

Este capítulo realiza una discusión sobre los métodos más importantes de modelización difusa. La sección 4.1 realiza una discusión sobre algunos de los aspectos básicos de modelización difusa. La sección 4.2 recoge algunas de las técnicas evolutivas de mayor interés para modelización difusa y la sección 4.3 resume los contenidos desarrollados a lo largo del capítulo.

4.1 Aspectos de Modelización Difusa

Los conceptos de la teoría de conjuntos difusos pueden emplearse para modelizar sistemas de diversas formas. Diversos aspectos de ingeniería y teoría de sistemas pueden llevarse al campo difuso dando como resultado distintas clases de modelos difusos.

Para realizar un modelo difuso existen dos fases fundamentales:

1. *Identificación de variables.*

En esta fase se identifican qué variables de entrada son significativas para calcular la salida del modelo. Esto permite descartar variables de entrada que no resultan significativas, por ser ruido o combinación de otras variables.

2. *Identificación de la estructura.*

Una vez se han identificado las variables de entrada al sistema, el siguiente paso consiste en identificar la estructura del propio modelo difuso. Dentro de esta fase se debe elegir el tipo de modelo difuso y los parámetros de dicho modelo.

4.1.1 Estructura del modelo difuso

Los modelos difusos más utilizados son los modelos difusos basados en reglas, es decir, sistemas en los que las relaciones entre variables se representan mediante reglas difusas del tipo *Si-entonces* junto con un mecanismo de inferencia difuso asociado.

Un sistema de inferencia difuso (*Fuzzy Inference System, FIS*) tiene, por tanto, un conjunto de reglas difusas Si-entonces que son, en general, de la forma:

Si *antecedente* **entonces** *consecuente*

donde *antecedente* y *consecuente* son proposiciones. Según la forma de los consecuentes y la estructura de la regla, se pueden distinguir dos tipos básicos de modelos difusos:

- *Modelo difuso lingüístico* o *Modelo difuso de Mamdani*, [Mam75] donde, tanto los antecedentes como los consecuentes, son proposiciones difusas.
- *Modelo difuso de Takagi-Sugeno-Kang* o *Modelo difuso TSK*, [Tak85, Sug88] donde los consecuentes son funciones no difusas.

La elección del tipo de modelo difuso depende de la aplicación concreta; un modelo aproximativo será más adecuado para modelización más precisa, mientras que un modelo lingüístico nos ofrecerá una descripción cualitativa del proceso. La interpretación de dichos modelos es también diferente: un modelo Takagi-Sugeno con consecuentes lineales puede interpretarse en términos de modelos locales lineales del sistema, mientras que un modelo lingüístico se interpreta utilizando lenguaje natural y términos cualitativos, que resulta más sencillo de interpretar, pero con los que se alcanza menor precisión para igualdad de número de reglas. Por tanto, podría ser útil desarrollar diferentes modelos para el mismo proceso, de forma que cada modelo sirva para un propósito diferente.

Modelo Difuso Lingüístico (Modelo Difuso de Mamdani)

El *sistema de inferencia difuso de Mamdani* [Mam75] es el primer intento para realizar el control de la presión dentro de un tanque mediante un conjunto de reglas de control lingüísticas obtenidas de operadores humanos con experiencia.

En un modelo difuso lingüístico (o modelo difuso de Mamdani), tenemos reglas en las cuales tanto los antecedentes como los consecuentes son proposiciones difusas, es decir tenemos un conjunto de M reglas de la forma:

$$R_i : \text{Si } x_1 \text{ es } A_{i1} \text{ y } \dots \text{ y } x_n \text{ es } A_{in} \text{ entonces } y \text{ es } B_i$$

donde:

$$i = 1, \dots, M,$$

$$\mathbf{x} = [x_1, \dots, x_n]^T \text{ es el vector de entrada,}$$

A_{ij} son los conjuntos difusos definidos en el espacio de los antecedentes por las funciones de pertenencia $\mu_{A_{ij}} : \mathcal{X}_j \rightarrow [0, 1]$, para $j = 1, \dots, n$

$$\mathcal{X}_j \text{ es el dominio de la variable de entrada } x_j, \text{ para } j = 1, \dots, n$$

B_i son los conjuntos difusos definidos en el espacio de los consecuentes con funciones de pertenencia $\mu_{B_i} : \mathcal{Y} \rightarrow [0, 1]$

$$\text{e } \mathcal{Y} \text{ es el dominio de la variable de salida } y.$$

El grado de disparo de la regla i -ésima se calcula aplicando la T-norma, donde un

caso particular puede ser calculado como el mínimo del disparo de cada variable:

$$\mu_i(\mathbf{x}) = \min_{j=1,\dots,n} \{\mu_{A_{ij}}(x_j)\}$$

La función de pertenencia del conjunto B_i queda, por tanto, modificada de la forma:

$$\mu_{B'_i}(y) = \min \{\mu_i(\mathbf{x}), \mu_{B_i}(y)\}$$

La función de pertenencia de la salida agregada se calcula aplicando la T-conorma, donde un caso particular se calcula como el máximo:

$$\mu(y) = \max_{i=1,\dots,M} \{\mu_{B'_i}(y)\}$$

Otra forma de calcular la función de pertenencia agregada es utilizar composición producto-suma. En este caso, el grado de disparo de la regla i -ésima se calcula como:

$$\mu_i(\mathbf{x}) = \prod_{j=1}^n \mu_{A_{ij}}(x_j) \quad (4.1)$$

la función de pertenencia de B_i es, por tanto, de la forma:

$$\mu_{B'_i}(y) = \mu_i(\mathbf{x})\mu_{B_i}(y)$$

y la salida se calcula como:

$$\mu(y) = \sum_{i=1}^M \mu_{B'_i}(y)$$

Para calcular el valor de salida, es necesario realizar el proceso de defuzzificación. Algunos de los métodos para realizar esto se describen en la sección A.5.

Modelo Difuso de Takagi-Sugeno-Kang (Modelo Difuso TSK)

El *modelo difusode Takagi-Sugeno-Kang*, también conocido como *modelo difuso TSK* es propuesto por Takagi, Sugeno y Kang [Tak85, Sug88] con el fin de desarrollar un método sistemático para generar reglas difusas a partir de un conjunto dado de datos entrada-salida. Una regla difusa en un modelo difuso TSK tiene la forma:

$$R_i : \text{Si } x_1 \text{ es } A_{i1} \text{ y } \dots \text{ y } x_n \text{ es } A_{in} \text{ entonces } y = f_i(\mathbf{x})$$

donde $f_i(\mathbf{x})$, $i = 1, \dots, M$ son las funciones del consecuente, normalmente funciones polinomiales, pero pueden ser cualquier función que describa de forma adecuada la salida del modelo en las regiones especificadas por los antecedentes de las reglas.

La salida total se calcula utilizando la *media ponderada*, evitando el proceso de defuzzificación del sistema de Mamdani que resulta muy costoso computacionalmente. La salida se calcula por tanto de la forma:

$$y = \frac{\sum_{i=1}^M \mu_i(\mathbf{x}) f_i(\mathbf{x})}{\sum_{i=1}^M \mu_i(\mathbf{x})}$$

donde $\mu_i(\mathbf{x})$ es el grado de disparo de la regla i -ésima que se calcula de igual forma que en el modelo difuso de Mamdani, por ejemplo según la ecuación (4.1).

En la práctica, con objeto de reducir aún más el coste computacional, muchas veces se reemplaza la media ponderada por la *suma ponderada*:

$$y = \sum_{i=1}^M \mu_i(\mathbf{x}) f_i(\mathbf{x})$$

Cuando $f_i(\mathbf{x})$ es un polinomio de primer orden, al sistema de inferencia resultante se le llama *modelo difuso TSK de primer orden*:

$$R_i : \text{Si } x_1 \text{ es } A_{i1} \text{ y } \dots \text{ y } x_n \text{ es } A_{in} \text{ entonces } y = \mathbf{a}_i^T \mathbf{x} + b_i \quad (4.2)$$

donde $\mathbf{a}_i = [a_{i1}, \dots, a_{in}]^T \in \Re^n$ y $b_i \in \Re$ para $i = 1, \dots, M$ son los parámetros del consecuente. Es decir, la regla desarrollada es de la forma:

$$R_i : \text{Si } x_1 \text{ es } A_{i1} \text{ y } \dots \text{ y } x_n \text{ es } A_{in} \text{ entonces } y = a_{i1}x_1 + \dots + a_{in}x_n + b_i$$

En el caso de que $f_i(\mathbf{x})$ sea una constante, entonces tenemos un *modelo difuso TSK de orden cero*:

$$R_i : \text{Si } x_1 \text{ es } A_{i1} \text{ y } \dots \text{ y } x_n \text{ es } A_{in} \text{ entonces } y = b_i$$

donde $b_i \in \Re$ para $i = 1, \dots, M$ son los parámetros del consecuente.

4.1.2 Estimación de los parámetros de los consecuentes

Existen diversos métodos para calcular los parámetros de los consecuentes. Uno de los métodos más difundido es el *método de mínimos cuadrados* que consiste en minimizar la suma del error cuadrado, calculando el error como la diferencia entre la salida real y la salida del modelo.

Dado un modelo difuso TSK de primer orden de la forma descrita en la ecuación (4.2), los valores de los parámetros de los consecuentes \mathbf{a}_i y b_i pueden estimarse de la siguiente forma:

Sean los vectores de datos:

$$\mathbf{z}_k = [\mathbf{x}_k^T; y_k]^T = [x_{k1}, \dots, x_{kn}, y_k]^T, \quad k = 1, \dots, N$$

Para formular el problema de forma matricial, se define la matriz $\mathbf{X} \in \mathbb{R}^{N \times n}$ con los vectores \mathbf{x}_k en sus columnas y el vector $\mathbf{y} \in \mathbb{R}^N$ con los escalares y_k :

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_N^T \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}$$

Añadiendo una columna unitaria a la \mathbf{X} tenemos la matriz extendida $\mathbf{X}_e \in \mathbb{R}^{N \times (n+1)}$:

$$\mathbf{X}_e = [\mathbf{X}; \mathbf{1}]$$

Los parámetros del consecuente de la regla i -ésima \mathbf{a}_i y b_i se concatenan en un solo vector de parámetros $\boldsymbol{\theta}_i \in \mathbb{R}^{n+1}$:

$$\boldsymbol{\theta}_i = [\mathbf{a}_i^T; b_i]^T, \quad i = 1, \dots, M$$

El vector de parámetros $\boldsymbol{\theta}' \in \mathbb{R}^{M(n+1)}$ se construye de la forma:

$$\boldsymbol{\theta}' = [\boldsymbol{\theta}_1^T; \dots; \boldsymbol{\theta}_M^T]^T$$

Construimos las matrices diagonales $\Gamma_i \in \mathbb{R}^{N \times N}$ de la forma:

$$\Gamma_i = \begin{bmatrix} \gamma_{i1} & 0 & \dots & 0 \\ 0 & \gamma_{i2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \gamma_{iN} \end{bmatrix}, \quad i = 1, \dots, M$$

donde el elemento k -ésimo de la diagonal es el grado de disparo normalizado:

$$\gamma_{ik} = \frac{\mu_{ik}}{\sum_{j=1}^N \mu_{jk}} = \frac{\mu_i(\mathbf{x}_k)}{\sum_{j=1}^N \mu_j(\mathbf{x}_k)}, \quad i = 1, \dots, M, \quad k = 1, \dots, N$$

Sea la matriz X' en $\Re^{N \times (MN)}$ compuesta por las matrices Γ_i y X_e :

$$X' = [(\Gamma_1 X_e); \dots; (\Gamma_M X_e)]$$

El problema de mínimos cuadrados resultante:

$$\mathbf{y} = X' \boldsymbol{\theta}' + \boldsymbol{\epsilon}$$

tiene la solución:

$$\boldsymbol{\theta}' = \left[(X')^T X' \right]^{-1} (X')^T \mathbf{y}$$

siendo $\boldsymbol{\epsilon}$ el vector de error. Los parámetros \mathbf{a}_i y b_i se obtienen de la forma:

$$\mathbf{a}_i = [\theta'_{q+1}, \dots, \theta'_{q+n}]^T, \quad b_i = \theta'_{q+n+1}, \quad i = 1, \dots, M$$

donde $q = (i - 1)(n + 1)$.

4.1.3 Transparencia en modelos difusos aproximativos

Uno de los aspectos que distinguen la modelización difusa de otras aproximaciones del tipo caja negra, es que los modelos difusos son, hasta cierto grado, transparentes a la interpretación y análisis. Sin embargo, cuando se utilizan modelos aproximativos contruidos mediante métodos numéricos (regresión por mínimos cuadrados, gradiente, etc) la transparencia no se consigue de forma automática; por tanto, si deseamos tener modelos interpretables, será necesario utilizar alguna técnica para mejorar la transparencia del modelo.

Cuando se generan modelos difusos a partir de los datos, siempre se tiene un cierto grado de redundancia y complejidad innecesaria. La redundancia típicamente ocurre cuando tenemos conjuntos difusos similares, de forma que dichos conjuntos se superponen describiendo prácticamente la misma región en el dominio de la variable. En tales casos, el modelo está utilizando más conjuntos difusos de los necesarios, puesto que ambos conjuntos representan el mismo concepto lingüístico. Por otra parte, también pueden obtenerse algunos conjuntos difusos similares al conjunto universal, que tampoco añaden información al modelo.

Setnes [Set95] desarrolla un algoritmo que utiliza una medida de similaridad para identificar conjuntos difusos similares y los reemplaza por un conjunto difuso común que represente a los conjuntos originales. Este conjunto difuso común sustituye a los conjuntos originales en la base de reglas, de forma que se reduce el número de conjuntos difusos que definen el modelo. Si la redundancia del modelo es alta, al mezclar conjuntos difusos similares puede ocurrir que aparezcan reglas similares que deben ser eliminadas, llevando a una reducción en el número de reglas.

Existen muchas medidas para cuantificar la similaridad entre dos conjuntos difusos. Una de las medidas más utilizadas es la siguiente:

$$S(A, B) = \frac{|A \cap B|}{|A \cup B|} \quad (4.3)$$

donde $|\cdot|$ es la cardinalidad del conjunto difuso. Es decir:

$$S(A, B) = \frac{\int_{\mathcal{X}} \mu_{A \cap B}(x) dx}{\int_{\mathcal{X}} \mu_{A \cup B}(x) dx} = \frac{\int_{\mathcal{X}} \min \{ \mu_A(x), \mu_B(x) \} dx}{\int_{\mathcal{X}} \max \{ \mu_A(x), \mu_B(x) \} dx}$$

siendo \mathcal{X} el dominio de los conjuntos difusos A y B .

El algoritmo de simplificación mezcla conjuntos difusos similares de forma iterativa utilizando dos valores umbrales: $\eta \in [0, 1]$ para mezclar conjuntos difusos similares y $\eta_r \in [0, 1]$ para eliminar conjuntos difusos similares al conjunto universal. En cada iteración, se calcula la similaridad s entre todos los conjuntos difusos para cada variable del antecedente y se mezclan los conjuntos difusos con mayor similaridad $s \geq \eta$. La base de reglas se actualiza sustituyendo el nuevo conjunto difuso por los dos conjuntos mezclados. El algoritmo continúa iterando hasta que ya no se encuentran más conjuntos difusos con similaridad $s \geq \eta$. Por último, los conjuntos difusos similares al conjunto universal son eliminados. El algoritmo es de la siguiente forma:

1. Seleccionar los dos conjuntos difusos con mayor similaridad.

- Calcular $s_{ijk} = S(A_{ij}, A_{kj})$, $j = 1, \dots, n$, $i, k = 1, \dots, M$
- Seleccionar A_{lq} y A_{mq} tales que $s_{lmq} = \max_{ijk, i \neq j} \{s_{ijk}\}$

2. Mezclar los conjuntos difusos y actualiza la base de reglas.

- Si $s_{lmq} \geq \eta$ entonces mezclar A_{lq} y A_{mq} para crear un nuevo conjunto difuso A y reemplazar A_{lq} y A_{mq} por A en la base de reglas.

3. Si hay más conjuntos difusos con $s_{ijk} > \eta$ entonces volver al paso 1.
4. Eliminar los conjuntos difusos similares al conjunto universal.
 - Para cada conjunto difuso A_{ij} calcular $s_{ij} = S(A_{ij}, U)$, donde U es el conjunto universal ($\mu_U(x_j) = 1 \forall x_j \in \mathcal{X}_j$).
 - Si $s_{ij} \geq \eta_r$ entonces eliminar A_{ij} de la base de reglas.

El algoritmo sólo mezcla un par de conjuntos difusos por cada iteración. La mezcla de dos conjuntos difusos A y B se realiza estableciendo el soporte del nuevo conjunto difuso C como el soporte de $A \cup B$. Esto garantiza que se preserve la cobertura del espacio del antecedente. El núcleo de C se calcula haciendo la media de los núcleos de A y B .

Setnes et al. [Set98] proponen una modificación del anterior algoritmo de forma que en cada iteración en lugar de mezclarse sólo los dos conjuntos con mayor similitud, se realiza la mezcla de todos los conjuntos cuya similaridad sea mayor que el umbral η . Es decir, en cada iteración se calcula, para cada variable de entrada $j = 1, \dots, n$, el conjunto de conjuntos difusos A_j de la forma:

$$A_j = \{A_{lj} | s_{jlm} \geq \eta, l, m = 1, \dots, M, l \neq m\}$$

y se calcula el conjunto general A_{Gj} como la mezcla de todos los conjuntos de A_j , de forma que el soporte de A_{Gj} sea igual al soporte de la unión de todos los conjuntos de A_j , y el núcleo de A_{Gj} sea la media de los núcleos de los conjuntos de A_j . Todos los conjuntos difusos pertenecientes a A_j son reemplazados por el conjunto A_{Gj} .

4.1.4 Modelización neuro-difusa

La hibridación de diferentes algoritmos es una técnica que permite aprovechar las ventajas de cada uno de los algoritmos, obteniendo mejores resultados que los obtenidos por cada uno de los algoritmos de forma independiente. El uso de redes neuronales en la modelización difusa permite obtener modelos difusos más precisos, por ello las técnicas de modelización neuro-difusa han resultado de gran interés [Jan97].

Moody y Darken [Moo88] proponen una estructura de red neuronal que utiliza neuronas de recepción locales para realizar aplicación de funciones. A las redes neuronales que siguen este esquema se les llama redes neuronales con funciones de base

radial (*Radial Basis Function Neural Network, Red Neuronal RBF*). La figura 4.1 muestra un diagrama esquemático de una red neuronal RBF con dos entradas, una salida y cuatro neuronas:

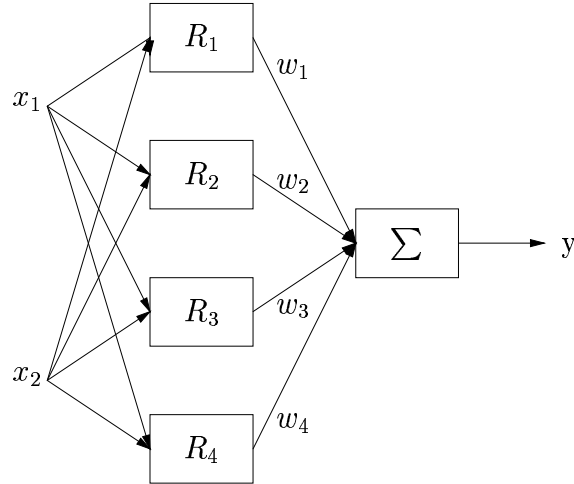


Figura 4.1: Red Neuronal RBF.

El nivel de activación de la neurona i -ésima viene dado por su función de base radial R_i , $i = 1, \dots, M$, siendo M el número de neuronas. Típicamente R_i es una función gaussiana:

$$R_i(\mathbf{x}) = \exp \left(-\frac{\|\mathbf{x} - \mathbf{c}_i\|^2}{2\sigma_i^2} \right)$$

o una función logística:

$$R_i(\mathbf{x}) = \frac{1}{1 + \exp \left(\frac{\|\mathbf{x} - \mathbf{c}_i\|^2}{\sigma_i^2} \right)}$$

donde \mathbf{x} es el vector de entrada,

\mathbf{c}_i es un vector con la misma dimensión de \mathbf{x} que indica el centro de dicha función, y

σ_i indica la varianza de la función.

Por tanto, el nivel de activación de la función de base radial calculada por la neurona i -ésima es máximo cuando el vector \mathbf{x} se encuentra en el centro \mathbf{c}_i de la función.

La salida de una red neuronal RBF puede calcularse de dos formas. El método más sencillo es calcular la *suma ponderada* de los valores de salida asociados con cada neurona:

$$y = \sum_{i=1}^M w_i R_i(\mathbf{x})$$

donde w_i es el peso asociado a la salida de la neurona i -ésima. Un método alternativo para calcular la salida es mediante la *media ponderada* de los valores de salida asociados con cada neurona:

$$y = \frac{\sum_{i=1}^M w_i R_i(\mathbf{x})}{\sum_{i=1}^M R_i(\mathbf{x})}$$

El valor w_i puede calcularse como una función general de los valores de entrada:

$$w_i = f_i(\mathbf{x})$$

En este caso, la salida total del modelo se calcula, por tanto, como:

$$y = \frac{\sum_{i=1}^M R_i(\mathbf{x}) f_i(\mathbf{x})}{\sum_{i=1}^M R_i(\mathbf{x})} \quad (4.4)$$

Equivalencia funcional entre una red neuronal RBF y un sistema de inferencia difuso (FIS)

Una extensión de la red neuronal RBF original de Moody-Darken consiste en asignar una función lineal a la salida de cada neurona:

$$w_i = \mathbf{a}_i^T \mathbf{x} + b_i$$

donde \mathbf{a}_i y b_i son los parámetros asociados a la salida de la neurona i -ésima. En este caso, la salida obtenida por la red neuronal RBF es idéntica a la producida por un sistema de inferencia difuso TSK de primer orden si cumple las siguientes condiciones:

- El número de neuronas de la red RBF es igual al número de reglas del FIS.
- Tanto la red neuronal RBF como el FIS tienen el mismo método de agregación (suma ponderada o media ponderada) para calcular la salida total.
- Las funciones de base radial de la red neuronal RBF deben ser iguales a la función de pertenencia compuesta de la premisa de una regla difusa del FIS.

Una forma de conseguir esto es utilizar una función de pertenencia gaussiana con la misma varianza en la regla difusa y aplicar el producto para calcular el grado de disparo.

- La salida de cada regla del FIS y de la red neuronal RBF deben tener la misma función de respuesta:
 - en el caso de una ecuación lineal de primer orden, $f_i(\mathbf{x}) = \mathbf{a}_i^T \mathbf{x} + b_i$, resulta equivalente a un sistema de inferencia TSK de primer orden,
 - en el caso de una constante, $f_i(\mathbf{x}) = b_i$, entonces resulta equivalente a un sistema de inferencia TSK de orden cero.

Entrenamiento de una red neuronal RBF

Se han propuesto diversos algoritmos de aprendizaje para identificar los parámetros \mathbf{c}_i , σ_i y w_i de una red neuronal RBF [Jan97]. El uso de este tipo de redes ha dado muy buenos resultados en problemas de modelización. Las posiciones centrales de las funciones \mathbf{c}_i pueden determinarse mediante técnicas de clustering, mientras que los valores de σ_i pueden establecerse de forma heurística tomando la distancia media de los vecinos más próximos a su centro.

Una vez estos parámetros han sido fijados, los pesos de salida w_i pueden establecerse utilizando un método de mínimos cuadrados o gradiente.

Otra forma de realizar el entrenamiento es mediante un método de gradiente aplicado a la red neuronal. El grado de disparo de una regla dada equivale al grado de disparo de una neurona y viene dado mediante la composición suma-producto de la forma:

$$R_i(\mathbf{x}) = \exp \left[- \sum_{j=1}^n \frac{|x_j - c_{ij}|^2}{2\sigma_{ij}^2} \right] = \prod_{j=1}^n \exp \left[- \frac{|x_j - c_{ij}|^2}{2\sigma_{ij}^2} \right] = \prod_{j=1}^n \mu_{A_{ij}}(x_j) = \mu_i(\mathbf{x})$$

donde:

$\mathbf{c}_i = [c_{i1}, \dots, c_{in}]^T$ es el vector de centros de la regla i -ésima,

$\boldsymbol{\sigma}_i = [\sigma_{i1}, \dots, \sigma_{in}]^T$ es el vector de varianzas de la regla i -ésima, y

A_{ij} , $j = 1, \dots, n$ son los conjuntos difusos del antecedente de la regla i -ésima, definido cada uno por una función de pertenencia gaussiana con parámetros (c_{ij}, σ_{ij}) .

La salida total del modelo se calcula sumando las contribuciones individuales de cada regla utilizando la media ponderada, tal como se muestra en la ecuación (4.4) donde $f_i(\mathbf{x})$ es la función definida en el consecuente de la regla i -ésima. En el caso de una función lineal de primer orden:

$$f_i(\mathbf{x}) = \theta_{i1}x_1 + \dots + \theta_{in}x_n + \theta_{i(n+1)}$$

donde $\boldsymbol{\theta}_i = [\theta_{i1}, \dots, \theta_{i(n+1)}]^T = [\mathbf{a}_i^T; b_i]^T$ es el vector de parámetros del consecuente.

El algoritmo de entrenamiento actualiza de forma incremental los parámetros en base a los patrones de entrenamiento que se le presentan. Los parámetros de la red son actualizados aplicando el método del descenso por gradiente en la función de error. Para el patrón de entrenamiento k -ésimo, el error E_k se define como:

$$E_k = \frac{1}{2} (y_k - t_k)^2$$

donde y_k es la salida real del modelo, y t_k es la salida deseada para el vector de entrada k -ésimo. La regla para realizar la actualización es la siguiente:

$$\mathbf{v}_i \leftarrow \mathbf{v}_i + \eta \Delta \mathbf{v}_i = \mathbf{v}_i - \eta \frac{\partial E_k}{\partial \mathbf{v}_i}$$

donde $i = 1, \dots, M$, $\mathbf{v}_i = [\mathbf{c}_i^T; \boldsymbol{\sigma}_i^T; \boldsymbol{\theta}_i^T]^T$ son los parámetros asociados a la regla i -ésima, y η es el ratio de aprendizaje. Esta regla se aplica durante un número de iteraciones (epochs). Los gradientes negativos de E_n con respecto a cada parámetro se calculan de la siguiente forma:

$$\begin{aligned} \Delta c_{ij} &= -\frac{\partial E_n}{\partial c_{ij}} = (t_k - y_k) \frac{f_i(\mathbf{x}) - y_k}{z} R_i(\mathbf{x}) \frac{x_j - c_{ij}}{\sigma_{ij}^2} \\ \Delta \sigma_{ij} &= -\frac{\partial E_n}{\partial \sigma_{ij}} = (t_k - y_k) \frac{f_i(\mathbf{x}) - y_k}{z} R_i(\mathbf{x}) \frac{(x_j - c_{ij})^2}{\sigma_{ij}^3} \\ \Delta \theta_{ij} &= -\frac{\partial E_n}{\partial \theta_{ij}} = (t_k - y_k) \frac{1}{z} R_i(\mathbf{x}) x_j \\ \Delta \theta_{i(n+1)} &= -\frac{\partial E_n}{\partial \theta_{i(n+1)}} = (t_k - y_k) \frac{1}{z} R_i(\mathbf{x}) \end{aligned}$$

donde $i = 1, \dots, M$, $j = 1, \dots, n$, y $z = \sum_{i=1}^M R_i(\mathbf{x})$.

4.2 Enfoques Evolutivos para Modelización Difusa

La hibridación de técnicas evolutivas con otros tipos de técnicas ha dado muy buenos resultados en modelización difusa [Rus98, Rou00, Ish01, Mag96, Cord97, Gom99a]. Existen muchos algoritmos que han utilizado diferentes aproximaciones evolutivas para obtener modelos difusos, utilizando normalmente hibridación con alguna otra técnica numérica. En esta sección veremos algunos ejemplos de algoritmos que utilizan estos métodos. Esta revisión permitirá detectar puntos a mejorar y dónde la optimización tendrá su papel.

4.2.1 FuGeNeSys. Sistema neuro-genético para modelización difusa

Russo [Rus98] desarrolla un sistema neuro-genético para modelización difusa llamado *FuGeNeSys* (*Fuzzy Genetic Neural System*) que es una técnica híbrida que hace uso de una combinación de algoritmos genéticos, redes neuronales y sistemas difusos. Este algoritmo permite realizar el aprendizaje supervisado de reglas difusas a partir de un conjunto de datos. Las principales características de FuGeNeSys son las siguientes:

- El número de reglas M que se obtiene es siempre bajo, casi siempre por debajo de diez.
- Con el método de codificación propuesto, el número de bits de cada individuo crece de forma proporcional al número de entradas y salidas, no de forma exponencial.
- Pueden generarse bases de reglas difusas simplificadas, es decir, el algoritmo es capaz de eliminar los antecedentes innecesarios en cada regla.
- El algoritmo es capaz de realizar selección de variables, identificando aquellas variables relevantes.
- Puede utilizarse para problemas de clasificación y de interpolación.

Representación del modelo difuso

Cada individuo de la población codifica un modelo difuso TSK de orden cero compuesto por un conjunto de M reglas para un número de n entradas en el antecedente. Tenemos por tanto A_{ij} , $i = 1, \dots, M$, $j = 1, \dots, n$ conjuntos difusos definidos por las funciones de pertenencia $\mu_{A_{ij}}$ que se han definido como gaussianas simétricas:

$$\mu_{A_{ij}}(x_j) = \exp \left[-\frac{(x_j - c_{ij})^2}{2\sigma_{ij}^2} \right]$$

por lo que para codificar estas funciones necesitamos los centros c_{ij} y las varianzas σ_{ij} de cada conjunto. FuGeNeSys codifica los valores c_{ij} y los valores inversos de las varianzas γ_{ij} . Al codificar la inversa de la varianza, se elimina la singularidad en la ecuación de la gaussiana cuando $\sigma_{ij} = 0$. El valor de la función de pertenencia se calcula, por tanto, como:

$$\mu_{A_{ij}}(x_j) = \exp \left[-\gamma_{ij}^2 (x_j - c_{ij})^2 \right]$$

El valor de disparo de la regla i -ésima se calcula utilizando el operador mínimo:

$$\mu_i(\mathbf{x}) = \min_{j=1}^n \{ \mu_{A_{ij}}(x_j) \}$$

Para codificar los consecuentes, asociamos un parámetro θ_i que define la constante de cada consecuente. El valor total de salida se calcula como la suma ponderada de la forma:

$$y = \frac{\sum_{i=1}^M \mu_i(\mathbf{x}) \theta_i}{\sum_{i=1}^M \mu_i(\mathbf{x})}$$

También se puede calcular la salida de una forma más sencilla:

$$y = \sum_{i=1}^M \mu_i(\mathbf{x}) \theta_i$$

Para un modelo con M reglas y n entradas, un individuo está formado por $M(2n + 1)$ números reales codificados como:

$$I = (\mathbf{c}_1, \boldsymbol{\gamma}_1, \dots, \mathbf{c}_M, \boldsymbol{\gamma}_M, \theta_1, \dots, \theta_M)$$

donde $\mathbf{c}_i = (c_{i1}, \dots, c_{in})$ y $\boldsymbol{\gamma}_i = (\gamma_{i1}, \dots, \gamma_{in})$ para $i = 1, \dots, M$.

Función de adecuación

La función de adecuación se ha construido agregando diversos parámetros que se desean optimizar mediante una serie de heurísticas. La expresión analítica de la función de adecuación para un individuo \mathbf{x} es la siguiente:

$$F(\mathbf{x}) = \frac{K_{num}}{K_{den} + eK_{err} + rK_{rid} + r_2K_{rid2} + fK_{tet}}$$

donde K_{xxx} son coeficientes numéricos definidos por el usuario.

El valor e representa el error de aprendizaje:

$$e = \left[\frac{1}{N} \sum_{k=1}^N \left(\frac{y_k - t_k}{\Delta_y} \right)^2 \right]^{1/2}$$

donde:

- N es el número de datos de aprendizaje,
- y_k es la salida del modelo para el dato k -ésimo,
- t_k es la salida deseada para el dato k -ésimo,
- y Δ_y es la diferencia entre el valor máximo y mínimo de y_k ,

El parámetro r indica el ratio entre el número de antecedentes y consecuentes utilizados en todas las reglas y el valor $M \cdot m$. Es decir, cuanto más pequeño sea el valor r , el sistema difuso representado será más sencillo.

El parámetro r_2 se calcula como el porcentaje de entradas utilizadas por el sistema difuso comparado con el número de entradas totales. Este parámetro también es un indicador de la sencillez del sistema difuso, pero, a diferencia de r que refuerza aquellos individuos con menor número de antecedentes y consecuentes, r_2 refuerza aquellos individuos que utilizan menor número de entradas, es decir está realizando una selección de variables.

El último parámetro f es un indicador del grado de disparo de cada regla para los datos utilizados, de forma que se refuerzan aquellas reglas que tienen un mayor grado de disparo, es decir, aquellas reglas que influyen más en el sistema de inferencia.

Algoritmo evolutivo

El algoritmo evolutivo divide el espacio de búsqueda en hipercubos y sitúa a cada individuos dentro de su hipercubo correspondiente. La selección de individuos se realiza según un esquema steady-state elitista y proporcional a su función de adecuación. El algoritmo tiene dos tipos de selección: global y local. Las dos formas de realizar la selección son similares, con la diferencia que la selección local se limita al hipercubo local.

Una vez se han seleccionado dos padres, éstos producen un solo descendiente aplicando el operador de cruce, que realiza el cruce por un solo punto y el operador de mutación. El descendiente reemplaza al individuo con peor valor de adecuación dentro de su propio hipercubo.

Por último, se utiliza un operador que aplica un método de entrenamiento numérico para mejorar la precisión. En primer lugar, se construye el sistema neurodifuso correspondiente al individuo y se entrenan los pesos utilizando un método de gradiente. Por último, el sistema entrenado se transforma nuevamente en un individuo que se vuelve a introducir en su hipercubo correspondiente.

El esquema del algoritmo es como sigue:

1. Dividir el espacio de búsqueda en hipercubos y generar una población inicial aleatoria, $t \leftarrow 0$.
2. Seleccionar dos individuos padres mediante steady-state elitista proporcional a la función de adecuación.
3. Generar un descendiente mediante cruce y mutación de los padres.
4. Aplicar un método de gradiente al descendiente.
5. Si el nuevo individuo es mejor que alguno de sus padres, entonces reemplazar el padre por el descendiente e insertar el descendiente en su hipercubo correspondiente.
6. $t \leftarrow t + 1$.
7. Si $t < T$ entonces volver al paso 2.

4.2.2 Algoritmo evolutivo para modelización difusa con reducción de la complejidad.

Roubos y Setnes [Rou00] proponen un algoritmo evolutivo que entrena un modelo difuso basado en reglas TSK de primer orden.

Representación del modelo difuso

Cada individuo de la población codifica un modelo difuso TSK de primer orden compuesto por un conjunto de M reglas para un número de n entradas en el antecedente. Tenemos, por tanto, A_{ij} , $i = 1, \dots, M$, $j = 1, \dots, n$ conjuntos difusos definidos por las funciones de pertenencia μ_{ij} que se han definido como funciones triangulares:

$$\mu_{A_{ij}}(x_j) = \max \left(\min \left(\frac{x_j - a_{ij}}{b_{ij} - a_{ij}}, \frac{c_{ij} - x_j}{c_{ij} - b_{ij}} \right), 0 \right)$$

por tanto para codificar estas funciones necesitamos los parámetros a_{ij} , b_{ij} y c_{ij} para cada conjunto.

El valor de disparo de la regla i -ésima se calcula utilizando el producto:

$$\mu_i(\mathbf{x}) = \prod_{j=1}^n \mu_{A_{ij}}(x_j)$$

Los consecuentes son funciones lineales de primer grado, por tanto para codificar los consecuentes, debemos asociar un vector de parámetros $\boldsymbol{\theta}_i = [\theta_{i1}, \dots, \theta_{i(n+1)}]^T$ que defina la función lineal de primer orden de cada consecuente:

$$f_i(\mathbf{x}) = \theta_{i1}x_1 + \dots + \theta_{in}x_n + \theta_{i(n+1)}$$

El valor total de salida se calcula como la suma ponderada de la forma:

$$y = \frac{\sum_{i=1}^M \mu_i(\mathbf{x}) f_i(\mathbf{x})}{\sum_{i=1}^M \mu_i(\mathbf{x})}$$

Para un modelo con M reglas y n entradas, un individuo estará formado por $N = M(4n + 1)$ números reales codificados como:

$$I = (\mathbf{a}_1, \mathbf{b}_1, \mathbf{c}_1, \dots, \mathbf{a}_M, \mathbf{b}_M, \mathbf{c}_M, \boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_M)$$

donde $\mathbf{a}_i = (a_{i1}, \dots, a_{in})$, $\mathbf{b}_i = (b_{i1}, \dots, b_{in})$, $\mathbf{c}_i = (c_{i1}, \dots, c_{in})$, y $\boldsymbol{\theta}_i = (\theta_{i1}, \dots, \theta_{i(n+1)})$ para $i = 1, \dots, M$.

Función de adecuación

Los aspectos que se quieren optimizar son la precisión y la similaridad del modelo. La precisión se mide mediante el error cuadrático medio:

$$E = \frac{1}{N} \sum_{k=1}^N (y_k - t_k)^2$$

donde:

N es el número de datos de entrenamiento,

y_k es la salida real del modelo para el dato k -ésimo, y

t_k la salida deseada para el dato k -ésimo.

La similaridad del modelo se calcula como la media de las similaridades máximas en cada entrada, es decir:

$$S = \frac{1}{n} \sum_{j=1}^n \left[\frac{\max_{i,k=1,\dots,n_i, j \neq k} \{S(A_{ij}, A_{kj})\}}{n_i - 1} \right]$$

donde:

n_i es el número de conjuntos difusos distintos para la entrada i -ésima, y

$S(A_{ij}, A_{kj})$ es la medida de similaridad entre los conjuntos A_{ij} y A_{kj} que se calcula tal como indica la ecuación (4.3).

La función objetivo a optimizar para un individuo \mathbf{x} se calcula como una agregación de las dos medidas anteriores de la forma:

$$F(\mathbf{x}) = (1 + \lambda S) E$$

donde $\lambda \in [-1, 1]$ es un factor de peso que determina si la similaridad es recompensada ($\lambda < 0$) o penalizada ($\lambda > 0$).

Algoritmo evolutivo

El algoritmo evolutivo realiza la selección mediante un método de rueda de ruleta elitista que selecciona en cada iteración un número de individuos n_c , manteniendo siempre el mejor individuo en la población (selección elitista).

Cuando un individuo es seleccionado se le aplican los operadores de cruce y mutación. Se definen tres operadores de cruce y tres operadores de mutación; se aplica uno de ellos de forma aleatoria. Los operadores de cruce para dos individuos $I_v = (v_1, \dots, v_N)$ y $I_w = (w_1, \dots, w_N)$ se definen de la siguiente forma:

- *Cruce aritmético simple*: $I'_v = (v_1, \dots, v_k, w_{k+1}, \dots, w_N)$ y $I'_w = (w_1, \dots, w_k, v_{k+1}, \dots, v_N)$ siendo k un entero aleatorio en $[2, N - 1]$.
- *Cruce aritmético completo*: $I'_v = rI_v + (1 - r)I_w$ y $I'_w = rI_w + (1 - r)I_v$, siendo r un número real aleatorio en $[0, 1]$.
- *Cruce heurístico*: $I'_v = I_v + r(I_w - I_v)$ y $I'_w = I_w + r(I_v - I_w)$, siendo r un número real aleatorio en $[0, 1]$.

Los operadores de mutación para un individuo $I = (v_1, \dots, v_N)$ se definen de la siguiente forma:

- *Mutación uniforme*: $I' = (v_1, \dots, v'_k, \dots, v_N)$ siendo k un entero aleatorio en $[1, N]$ y v'_k un real aleatorio en el rango del parámetro k -ésimo, $[v_k^{min}, v_k^{max}]$.
- *Mutación uniforme múltiple*: se realiza la mutación uniforme de n elementos seleccionados aleatoriamente, donde n es un entero aleatorio en $[1, N]$.
- *Mutación gaussiana*: $I' = (v'_1, \dots, v'_N)$ donde $v'_k = v_k + f_k$, $k = 1, \dots, N$ siendo f_k un número aleatorio de una gaussiana con media cero y varianza adaptativa:

$$\sigma_k = \frac{T - t}{T} \cdot \frac{v_k^{max} - v_k^{min}}{3}$$

donde T es el número de iteraciones total y t el número de iteración actual. Este parámetro consigue un ajuste más fino en las últimas generaciones del algoritmo.

Después de aplicar los operadores de cruce y mutación se realiza una simplificación de la base de reglas utilizando el método para mejora de la transparencia descrito en el apartado 4.1.3.

La población inicial se genera calculando un modelo difuso original utilizando clustering para generar los conjuntos difusos de los antecedentes y mínimos cuadrados para establecer el valor de los parámetros de los consecuentes. A partir de dicho

modelo, se generan aleatoriamente los individuos de la población inicial, realizando variaciones de los parámetros dentro de unos márgenes dados.

El esquema del algoritmo es, por tanto, como sigue:

1. Calcular la población inicial a partir de un modelo original calculado mediante clustering y mínimos cuadrados, $t \leftarrow 0$.
2. Seleccionar n_c individuos mediante selección de rueda de ruleta elitista proporcional al valor de adecuación.
3. Realizar el cruce y mutación de los individuos seleccionados.
4. Realizar la simplificación de los individuos.
5. $t \leftarrow t + 1$.
6. Si $t < T$ entonces volver al paso 2.

4.2.3 Algoritmo evolutivo para modelos lingüísticos

Ishibuchi [Ish01] propone un algoritmo evolutivo para modelizar un sistema difuso lingüístico del tipo Mamdani.

Representación del modelo difuso

Un individuo codifica un conjunto de M reglas lingüísticas (R_1, \dots, R_M) siendo R_i la regla i -ésima codificada mediante sus n antecedentes y su consecuente de la forma:

$$R_i = (A_{i1}, \dots, A_{in}, B_i), \quad i = 1, \dots, M$$

donde cada conjunto difuso A_{ij} y B_i viene definido por los l parámetros de su función de pertenencia:

$$A_{ij} = (a_{ij1}, \dots, a_{ijl}), \quad i = 1, \dots, M, \quad j = 1, \dots, n$$

$$B_i = (b_{i1}, \dots, b_{il}), \quad i = 1, \dots, M$$

Puesto que el número de reglas M es variable, los individuos van a ser de longitud variable $M(n+1)l$.

Función de adecuación

Con el objetivo de conseguir un modelo difuso preciso y también interpretable, se han tenido en cuenta los siguientes parámetros:

- *Precisión*: $f_1(\mathbf{x})$ se evalúa mediante el valor del error mínimo cuadrático.
- *Compactitud*: $f_2(\mathbf{x})$ se evalúa mediante el número de reglas del modelo.
- *Simplicidad de las reglas*: $f_3(\mathbf{x})$ se evalúa mediante el número de conjuntos difusos.

Para tener en cuenta estos tres criterios se define la función de adecuación mediante una combinación lineal de los mismos:

$$F(\mathbf{x}) = w_1 f_1(\mathbf{x}) + w_2 f_2(\mathbf{x}) + w_3 f_3(\mathbf{x})$$

donde w_1, w_2, w_3 son números reales que ponderan cada uno de los objetivos.

Algoritmo evolutivo

El algoritmo evolutivo realiza la selección de individuos en base a la adecuación de cada individuo; una vez seleccionados dos individuos, se realiza el cruce y mutación. Los operadores de variación que se definen son los siguientes:

- *Cruce de reglas*: El cruce se realiza a nivel de reglas, de forma que dados dos individuos:

$$I = (R_1^I, \dots, R_{M_I}^I) \text{ y } J = (R_1^J, \dots, R_{M_J}^J)$$

se generan otros dos individuos:

$$I' = (R_1^I, \dots, R_k^I, R_1^J, \dots, R_l^J) \text{ y } J' = (R_{k+1}^I, \dots, R_{M_I}^I, R_{l+1}^J, \dots, R_{M_J}^J)$$

donde k, l son enteros aleatorios, $k \in \{1, \dots, M_I - 1\}$ y $l \in \{1, \dots, M_J - 1\}$.

- *Mutación de los valores lingüísticos*: Este operador realiza la mutación de los valores lingüísticos con una probabilidad dada. El operador de mutación está definido en el dominio de los parámetros que definen los valores lingüísticos.
- *Mutación las reglas*: Con una probabilidad dada, se elimina una regla aleatoria del conjunto de reglas.

Después de aplicar estas operaciones de cruce y mutación, se aplican dos procedimientos heurísticos a los conjuntos de reglas:

- *Heurística 1:* Reemplaza los valores lingüísticos de los consecuentes de cada regla por otros valores que se adecúen mejor a los datos de entrenamiento. El reemplazo se realiza intercambiando los consecuentes de forma probabilista, donde la probabilidad de intercambiar el valor lingüístico del consecuente B_j por B_k se calcula como:

$$P_{B_j, B_k} = \frac{\lambda_{B_k}}{\sum_{l=1}^M \lambda_{B_l}}$$

donde λ_{B_k} es el grado de compatibilidad del valor lingüístico B_k :

$$\lambda_{B_k} = \sum_{i=1}^N \mu_j(\mathbf{x}_i) \mu_{B_k}(y_i), \quad k = 1, \dots, M$$

siendo (\mathbf{x}_i, y_i) , $i = 1, \dots, N$ los pares de datos entrada-salida del conjunto de entrenamiento.

- *Heurística 2:* El segundo procedimiento genera nuevas reglas lingüísticas a partir de los datos de entrada-salida. Para ello, se calcula el error para cada par de datos entrada-salida y se identifica el par de datos (\mathbf{x}_p, y_p) para el cual el error es máximo. Entonces se genera una regla lingüística determinando los antecedentes y consecuentes mediante los valores lingüísticos más compatibles con el par de datos (\mathbf{x}_p, y_p) .

El esquema del algoritmo es como sigue:

1. Generar una población inicial P^0 aleatoria a partir de un conjunto de reglas dado, $t \leftarrow 0$.
2. Seleccionar dos individuos de la población P^t de forma probabilista según su valor de adecuación.
3. Realizar el cruce y mutación de los individuos seleccionados.
4. Aplicar los procedimientos heurísticos 1 y 2.
5. Introducir los nuevos individuos en la población P^{t+1} .

6. Si P^{t+1} no está completa entonces volver al paso 3.
7. $t \leftarrow t + 1$.
8. Si $t < T$ entonces volver al paso 2.

4.2.4 Algoritmos evolutivos para modelos con etiquetas fijas

Magdalena y Velasco [Mag96] proponen dos aproximaciones evolutivas para realizar el entrenamiento de sistemas difusos, especialmente orientados al caso de los controladores difusos. Un controlador difuso se caracteriza normalmente por la siguiente estructura general:

- *Funciones de normalización.* Las variables de entrada son normalizadas linealmente de sus intervalos reales al intervalo $[-1, 1]$.
- *Base de reglas difusas.* Estas son las reglas difusas que definen el comportamiento del controlador. La base de regla difusa tiene dos partes diferenciadas:
 - *Conjuntos difusos.* Son los conjuntos difusos fijos y variables del sistema definidos por sus funciones de pertenencia. En este caso se utilizan conjuntos trapezoidales.
 - *Reglas difusas.* Son las reglas que indican cómo realizar la inferencia en el sistema.

En la primera aproximación descrita por Magdalena y Velasco, cada individuo de la población codifica una base de reglas completa, mientras que en la segunda aproximación, cada individuo codifica una sola regla difusa.

Bases de reglas como individuos de la población

Codificación de los individuos. La codificación de los individuos se realiza mediante dos componentes:

1. *Base de datos.* Deben codificarse tres elementos:

- *Parámetros del controlador:*
 - Número de variables de entrada: N ,
 - Número de variables de salida: M ,
 - Número de conjuntos difusos asociados con cada variable:
 $\mathbf{n} = (n_1, \dots, n_N)$ donde n_i representa el número de conjuntos difusos asociados con la variable de entrada i -ésima, y
 $\mathbf{m} = (m_1, \dots, m_M)$ donde m_j representa el número de conjuntos difusos asociados con la variable de salida j -ésima.
- *Límites de normalización:* Un array de $2(N + M)$ números reales representa el conjunto de límites de normalización. Cada fila del array contiene los límites de una variable de entrada o de salida del sistema ($\{v_{min}, v_{max}\}$).
- *Conjunto de funciones de pertenencia:* Contiene las funciones de pertenencia de los L conjuntos difusos, donde L se calcula de la forma:

$$L_a = \sum_{i=1}^N n_i, \quad L_b = \sum_{i=1}^M m_i, \quad L = L_a + L_b$$

Cada conjunto difuso viene representado por los 4 parámetros que definen una función trapezoidal, por tanto este conjunto viene definido por $4L$ números reales en el rango $[-1, 1]$.

2. *Base de reglas.* La estructura de las reglas de control difusa es de la forma:

Si x_i es A_{oi} **y** ... **y** x_k es A_{pk} **entonces** y_j es B_{qj} **y** ... **y** y_l es B_{rl}

donde:

x_i es la variable de entrada i -ésima,

A_{oi} es un conjunto difuso asociado con x_i , ($o \leq n_i$),

y_j es la variable de salida j -ésima, y

B_{qj} es el conjunto difuso asociado con y_j , ($q \leq m_j$).

Cada regla del sistema se codifica en dos cadenas de bits: una cadena de longitud L_a para el antecedente (un bit por cada término lingüístico asociado a cada variable de entrada) y una cadena de longitud L_c para el consecuente.

Para codificar el antecedente se empieza con una cadena de L_a bits todos con un valor inicial 0. Si el antecedente de una regla contiene una entrada difusa de la forma x_j es A_{ij} entonces se pone un 1 en la posición p de la cadena:

$$p = i + \sum_{k=1}^{j-1} n_k$$

Se repite este proceso para todas las entradas difusas de la regla. El proceso para codificar el consecuente es similar al anterior, reemplazando n por m .

Función de adecuación. El controlador difuso se diseña mediante un sistema supervisor y su objetivo se describe por medio de un cierto criterio de eficiencia definido cuando se diseña una aplicación específica. Este criterio de eficiencia se basa en el comportamiento global cubriendo, no un solo ciclo de control, sino un periodo más largo. La función de adecuación, que también mantiene un sentido global, se obtiene directamente de este criterio de eficiencia.

Algoritmo evolutivo. El algoritmo evolutivo utiliza dos operadores de selección diferentes dentro de un proceso elitista. El primer operador de selección asigna a los individuos con mayor adecuación un número mayor de copias; el segundo operador de selección asigna a cada individuo una probabilidad proporcional a su adecuación de recibir o no una sola copia.

El cruce entre individuos se realiza mediante un punto de corte simple. Dados dos individuos que codifican dos bases de reglas:

$$\begin{aligned} I &= \{R_1^I, \dots, R_{M_I}^I\} \\ J &= \{R_1^J, \dots, R_{M_J}^J\} \end{aligned}$$

se obtienen otros dos individuos:

$$\begin{aligned} I' &= \{R_1^I, \dots, R_k^I, R_{l+1}^J, \dots, R_{M_J}^J\} \\ J' &= \{R_1^J, \dots, R_l^J, R_{k+1}^I, \dots, R_{M_I}^I\} \end{aligned}$$

donde $k \in 1, \dots, M_I$ y $l \in 1, \dots, M_J$ son números aleatorios obtenidos de forma independiente para cada individuo.

La mutación se compone de dos procesos diferentes:

1. *Mutación de reglas.* Se realiza en los $L_a + L_b$ bits que componen una regla y es similar a la mutación genética clásica aplicada a una cadena de bits.
2. *Mutación de rangos.* Si una variable tiene el rango $[\lambda_l, \lambda_u]$, la mutación se realiza de la forma:

$$\begin{aligned}\lambda'_l &= \lambda_l + K P_1 S_1 (\lambda_u - \lambda_l) / 2 \\ \lambda'_u &= \lambda_u + K P_2 S_2 (\lambda_u - \lambda_l) / 2\end{aligned}$$

donde:

$K \in [0, 1]$ es un parámetro que define la variación máxima,

P_1 y P_2 son valores aleatorios distribuidos uniformemente en $[0, 1]$, y

S_1 y S_2 toman valores -1 o 1 con probabilidad 0.5 .

Para aquellas variables que tienen un rango simétrico antes de la mutación, se añaden las condiciones $P_2 = P_1$ y $S_2 = -S_1$ con objeto de mantener los rangos simétricos después de la mutación.

Reglas como individuos de la población

Codificación de los individuos. En este sistema, las reglas tienen una estructura de la forma:

$$\text{Si } \langle \text{condición} \rangle \text{ entonces } \langle \text{acción} \rangle$$

donde $\langle \text{condición} \rangle$ es un conjunto de términos $\{V_x \in [a, b, c, d]\}$ y $\langle \text{acción} \rangle$ tiene la forma $\{V_y \in [e, f, g, h]\}$.

Cada término $\{V_x \in [a, b, c, d]\}$ se interpreta como que V_x pertenece al conjunto difuso trapezoidal definido por los parámetros $\{a, b, c, d\}$. Existe un número indeterminado de términos en la condición. Por otra parte, existe un único término para definir la acción. Esto se debe a que la adecuación de una regla está relacionada con la acción propuesta, por lo que si tuviéramos diferentes acciones, tendríamos diferentes adecuaciones para la regla.

Función de adecuación. El objetivo del sistema es guiar a una variable específica X (función de coste) a su mínimo valor. El sistema propuesto minimiza la variable teniendo en cuenta los últimos N casos. El rendimiento del sistema se calcula como:

$$SP_t = \begin{cases} 1, & \text{si } X_t \leq X_{min} \\ \frac{X_t - X_{t-1}}{X_{min} - X_{t-1}}, & \text{si } X_t > X_{min} \text{ y } X_t \leq X_{t-1} \\ \frac{X_{t-1} - X_t}{X_{max} - X_{t-1}}, & \text{si } X_t < X_{max} \text{ y } X_t > X_{t-1} \\ -1, & \text{si } X_t > X_{max} \end{cases}$$

donde:

X_t es el valor de la variable X en el instante t ,

X_{t-1} es el valor de la variable X en el instante $t - 1$,

X_{min} es el valor mínimo de X en los últimos N casos, y

X_{max} es el valor máximo de X en los últimos N casos.

Para evaluar la influencia de cada regla R , se utiliza el conjunto difuso A propuesto para la variable V_a en la parte de acción de la regla, y los valores reales para esta variable X en el tiempo t y $t - 1$. La influencia de una regla se calcula como:

$$RI_{R,t} = \begin{cases} \mu_A(X_t), & \text{si } \mu_A(X_t) > 0 \\ 0, & \text{si } \mu_A(X_t) = 0 \text{ y } |A - X_t| \leq |A - X_{t-1}| \\ \frac{X_t - X_{t-1}}{X_{t-1} - X_t}, & \text{si } \mu_A(X_t) = 0 \text{ y } |A - X_t| > |A - X_{t-1}| \end{cases}$$

donde:

$|A - X|$ es la distancia de X al conjunto difuso A , calculada como la distancia desde X al punto más cercano del soporte de A , y

X_t es el límite superior (si $X_t > X_{t-1}$) o inferior (si $X_t < X_{t-1}$) del rango de la variable X .

Para cada regla se obtiene un valor de evaluación multiplicando su influencia por el rendimiento del sistema:

$$E_{R,t} = SP_t \times RI_{R,t}$$

Si el rendimiento del sistema es positivo, la evaluación de la regla se calcula en función de su influencia: si la regla ha sido aplicada, la evaluación final es positiva; si no, es negativa. Para el caso en que la eficiencia sea negativa se realizan consideraciones similares.

Después de obtener el valor de evaluación, se modifica el valor de adecuación de la regla de forma que se incrementa dicho valor si la evaluación es positiva y se decrementa en otro caso. La adecuación se obtiene de la siguiente manera:

$$S_{R,t} = \begin{cases} S_{R,t-1} + KT_R E_{R,t} (1 - S_{R,t-1}), & \text{si } E_{R,t} \geq 0 \\ S_{R,t-1} + KT_R E_{R,t} S_{R,t-1}, & \text{si } E_{R,t} < 0 \end{cases}$$

donde:

K es una constante que permite fijar la memoria del sistema (valores de K cercanos a cero producen variaciones lentas y vice-versa), y

T_R es el valor de verdad de la regla [Mag96].

Algoritmo evolutivo. A diferencia de los sistemas tradicionales, en este esquema el algoritmo evolutivo no se ejecuta en cada ciclo del sistema. De hecho, el proceso de generación de reglas sólo se ejecuta cada cierto tiempo cuando una base de reglas ha sido utilizada varias veces. La aplicación del algoritmo tiene dos etapas: generación e inserción.

- *Generación.* Para generar nuevas reglas se diseñan un conjunto de operadores de variación adaptados:
 - *Selección.* La selección se realiza asignando a un individuo una probabilidad de selección proporcional a su adecuación.
 - *Cruce uniforme.* Es el cruce uniforme tradicional aplicado al conjunto de términos que forman el individuo.
 - *Mutación.* Cambia los conjuntos difusos mutando los valores $[a, b, c, d]$ para crear nuevos conjuntos difusos.
- *Inserción.* Las nuevas reglas generadas se ponen en un lugar espacial llamado *Limbo*, donde son probadas y evaluadas sin afectar la salida del sistema. Sólo cuando se ha comprobado que una regla funciona bien, es aceptada y se inserta en la base de reglas del sistema.

4.2.5 Algoritmo evolutivo para entrenar modelos aproximativos y descriptivos en control difuso

Cordón y Herrera [Cord97] proponen un proceso evolutivo en tres etapas para realizar el entrenamiento de controladores difusos descriptivos y aproximativos utilizando modelos de Mamdani. Si se utiliza una base de conocimiento en la cual los conjuntos difusos que dan significado semántico a las etiquetas lingüísticas se encuentran distribuidos uniformemente, entonces dicha base de reglas es un modelo *descriptivo*. Por otra parte, si las variables lingüísticas pueden tomar cualquier valor se dice que el modelo es *aproximativo*. En el trabajo de Cordón y Herrera se han tratado ambos tipos de modelos.

El proceso de aprendizaje se divide en tres fases diferentes, de forma que se simplifica el espacio de búsqueda. Las tres fases del método propuesto son las siguientes:

1. *Proceso de generación evolutivo*. En esta fase se generan las reglas de control difusas. Este proceso obtiene un conjunto de reglas que cubren el conjunto de entrenamiento de una forma adecuada.
2. *Proceso de simplificación genético*. En esta segunda fase se seleccionan reglas mediante un algoritmo genético con codificación binaria y una medida de la eficiencia del controlador difuso en el sistema que se está identificando. Esto permite evitar el sobreentrenamiento que la fase anterior podría causar eliminando aquellas reglas redundantes.
3. *Proceso de ajuste genético*. Por último se utiliza un algoritmo genético con codificación de números reales y una medida de eficiencia del controlador difuso. Este proceso realiza un ajuste de las funciones de pertenencia de cada regla de control difusa si el modelo es aproximativo, o un ajuste de la base de regla completa si tenemos un modelo descriptivo.

Proceso de generación evolutivo

Para generar las reglas de control difusas se tienen dos procesos:

- *Un método de generación de reglas difusas*, que encuentra la mejor regla entre un conjunto de ejemplo de acuerdo a las características incluidas en una función

de adecuación; puede incluir una estrategia de evolución opcional para ajustar localmente las reglas.

- *Un método de cubrimiento*, para obtener un conjunto de reglas difusas que cubran el conjunto de ejemplos.

Método de generación de reglas difusas. Este método obtiene la mejor regla entre un conjunto de ejemplo de acuerdo a las características incluidas en una función de adecuación; puede incluir una estrategia de evolución opcional para ajustar localmente las reglas; si se utiliza dicha estrategia evolutiva se obtiene un modelo aproximativo, si no se utiliza, el modelo es descriptivo.

Cada vez que se ejecuta este método, se produce un conjunto de reglas difusas candidatas generando las reglas que mejor cubren cada ejemplo del conjunto de entrenamiento. Estas reglas se obtienen tomando la partición difusa que mejor se adapta al valor del ejemplo para cada variable.

Para evaluar la adecuación de cada regla candidata, se utiliza una función de adecuación con tres criterios diferentes que nos aseguran que el conjunto final es completo y consistente. Finalmente se selecciona la regla difusa con mejor adecuación.

Los criterios utilizados son:

1. *Maximizar el valor de frecuencia.* La frecuencia de una regla difusa R_i sobre un conjunto de ejemplos E_p se define como:

$$\Psi_{E_p}(R_i) = \frac{\sum_{l=1}^p R_i(e_l)}{p}$$

2. *Maximizar el grado de cubrimiento medio sobre los ejemplos positivos.* El grado medio de cubrimiento sobre el conjunto de ejemplos positivos se define como:

$$G_w(R_i) = \sum_{e_l \in E_w^+(R_i)} \frac{R_i(e_l)}{n_w^+(R_i)}$$

siendo $E_w^+(R_i)$ el conjunto de ejemplos positivos a R_i con un grado de compatibilidad mayor o igual que ω y $n_w^+(R_i)$ igual a $|E_w^+(R_i)|$.

3. *Minimizar el conjunto de ejemplos negativos.* Se establece la función de penalización sobre los ejemplos negativos como:

$$g_n(R_i) = \begin{cases} 1, & \text{si } n_{R_i}^- \leq kn_w^+(R_i) \\ \frac{1}{n_{R_i}^- - kn_w^+(R_i) + \exp(1)}, & \text{en otro caso} \end{cases}$$

siendo n_{R_i} el número de ejemplos negativos y $k \in [0, 1]$ un parámetro que determina la fracción del número de ejemplos positivos que se permite de ejemplos negativos.

Estos tres criterios se combinan en una función de adecuación. Cordón y Herrera proponen la siguiente:

$$F(R_i) = \Psi_{E_p}(R_i) G_w(R_i) g_n(R_i)$$

El método anterior genera reglas de naturaleza descriptiva. Si deseamos mayor precisión podemos obtener un modelo aproximativo realizando un ajuste de las reglas. Cordón y Herrera proponen utilizar una estrategia de evolución para realizar dicho ajuste. La función de adecuación que se utiliza en este caso es la siguiente:

$$F(R_i) = \Psi_{E_p}(R_i) G_w(R_i) g_n(R_i) LNIR(R_i)$$

donde $LNIR(R_i) \in [0, 1]$ es un valor de penalización que determina el grado de interacción entre reglas, de forma que es igual a 1 cuando la regla R_i no interacciona con ninguna otra regla y toma el mínimo valor cuando la regla R_i es igual a alguna otra regla.

El método de generación puede ser resumido en el siguiente algoritmo:

1. Inicializar el conjunto de reglas difusas candidatas B^c como vacío.
2. Para cada valor $e_l \in E_p$, generar la regla difusa R_c que mejor cubra dicho valor. Si $R_c \notin B^c$, añade R_c a B^c .
3. Evaluar todas las reglas difusas contenidas en B^c y seleccionar aquella regla R_r con mayor valor de adecuación $F(R_r)$.
4. Si se desea, optimizar la regla R_r utilizando la estrategia de evolución, obteniendo una regla aproximativa.

Método de cubrimiento. El método de cubrimiento obtiene un conjunto de reglas difusas que cubren el conjunto de ejemplo. En cada iteración, se ejecuta el método de generación anterior para escoger la mejor regla de control difusa de acuerdo al estado actual del conjunto de entrenamiento, considerando el valor relativo de cubrimiento de esta regla sobre el conjunto de ejemplos, y se eliminan los ejemplos con un valor de cubrimiento mayor que un valor ϵ dado por el diseñador. El método de cubrimiento es de la siguiente forma:

1. Inicialización
 - (a) Introducir los valores para k , ω y ϵ .
 - (b) Establecer el grado de cubrimiento del ejemplo, $CV[l] \leftarrow 0$, $l = 1, \dots, p$.
 - (c) Inicializar el conjunto de reglas final B^g como vacío.
2. Aplicar el método de generación sobre el conjunto de ejemplos E_p , obteniendo como salida la mejor regla de control R_r , de acuerdo al estado actual de E_p .
3. Introducir R_r en B^g .
4. Para cada $e_l \in F_p$ hacer
 - (a) $CV[l] \leftarrow CV[l] + R_r(e_l)$.
 - (b) Si $CV[l] \geq \epsilon$, entonces eliminar R_r de E_p .
5. Si E_p es vacío, entonces terminar, en otro caso volver al paso 2.

Proceso de simplificación genético

Durante el proceso de generación de reglas es posible que ocurra un sobreaprendizaje, es decir, que algunos ejemplos sean cubiertos con un grado mayor que el deseado, lo que provoca que la base de regla se comporte peor debido a la existencia de reglas redundantes. Para solucionar este problema es necesario simplificar el conjunto de reglas obtenido en el proceso de generación, eliminando aquellas reglas redundantes.

El proceso de simplificación se basa en un algoritmo genético codificado en binario, en el cual la selección de individuos se realiza utilizando el procedimiento de muestreo estocástico universal junto con un esquema elitista, y la generación de

los nuevos individuos se realiza utilizando los operadores de cruce binario multipunto (con dos puntos) y mutación uniforme.

Las reglas se codifican mediante una cadena binaria de longitud fija. Considerando las reglas contenidas en el conjunto B^g derivado en el paso anterior, un subconjunto B^s de B^g se codifica como una cadena de M bits: $C = (c_1, \dots, c_M)$ de forma que si $c_i = 1$ entonces $R_i \in B^s$, en otro caso $R_i \notin B^s$.

La población inicial se genera introduciendo un individuo representando el conjunto B^g completo, es decir, con $c_i = 1$, para $i = 1, \dots, M$. El resto de individuos se seleccionan aleatoriamente.

Como función de adecuación se utiliza el error cuadrático medio sobre un conjunto de datos de entrenamiento E_{TDS} , lo que se calcula mediante la siguiente expresión:

$$E(C_j) = \frac{1}{2|E_{TDS}|} \sum_{e_l \in E_{TDS}} [ey^l - S(ex^l)]^2$$

donde $S(ex^l)$ es el valor de salida obtenido por el controlador difuso utilizando la base de reglas codificada en C_j , $R(C_j)$, cuando los valores de las variables de estado son ex^l , y ey^l es el valor deseado.

Para mantener la propiedad de completitud, se fuerza a que todos los ejemplos sean cubiertos con un grado mayor o igual que τ , el grado mínimo de completitud aceptado. Usualmente τ es menor o igual que ω , el grado de compatibilidad utilizado en el proceso de generación. La función de adecuación por tanto se define de la siguiente forma:

$$F(C_j) = \begin{cases} E(C_j), & \text{si } TSCD(R(C_j), E_{TDS}) \geq \tau \\ \frac{1}{2} \sum_{e_l \in E_{TDS}} (ey^l)^2, & \text{en otro caso} \end{cases}$$

siendo $TSCD(R(C_j), E_{TDS})$ el grado de completitud de $R(C_j)$ sobre el conjunto de ejemplos E_{TDS} .

Proceso de ajuste genético

El proceso de ajuste genético tiene dos variaciones, según el modelo difuso sea aproximativo o descriptivo. La diferencia entre ambos está en el esquema de codificación. En el primer caso, cada individuo codifica la base de reglas completa, mientras que en el segundo solo se codifican las particiones difusas con objeto de ajustar las funciones de pertenencia para todas las reglas de control difusas contenidas en la base de reglas.

En ambos casos, el algoritmo genético utiliza una codificación real y el procedimiento de selección por muestreo estocástico universal junto con un esquema elitista. Los operadores de variación empleados son la mutación no uniforme de Michalewicz [Mic92] y el cruce aritmético max-min [Her95]. Como función de adecuación se utiliza la misma función $F(C_j)$ definida para el proceso de simplificación.

Proceso de ajuste genético aproximativo. En este caso, cada individuo codifica una base de reglas completa, en concreto codifican la base de reglas R^s obtenida con el proceso de simplificación. Las diferencias entre unos individuos y otros se encuentran en las funciones de pertenencia.

Utilizando funciones de pertenencia triangulares, cada conjunto difuso se codifica mediante tres parámetros y cada una de las reglas se codifica de la forma:

$$C_i = (a_{i1}, b_{i1}, c_{i1}, \dots, a_{in}, b_{in}, c_{in}, a_i, b_i, c_i), \quad i = 1, \dots, M$$

Por tanto, un individuo se representa como:

$$(C_1, C_2, \dots, C_M)$$

Proceso de ajuste genético descriptivo. Este segundo proceso de ajuste genético es una versión modificada del anterior. En este caso, cada cromosoma codifica una base de reglas diferente. Se representa una partición difusa primaria como un vector compuesto por $3N_i$ valores reales, siendo N_i el número de términos que forman el conjunto de términos lingüísticos para la variable i -ésima. La base de datos completa para un problema, en el cual tenemos M variables, se codifica con parámetros reales y longitud fija uniendo las representaciones parciales de cada una de las particiones difusas de la siguiente forma:

$$C_i = (a_{i1}, b_{i1}, c_{i1}, \dots, a_{iN_i}, b_{iN_i}, c_{iN_i})$$

$$(C_1, C_2, \dots, C_M)$$

4.2.6 Optimalidad Pareto en modelización difusa

Gómez-Skarmeta et al. [Gom98] proponen utilizar un algoritmo evolutivo basado en el concepto de óptimo Pareto para realizar modelización difusa.

Representación del modelo difuso

Cada individuo representa un modelo difuso de Mamdani con un número fijo de reglas MAX para un número de n entradas. Un individuo es de la forma:

$$(R_1, \dots, R_{MAX}, d_1, \dots, d_{MAX})$$

donde (R_1, \dots, R_{MAX}) es el conjunto de reglas total, siendo R_i la regla i -ésima codificada de forma:

$$(A_{i1}, \dots, A_{in}, B_i), \quad i = 1, \dots, MAX$$

siendo A_{ij} y B_i la codificación de los conjuntos difusos correspondientes; en este caso se elige una función de pertenencia trapezoidal, de forma que cada conjunto difuso viene definido por cuatro parámetros:

$$A_{ij} = (a_{ij1}, a_{ij2}, a_{ij3}, a_{ij4}), \quad i = 1, \dots, MAX, \quad j = 1, \dots, n$$

$$B_i = (b_{i1}, b_{i2}, b_{i3}, b_{i4}), \quad i = 1, \dots, MAX$$

El conjunto de control (d_1, \dots, d_{MAX}) , donde $d_i \in \{0, 1\}$ indica si la regla i -ésima se encuentra activa en el individuo. Por tanto, el número M de reglas en un individuo viene dado por el número de valores d_i iguales a 1.

Función de adecuación

La adecuación de un individuo en este caso no viene determinada por un solo valor, sino por varios valores, correspondientes a los diferentes parámetros que deseamos optimizar en el modelo difuso. Se proponen dos funciones objetivo:

1. La primera función objetivo f_1 sirve para optimizar la precisión del modelo, se establece como el error cuadrático medio.
2. La segunda función objetivo f_2 optimiza la simplificación del modelo, siendo igual al número M de reglas.

Para comparar dos individuos se utiliza el concepto de dominación Pareto teniendo en cuenta todas las funciones objetivo. Es decir, al optimizar el modelo se tiene en cuenta tanto su precisión como su simplicidad, obteniéndose un modelo preciso con el menor número de reglas.

Algoritmo evolutivo

El algoritmo propuesto es un algoritmo multi-objetivo que realiza la selección mediante un torneo con dominación Pareto de igual forma que el algoritmo NPGA de Horn y Nafpliotis [Hor93]. También establece un mecanismo de nichos para mantener la diversidad en la población, de forma que entre dos individuos no dominados entre ellos, se elige aquel con mejor diversidad.

Se aplica el cruce y mutación a los individuos seleccionados y los descendientes se copian a una nueva población. Este proceso se repite hasta que la población está completa. El algoritmo también es elitista, pues en cada generación se seleccionan un número aleatorio de individuos no dominados.

La población inicial se genera de forma aleatoria, pero si tenemos una base de reglas inicial obtenida, por ejemplo, con un método de clustering, dichas reglas pueden ser introducidas de forma aleatoria en la población inicial.

Los operadores de variación se han definido en dos niveles, puesto que los individuos tienen dos tipos de información diferente (el conjunto de reglas y el conjunto de control). Para el primero, se definen diversos tipos de operadores de cruce (cruce simple, cruce aritmético, cruce max-min) y de mutación (mutación simple y mutación no uniforme). En cuanto al conjunto de control, al estar compuesto por un conjunto de bits puede aplicarse el cruce y mutación simples binario [Gol89].

El esquema del algoritmo es el siguiente:

1. Generar una población inicial P^0 , $t \leftarrow 0$.
2. Seleccionar un número aleatorio de individuos no dominados de la población actual P^t y copiarlos a la nueva población P^{t+1} .
3. Seleccionar dos individuos mediante torneo con dominación Pareto con nichos.
4. Cruzar y mutar los individuos seleccionados para generar descendientes que copiar a la nueva población P^{t+1} .
5. Si P^{t+1} no está completa entonces volver al paso 3.
6. $t \leftarrow t + 1$.
7. Si $t < T$ entonces volver al paso 2.

4.3 Sumario

Este capítulo realiza una discusión sobre diferentes aspectos de modelización difusa. Se distinguen dos problemas fundamentales dentro de la modelización difusa: (i) identificación de variables, e (ii) identificación de la estructura. Se revisan dos de las estructuras fundamentales dentro de la modelización difusa: (i) modelo descriptivo (Mamdani), y (ii) modelo aproximativo (TSK). En esta tesis se va a trabajar sobre todo con modelos aproximativos. Estos modelos resultan interesantes por ser precisos; sin embargo, no siempre resultan interpretables. Para conseguir modelos aproximativos interpretables se plantean diversas técnicas para mejorar la transparencia en modelos aproximativos. Por otra parte, se puede realizar una equivalencia entre modelos difusos TSK y redes neuronales de base radial, lo cual permite aplicar el método del gradiente para conseguir modelos con gran precisión.

En la segunda parte del capítulo se revisan distintos enfoques evolutivos para modelización difusa. La hibridación de técnicas evolutivas con métodos clásicos como el gradiente permite aprovechar las ventajas de ambos, la robustez de las técnicas evolutivas y la eficacia de los métodos numéricos. Por otra parte, la utilización del concepto Pareto en las técnicas evolutivas multiobjetivo permiten obtener modelos difusos precisos y también transparentes.

En conclusión, se ha repasado el uso de algoritmos evolutivos en modelización difusa, analizando cómo hibridar con otras técnicas, en concreto con redes neuronales RBF y, por tanto, se va a plantear un esquema de integración en el que se obtengan modelos difusos precisos que siguen el modelo difuso TSK, pero que son, al mismo tiempo, descriptivos como los modelos de Mamdani.

Capítulo 5

Nuevos Algoritmos Evolutivos Multiobjetivo para Modelización Difusa

En los últimos años, la modelización difusa como complemento a las técnicas de modelización convencionales se ha convertido en un tema de investigación de gran importancia y se han realizado aplicaciones con éxito en diversas áreas. La combinación de sistemas difusos y computación evolutiva ha obtenido muy buenos resultados de investigación en las últimas décadas [Rus98, Rou00, Ish01, Cord97, Gom98, Jim01]. Ambas tecnologías son apropiadas para entornos de optimización en los que un decisor prefiere obtener soluciones satisfactorias de acuerdo a sus preferencias en lugar de soluciones optimales; también resultan especialmente adecuadas cuando tenemos problemas de optimización donde aparece un cierto grado de incertidumbre y no existen técnicas adecuadas para resolverlos. Los algoritmos evolutivos se han utilizado para obtener los modelos difusos encontrando los valores de los antecedentes y consecuentes de las reglas difusas y se han aplicado tanto a modelos difusos con un número fijo de reglas, como modelos con un número variable de las mismas [Wan98, Ish99]. Por otra parte, los algoritmos evolutivos han sido combinados con otras técnicas como clustering difuso [Hwa98, Gom99a] y redes neuronales [Jagi99, Rus98], obteniéndose algoritmos complejos en los que, tal como se reconoce en [Vale99] y [Set98], los aspectos de interpretabilidad del modelo, tales como la transparencia y la compactitud del conjunto de reglas, no se consideran parámetros de importancia y son relegados a un

segundo plano prevaleciendo sólo aquellos aspectos relacionados con la precisión. De esta forma, el modelo difuso se vuelve una caja negra, las reglas difusas no resultan interpretables y podemos preguntarnos en estos casos qué sentido tiene realizar una modelización difusa en lugar de utilizar otras técnicas.

La transparencia y la interpretabilidad para modelos difusos ha recibido una gran atención en la literatura reciente [Pom00, Jin00, Cord00, Joh00]. En base a esta literatura, los criterios fundamentales para optimizar al identificar un modelo difuso son precisión, transparencia y compactitud. La precisión es el primer criterio que siempre debe tenerse en cuenta, puesto que deseamos obtener modelos precisos que se ajusten lo máximo posible al sistema real que se modeliza. No obstante, si se desea obtener modelos interpretables, además de la precisión, es necesario tener en cuenta criterios de interpretabilidad. Para obtener un modelo difuso con una buena interpretabilidad, es necesario utilizar estructuras de reglas transparentes y funciones de pertenencia difusa bien caracterizadas. La compactitud del modelo es otro aspecto importante para su interpretabilidad y dicha compactitud está directamente relacionada con el tamaño del modelo, es decir, el número de reglas, número de conjuntos difusos y número de entradas para cada conjunto, mientras que la transparencia está relacionada con la interpretabilidad lingüística de los conjuntos difusos y la localización de las reglas difusas [Vale99, Set98]. Aparecen, por tanto, múltiples criterios para realizar un buen modelo difuso y, por tanto, los algoritmos evolutivos multiobjetivo aparecen como una opción apropiada para resolver problemas de este tipo.

La mayoría de aproximaciones evolutivas a la modelización difusa consisten en múltiples algoritmos evolutivos, normalmente diseñados para realizar una sola tarea cada uno de ellos. En estos casos, cada algoritmo evolutivo optimiza el modelo atendiendo a uno de los criterios de forma separada, lo cual es un impedimento para la búsqueda global. Resulta más apropiado realizar la optimización de todos los criterios de forma simultánea. Para ello, algunos autores han utilizado aproximaciones basadas en técnicas multiobjetivo clásicas en las que los diversos objetivos son agregados para formar una sola función, la cual se optimiza [Gom99a, Rou01]. De esta forma, el algoritmo evolutivo obtiene una solución de compromiso que consiste en la optimización de la suma ponderada de todos los criterios.

Otra aproximación más interesante en optimización multiobjetivo consiste en los algoritmos evolutivos basados en el concepto de óptimo Pareto, con los que pueden

obtenerse múltiples soluciones no dominadas con respecto a la precisión, transparencia y compactitud.

Diferentes aproximaciones evolutivas multiobjetivo basadas en dominación Pareto se han aplicado para modelización difusa [Ish97, Jim01]. En este capítulo se van a presentar nuevas soluciones evolutivas multiobjetivo para realizar modelos difusos. El capítulo comienza utilizando un algoritmo evolutivo para realizar selección de variables en la sección 5.1. En la sección 5.2 se propone un algoritmo evolutivo multiobjetivo que encuentra múltiples soluciones no dominadas para problemas de modelización difusa. Los criterios que se tienen en cuenta para identificar el modelo difuso son la precisión, transparencia y compactitud. Se propone un modelo de optimización y un proceso de decisión a posteriori. En la sección 5.3 se propone una mejora del algoritmo anterior introduciendo conjuntos gaussianos y realizando hibridación con el método de descenso de gradiente para lograr un mejor ajuste de la precisión. El resultado es un algoritmo neuro-evolutivo multiobjetivo. Se plantea también una adaptación del algoritmo para calcular modelos difusos aproximativos, es decir, teniendo en cuenta solamente el criterio de precisión. Tanto en la sección 5.2 como en la sección 5.3 se prueban los algoritmos propuestos utilizando un par de problemas clásicos. La sección 5.4 plantea un conjunto de problemas adicionales, entre los que cabe destacar un problema real que consiste en realizar un modelo para predicción de riego y una aplicación del algoritmo neuro-evolutivo para resolver problemas de clasificación. Por último se cierra el capítulo realizando un resumen de todos los contenidos en la sección 5.5.

5.1 Algoritmo Evolutivo para Selección de Variables

La selección de variables es uno de los problemas que aparecen dentro de modelización difusa y consiste en realizar un preprocesamiento de datos para seleccionar las variables más adecuadas a partir de un conjunto de posibles variables.

El problema de identificación de variables puede formularse como un problema de optimización con restricciones [Buc96, Gom99b]. Dado un conjunto de datos normalizados:

$$S = \{(x_{11}, \dots, x_{n1}, y_1), \dots, (x_{1N}, \dots, x_{nN}, y_N)\}$$

es decir, un conjunto de N datos, cada uno compuesto por n variables, se deben determinar los pesos (w_1, \dots, w_n) relacionados con cada variable de forma que se minimize el error cuadrático medio. Matemáticamente se expresa como:

$$\begin{aligned} \text{Minimizar } MSE &= \sqrt{\frac{\sum_{k=1}^N \left(y_k - \sum_{j=1}^p w_j x_{jk} \right)^2}{N}} \\ \text{sujeeto a : } &\sum_{j=1}^n w_j = 1, \\ &0 \leq w_j \leq 1, \quad j = 1, \dots, n \end{aligned}$$

De esta forma, solamente aquellas variables con un peso suficientemente grande forman parte del conjunto de variables de entrada que se tiene en cuenta al realizar el modelo final.

Otra forma de plantear el problema consiste en utilizar el modelo de la media generalizada. En este caso, el problema de identificación de variables se formula de la siguiente forma:

$$\begin{aligned} \text{Minimizar } MSE &= \sqrt{\frac{\sum_{k=1}^N \left(y_k - \left(\sum_{j=1}^n w_j (x_{jk})^f \right)^{1/f} \right)^2}{N}} \\ \text{sujeeto a : } &\sum_{j=1}^n w_j = 1, \\ &0 \leq w_j \leq 1, \quad j = 1, \dots, n, \\ &f_l \leq f \leq f_u \end{aligned} \tag{5.1}$$

donde $f \in [f_l, f_u]$ ($-\infty \leq f_l \leq f_u \leq +\infty$) es el grado de borrosidad de la redundancia/ complementaridad de los datos. El grado de borrosidad f puede ser establecido por un decisor o requerido por el problema. En la formulación de la ecuación (5.1) se puede establecer un grado de borrosidad a un valor concreto f^* haciendo $f_l = f^* = f_u$.

El problema planteado en la ecuación (5.1) es un problema no lineal con restricciones lineales y pueden utilizarse diferentes técnicas para resolverlo. El método de gradiente es una técnica que puede utilizarse; sin embargo, con esta técnica se realiza una optimización local. Por otra parte, utilizando algoritmos evolutivos es posible realizar una optimización global, lo que permite obtener mejores resultados. En esta sección se va a utilizar un algoritmo evolutivo para resolver el problema de la identificación de variables tal como se plantea en la ecuación (5.1).

5.1.1 Representación de soluciones

Un individuo I de la población se representa mediante un conjunto de $n + 1$ números reales:

$$I = (w_1, \dots, w_n, f)$$

donde (w_1, \dots, w_n) son los pesos relacionados con las n variables de entrada y f es el grado de borrosidad asociado.

5.1.2 Población inicial

Los individuos se generan de forma aleatoria, pero satisfaciendo las restricciones del problema. La inicialización de un individuo $I = (w_1, \dots, w_n, f)$ se realiza de la siguiente forma:

$$(w_1, \dots, w_n) \leftarrow \text{pesos1 ó pesos2 aleatoriamente}$$

$$f \leftarrow \text{aleatorio} \in [f_l, f_u]$$

Para inicializar los pesos se utilizan con igual probabilidad dos procedimientos de inicialización. El primer procedimiento asigna n pesos de forma aleatoria de forma que la suma de todos los pesos sea igual a uno.

Procedimiento *pesos1*

$$w_i \leftarrow \text{aleatorio} \in [0, 1], \quad i = 1, \dots, n$$

$$w_i \leftarrow \frac{w_i}{\sum_{j=1}^n w_j}, \quad i = 1, \dots, n$$

El segundo procedimiento es similar al primero, pero asignando a algunos pesos el valor cero de forma aleatoria.

Procedimiento *pesos2*

$$w_i \leftarrow 0, \quad \text{si } i \in C \subseteq \{1, \dots, n\}, \text{ siendo } C \text{ un subconjunto aleatorio de } \{1, \dots, n\}$$

$$w_i \leftarrow \text{pesos1}, \text{ en otro caso}$$

5.1.3 Operadores de variación

Se han considerado cuatro operadores de variación, un operador de cruce y tres operadores de mutación. En todos los casos, los operadores de variación mantienen las restricciones de los individuos.

Cruce aritmético

El cruce que se ha utilizado es el cruce aritmético. Este cruce mantiene las restricciones de los individuos. Dados dos individuos I_1 e I_2 de la forma:

$$I_1 = (w_1^1, \dots, w_n^1, f^1) \quad I_2 = (w_1^2, \dots, w_n^2, f^2)$$

se producen otros dos individuos I_3 e I_4 de la forma:

$$I_3 = (w_1^3, \dots, w_n^3, f^3) \quad I_4 = (w_1^4, \dots, w_n^4, f^4)$$

donde:

$$\begin{aligned} w_i^3 &= \alpha w_i^1 + (1 - \alpha) w_i^2, & w_i^4 &= \alpha w_i^2 + (1 - \alpha) w_i^1, & i &= 1, \dots, n \\ f^3 &= \alpha f^1 + (1 - \alpha) f^2, & f^4 &= \alpha f^2 + (1 - \alpha) f^1 \end{aligned}$$

siendo α una variable aleatoria entre 0 y 1.

Mutación uno

La primera mutación propuesta realiza un pequeño cambio en el individuo intercambiando dos pesos aleatorios. El componente f del individuo siempre se muta mediante una mutación no uniforme. Dado un individuo I :

$$I = (w_1, \dots, w_i, \dots, w_j, \dots, w_n, f)$$

se produce otro individuo I' :

$$I' = (w_1, \dots, w_j, \dots, w_i, \dots, w_n, f')$$

donde i, j son índices aleatorios entre 1 y n y el valor de f' se calcula mediante la mutación no uniforme (ver apartado 3.1.4).

Mutación dos y tres

La segunda y tercera mutaciones propuestas utilizan los procedimientos *pesos1* y *pesos2* respectivamente para modificar un subconjunto aleatorio de pesos. El valor f se modifica según la mutación no uniforme en ambos casos. Dado un individuo I :

$$I = (w_1, \dots, w_n, f)$$

se produce otro individuo I' :

$$I' = (w'_1, \dots, w'_n, f')$$

donde:

$w'_i \leftarrow w_i$, si $i \in C \subseteq \{1, \dots, n\}$, siendo C un subconjunto aleatorio de $\{1, \dots, n\}$,
 $w'_i \leftarrow \text{pesos1} \text{ ó } \text{pesos2} \text{ aleatoriamente}$, si $i \notin C$, y
 f' se calcula mediante la mutación no uniforme.

5.1.4 Selección y reemplazo generacional

La selección y reemplazo generacional se realiza mediante un procedimiento de pre-selección simple steady-state con $n = 2$ de la siguiente forma:

1. Seleccionar dos individuos aleatorios I_1 e I_2 como padres.
2. Cruzar y mutar ambos individuos para producir dos hijos I_3 e I_4 .
3. Si I_3 es mejor que I_1 , entonces reemplazar I_1 por I_3 .
4. Si I_4 es mejor que I_2 , entonces reemplazar I_2 por I_4 .

Otra opción para realizar la selección consiste en utilizar la técnica de selección de ENORA aplicada al caso de un solo objetivo. Se prueba este método para comprobar que resulta válido como un esquema de selección general, no solamente para el caso de tener más de un objetivo, sino también para aquellos casos con un solo objetivo.

5.1.5 Experimentos y resultados

Para validar el algoritmo evolutivo propuesto se ejecuta sobre diversos problemas de prueba. En este apartado se muestran los resultados obtenidos en cada experimento utilizando preselección simple y la técnica de selección de ENORA.

Experimento 1

Se considera el problema test standard propuesto por Dyckhoff y Pedrycz [Dyc84] y utilizado por Buczak y Uhrig [Buc96]. El conjunto de datos consiste en dos variables de entrada x_1 y x_2 y una variable de salida. Para comprobar el funcionamiento de nuestro algoritmo se añaden seis nuevas variables x_3 , x_4 , x_5 , x_6 , x_7 , y x_8 construidas a partir de x_1 y x_2 añadiendo un 10% de ruido uniforme para x_3 y x_4 , un 40% para x_5 y x_6 , y un 100% para x_7 y x_8 . Ejecutamos el algoritmo con los siguientes parámetros:

- Tamaño de la población, $N = 100$,
- Probabilidad de cruce, $p_{Cross} = 0.9$,
- Probabilidad de mutación, $p_{Mutate} = 0.9$,
- Número de generaciones totales,
 - Utilizando preselección simple:

$$T = 10^5 \text{ para } f \text{ constante, y } T = 10^6 \text{ para } f \text{ variable.}$$
 - Utilizando técnica de selección de ENORA:

$$T = 10^4 \text{ para } f \text{ constante, y } T = 10^5 \text{ para } f \text{ variable.}$$

Todos los operadores de cruce y mutación se aplican con la misma probabilidad. El algoritmo termina cuando se alcanza el número de generaciones totales.

La tabla 5.1 muestra los resultados obtenidos con nuestro algoritmo utilizando $f = 1.0$ y $f = 2.0$ y la tabla 5.2 muestra los resultados obtenidos utilizando f variable dentro del intervalo $[f_l, f_u] = [-10.0, 10.0]$ y $[f_l, f_u] = [-20.0, 20.0]$. Se comparan nuestros resultados con los resultados obtenidos por Buczak y Uhrig [Buc96] y por Gómez-Skarmeta et al. [Gom99b]. Se puede observar que los resultados obtenidos por ambos algoritmos son parecidos. El valor MSE obtenido es similar y se identifican correctamente las variables x_1 y x_2 en todos los casos.

	$f = 1.0$			$f = 2.0$		
	[Gom99b]	Método ¹	Método ²	[Gom99b]	Método ¹	Método ²
w_1	0.3831	0.4234	0.4229	0.3146	0.3158	0.3274
w_2	0.5679	0.5766	0.5771	0.6150	0.6842	0.6726
w_3	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
w_4	0.0000	0.0000	0.0000	0.0704	0.0000	0.0000
w_5	0.0317	0.0000	0.0000	0.0000	0.0000	0.0000
w_6	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
w_7	0.0173	0.0000	0.0000	0.0000	0.0000	0.0000
w_8	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
MSE	0.0916	0.0926	0.0926	0.1606	0.1607	0.1607

Tabla 5.1: Resultados obtenidos con nuestro algoritmo y en [Gom99b] para el problema de la selección de variables con f constante.

		$f \in [-10.0, 10.0]$			$f \in [-20.0, 20.0]$		
	[Buc96]	[Gom99b]	Método ¹	Método ²	[Gom99b]	Método ¹	Método ²
f^*	0.39	0.398	0.3765	0.3694	0.382	0.3785	0.3765
w_1	0.25	0.3598	0.4377	0.3945	0.4149	0.4364	0.4442
w_2	0.522	0.4847	0.5472	0.4959	0.5138	0.5474	0.5155
w_3	0.142	0.0716	0.0081	0.0508	0.0050	0.0116	0.0152
w_4	0.003	0.0083	0.0022	0.0479	0.0109	0.0006	0.0190
w_5	0.031	0.0083	0.0047	0.0057	0.0000	0.0040	0.0024
w_6	0.000	0.0041	0.0000	0.0032	0.0166	0.0000	0.0019
w_7	0.021	0.0136	0.0000	0.0014	0.0207	0.0000	0.0015
w_8	0.031	0.0521	0.0000	0.0006	0.0180	0.0000	0.0002
MSE	0.0625	0.0521	0.0525	0.0540	0.0518	0.0525	0.0534

Tabla 5.2: Resultados obtenidos con nuestro algoritmo y en [Gom99b] para el problema de la selección de variables con f variable.

¹ Utilizando preselección simple.

² Utilizando técnica de selección de ENORA.

	Gradiente	Método ¹	Diferencia	Método ²	Diferencia
$[f_l, f_u] = [-10, 10]$	0.1760	0.0525	-70.17%	0.0540	-69.32%
$[f_l, f_u] = [-20, 20]$	0.2056	0.0525	-74.46%	0.0534	-74.03%

Tabla 5.3: Comparación entre el método del gradiente y nuestro algoritmo evolutivo.

¹ Utilizando preselección simple.

² Utilizando técnica de selección de ENORA.

Para comparar los resultados con otras técnicas no evolutivas mostramos en la tabla 5.3 los resultados de precisión obtenidos con nuestro algoritmo evolutivo y con el método de gradiente utilizado por Gómez-Skarmeta et al. [Gom99b] dentro de los intervalos $[f_l, f_u] = [-10.0, 10.0]$ y $[f_l, f_u] = [-20.0, 20.0]$. Se puede ver que el algoritmo evolutivo obtiene mejores resultados que el método de gradiente tanto utilizando preselección simple como la técnica de selección de ENORA, con una mejora media del 72.31% y del 71.67% respectivamente.

Experimento 2

Con objeto de probar nuestro algoritmo con otro ejemplo, se ha ejecutado también con un conjunto de datos diferentes. En este caso el problema tiene dos variables de entrada, x_1 y x_2 , que son variables aleatorias uniforme en los intervalos $[0, 0.5]$ y $[0, 0.3]$ respectivamente y una variable de salida y , que es función lineal de la forma $y = 0.7x_1 + 0.3x_2$. Se añaden dos nuevas variables x_3 y x_4 . La variable x_3 se construye como combinación lineal de las variables x_1 y x_2 agregando un ruido uniforme ($x_3 = 0.4x_1 + 0.2x_2 + 0.3r$, siendo r una variable aleatoria uniforme en el intervalo $[0, 0.3]$) y la variable x_4 se construye como combinación lineal de x_1 y x_2 ($x_4 = 0.2x_1 + 0.7x_2$). Los parámetros del algoritmo son los mismos que en el experimento anterior, excepto el número de iteraciones, que en este caso se establece como $T = 20.000$, para f constante y $T = 200.000$ para f variable tanto utilizando preselección simple como la técnica de selección de ENORA. Las tablas 5.4 y 5.5 muestran los resultados obtenidos con f constante y con f variable respectivamente. Puede verse como en todos los casos se seleccionan las variables x_1 y x_2 de forma correcta, excepto para el caso $f = 2.0$ que resulta seleccionada la variable x_4 en lugar de x_2 , lo cual resulta también correcto al ser x_4 una combinación lineal de x_1 y x_2 sin ruido añadido.

	$f = 1.0$		$f = 2.0$	
	Método ¹	Método ²	Método ¹	Método ²
w_1	0.6999	0.6927	0.5712	0.5711
w_2	0.2995	0.2799	0.0000	0.0000
w_3	0.0000	0.0030	0.0005	0.0005
w_4	0.0006	0.0244	0.4282	0.0428
MSE	0.0000	0.0004	0.0085	0.0085

Tabla 5.4: Resultados obtenidos con nuestro algoritmo para el segundo problema de selección de variables con f constante.

	$f \in [-10.0, 10.0]$		$f \in [-20.0, 20.0]$	
	Método ¹	Método ²	Método ¹	Método ²
f^*	1.0227	1.0295	0.9920	0.9784
w_1	0.6781	0.6829	0.6709	0.6841
w_2	0.2413	0.2577	0.2038	0.2277
w_3	0.0076	0.0043	0.0144	0.0083
w_4	0.0730	0.0551	0.1110	0.0798
MSE	0.0011	0.0008	0.0018	0.0013

Tabla 5.5: Resultados obtenidos con nuestro algoritmo para el segundo problema de selección de variables con f variable.

¹ Utilizando preselección simple.

² Utilizando técnica de selección de ENORA.

5.1.6 Conclusiones

En esta sección se plantea resolver el problema de la selección de variables. Para ello, se sigue una definición del problema basada en pesos y se diseña un algoritmo evolutivo *ad hoc* con el que se obtienen resultados que mejoran los obtenidos mediante otras técnicas no evolutivas tales como el gradiente. El algoritmo propuesto es, por tanto, una herramienta útil para resolver el problema de la selección de variables, que es el primer paso en modelización difusa antes de realizar la identificación del modelo.

5.2 Algoritmo Evolutivo Multiobjetivo para Modelización Difusa

Una vez abordado en la sección anterior el problema de la selección de variables, en esta sección se trata el problema de la identificación dentro de la modelización difusa utilizando algunas de las técnicas desarrolladas en el capítulo tercero. En concreto se propone un algoritmo evolutivo multiobjetivo basado en el concepto de óptimo Pareto. Se considera un modelo difuso basado en reglas del tipo Takagi-Sugeno. Los antecedentes son conjuntos difusos trapezoidales y los consecuentes son funciones lineales. El modelo de optimización que se propone tiene en cuenta diferentes criterios: precisión, transparencia y compactitud. Se discute un método de simplificación de conjuntos difusos para mejorar la transparencia y compactitud. Los operadores de variación realizan una búsqueda manteniendo las restricciones del modelo difuso y se propone un esquema de preselección para mantener la diversidad entre las soluciones junto con otras técnicas de carácter explícito. Por último, se propone un proceso de decisión a posteriori para elegir una solución entre todas las soluciones no dominadas encontradas por el algoritmo.

5.2.1 Identificación del modelo difuso

Se va a considerar el modelo difuso basado en reglas del tipo Takagi-Sugeno (TS) [Tak85] de primer orden. Los consecuentes de las reglas son funciones lineales de las entradas. Las reglas tienen, por tanto, la siguiente forma:

$$R_i : \text{Si } x_1 \text{ es } A_{i1} \text{ y } \dots \text{ y } x_n \text{ es } A_{in} \text{ entonces } y = \theta_{i1}x_1 + \dots + \theta_{in}x_n + \theta_{i(n+1)}$$

donde:

$i = 1, \dots, M$, M es el número de reglas,

$\mathbf{x} = (x_1, \dots, x_n)$ es el vector de entrada,

A_{ij} son los conjuntos difusos definidos en el espacio de los antecedentes por las funciones de pertenencia $\mu_{A_{ij}} : \mathcal{X}_j \rightarrow [0, 1]$, siendo \mathcal{X}_j el dominio de la variable de entrada x_j , y

$\theta_{ij} \in \mathbb{R}$ son los parámetros del consecuente.

La salida total del modelo se calcula sumando las contribuciones individuales de cada regla utilizando la media ponderada:

$$y = \frac{\sum_{i=1}^M \mu_i(\mathbf{x}) f_i(\mathbf{x})}{\sum_{i=1}^M \mu_i(\mathbf{x})}$$

siendo $\mu_i(\mathbf{x})$ el grado de disparo de la regla i -ésima:

$$\mu_i(\mathbf{x}) = \prod_{j=1}^n \mu_{A_{ij}}(x_j) \quad (5.2)$$

y $f_i(\mathbf{x})$ la función definida en el consecuente de la regla i -ésima:

$$f_i(\mathbf{x}) = \theta_{i1}x_1 + \dots + \theta_{in}x_n + \theta_{i(n+1)}$$

Cada conjunto difuso A_{ij} se describe mediante su función de pertenencia, que en este caso es trapezoidal con parámetros $\{a_{ij}, b_{ij}, c_{ij}, d_{ij}\}$:

$$\mu_{A_{ij}}(x_j) = \max \left(0, \min \left(\frac{x_j - a_{ij}}{b_{ij} - a_{ij}}, 1, \frac{c_{ij} - x_j}{c_{ij} - d_{ij}} \right) \right)$$

donde $a_{ij}, b_{ij}, c_{ij}, d_{ij} \in [l_j, u_j]$,

con $a_{ij} \leq b_{ij} \leq c_{ij} \leq d_{ij}$,

para $i = 1, \dots, M$ y $j = 1, \dots, n$.

5.2.2 Criterios para la modelización difusa

Se desea obtener modelos difusos que sean precisos y al mismo tiempo interpretables, para ello se deben tener en cuenta tres criterios fundamentales:

1. Precisión.
2. Transparencia.
3. Compactitud.

Es necesario establecer medidas cualitativas para estos tres criterios mediante funciones objetivo apropiadas que definan la identificación del modelo difuso completo.

Precisión. La precisión de un modelo puede medirse mediante el *error cuadrático medio* (*mean squared error*, *MSE*):

$$MSE = \frac{1}{N} \sum_{k=1}^N (y_k - t_k)^2 \quad (5.3)$$

donde:

y_k es la salida real del modelo para el vector de entrada k -ésimo,

t_k es la salida deseada para el vector de entrada k -ésimo, y

N es el número de muestras de datos.

Transparencia. Para el segundo de los criterios, la transparencia, existen diversas medidas posibles, sin embargo una de las medidas más extendidas es la *similaridad* [Set95]. La similaridad S puede expresarse de la forma:

$$S = \max_{\substack{i, j, k \\ A_{ij} \neq A_{kj}}} S(A_{ij}, A_{kj})$$

donde:

$i = 1, \dots, M$, $j = 1, \dots, n$, y $k = 1, \dots, M$.

Para medir la similaridad entre dos conjuntos difusos distintos A y B se pueden utilizar diferentes medidas. En nuestro trabajo se utiliza la siguiente medida:

$$S(A, B) = \frac{|A \cap B|}{|A \cup B|} \quad (5.4)$$

El valor de S es, por tanto, una medida de agregación de la similaridad para el modelo difuso basado en reglas. El objetivo es minimizar la máxima similaridad entre conjuntos difusos en cada dominio de entrada, puesto que si tenemos conjuntos poco similares, eso significa que el modelo es más transparente, al utilizarse conjuntos difusos diferentes entre sí.

Compactitud. Por último, para el tercer criterio, pueden tomarse como medidas el número de reglas M y el número de conjuntos difusos diferentes L del modelo, puesto que se asume que aquellos modelos con un número pequeño de reglas y de conjuntos difusos son compactos y, por tanto, más interpretables.

Criterio	Medidas
Precisión	MSE
Transparencia	S
Compactitud	M, L

Tabla 5.6: Criterios para la modelización difusa y sus medidas correspondientes.

La tabla 5.6 muestra el resumen de los tres criterios que se han considerado para realizar la modelización difusa y las medidas definidas para dichos criterios.

5.2.3 Simplificación del conjunto de reglas difusas

Las aproximaciones automáticas a la modelización difusa a menudo introducen redundancia en forma de conjuntos difusos similares que describen prácticamente la misma región en el dominio de alguna variable. Es, por tanto, interesante apoyarse en un proceso de simplificación del conjunto de reglas. De acuerdo a alguna medida de similaridad, dos conjuntos difusos similares pueden fusionarse para crear un nuevo conjunto difuso que represente a los dos conjuntos anteriores. Este nuevo conjunto difuso sustituye a los conjuntos anteriores. El proceso de fusión puede repetirse hasta que se obtiene un modelo difuso con un valor adecuado de similaridad; en ese momento puede ser que tengamos varias reglas idénticas, en cuyo caso dichas reglas deben eliminarse del conjunto de reglas. El algoritmo es el siguiente:

1. Mientras existan i, j, k tales que $S(A_{ij}, A_{kj}) > \eta_1$

Calcular C como fusión de A_{ij} y A_{kj} .

Sustituir A_{ij} y A_{kj} por C .

2. Mientras existan i, k tales que los antecedentes de las reglas R_i y R_k sean iguales

Calcular un nuevo consecuente haciendo la media de los parámetros de los consecuentes de R_i y R_k .

Sustituir el consecuente de R_i por el nuevo consecuente.

Eliminar R_k .

Para medir la similaridad entre dos conjuntos difusos A y B , utilizaremos la medida de similaridad descrita por la ecuación (5.4). El valor η_1 ($0 < \eta_1 < 1$) es el umbral para realizar la fusión (en nuestro caso utilizamos $\eta_1 = 0.6$). Si $S(A, B) > \eta_1$, los conjuntos difusos $A = \{a_A, b_A, c_A, d_A\}$ y $B = \{a_B, b_B, c_B, d_B\}$ se fusionan para formar un nuevo conjunto difuso $C = \{a_C, b_C, c_C, d_C\}$ de la forma:

$$\begin{aligned} a_C &= \min \{a_A, a_B\} \\ b_C &= \alpha b_A + (1 - \alpha) b_B \\ c_C &= \alpha c_A + (1 - \alpha) c_B \\ d_C &= \max \{d_A, d_B\} \end{aligned}$$

donde $\alpha \in [0, 1]$ es un valor aleatorio que determina la influencia de A y B en el nuevo conjunto difuso C

Por ejemplo, supongamos que tenemos las siguientes reglas:

R_i : **Si** x_1 es $A_{i1} = \{1, 2, 3, 5\}$ **y** x_2 es $A_{i2} = \{3, 5, 6, 8\}$ **entonces** $y = 5x_1 + 2x_2 + 5$

R_j : **Si** x_1 es $A_{j1} = \{2, 4, 5, 6\}$ **y** x_2 es $A_{j2} = \{4, 7, 8, 9\}$ **entonces** $y = 3x_1 + 4x_2 + 3$

Los conjuntos A_{i1} y A_{j1} pueden unirse en otro conjunto C_1 :

$$\begin{aligned} A_{i1} &= \{1, 2, 3, 5\} \\ A_{j1} &= \{2, 4, 5, 6\} \end{aligned} \Rightarrow C_1 = \{1, 3, 4, 6\}$$

y los conjuntos A_{i2} y A_{j2} pueden unirse para formar el conjunto C_2 :

$$\begin{aligned} A_{i2} &= \{3, 5, 6, 8\} \\ A_{j2} &= \{4, 7, 8, 9\} \end{aligned} \Rightarrow C_2 = \{3, 6, 7, 9\}$$

calculando estos valores para $\alpha = 0.5$.

Sustituyendo por los nuevos conjuntos, las reglas quedan de la forma:

R_i : **Si** x_1 es $C_1 = \{1, 3, 4, 6\}$ **y** x_2 es $C_2 = \{3, 6, 7, 9\}$ **entonces** $y = 5x_1 + 2x_2 + 5$

R_j : **Si** x_1 es $C_1 = \{1, 3, 4, 6\}$ **y** x_2 es $C_2 = \{3, 6, 7, 9\}$ **entonces** $y = 3x_1 + 4x_2 + 3$

Como los antecedentes de ambas reglas son ahora iguales, podemos eliminar R_j , sustituyendo el consecuente de R_i por la media de los consecuentes anteriores:

$$\begin{aligned} y &= 5x_1 + 2x_2 + 5 \\ y &= 3x_1 + 4x_2 + 3 \end{aligned} \Rightarrow y = 4x_1 + 3x_2 + 4$$

Por tanto, las dos reglas se unen en una sola regla de la forma:

R_i : **Si** x_1 es $C_1 = \{1, 3, 4, 6\}$ **y** x_2 es $C_2 = \{3, 6, 7, 9\}$ **entonces** $y = 4x_1 + 3x_2 + 4$

5.2.4 Características del algoritmo evolutivo multiobjetivo

El algoritmo propuesto es un algoritmo evolutivo multiobjetivo basado en dominación Pareto para modelización difusa. Se han utilizado, por tanto, algunas de las técnicas desarrolladas en el capítulo tercero para optimización multiobjetivo. El algoritmo ha sido diseñado para encontrar, en una sola ejecución, múltiples soluciones no dominadas de acuerdo con la estrategia de decisión Pareto. No existe dependencia entre las funciones objetivo y el diseño del algoritmo; por tanto, cualquier función objetivo puede incorporarse fácilmente. Sin falta de generalidad, el algoritmo evolutivo minimiza todas las funciones objetivo. Las principales características del algoritmo evolutivo multiobjetivo son las siguientes:

1. Representación de números reales con longitud variable. Cada individuo de la población contiene un número de reglas variable entre 1 y max , donde max viene definido por el decisor. Los conjuntos difusos de los antecedentes y los parámetros de los consecuentes se codifican mediante números reales.
2. La población inicial se genera aleatoriamente con una distribución uniforme dentro de los límites del espacio de búsqueda, definido por los datos de aprendizaje y las restricciones del modelo.
3. Las restricciones con respecto a la estructura del modelo difuso se satisfacen incorporando conocimiento específico sobre el problema. El procedimiento de inicialización y los operadores de variación generan individuos que siempre satisfacen dichas restricciones.
4. La selección y reemplazo de cromosomas se realiza mediante una variante del esquema de preselección. Esta técnica es, implícitamente, una técnica de formación de nichos y una estrategia elitista. Sin embargo, con objeto de mantener una mayor diversidad con respecto al número de reglas de los individuos, se ha añadido una técnica de formación de nichos explícita. La supervivencia de los individuos está basada siempre en el concepto de dominación Pareto.
5. Los operadores de variación del algoritmo evolutivo afectan a los individuos a tres niveles diferentes: (i) nivel del conjunto de reglas, (ii) nivel de reglas, y (iii) nivel de parámetros.

6. Después de realizar la inicialización y también después de aplicar los operadores de variación, se trata cada individuo con la técnica para simplificación del conjunto de reglas difusas descrita en el apartado 5.2.3. Este procedimiento es una técnica ad hoc añadida para mejorar la transparencia y compactitud del modelo difuso.

5.2.5 Representación de soluciones

Para este problema, un individuo está formado por un conjunto de M reglas de la forma:

$$\begin{aligned} R_1 : & \quad A_{11} \ \dots \ A_{1n} \quad \theta_{11} \ \dots \ \theta_{1n} \ \theta_{1(n+1)} \\ & \dots \\ R_M : & \quad A_{M1} \ \dots \ A_{Mn} \quad \theta_{M1} \ \dots \ \theta_{Mn} \ \theta_{M(n+1)} \end{aligned} \tag{5.5}$$

Se utilizan conjuntos difusos trapezoidales en los antecedentes y funciones lineales en los consecuentes. Por tanto, con M reglas y n variables de entrada, un individuo se representa mediante los siguientes números reales:

- Parámetros de los conjuntos difusos trapezoidales A_{ij} de los antecedentes:

$$a_{ij}, b_{ij}, c_{ij}, d_{ij}, \ i = 1, \dots, M, \ j = 1, \dots, n$$

- Coeficientes para las funciones lineales de los consecuentes:

$$\theta_{ij}, \ i = 1, \dots, M, \ j = 1, \dots, n + 1$$

Las restricciones en el dominio de las variables para un modelo difuso vienen dadas por la semántica del conjunto difuso. Un conjunto difuso A_{ij} se representa mediante cuatro valores reales $a_{ij}, b_{ij}, c_{ij}, d_{ij} \in [l_j, u_j]$, con $a_{ij} \leq b_{ij} \leq c_{ij} \leq d_{ij}$.

Los parámetros de los consecuentes son también números reales restringidos a un dominio, es decir, $\theta_{ij} \in [l, u]$.

Otra restricción es el número de reglas M del modelo, que se define entre un número mínimo 1 y un número máximo max fijado por el decisor.

5.2.6 Población inicial

La población inicial se genera aleatoriamente. El número de individuos con M reglas, para $M \in [1, max]$, debe estar entre $minNS$ y $maxNS$ para asegurar la diversidad

con respecto al número de reglas, donde $minNS$ y $maxNS$, con $0 \leq minNS \leq PS/max \leq maxNS \leq PS$ (PS es el tamaño de la población), son los tamaños mínimo y máximos de nicho respectivamente (ver apartado 5.2.8).

Para generar un individuo con M reglas, el procedimiento es como sigue:

1. Para cada conjunto difuso A_{ij} ($i = 1, \dots, M$, $j = 1, \dots, n$), generar cuatro valores reales en el intervalo $[l_j, u_j]$ y ordenarlos para satisfacer las restricciones $a_{ij} \leq b_{ij} \leq c_{ij} \leq d_{ij}$.
2. Generar los parámetros θ_{ij} ($i = 1, \dots, M$, $j = 1, \dots, n + 1$) como valores reales aleatorios en el intervalo $[l, u]$.
3. Simplificar el individuo según el procedimiento descrito en el apartado 5.2.3.

5.2.7 Operadores de variación

Como ya se ha indicado, un individuo es un conjunto de M reglas. Una regla es una colección de n conjuntos difusos (antecedente) y $n+1$ parámetros reales (consecuente). Con objeto de conseguir una adecuada exploración de todas las posibles soluciones en el espacio de búsqueda, es necesario que los operadores de variación trabajen en los diferentes niveles del individuo. En este sentido, se han establecido operadores de variación en los tres niveles del individuo:

1. Nivel del conjunto de reglas.
2. Nivel de reglas.
3. Nivel de parámetros.

Los operadores de variación en el nivel del conjunto de reglas transforman la combinación de reglas para formar el conjunto; los operadores de variación en el nivel de reglas modifican la combinación de conjuntos en los antecedentes y de consecuentes dentro de las reglas y, por último, los operadores de variación en el nivel de parámetros modifican los parámetros dentro de los conjuntos de los antecedentes y dentro de los consecuentes.

Operadores de Variación del Nivel del Conjunto de Reglas

Cruce del Conjunto de Reglas. Este operador intercambia un número aleatorio de reglas entre los padres, pero no intercambia información interna de las reglas. Dados dos padres $I_1 = (R_1^1 \dots R_{M_1}^1)$ e $I_2 = (R_1^2 \dots R_{M_2}^2)$ se generan dos hijos:

$$I_3 = (R_1^1 \dots R_a^1 R_1^2 \dots R_b^2)$$

$$I_4 = (R_{a+1}^1 \dots R_{M_1}^1 R_{b+1}^2 \dots R_{M_2}^2)$$

siendo:

$$a = \text{round}(\alpha M_1)$$

$$b = \text{round}((1 - \alpha) M_2)$$

donde α es un número real aleatorio entre 0 y 1. El número de reglas de los hijos se encuentra, por tanto, entre M_1 y M_2 .

Cruce con Incremento del Conjunto de Reglas. Este operador incrementa el número de reglas de los dos hijos añadiendo a cada padre un número aleatorio de reglas tomadas del otro padre. Dados dos padres $I_1 = (R_1^1 \dots R_{M_1}^1)$ e $I_2 = (R_1^2 \dots R_{M_2}^2)$ se generan dos hijos:

$$I_3 = (R_1^1 \dots R_{M_1}^1 R_1^2 \dots R_a^2)$$

$$I_4 = (R_1^2 \dots R_{M_2}^2 R_1^1 \dots R_b^1)$$

siendo:

$$a = \min\{\max - M_1, M_2\}$$

$$b = \min\{\max - M_2, M_1\}$$

Mutación del Conjunto de Reglas. Este operador borra o añade, con igual probabilidad, una regla del conjunto de reglas. Para borrar, elige una regla aleatoriamente. Para añadir, genera aleatoriamente una regla de acuerdo con el procedimiento de inicialización y la añade al conjunto de reglas. Dado un individuo $I = (R_1 \dots R_M)$ se crea otro individuo I' de la forma:

$$I' = (R_1, \dots, R_{a-1} R_{a+1} \dots R_M), \quad \text{si } \alpha \leq 0.5$$

$$I' = (R_1, \dots, R_M R_{M+1}), \quad \text{en otro caso}$$

siendo α un número real aleatorio entre 0 y 1,

a un índice aleatorio entre 1 y M , y

R_{M+1} una nueva regla creada aleatoriamente según el procedimiento de inicialización.

Operadores de Variación del Nivel de Reglas

Cruce Aritmético de Reglas. Realiza un cruce aritmético entre dos reglas aleatorias. Dados dos padres $I_1 = (R_1^1 \dots R_{M_1}^1)$ e $I_2 = (R_1^2 \dots R_{M_2}^2)$ se generan dos hijos:

$$I_3 = (R_1^1 \dots R_i^3 \dots R_{M_1}^1)$$

$$I_4 = (R_1^2 \dots R_j^4 \dots R_{M_2}^2)$$

donde R_i^3 y R_j^4 se obtienen mediante *cruce aritmético*:

$$R_i^3 = \alpha R_i^1 + (1 - \alpha) R_j^2$$

$$R_j^4 = \alpha R_j^2 + (1 - \alpha) R_i^1$$

siendo α es un número real aleatorio entre 0 y 1,

i, j son índices aleatorios en los intervalos $[1, M_1]$ y $[1, M_2]$ respectivamente.

El producto αR_i se define como otra regla de la forma:

$$\alpha R_i : \alpha A_{i1} \dots \alpha A_{in} \quad \alpha \theta_{i1} \dots \alpha \theta_{in} \quad \alpha \theta_{i(n+1)}$$

El conjunto αA_{ij} se define como otro conjunto difuso con los siguientes parámetros:

$$\alpha A_{ij} = \{\alpha a_{ij}, \alpha b_{ij}, \alpha c_{ij}, \alpha d_{ij}\}$$

Cruce Uniforme de Reglas. Realiza un cruce uniforme entre dos reglas aleatorias. Dados dos padres $I_1 = (R_1^1 \dots R_{M_1}^1)$ e $I_2 = (R_1^2 \dots R_{M_2}^2)$ se generan dos hijos:

$$I_3 = (R_1^1 \dots R_i^3 \dots R_{M_1}^1)$$

$$I_4 = (R_1^2 \dots R_j^4 \dots R_{M_2}^2)$$

donde R_i^3 y R_j^4 se obtienen mediante *cruce uniforme*,

i, j son índices aleatorios en los intervalos $[1, M_1]$ y $[1, M_2]$ respectivamente.

Operadores de Variación del Nivel de Parámetros

Cruce Aritmético. Dados dos padres, elige aleatoriamente una regla de cada padre y realiza un cruce aritmético de los conjuntos difusos correspondientes a una variable aleatoria o de los parámetros del consecuente.

Mutación Uniforme. Este operador cambia el valor de uno de los conjuntos difusos o del consecuente de una regla aleatoria. El nuevo número difuso o el nuevo consecuente se genera aleatoriamente.

Mutación No Uniforme. Este operador cambia mediante mutación no uniforme (ver apartado 3.1.4) el valor de uno de los conjuntos difusos o de un parámetro del consecuente de una regla elegida aleatoriamente.

Mutación Mínima. Similar al anterior, pero con mutación mínima. La mutación mínima (ver apartado 3.1.4) produce un cambio pequeño en los parámetros del individuo y resulta muy adecuada para el ajuste fino de los parámetros reales.

5.2.8 Selección y reemplazo generacional

En cada iteración, el algoritmo evolutivo realiza los siguiente pasos:

1. Seleccionar dos individuos aleatorios de la población como padres.
2. Repetir un número de veces igual a $nChildren$
 - Realizar el cruce y mutación de los dos individuos padres para producir dos individuos hijos.
 - Simplificar los hijos.
3. Reemplazar el primer padre por el mejor de los primeros hijos y el segundo padre por el mejor de los segundos hijos sólo si:
 - el hijo es mejor que el padre, y

- el número de reglas del hijo es igual al número de reglas del padre, o la cuenta de nicho del padre es mayor que $minNS$ y la cuenta de nicho del hijo es menor que $maxNS$.

Un individuo I es mejor que otro individuo J si I domina a J . La cuenta de nicho de un individuo I es el número de individuos de la población con el mismo número de reglas que I .

El esquema de preselección permite mantener cierta diversidad en la población; sin embargo, es necesario añadir un mecanismo de diversidad explícito con respecto al número de reglas de los individuos; dicho mecanismo asegura que el número de individuos con M reglas para todo $M \in [1, max]$, sea siempre mayor o igual que $minNS$ y menor o igual que $maxNS$. Por otra parte, este esquema es también una estrategia elitista, puesto que un individuo de la población es reemplazado sólo por otro individuo mejor.

5.2.9 Modelo de optimización y proceso de decisión

Para establecer el modelo de optimización y el proceso de decisión se han tenido en cuenta las siguientes consideraciones:

1. En lugar de minimizar el número de reglas M se ha decidido buscar soluciones con un número de reglas dentro de un intervalo $[1, max]$ elegido por un decisor. La técnica de formación explícita de nichos asegura que el algoritmo evolutivo siempre contiene en la población un número mínimo de soluciones para cada número de reglas. Por tanto, durante la ejecución del algoritmo, no se realiza la minimización del número de reglas, sino que esto se tendrá en cuenta al final de la ejecución del algoritmo, en un proceso de decisión a posteriori aplicado a la última población.
2. Un modelo que sea muy transparente no será aceptado por un decisor si dicho modelo no es preciso. En la mayoría de problemas de modelización difusa, los valores excesivamente bajos de similaridad impiden obtener buena precisión, por lo que dichos modelos son rechazados. Una estrategia de decisión alternativa, tal como la *programación por metas*, nos permite reducir el dominio de las funciones objetivo de acuerdo a las preferencias de un decisor. Así, podemos

establecer una meta g_s para la similaridad, de forma que la minimización de la similaridad parará en las soluciones que alcancen dicho valor g_s . Se propone $g_s = 0.25$ como valor de meta adecuado para la similaridad, ya que este valor significa que no habrá dos conjuntos difusos solapados en mas de un 25%, lo cual ofrece un modelo con suficiente transparencia.

3. La medida L (número de conjuntos difusos diferentes) se reduce mediante la técnica de simplificación del conjunto de reglas difusas. Por tanto, no se define una función objetivo explícita para minimizar L .

De acuerdo a las consideraciones anteriores, finalmente se establece el siguiente modelo de optimización:

$$\begin{aligned} \text{Minimizar } f_1 &= MSE \\ \text{Minimizar } f_2 &= \max \{g_s, S\} \end{aligned} \quad (5.6)$$

Al final de la ejecución, consideramos el siguiente proceso de decisión a posteriori aplicado a la última población para obtener una solución de compromiso final:

1. Identificar el conjunto $X^* = \{x_1^*, \dots, x_p^*\}$ de soluciones no dominadas de acuerdo al modelo:

$$\begin{aligned} \text{Minimizar } f_1 &= MSE \\ \text{Minimizar } f_2 &= \max \{g_s, S\} \\ \text{Minimizar } f_3 &= M \end{aligned} \quad (5.7)$$

2. Elegir de X^* la solución más precisa x_i^* ; eliminar x_i^* de X^* .
3. Si la solución x_i^* no es suficientemente precisa, o si no existe ninguna solución en el conjunto X^* entonces terminar (no se ha encontrado ninguna solución satisfactoria).
4. Si la solución x_i^* no es suficientemente transparente o compacta, entonces volver al paso 2.
5. Mostrar la solución x_i^* como resultado.

La visualización de los resultados mediante gráficos por ordenador puede resultar una ayuda importante para realizar las decisiones en los pasos 3 y 4.

5.2.10 Experimentos y resultados

En este apartado se aplica el algoritmo evolutivo multiobjetivo para identificar modelos difusos de dos conjuntos de datos estudiados por otros autores. Se utilizan los siguientes valores para los parámetros en la ejecución del algoritmo:

- Tamaño de la población: $N = 100$,
- Número mínimo de individuos para cada número de reglas: $\min NS = 5$,
- Número máximo de individuos para cada número de reglas: $\max NS = 20$,
- Número de descendientes para el esquema de preselección: $nChildren = 10$,
- Probabilidad de cruce: $pCross = 0.8$, y
- Probabilidad de mutación: $pMutate = 0.4$.

Todos los operadores de cruce y mutación se aplican con la misma probabilidad.

Mostramos los resultados obtenidos con el algoritmo utilizando el modelo de optimización descrito en la ecuación (5.6) con los siguientes valores:

- Valor umbral de similaridad: $g_S = 0.25$, y
- Número máximo de reglas: $\max = 5$.

Ejemplo 1

Consideramos la planta no lineal de segundo orden estudiada por Wang y Yen en [Wan99, Yen98]:

$$\begin{aligned}
 y(k) &= g(y(k-1), y(k-2)) + u(k) \\
 \text{con} \quad g(y(k-1), y(k-2)) &= \frac{y(k-1)y(k-2)(y(k-1) - 0.5)}{1 + y^2(k-1) + y^2(k-2)}
 \end{aligned} \tag{5.8}$$

El objetivo es aproximar el componente no lineal de la planta $g(y(k-1), y(k-2))$ mediante un modelo difuso. De igual forma que en [Wan99], se generan 400 puntos de simulación a partir del modelo de planta descrito en la ecuación (5.8). A partir del estado de equilibrio $(0, 0)$, se obtienen 200 muestras de datos de identificación con una

señal de entrada aleatoria $u(k)$ distribuida uniformemente en el intervalo $[-1.5, 1.5]$, seguidos por 200 muestras de datos de evaluación obtenidos utilizando una señal de entrada sinusoidal $u(k) = \sin(2\pi k/25)$. Las señales resultantes y la superficie real se muestran en la figura 5.1.

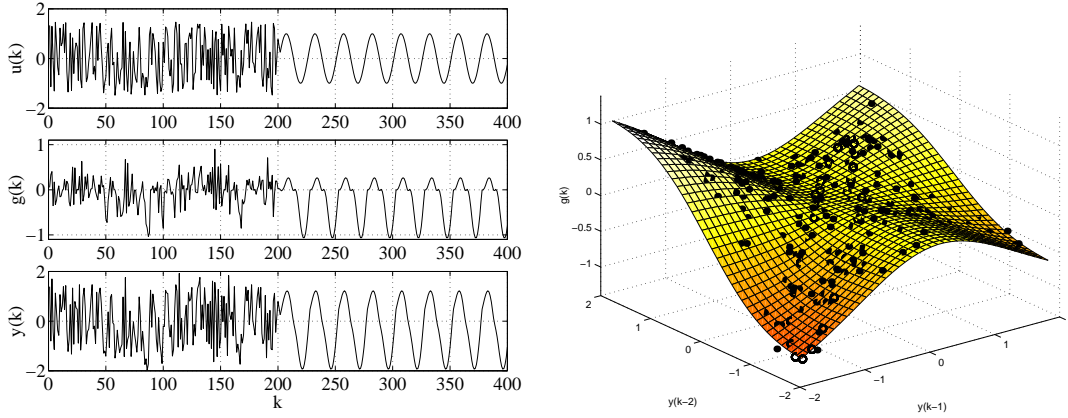


Figura 5.1: Señales resultantes y superficie real para la planta no lineal de segundo orden estudiada por Wang y Yen en [Wan99, Yen98] descrito en la ecuación (5.8). *Izquierda:* Entrada $u(k)$, sistema sin forzar $g(k)$, y salida de la planta $y(k)$ en la ecuación (5.8). *Derecha:* Superficie real.

La tabla 5.7 muestra las soluciones no dominadas de la última población de acuerdo al modelo de optimización descrito en la ecuación (5.7) obtenidas utilizando el algoritmo evolutivo multiobjetivo.

Se puede apreciar la efectividad de la técnica de selección y de la técnica de formación de nichos para mantener la diversidad en la población. El modelo basado en metas tiene la desventaja de que es necesario elegir, a priori, unas buenas metas para el problema, si bien este valor es representativo del grado máximo de solapamiento que el decisor permite a los conjuntos difusos.

De acuerdo con el proceso de decisión descrito, finalmente se elige una solución de compromiso. En este caso se elige la solución con 5 reglas. Se muestra dicha solución en la figura 5.2 mediante diferentes gráficos del modelo obtenido. La figura 5.2(a) muestra los modelos locales aproximados independientemente por cada una de las 5 reglas. Puede observarse cómo dichos modelos locales cubren todo el espacio del modelo global. La superficie generada por el modelo global se muestra en la figura

M	MSE	S
1	$2.783 \cdot 10^{-2}$	0.0
2	$1.116 \cdot 10^{-2}$	0.0
3	$2.238 \cdot 10^{-3}$	0.205
4	$8.505 \cdot 10^{-4}$	0.211
5	$5.926 \cdot 10^{-4}$	0.244

Tabla 5.7: Soluciones no dominadas de acuerdo al modelo de optimización descrito en la ecuación (5.7) obtenidas con el algoritmo evolutivo multiobjetivo para la planta no lineal de segundo orden estudiada por Wang y Yen en [Wan99, Yen98] descrito en la ecuación (5.8).

5.2(b). Por último, los conjuntos difusos para cada variable se muestran en la figura 5.2(c). Para cada variable de entrada se tienen 3 conjuntos difusos y se podría asignar a dichos conjuntos etiquetas lingüísticas de la forma *bajo*, *medio* y *alto*, por lo que resulta un modelo interpretable.

Comparamos nuestros resultados con los obtenidos mediante las cuatro diferentes aproximaciones propuestas en [Yen98] y [Rou00]. Los mejores resultados obtenidos para cada caso se resumen en la tabla 5.8, indicando el número de reglas, número de conjuntos difusos diferentes, tipo de consecuente y error cuadrático medio MSE obtenido para los datos de entrenamiento y de evaluación. En [Yen98], el bajo valor obtenido para MSE en los datos de entrenamiento contrasta con el valor de MSE para los datos de evaluación, lo cual indica un sobreentrenamiento. La solución en [Rou00] es similar a las soluciones obtenidas en este trabajo con respecto a la precisión, transparencia y compactitud, pero en [Rou00] es necesario utilizar un conjunto de técnicas híbridas (clustering difuso inicial y una secuencia de algoritmos evolutivos específicos). Las soluciones en este trabajo se obtienen con un solo algoritmo evolutivo y han sido elegidas entre diferentes alternativas, lo cual es una ventaja para un proceso de decisión adecuado.

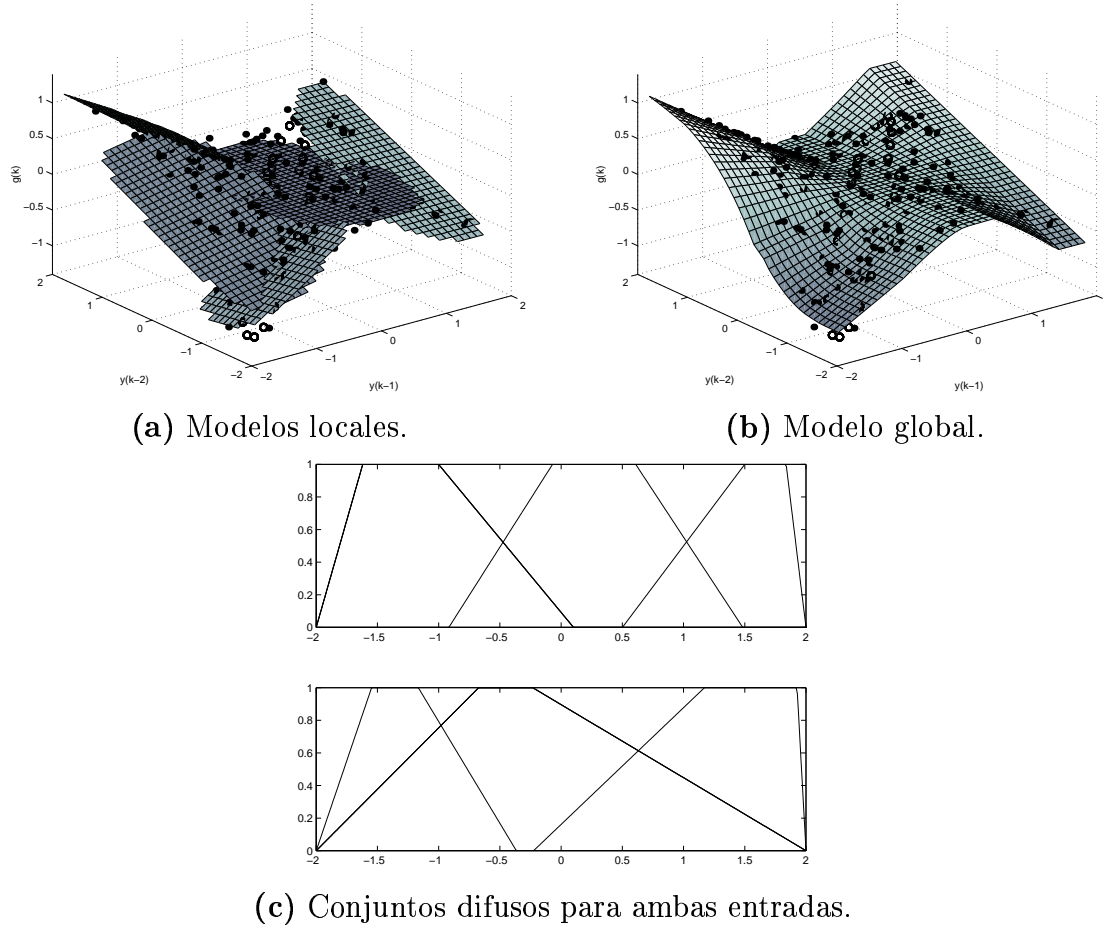


Figura 5.2: Modelo difuso preciso, transparente y compacto obtenido mediante el algoritmo evolutivo multiobjetivo para la planta no lineal de segundo orden estudiada por Wang y Yen en [Wan99, Yen98] descrita en la ecuación (5.8).

Referencia	M	L	MSE^1	MSE^2
[Yen98]	36 (inicial)	12 (B -splines)	$1.9 \cdot 10^{-6}$	$2.9 \cdot 10^{-3}$
	24 (optimizado)	-	$2.0 \cdot 10^{-6}$	$6.4 \cdot 10^{-4}$
[Rou00]	7 (inicial)	14 (triangular)	$1.8 \cdot 10^{-3}$	$1.0 \cdot 10^{-3}$
	5 (optimizado)	5 (triangular)	$5.0 \cdot 10^{-4}$	$4.2 \cdot 10^{-4}$
Nuestro modelo ³	5	6 (trapezoidal)	$5.9 \cdot 10^{-4}$	$8.8 \cdot 10^{-4}$

Tabla 5.8: Modelos difusos para la planta no lineal de segundo orden estudiada por Wang y Yen en [Wan99, Yen98] descrita en la ecuación (5.8). Todos los modelos son del tipo Takagi-Sugeno con consecuentes lineales.

¹ Error cuadrático medio de entrenamiento.

² Error cuadrático medio de evaluación.

³ Solución correspondiente al modelo con 5 reglas en la tabla 5.7, y figura 5.2.

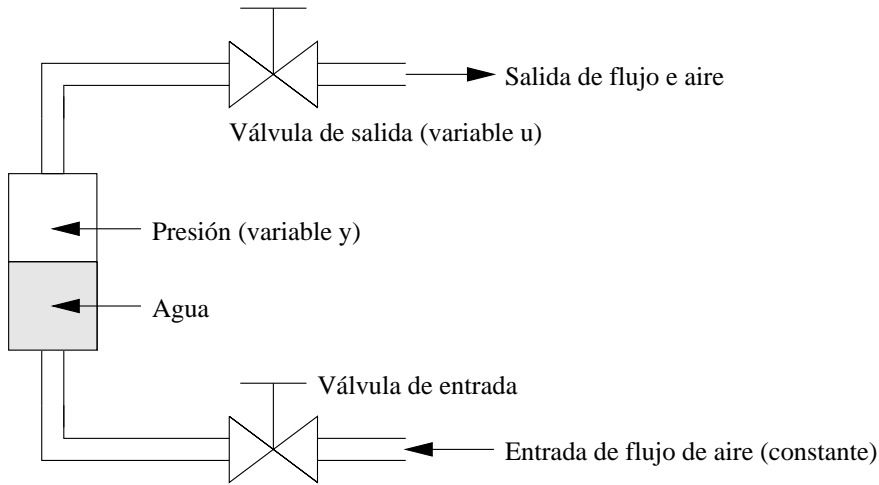


Figura 5.3: Diagrama esquemático del experimento del fermentador.

Ejemplo 2

Este ejemplo está tomado del trabajo de Babuska y Verbruggen [Bab94], donde se trata de identificar el modelo difuso para la dinámica de presión de un fermentador de laboratorio. La figura 5.3 muestra un diagrama esquemático del experimento.

La presión en el tanque de fermentación (variable y) se controla abriendo la válvula de salida (variable u). Asumiendo una serie de simplificaciones, se puede derivar un modelo físico a partir de este proceso. Con un flujo de aire de entrada constante, el establecimiento de la válvula de salida genera un comportamiento transitorio de la presión que puede describirse mediante una ecuación diferencial no lineal de primer orden. Tal como se muestra en [Bab94], un modelo lineal difuso proporciona una solución simple y elegante a este problema. De esta forma, el proceso puede representarse mediante un conjunto de reglas Takagi-Sugeno de la siguiente forma:

$$\begin{aligned}
 R_i : & \text{ Si } y(k) \text{ es } A_i \text{ y } u(k) \text{ es } B_i \text{ entonces} \\
 & y(k+1) = a_i y(k) + b_i u(k) + c_i, \quad i = 1, \dots, M.
 \end{aligned} \tag{5.9}$$

Esta base de reglas representa un modelo de regresión no lineal de primer orden $y(k+1) = f(y(k), u(k))$, donde $y(k)$ y $u(k)$ son la presión y la posición de la válvula de escape en el tiempo k , respectivamente. Las funciones de pertenencia de los antecedentes A_i y B_i , así como los parámetros de los consecuentes a_i , b_i y c_i se

estiman a partir de los datos mediante el algoritmo evolutivo multiobjetivo propuesto en este trabajo.

Para realizar la identificación del modelo, la entrada al sistema se excita mediante una señal sinusoidal con cierta cantidad de frecuencias armónicas elevadas. Por razones técnicas, se aplican diferentes señales de entrada alrededor de los puntos de operación de alta y baja presión separadamente.

Comparamos nuestros resultados con los obtenidos mediante las dos diferentes aproximaciones propuestas en [Bab95] y [Del95]. La solución en [Bab95] se obtiene mediante un clustering difuso de hiperplanos (algoritmo de Gustafson-Kessel [Gus79]) con proyecciones de los clusters difusos en cada dominio y haciendo el cierre extensional de los conjuntos difusos obtenidos para aproximarlos mediante conjuntos difusos trapezoidales. La solución obtenida es transparente y compacta, conteniendo 3 reglas y 6 conjuntos difusos con un error cuadrático medio $MSE = 4.516 \cdot 10^{-4}$. En [Del95], se obtienen los coeficientes de los consecuentes lineales utilizando el algoritmo de mínimos cuadrados recursivo (o un filtro Kalman estacionario), el grado de pertenencia de los datos a los conjuntos difusos de los antecedentes se calcula utilizando directamente el grado de pertenencia de los datos a los clusters difusos encontrados en el espacio producto de las variables de entrada y salida. Más aún, como en este tipo de modelos estamos buscando modelos lineales locales presentes en los datos, se adopta una modificación que consiste en utilizar el algoritmo de las C-medias difuso para inicializar el algoritmo de clustering difuso de Gustafson-Kessel. De esta forma, este algoritmo es adecuado para detectar las particiones difusas que mejor describen los modelos difusos lineales. La solución obtenida es compacta, pero no transparente, contiene 4 reglas y 4 conjuntos difusos n -dimensionales, con un error cuadrático medio $MSE = 6.4 \cdot 10^{-5}$.

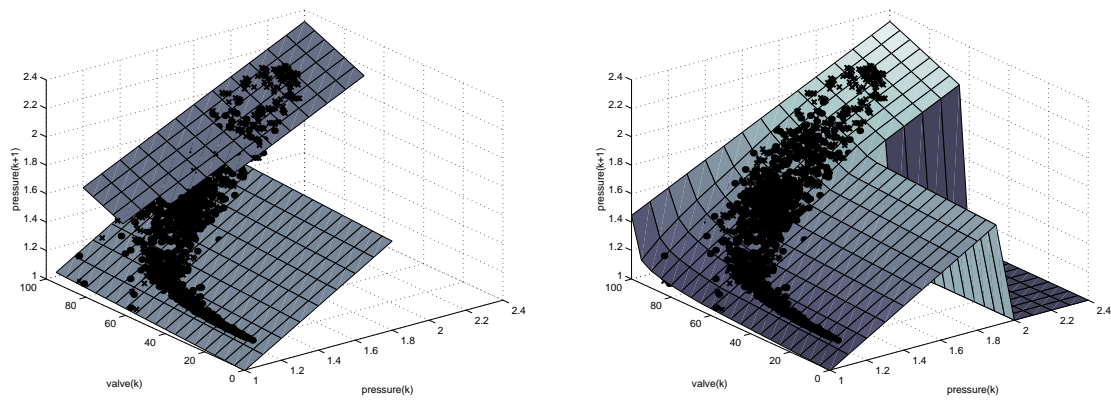
Las soluciones no dominadas de acuerdo al modelo de optimización descrito en la ecuación (5.7) obtenidas con el algoritmo evolutivo multiobjetivo se resumen en la tabla 5.9, donde se indican el número de reglas, número de conjuntos difusos diferentes, error cuadrático medio MSE obtenido para los datos de entrenamiento y similaridad S de los conjuntos difusos.

La solución precisa, transparente y compacta finalmente elegida de acuerdo con la estrategia de decisión a posteriori propuesta para el algoritmo evolutivo multiobjetivo tiene 2 reglas, 4 conjuntos difusos diferentes, error cuadrático medio $MSE = 4.845 \cdot 10^{-5}$ y similaridad $S = 0.235$. Esta solución se muestra en la figura 5.4 median-

M	L	MSE	S
1	2	$9.989 \cdot 10^{-4}$	0.0
2	4	$4.845 \cdot 10^{-5}$	0.235
3	5	$2.778 \cdot 10^{-5}$	0.248
4	6	$2.470 \cdot 10^{-5}$	0.232
5	7	$2.306 \cdot 10^{-5}$	0.232

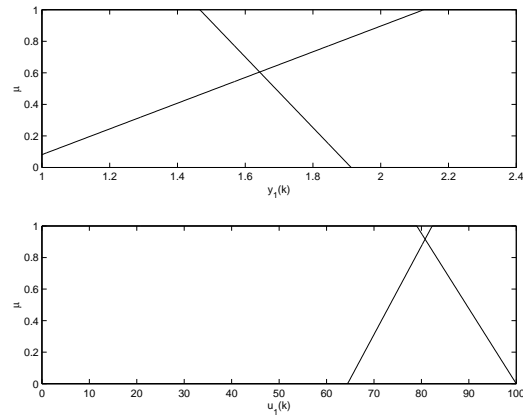
Tabla 5.9: Soluciones no dominadas de acuerdo al modelo de optimización descrito en la ecuación (5.7) obtenidas con el algoritmo evolutivo multiobjetivo para la dinámica de presión de un fermentador de laboratorio descrito en la ecuación (5.9).

te diferentes gráficos para el modelo obtenido. La figura 5.4(a) muestra los modelos locales aproximados independientemente por cada una de las 2 reglas. Puede observarse cómo es suficiente con 2 reglas para cubrir el espacio ocupado por la mayoría de soluciones. La superficie generada por el modelo global se muestra en la figura 5.4(b). Por último, tenemos 2 conjuntos difusos para cada variable de entrada, tal como se muestran en la figura 5.4(c). En este caso, para los conjuntos de la primera variable podrían asignarse etiquetas lingüísticas del tipo *bajo* y *alto*, mientras que para la segunda variable podrían asignarse etiquetas *bajo-medio* y *alto*, dando lugar por tanto a un modelo interpretable.



(a) Modelos locales.

(b) Modelo global.



(c) Conjuntos difusos para ambas entradas.

Figura 5.4: Modelo difuso preciso, transparente y compacto obtenido mediante el algoritmo evolutivo multiobjetivo para la dinámica de presión de un fermentador de laboratorio descrito en la ecuación (5.9).

5.3 Algoritmo Neuro-Evolutivo Multiobjetivo para Modelización Difusa

Si bien se han alcanzado resultados satisfactorios con el algoritmo evolutivo multiobjetivo propuesto en la sección anterior, un aspecto a analizar es si resulta posible mediante hibridación mejorar los resultados obtenidos. Otro aspecto a mejorar del algoritmo evolutivo previo es la técnica de simplificación del conjunto de reglas difusas, ya que ésta tiene repercusiones directas en la transparencia y compactitud de los modelos difusos obtenidos. En esta sección se propone un algoritmo evolutivo multiobjetivo hibridado con una técnica de gradiente con objeto de obtener mayor precisión que con el algoritmo evolutivo multiobjetivo propuesto en la sección 5.2. El algoritmo evolutivo propuesto en esta sección incluye también una técnica para mejorar la transparencia y la compactitud, que es una modificación de la técnica de simplificación, junto con una nueva medida de similaridad. Se considera un modelo difuso basado en reglas del tipo Takagi-Sugeno con consecuentes lineales, pero los antecedentes son conjuntos difusos gaussianos. El modelo de optimización y el proceso de decisión que se sigue es el mismo.

La técnica para mejorar la transparencia y compactitud y el método del gradiente se aplican a cada individuo después de realizar la inicialización y también después de aplicar los operadores de variación. Estos dos métodos permiten mejorar la similaridad y la precisión respectivamente de los individuos.

5.3.1 Identificación del modelo difuso

El modelo difuso que se identifica en este caso está formado por funciones de pertenencia que son gaussianas asimétricas. Se han elegido las funciones gaussianas como un método de representación alternativo a las funciones trapezoidales que, además, permite realizar un entrenamiento de sus parámetros mediante el método de gradiente. Por otra parte, se han elegido gaussianas asimétricas porque esto permite una mayor flexibilidad de los modelos. Cada conjunto difuso A_{ij} se describe mediante

una gaussiana asimétrica con parámetros $\{c_{ij}, \sigma_{ij}^l, \sigma_{ij}^r\}$:

$$\mu_{A_{ij}}(x_j) = \begin{cases} \exp \left[-\frac{1}{2} \left(\frac{x_j - c_{ij}}{\sigma_{ij}^l} \right)^2 \right], & \text{si } x_j \leq c_{ij} \\ \exp \left[-\frac{1}{2} \left(\frac{x_j - c_{ij}}{\sigma_{ij}^r} \right)^2 \right], & \text{si } x_j \geq c_{ij} \end{cases}$$

donde:

$c_{ij} \in [l_j, u_j]$, es el centro,

$\sigma_{ij}^l > 0$, es la varianza izquierda, y

$\sigma_{ij}^r > 0$, es la varianza derecha,

para $i = 1, \dots, M$ y $j = 1, \dots, n$.

Un conjunto difuso gaussiano asimétrico A_{ij} puede describirse también mediante tres valores reales $\{c_{ij}^l, c_{ij}, c_{ij}^r\} \in [l_j, u_j]$, con $c_{ij}^l \leq c_{ij} \leq c_{ij}^r$ cumpliendo la siguiente relación:

$$c_{ij}^l = c_{ij} - n_\sigma \sigma_{ij}^l, \quad c_{ij}^r = c_{ij} + n_\sigma \sigma_{ij}^r$$

Por tanto:

$$\sigma_{ij}^l = \frac{c_{ij} - c_{ij}^l}{n_\sigma}, \quad \sigma_{ij}^r = \frac{c_{ij}^r - c_{ij}}{n_\sigma} \quad (5.10)$$

donde n_σ es una constante, en nuestro caso $n_\sigma = 3$.

Este modelo difuso se define mediante una red neuronal de base radial. El número de neuronas en la capa oculta de una red neuronal RBF es igual al número de reglas del modelo difuso. El grado de disparo de la neurona i -ésima en la capa oculta corresponde con el grado de disparo de la regla i -ésima en el modelo difuso. Utilizamos como función de pertenencia una gaussiana asimétrica definida por sus tres parámetros $\{c, \sigma^l, \sigma^r\}$. Por tanto, cada neurona en la capa oculta tiene estos tres parámetros que definen el valor de su grado de disparo. Las neuronas en la capa de salida realizan los cálculos de la función lineal de primer orden descrita en los consecuentes del modelo difuso, por tanto, la neurona i -ésima en la capa de salida tiene los parámetros $\theta_i = (\theta_{i1}, \dots, \theta_{i(n+1)})$ que corresponden a la función lineal definida en la regla i -ésima del modelo difuso.

5.3.2 Criterios para la modelización difusa

Los criterios para la modelización difusa son los mismos que se proponen en el apartado 5.2.2. La única diferencia es la forma de calcular la similaridad entre dos conjuntos

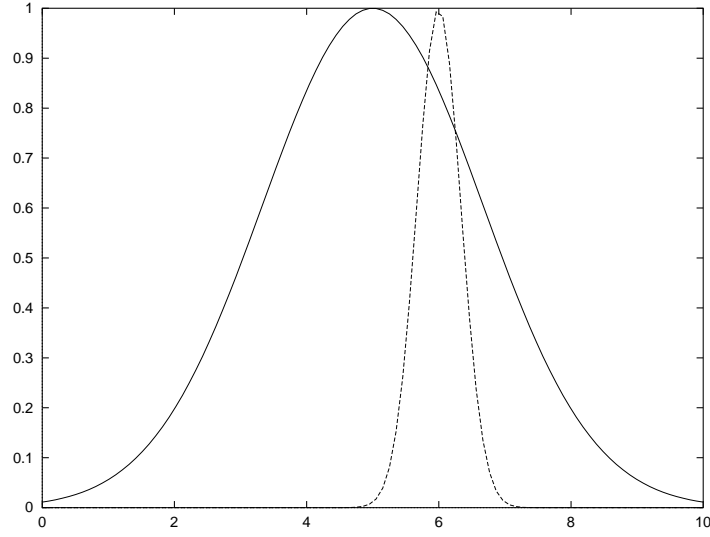


Figura 5.5: Ejemplo de dos conjuntos difusos gaussianos con varianza muy distinta en el que el conjunto de mayor varianza incluye al conjunto de menor.

difusos diferentes. En este caso, para medir la similaridad entre dos conjuntos difusos A y B , se utiliza la siguiente medida de similaridad:

$$S(A, B) = \max \left\{ \frac{|A \cap B|}{|A|}, \frac{|A \cap B|}{|B|} \right\} \quad (5.11)$$

El motivo de variar la forma de calcular la similaridad es para tratar el caso de dos conjuntos difusos con anchura muy distinta, tal como se muestra en la figura 5.5. En este caso el conjunto difuso más ancho puede absorber al conjunto difuso más estrecho, resultando que ambos conjuntos no aportan transparencia al modelo y deben fusionarse en uno sólo. Sin embargo, al calcular la similaridad como la intersección dividida por la unión, se obtiene un valor bajo en este caso, por lo que no se detecta que ambos conjuntos deben fusionarse. Este problema se acentúa y se deja notar cuando los conjuntos difusos son gaussianos. Definiendo la similaridad de la forma mostrada en la ecuación (5.11) se obtiene un valor bajo en este caso, por lo que ambos conjuntos se fusionan en uno sólo.

5.3.3 Mejora de la transparencia y la compactitud del modelo difuso

Como se ha visto en el apartado 5.2.3, dos conjuntos difusos similares pueden fusionarse. Por otra parte, si tenemos dos conjuntos difusos que son parecidos, pero no demasiado, la mejor aproximación consiste en separar ambos conjuntos, de forma que mejore su medida de similaridad. En este caso se sigue un proceso de fusión-separación que se repite hasta obtener un modelo difuso con un valor adecuado de similaridad; en ese momento puede ser que tengamos varias reglas idénticas; en tal caso dichas reglas deben eliminarse del conjunto de reglas. El algoritmo propuesto para mejorar la transparencia y la compactitud del modelo difuso es, por tanto, el siguiente:

1. Mientras existan i, j, k tales que $S(A_{ij}, A_{kj}) > \eta_2$

Si $S(A_{ij}, A_{kj}) > \eta_1$ entonces

Calcular C como fusión de A_{ij} y A_{kj} .

Sustituir A_{ij} y A_{kj} por C .

en otro caso

Separar A_{ij} y A_{kj} .

2. Mientras existan i, k tales que los antecedentes de las reglas R_i y R_k sean iguales

Calcular un nuevo consecuente haciendo la media de los parámetros de los consecuentes de R_i y R_k .

Sustituir el consecuente de R_i por el nuevo consecuente.

Eliminar R_k .

Para medir la similaridad entre dos conjuntos difusos, $S(A, B)$, se utiliza la medida de similaridad definida en la ecuación (5.11). Los valores η_1 y η_2 son los umbrales para realizar la fusión o la separación respectivamente y deben cumplir que $0 < \eta_1 < \eta_2 < 1$.

Si $S(A, B) > \eta_1$ (en nuestro algoritmo utilizamos $\eta_1 = 0.9$), los conjuntos difusos $A = \{c_A, c_A^l, c_A^r\}$ y $B = \{c_B, c_B^l, c_B^r\}$ se fusionan para formar un nuevo conjunto difuso

$C = \{c_C, c_C^l, c_C^r\}$ de la forma:

$$\begin{aligned} c_C &= \alpha c_A + (1 - \alpha) c_B \\ c_C^l &= \min \{c_A^l, c_B^l\} \\ c_C^r &= \max \{c_A^r, c_B^r\} \end{aligned}$$

donde $\alpha \in [0, 1]$ determina la influencia de A y B en el nuevo conjunto difuso C :

$$\alpha = \frac{c_A^r - c_A^l}{c_A^r - c_A^l + c_B^r - c_B^l}$$

Si $\eta_2 < S(A, B) < \eta_1$ (utilizamos $\eta_2 = 0.6$), los conjuntos difusos A y B se separan de la forma:

$$\begin{aligned} \text{si } c_A < c_B \text{ entonces } & \sigma_A^r \leftarrow \sigma_A^r (1 - \beta), \quad \sigma_B^l \leftarrow \sigma_B^l (1 - \beta) \\ \text{en otro caso } & \sigma_A^l \leftarrow \sigma_A^l (1 - \beta), \quad \sigma_B^r \leftarrow \sigma_B^r (1 - \beta) \end{aligned}$$

donde $\beta \in [0, 1]$ indica la cantidad de separación entre A y B ; en nuestro algoritmo, utilizamos $\beta = 0.1$.

5.3.4 Entrenamiento de la red neuronal RBF

Tal como vimos en el apartado 4.1.4, la red neuronal RBF asociada con el modelo difuso puede entrenarse mediante un método de gradiente para obtener más precisión. Sin embargo, con el fin de mantener la transparencia y compactitud de los conjuntos difusos, sólo se realiza el entrenamiento de los parámetros de los consecuentes.

El algoritmo de entrenamiento actualiza de forma incremental los parámetros en base a los patrones de entrenamiento que se le presentan. Los parámetros de la red neuronal se actualizan aplicando a la función de error el método del descenso por gradiente. La función de error para el patrón de entrenamiento i -ésimo viene dada por la función MSE definida en la ecuación (5.3). La regla para realizar la actualización es la siguiente:

$$\theta_{ij} \leftarrow \theta_{ij} + \eta \Delta \theta_{ij} = \theta_{ij} - \eta \frac{\partial MSE}{\partial \theta_{ij}}$$

donde $i = 1, \dots, M$, $j = 1, \dots, n + 1$ y η es el ratio de aprendizaje. Esta regla se aplica durante un número de iteraciones (epochs). Nuestro algoritmo utiliza un valor

$\eta = 0.01$ y un número de 10 epochs. Los gradientes negativos del MSE respecto a cada parámetro se calculan de la siguiente forma:

$$\Delta\theta_{ij} = -\frac{\partial MSE}{\partial\theta_{ij}} = (t_k - y_k) \frac{1}{z} \mu_i(\mathbf{x}) x_j$$

$$\Delta\theta_{i(n+1)} = -\frac{\partial MSE}{\partial\theta_{i(n+1)}} = (t_k - y_k) \frac{1}{z} \mu_i(\mathbf{x})$$

donde $i = 1, \dots, M$, $j = 1, \dots, n$, $\mu_i(\mathbf{x})$ es el valor de disparo para la regla i -ésima definida tal como indica la ecuación (5.2), y $z = \sum_{i=1}^M \mu_i(\mathbf{x})$.

5.3.5 Representación de soluciones

La representación de soluciones se realiza codificando los números difusos gaussianos asimétricos que forman en conjunto una red neuronal RBF. Un individuo está formado por un conjunto de M reglas definidas por los pesos de la red neuronal RBF. Con n variables de entrada, tenemos, para cada individuo, los siguientes parámetros:

- Parámetros de los conjuntos difusos gaussianos A_{ij} de los antecedentes:

$$c_{ij}, \sigma_{ij}^l, \sigma_{ij}^r, i = 1, \dots, M, j = 1, \dots, n$$

- Coeficientes para las funciones lineales de los consecuentes:

$$\theta_{ij}, i = 1, \dots, M, j = 1, \dots, n + 1$$

5.3.6 Población inicial

La inicialización de los individuos se realiza generando individuos con diferente número de reglas. Para generar un individuo con M reglas, el procedimiento es como sigue:

1. Para cada conjunto difuso A_{ij} ($i = 1, \dots, M$, $j = 1, \dots, n$), generar tres valores reales en el intervalo $[l_j, u_j]$ y ordenarlos para satisfacer las restricciones $c_{ij}^l \leq c_{ij} \leq c_{ij}^r$. Los parámetros del conjunto gaussiano σ_{ij}^l y σ_{ij}^r se calculan de la forma descrita en la ecuación (5.10).
2. Generar los parámetros θ_{ij} ($i = 1, \dots, M$, $j = 1, \dots, n + 1$) como valores reales aleatorios en el intervalo $[l, u]$.

3. Tratar el individuo con la técnica para mejorar la transparencia y la compactitud descrita en el apartado 5.3.3.
4. Entrenar el individuo mediante la técnica de gradiente descrita en el apartado 5.3.4.

5.3.7 Operadores de variación

Los operadores de variación se definen de igual forma que en el apartado 5.2.7. La diferencia se encuentra en los operadores que trabajan con los parámetros de los conjuntos difusos, puesto que ahora están definidos sobre conjuntos gaussianos, por lo que la forma de inicializarlos y sus restricciones se adaptan a los parámetros de los conjuntos difusos gaussianos.

5.3.8 Selección y reemplazo generacional

Para realizar la selección y reemplazo generacional, en cada iteración el algoritmo neuro-evolutivo multiobjetivo realiza los siguiente pasos:

1. Seleccionar dos individuos aleatorios de la población como padres.
2. Repetir un número de veces igual a $nChildren$
 - Realizar el cruce y mutación de los dos individuos padres para producir dos individuos hijos.
 - Ejecutar la técnica para mejorar la transparencia y compactitud en los hijos.
 - Realizar el entrenamiento de los hijos mediante la técnica del gradiente.
3. Reemplazar el primer padre por el mejor de los primeros hijos y el segundo padre por el mejor de los segundos hijos sólo si:
 - el hijo es mejor que el padre, y
 - el número de reglas del hijo es igual al número de reglas del padre, o la cuenta de nicho del padre es mayor que $minNS$ y la cuenta de nicho del hijo es menor que $maxNS$.

5.3.9 Modelo de optimización y proceso de decisión

El modelo de optimización y el proceso de decisión a posteriori es el mismo que se propone en el apartado 5.2.9, pero con la diferencia de que el valor de similaridad S se calcula según la ecuación (5.11). En este caso S tomará valores ligeramente superiores para modelos con similar transparencia. El valor de meta g_s que se propone debe ser, por tanto, algo más alto que el propuesto en el apartado 5.2.9. En este caso el valor $g_s = 0.35$ da resultados de transparencia similares a los obtenidos con el modelo anterior.

5.3.10 Experimentos y resultados

En esta sección se aplica el algoritmo neuro-evolutivo multiobjetivo para identificar modelos difusos en los mismos conjuntos de datos planteados en el apartado 5.2.10. Los valores de los parámetros utilizados para la ejecución del algoritmo son los siguientes:

- Tamaño de la población: $N = 100$,
- Número mínimo de individuos para cada número de reglas: $minNS = 5$,
- Número máximo de individuos para cada número de reglas: $maxNS = 30$,
- Número de descendientes para el esquema de preselección: $nChildren = 10$,
- Probabilidad de cruce: $pCross = 0.9$, y
- Probabilidad de mutación: $pMutate = 0.9$.

Todos los operadores de cruce y mutación se aplican con la misma probabilidad.

Mostramos los resultados obtenidos con el algoritmo utilizando el modelo de optimización descrito en la ecuación (5.6) con los siguientes valores:

- Valor umbral de similaridad: $g_s = 0.35$, y
- Número máximo de reglas: $max = 5$.

Ejemplo 1

Se trata de resolver el mismo problema propuesto como primer ejemplo en el apartado 5.2.10, es decir, aproximar el componente no lineal de la planta estudiada por Wang y Yen en [Wan99, Yen98] y descrita en la ecuación (5.8).

La tabla 5.10 muestra las soluciones no dominadas de la última población de acuerdo al modelo de optimización descrito en la ecuación (5.7) obtenidas utilizando el algoritmo neuro-evolutivo multiobjetivo.

De acuerdo con el proceso de decisión descrito, finalmente se elige la solución con 5 reglas. Esta solución se muestra en la figura 5.6 mediante diferentes gráficos del modelo obtenido. La figura 5.6(a) muestra los modelos locales aproximados independientemente por cada una de las reglas que, como puede observarse, cubren todo el espacio ocupado por las soluciones. La superficie generada por el modelo global se muestra en la figura 5.6(b). Finalmente, los conjuntos difusos para cada variable de entrada se muestran en la figura 5.6(c). Se tienen 3 conjuntos difusos para la primera variable de entrada a los que podrían asignarse las etiquetas lingüísticas *bajo*, *alto* y *muy alto* y 2 conjuntos difusos para la segunda variable de entrada a los que pueden asignarse las etiquetas *bajo* y *alto*. El modelo es, por tanto, interpretable.

Si comparamos los resultados obtenidos por los dos algoritmos propuestos en este trabajo, podemos observar que ambos algoritmos obtienen modelos transparentes, si bien el algoritmo neuro-evolutivo consigue un mejor valor de precisión.

M	L	MSE	S
1	2	$4.273 \cdot 10^{-2}$	0.0
2	3	$5.594 \cdot 10^{-3}$	0.350
3	5	$4.221 \cdot 10^{-3}$	0.350
4	4	$1.888 \cdot 10^{-4}$	0.346
5	5	$1.765 \cdot 10^{-4}$	0.348

Tabla 5.10: Soluciones no dominadas de acuerdo al modelo de optimización descrito en la ecuación (5.7) obtenidas con el algoritmo neuro-evolutivo multiobjetivo para la planta no lineal de segundo orden estudiada por Wang y Yen en [Wan99, Yen98] descrita en la ecuación (5.8).

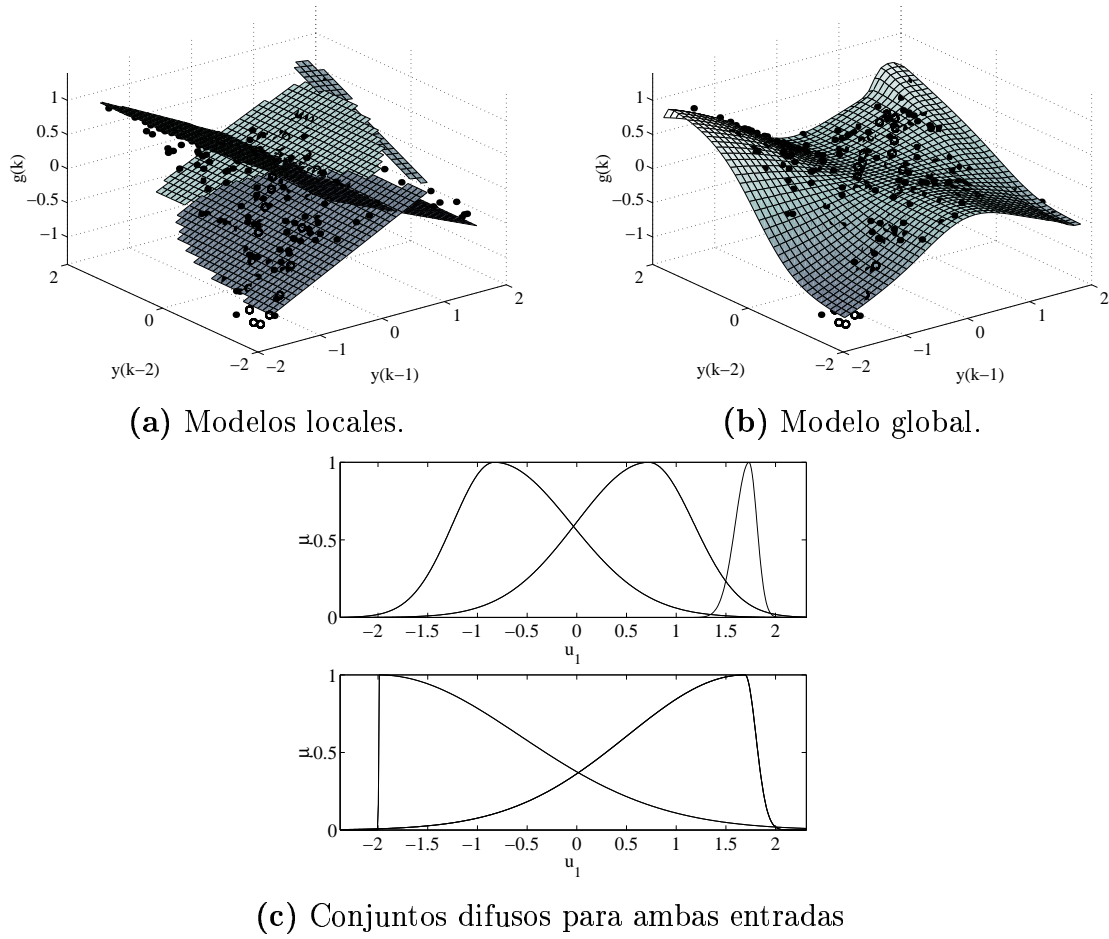


Figura 5.6: Modelo difuso preciso, transparente y compacto obtenido mediante el algoritmo neuro-evolutivo multiobjetivo para la planta no lineal de segundo orden estudiada por Wang y Yen en [Wan99, Yen98] descrita en la ecuación (5.8).

Ejemplo 2

Este es el mismo problema propuesto como segundo ejemplo en el apartado 5.2.10. Se trata de realizar un modelo difuso para la dinámica de presión de un fermentador de laboratorio descrito en la ecuación (5.9).

Las soluciones no dominadas de acuerdo al modelo de optimización descrito en la ecuación (5.7) obtenidas con el algoritmo neuro-evolutivo multiobjetivo se muestran en la tabla 5.11.

La solución elegida para el algoritmo neuro-evolutivo multiobjetivo de acuerdo con la estrategia de decisión a posteriori tiene 3 reglas, 5 conjuntos difusos diferentes, error cuadrático medio $MSE = 2.737 \cdot 10^{-5}$ y similaridad $S = 0.350$. Esta solución se muestra en la figura 5.7 mediante diferentes gráficos para el modelo obtenido. La figura 5.7(a) muestra los modelos locales aproximados independientemente por cada una de las 3 reglas. La superficie generada por el modelo global se muestra en la figura 5.7(b). Por último, los conjuntos difusos para cada variable de entrada se muestran en la figura 5.7(c). Se obtienen 2 conjuntos difusos para la primera variable, a los que se puede asignar las etiquetas lingüísticas *bajo-medio* y *alto* y 3 conjuntos difusos para la segunda variable, que podrían tener las etiquetas *bajo*, *medio* y *alto*. El modelo, por tanto, resulta interpretable. Comparándola con la solución obtenida con el algoritmo evolutivo, vemos que esta solución tiene valores similares en cuanto a transparencia y compactitud, siendo más precisa.

M	L	MSE	S
1	2	$6.540 \cdot 10^{-4}$	0.0
2	4	$6.374 \cdot 10^{-5}$	0.188
3	5	$2.737 \cdot 10^{-5}$	0.350
4	6	$2.399 \cdot 10^{-5}$	0.350
5	7	$2.266 \cdot 10^{-5}$	0.349

Tabla 5.11: Soluciones no dominadas de acuerdo al modelo de optimización descrito en la ecuación (5.7) obtenidas con el algoritmo neuro-evolutivo multiobjetivo para la dinámica de presión de un fermentador de laboratorio descrito en la ecuación (5.9).

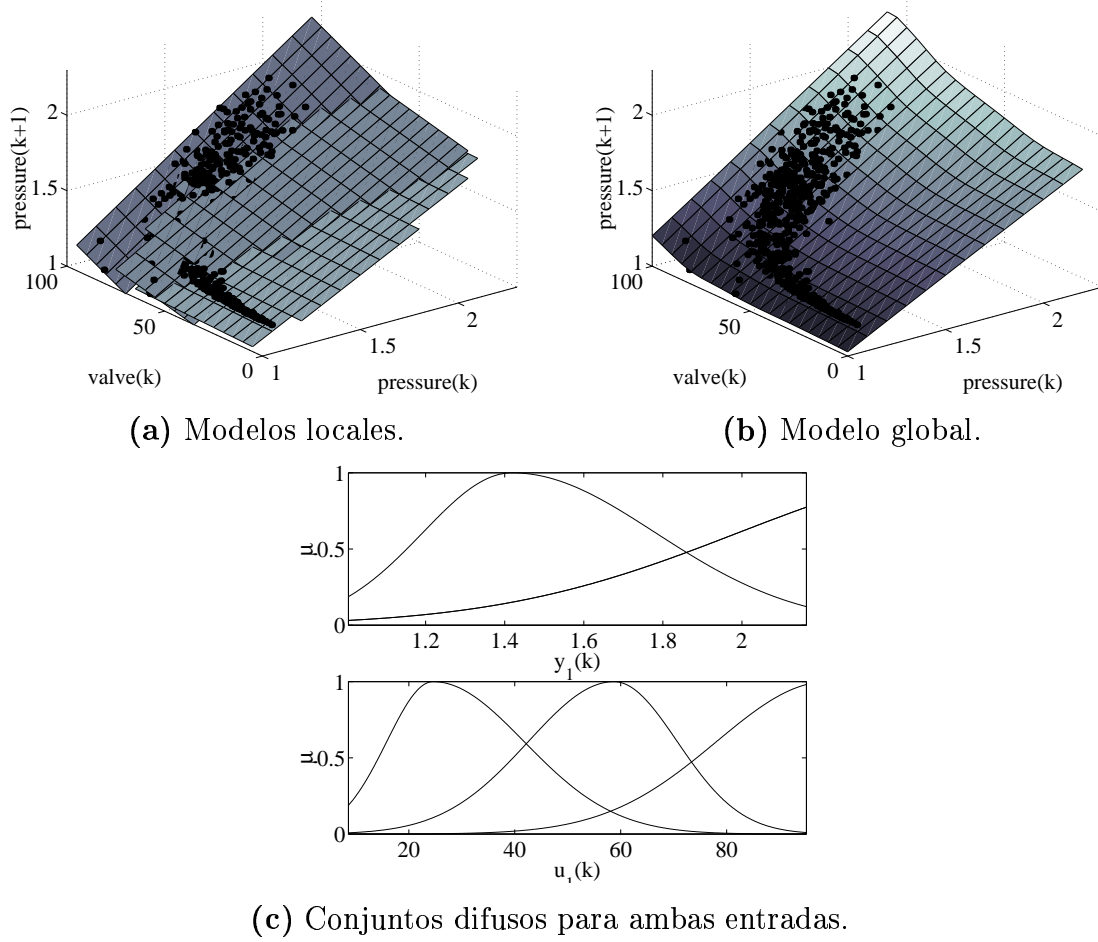


Figura 5.7: Modelo difuso preciso, transparente y compacto obtenido mediante el algoritmo neuro-evolutivo multiobjetivo para la dinámica de presión de un fermentador de laboratorio descrito en la ecuación (5.9).

5.3.11 Adaptación del algoritmo para la obtención de modelos aproximativos

El algoritmo neuro-evolutivo puede utilizarse teniendo en cuenta solamente la precisión. En este caso no se tienen en cuenta aspectos de interpretabilidad y se obtienen modelos aproximativos, con mayor precisión, pero que no resultan interpretables. Para ello, se ejecuta el algoritmo con las siguientes características:

- Se establece el valor $g_s = 1.0$, que es lo mismo que eliminar el objetivo de similaridad.
- Se elimina el procedimiento para mejorar la transparencia de los conjuntos difusos.
- Se añade entrenamiento por gradiente también de los conjuntos difusos de los antecedentes. La regla para realizar la actualización de los parámetros de los conjuntos antecedentes es la siguiente:

$$c_{ij} \leftarrow c_{ij} + \eta \Delta c_{ij}, \quad \sigma_{ij}^l \leftarrow \sigma_{ij}^l + \eta \Delta \sigma_{ij}^l, \quad \sigma_{ij}^r \leftarrow \sigma_{ij}^r + \eta \Delta \sigma_{ij}^r$$

donde $i = 1, \dots, M$, $j = 1, \dots, n$ y η es el ratio de aprendizaje. Los gradientes negativos del MSE respecto a cada uno de los parámetros de los antecedentes se calculan de la siguiente forma:

$$\Delta c_{ij} = -\frac{\partial MSE}{\partial c_{ij}} = \begin{cases} (t_k - y_k) \frac{f_i(\mathbf{x}) - y_k}{z} \mu_i(\mathbf{x}) \frac{x_j - c_{ij}}{(\sigma_{ij}^l)^2}, & \text{si } x_j < c_{ij} \\ (t_k - y_k) \frac{f_i(\mathbf{x}) - y_k}{z} \mu_i(\mathbf{x}) \frac{x_j - c_{ij}}{(\sigma_{ij}^r)^2}, & \text{en otro caso} \end{cases}$$

$$\Delta \sigma_{ij}^l = -\frac{\partial MSE}{\partial \sigma_{ij}^l} = \begin{cases} (t_k - y_k) \frac{f_i(\mathbf{x}) - y_k}{z} \mu_i(\mathbf{x}) \frac{(x_j - c_{ij})^2}{(\sigma_{ij}^l)^3}, & \text{si } x_j < c_{ij} \\ 0, & \text{en otro caso} \end{cases}$$

$$\Delta \sigma_{ij}^r = -\frac{\partial MSE}{\partial \sigma_{ij}^r} = \begin{cases} 0, & \text{si } x_j < c_{ij} \\ (t_k - y_k) \frac{f_i(\mathbf{x}) - y_k}{z} \mu_i(\mathbf{x}) \frac{(x_j - c_{ij})^2}{(\sigma_{ij}^r)^3}, & \text{en otro caso} \end{cases}$$

donde $i = 1, \dots, M$, $j = 1, \dots, n$, $\mu_i(\mathbf{x})$ es el valor de disparo para la regla i -ésima definida tal como indica la ecuación (5.2), y $z = \sum_{i=1}^M \mu_i(\mathbf{x})$.

5.4 Experimentos y Resultados Adicionales

Con objeto de ampliar la evaluación de los algoritmos propuestos a otros casos, en esta sección se plantean dos nuevos problemas clásicos de la literatura y un problema de una aplicación real, y se ejecutan los algoritmos para realizar modelos difusos.

Los valores de los parámetros utilizados en la ejecución de los algoritmos son los mismos que se muestran en los apartados 5.2.10 y 5.3.10.

Ejemplo 3

Para este ejemplo hemos considerado la modelización de la base de reglas dada por Sugeno en [Sug88]:

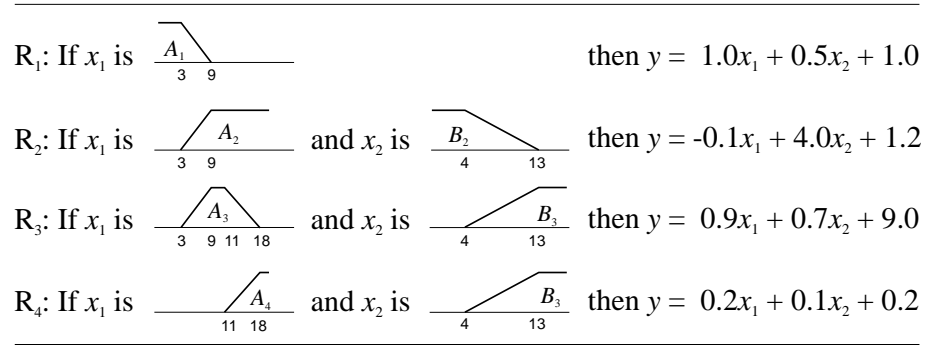


Figura 5.8: Modelización de la base de reglas dado en [Sug88].

La superficie correspondiente se muestra en la figura 5.9.

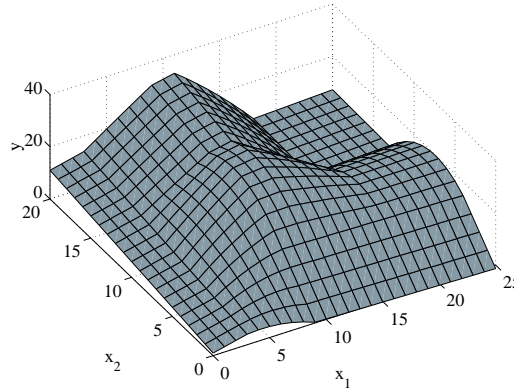


Figura 5.9: Superficie real para el ejemplo en [Sug88].

En [Set99] se identifica un modelo con cuatro reglas a partir de los datos de muestra ($N = 546$) mediante el algoritmo de clustering supervisado, el cual se inicializa con 12 clusters. Este modelo se optimiza utilizando un algoritmo evolutivo para obtener al final un MSE igual a 1.6. La tabla 5.12 muestra los resultados obtenidos con el algoritmo evolutivo multiobjetivo y la tabla 5.13 muestra los resultados obtenidos con el algoritmo neuro-evolutivo multiobjetivo. Comparando los resultados obtenidos con ambos algoritmos se puede observar que ambos algoritmos obtienen modelos transparentes alcanzando en ambos casos los valores óptimos de similaridad. Ambos algoritmos obtienen también buenos valores de precisión, aunque el algoritmo neuro-evolutivo obtiene soluciones más precisas.

Finalmente, escogemos una solución de compromiso. Para ambos algoritmos se elige el modelo difuso con 4 reglas. La figura 5.10 muestra los modelos locales aproximados independientemente por cada una de las reglas, la superficie generada por el modelo global, y los conjuntos difusos para cada variable para la solución escogida calculada por el algoritmo evolutivo multiobjetivo. La figura 5.11 muestra lo mismo para la solución escogida calculada por el algoritmo neuro-evolutivo multiobjetivo. En ambos casos tenemos 5 conjuntos difusos, 3 conjuntos para la primera variable de entrada a los que se podrían asignar las etiquetas *bajo*, *medio* y *alto* y 2 conjuntos para la segunda variable de entrada a los que se podría etiquetar como *bajo* y *alto*. Ambos modelos resultan, por tanto, interpretables.

M	L	MSE	S
1	2	56.696	0.0
2	3	14.226	0.042
3	4	2.945	0.134
4	5	1.233	0.249
5	5	0.993	0.247

Tabla 5.12: Soluciones no dominadas de acuerdo al modelo de optimización descrito en la ecuación (5.7) obtenidas con el algoritmo evolutivo multiobjetivo para la modelización de la base de reglas dada en [Sug88] y mostrado en la figura 5.8.

M	L	MSE	S
1	2	125.291	0.0
2	4	25.606	0.348
3	5	2.187	0.349
4	5	1.017	0.349
5	5	0.910	0.350

Tabla 5.13: Soluciones no dominadas de acuerdo al modelo de optimización descrito en la ecuación (5.7) obtenidas con el algoritmo neuro-evolutivo multiobjetivo para la modelización de la base de reglas dada en [Sug88] y mostrado en la figura 5.8.

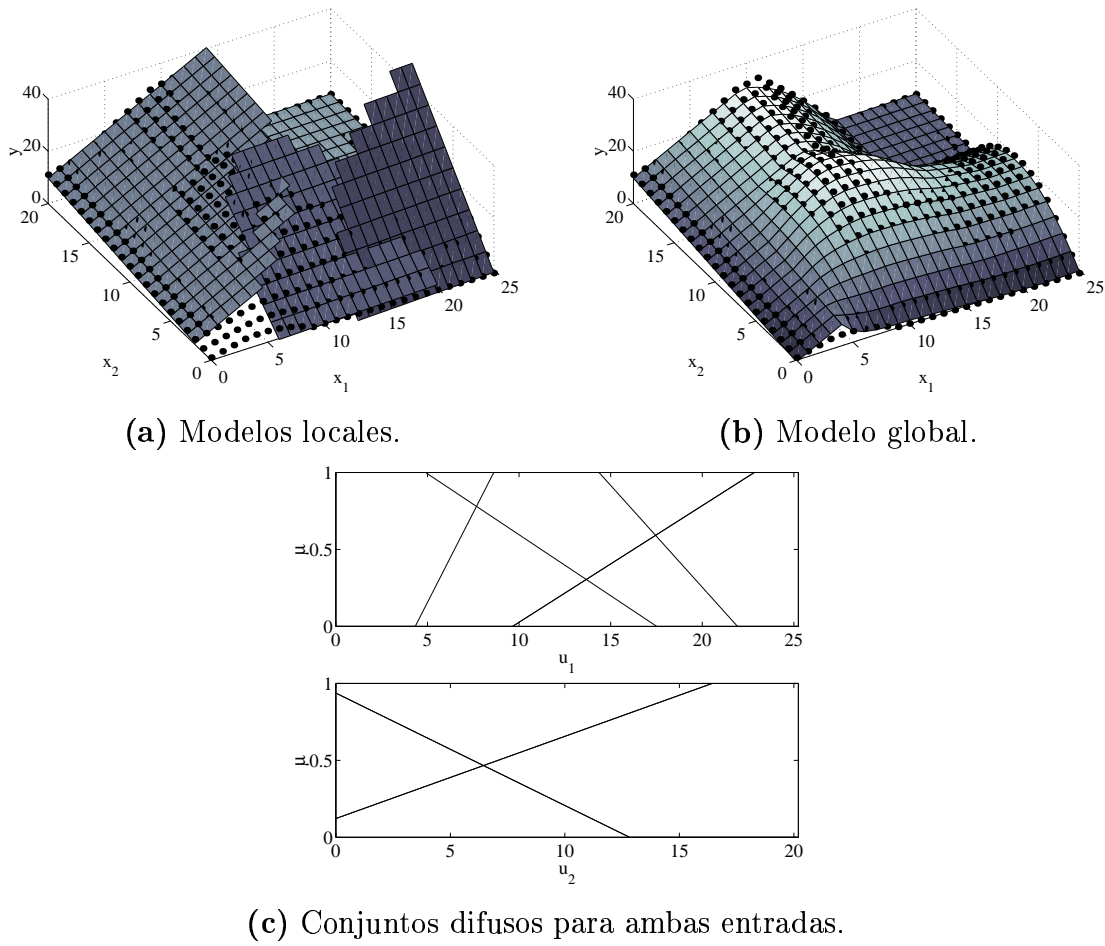


Figura 5.10: Modelo difuso preciso, transparente y compacto obtenido mediante el algoritmo multiobjetivo para la modelización de la base de reglas dada en [Sug88] y mostrada en la figura 5.8.

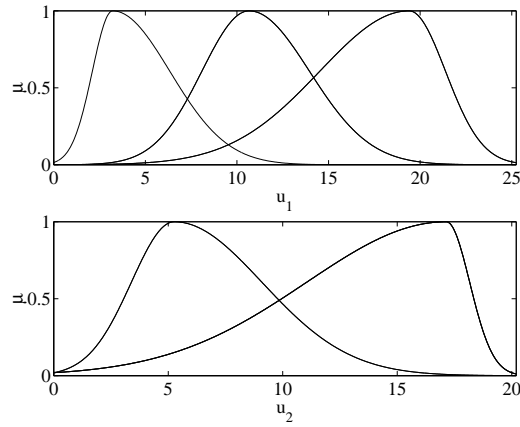
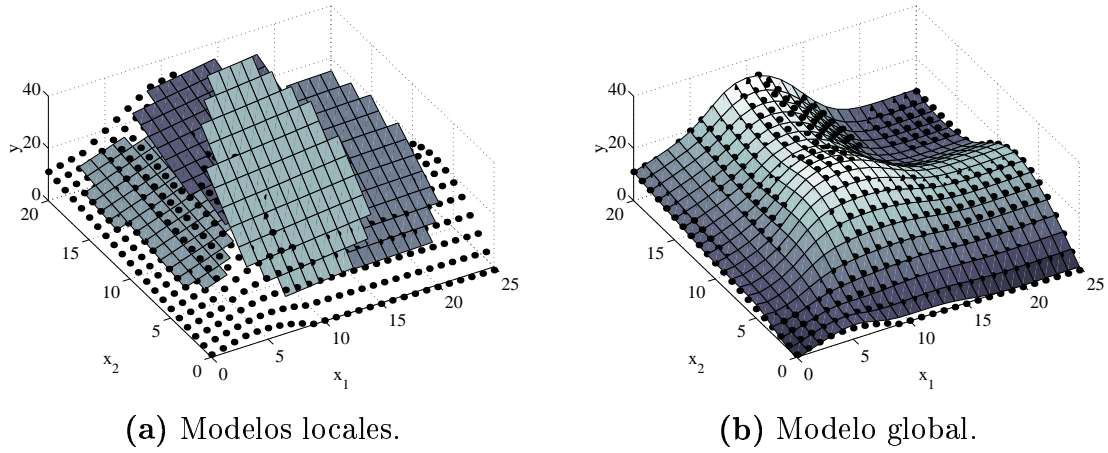


Figura 5.11: Modelo difuso preciso, transparente y compacto obtenido mediante el algoritmo neuro-evolutivo multiobjetivo para la modelización de la base de reglas dada en [Sug88] y mostrada en la figura 5.8.

M	MSE
1	63.2826
2	5.9119
3	0.6766
4	0.2078
5	0.1403

Tabla 5.14: Soluciones no dominadas de acuerdo al modelo de optimización descrito en la ecuación (5.7) sin tener en cuenta la similaridad ($g_s = 1.0$) obtenidas con el algoritmo neuro-evolutivo multiobjetivo para la modelización de la base de reglas dada en [Sug88] y mostrada en la figura 5.8.

Se ha ejecutado también el algoritmo neuro-evolutivo para obtener modelos aproximativos, según la adaptación del algoritmo descrita en el apartado 5.3.11. Los resultados obtenidos para un número máximo de 5 neuronas internas se muestran en la tabla 5.14. Puede observarse que, tal como era de esperar, se produce una mejora sustancial en la precisión de los modelos obtenidos, aunque se pierden los aspectos de interpretabilidad. Esta aproximación puede resultar interesante cuando se desea obtener la máxima precisión en un modelo sin tener en cuenta criterios de interpretabilidad. Por otra parte, también puede resultar una aproximación interesante si se desea reducir el número de neuronas internas, puesto que en este caso con un menor número de neuronas se obtiene una mayor precisión.

Ejemplo 4

La mayoría del agua consumida por las plantas se evapora a la atmósfera de dos formas: a través de la superficie de la planta, mediante un proceso denominado transpiración, y por evaporación desde la superficie de cultivo. El proceso global se denomina evapotranspiración y determina la cantidad de agua que necesita la planta.

Para determinar las necesidades de los cultivos, a partir de un conjunto de datos agroclimáticos y utilizando expresiones definidas por organismos internacionales como la FAO¹, se calcula el valor del parámetro ET_0 (evapotranspiración de referencia). Dicho valor es fundamental, ya que es la base para determinar las necesidades hídricas de los cultivos.

En este ejemplo, para aproximar el valor del parámetro ET_0 se utiliza el *modelo evaporimétrico de cubeta Clase A* [Smi96]. Este modelo se basa en el empleo de una cubeta calibrada que contiene un determinado volumen de agua y está convenientemente localizada en las proximidades de una estación climática. Mediante la observación de diferencias en el nivel de agua de la cubeta se puede medir el nivel de evaporación. A partir de estas medidas y de ciertas correcciones se puede calcular un determinado valor de ET_0 . La ecuación principal de este modelo es la siguiente:

$$ET_0 = K_p E_0 \quad (5.12)$$

donde E_0 es el nivel de evaporación de la cubeta, en $mm/día$, y

K_p es el coeficiente corrector del tanque definido por:

$$K_p = a_0 + a_1 U + a_2 H_r + a_3 d + a_4 H_r^2 + a_5 d^2 + a_6 U H_r^2 + a_7 H_r^2 \quad (5.13)$$

siendo U la velocidad media del viento,

H_r la humedad relativa media del aire, y

d la distancia a barlovento de la cubeta a la cubierta vegetal.

El ajuste de la ecuación (5.13) depende de las condiciones particulares de cada zona y no resulta una tarea sencilla. Los expertos deben ajustar los modelos teniendo en cuenta básicamente su experiencia, usando un sistema de ensayo y error. Por ello, dentro del grupo de investigación de Sistemas Inteligente del Departamento de

¹Food and Agriculture Organization of the United Nations.

M	L	MSE	S
1	3	0.1194	0.0
2	4	0.0917	0.334
3	7	0.0839	0.344

Tabla 5.15: Soluciones no dominadas de acuerdo al modelo de optimización descrito en la ecuación (5.7) obtenidas con el algoritmo neuro-evolutivo multiobjetivo para estimar la evapotranspiración según el modelo de cubeta clase A mostrado en las ecuaciones (5.12) y (5.13).

Ingeniería de la Información y las Comunicaciones de la Universidad de Murcia² se ha planteado utilizar diferentes técnicas inteligentes para obtener modelos que ajusten el valor de ET_0 . Las técnicas propuestas son:

- Cubeta A, utilizar directamente el modelo clásico de Cubeta Clase A,
- Red Neuronal MLP (Perceptrón Multicapa), entrenada mediante gradiente,
- Red Neuronal RBF (Función de Base Radial), entrenada mediante gradiente,
- Red Neuronal ANFIS (Sistema de inferencia neuro-difuso), entrenada mediante un clustering previo más gradiente.

Otra opción es utilizar el algoritmo neuro-evolutivo multiobjetivo para obtener un modelo difuso y también interpretable. Los resultados se han obtenido utilizando la base de datos agroclimáticos del CIDA³ procedentes de las 42 estaciones automáticas que componen la Red Climática de la región de Murcia y que dispone de series temporales horarias desde 1996 sobre diversas variables climáticas. En concreto, utilizamos un conjunto de 2580 datos desde que van desde el año 1996 hasta el año 1999.

Las soluciones no dominadas encontradas por el algoritmo neuro-evolutivo se muestran en la tabla 5.15. Se elige como solución más adecuada la que tiene 3 reglas con $MSE = 0.0839$. Esta solución es transparente y precisa. Se ejecuta

²Trabajo parcialmente financiado por la Comisión Europea y la Comisión Interministerial de Ciencia y Tecnología CICYT a través del proyecto FEDER 1FD97-0255-C03-01 y el proyecto CICYT TIC97-1343-C002-02.

³Centro de Investigación y Desarrollo Agroalimentario.

$max = 5$		$max = 10$			
M	MSE	M	MSE	M	MSE
1	0.1192	1	0.1036	6	0.0813
2	0.0915	2	0.0921	7	0.0796
3	0.0811	3	0.0875	8	0.0786
4	0.0797	4	0.0853	9	0.0772
5	0.0790	5	0.0815	10	0.0763

Tabla 5.16: Soluciones no dominadas de acuerdo al modelo de optimización descrito en la ecuación (5.7) sin tener en cuenta la similaridad ($g_s = 1.0$) obtenidas con el algoritmo neuro-evolutivo multiobjetivo para estimar la evapotranspiración según el modelo de cubeta clase A mostrado en las ecuaciones (5.12) y (5.13).

también el algoritmo teniendo en cuenta solamente la precisión según la adaptación del algoritmo neuro-evolutivo descrita en el apartado 5.3.11. Los resultados obtenidos para un número máximo de 5 y de 10 reglas se muestran en la tabla 5.16. Se puede observar que el modelo con mayor precisión se obtiene con un número de 10 reglas. Sin embargo, si se desea un menor número de reglas, las soluciones obtenidas estableciendo un máximo de 5 reglas resultan más precisas (para un mismo número de reglas) que las obtenidas con un máximo de 10 reglas. Es decir, para un problema dado, resulta conveniente estudiar cual es el número de reglas que se desea obtener para el modelo difuso y establecer dicho número como el máximo.

Comparamos los resultados con los obtenidos mediante otros algoritmos. Aplicando directamente las ecuaciones clásicas del modelo evaporimétrico de cubeta clase A se obtiene un primer resultado, con $MSE = 2.8613$. Este cálculo resulta rápido y sencillo, pero el modelo que se obtiene no es muy preciso. El segundo resultado se obtiene entrenando mediante el método de gradiente durante 10000 epochs una red neuronal MLP (Perceptrón Multicapa) con 10 neuronas en la capa interna⁴. La siguiente opción utiliza una red neuronal RBF con 10 neuronas en la capa interna entrenada mediante el método de gradiente durante 1000 epochs. Otra aproximación diferente es utilizar una red neuronal ANFIS; en este caso se obtiene el número de reglas mediante un clustering previo y posteriormente un entrenamiento mediante

⁴El número de neuronas en la capa interna es equivalente al número de reglas.

Método	MSE	M
Cubeta Clase A	2.8613	—
MLP	0.1169	10
RBF	0.0694	10
ANFIS	0.0813	12
ANEM ¹	0.0839	3
ANEM ²	0.0790	5
ANEM ³	0.0763	10

Tabla 5.17: Comparación de los resultados obtenidos en este trabajo y con otros algoritmos para estimar la evapotranspiración.

¹ Algoritmo Neuro-Evolutivo Multiobjetivo con similaridad y $max = 5$.

² Algoritmo Neuro-Evolutivo Multiobjetivo sin similaridad y $max = 5$.

³ Algoritmo Neuro-Evolutivo Multiobjetivo sin similaridad y $max = 10$.

gradiente para los antecedentes y mínimos cuadrados para los consecuentes con 100 epochs, obteniendo un valor $MSE = 0.0813$ con 12 reglas. Con nuestro algoritmo neuro-evolutivo es posible obtener un modelo interpretable con tan solo 3 reglas y con una precisión mejor que la obtenida por el modelo MLP con 10 neuronas en la capa interna y similar a la precisión obtenida con ANFIS con 12 reglas. Por otra parte, si utilizamos el algoritmo neuro-evolutivo teniendo en cuenta solo la precisión, obtenemos modelos con 5 y 10 reglas, ambos con mejor precisión que ANFIS y obteniendo valores de precisión similares a los obtenidos con una red neuronal RBF. En la tabla 5.17 se muestran estos resultados.

Ejemplo 5

Finalmente y para abordar la modelización desde otra perspectiva, en este ejemplo se resuelve un problema de clasificación. Para ello se utiliza la base de datos *Iris*. Este conjunto de datos es muy utilizado en la literatura y se encuentra disponible en la universidad de California en Irvine [Bla98]. El conjunto de datos *Iris* contiene 150 datos con 4 variables de características pertenecientes a 3 clases distintas con 50 datos en cada clase.

La evaluación se realiza utilizando la tasa de clasificación (*Classification Rate*, CR) que indica el porcentaje de individuos correctamente clasificados. Dicho valor toma el lugar del MSE .

Las soluciones no dominadas obtenidas de acuerdo al modelo de optimización para el conjunto de datos *Iris* se muestran en la tabla 5.18. Se muestran también el número de datos mal clasificados (MC) y no clasificados (NC). Se han obtenido también soluciones con 4 y 5 reglas, pero su tasa de clasificación obtiene el mismo valor que el modelo con 3 reglas, y al tener éste un menor número de reglas, resulta un modelo más compacto por lo que, según la ecuación (5.7), estas soluciones son dominadas y por ello no se muestran.

Gómez-Skarmeta et al. [Gom01] obtienen diversos modelos difusos para clasificar el conjunto de datos *Iris*. La mayor precisión la obtienen mediante un modelo aproximativo. En este modelo las reglas se generan utilizando el algoritmo de c-medias difuso [Bez81] para asignar los conjuntos difusos y el algoritmo de frecuencia difusa [Del96] para asignar el grado de certidumbre de cada regla. A continuación se utiliza un algoritmo evolutivo diferencial [Sto97] para ajustar las reglas. Se consigue un modelo con 26 reglas, y $MC/NC = 1/0$. Con nuestro algoritmo se consigue la misma tasa de error tan sólo con 3 reglas, siendo un modelo transparente. Otro modelo obtenido por Gómez-Skarmeta et al. [Gom01] es un modelo descriptivo que utiliza etiquetas lingüísticas. En este caso, se obtiene un modelo interpretable con 4 reglas y $MC/NC = 3/0$. Nuestro algoritmo neuro-evolutivo obtiene dicha tasa de error con una sola regla. Por último, para obtener una mayor precisión, se ejecuta el algoritmo para obtener un modelo aproximativo según la adaptación descrita en el apartado 5.3.11. En ese caso podemos llegar a obtener un modelo con 3 reglas capaz de clasificar correctamente todos los datos, tal como se muestra en la tabla 5.19.

M	L	CR	S	MC/NC
1	4	98.00%	0.0	3/0
2	6	98.67%	0.263	2/0
3	7	99.33%	0.328	1/0

Tabla 5.18: Soluciones no dominadas de acuerdo al modelo de optimización descrito en la ecuación (5.7) obtenidas con el algoritmo neuro-evolutivo multiobjetivo para clasificar el conjunto de datos *Iris*.

M	CR	MC/NC
1	98.00%	3/0
2	99.33%	1/0
3	100%	0/0

Tabla 5.19: Soluciones no dominadas de acuerdo al modelo de optimización descrito en la ecuación (5.7) sin tener en cuenta la similaridad ($g_s = 1.0$) obtenidas con el algoritmo neuro-evolutivo multiobjetivo para clasificar el conjunto de datos *Iris*.

Conclusiones de los experimentos realizados

En esta sección se han aplicado los algoritmos evolutivos propuestos en las secciones 5.2 y 5.3 para resolver un problema test clásico adicional (el problema propuesto por Sugeno [Sug88]). Se ha comprobado que, tanto en este problema como en los problemas anteriores (modelo de planta no lineal estudiada por Wang [Wan98] y dinámica de presión de un fermentador [Bab94]), ambos algoritmos obtienen resultados más satisfactorios atendiendo a los criterios de precisión, transparencia y compactitud que los resultados obtenidos por otros autores. Por otra parte, comparando los resultados obtenidos por ambos algoritmos, se ha probado que, tal como se esperaba, el algoritmo neuro-evolutivo al realizar hibridación con el método de gradiente, obtiene modelos más precisos, sin disminuir por ello la interpretabilidad de los mismos.

Con objeto de continuar la evaluación del algoritmo neuro-evolutivo se ha aplicado este algoritmo para resolver un par de problemas adicionales. En primer lugar se ha aplicado para realizar un modelo de la evapotranspiración que permita determinar las necesidades de riego de los cultivos. Este problema resulta especialmente interesante por ser un problema real y la importancia de su aplicación. En segundo lugar se realiza una adaptación del algoritmo para resolver problemas de clasificación, aplicándolo para clasificar en concreto el conjunto de datos Iris. En ambos problemas, el algoritmo neuro-evolutivo ha obtenido modelos más precisos e interpretables que los modelos obtenidos mediante otras técnicas.

Otro aspecto de interés es la evaluación del algoritmo neuro-evolutivo para obtener modelos aproximativos, es decir, sin tener en cuenta aspectos de similitud. Se ha ejecutado esta variante del algoritmo para el problema de Sugeno, cálculo de la evapotranspiración y para clasificación. En todos los casos se han obtenido modelos aproximativos más precisos que los modelos anteriores.

5.5 Sumario

Este capítulo comienza proponiendo un algoritmo evolutivo para identificación de variables. Se comprueba el funcionamiento del mismo realizando diversas ejecuciones con un conjunto de datos de un problema test estándar y comparando los resultados con los obtenidos mediante otros algoritmos propuestos en la literatura.

A continuación se realiza una discusión sobre modelización difusa y se proponen un par de algoritmos evolutivos multiobjetivo para calcular modelos difusos. Se pretende hallar modelos difusos que sean precisos, pero también interpretables. Aparecen dos criterios a optimizar, que resultan antagónicos en muchos casos. Este es, por tanto, un problema donde las técnicas evolutivas multiobjetivo pueden aplicarse para obtener buenos resultados.

Los algoritmos evolutivos propuestos utilizan, por tanto, técnicas multiobjetivo para calcular un conjunto de soluciones no dominadas atendiendo a un modelo de optimización que tiene en cuenta criterios de precisión y transparencia. Por otra parte, se utiliza un número variable de reglas preservando un conjunto mínimo y máximo de modelos para cada número de reglas, lo cual permite mantener la diversidad con respecto al número de reglas y obtener en la solución final modelos para cada número de reglas en el intervalo definido. Se propone asimismo un proceso de decisión a posteriori para elegir entre todas las soluciones no dominadas, aquella que resulte más interesante.

El primer algoritmo propuesto es una primera aproximación, si bien recoge técnicas validadas en optimización multiobjetivo. Utiliza modelos difusos del tipo Takagi-Sugeno con consecuentes lineales y conjuntos difusos trapezoidales en los antecedentes. Para mejorar la transparencia utiliza una técnica de simplificación del conjunto de reglas difusas.

El segundo algoritmo se presenta como un perfeccionamiento del primero para mejorar precisión, transparencia y compactitud. Utiliza también modelos difusos del tipo Takagi-Sugeno con consecuentes lineales pero, a diferencia del algoritmo anterior, utiliza conjuntos difusos gaussianos asimétricos. Para mejorar la precisión se realiza una hibridación con el método del gradiente para entrenar la red neuronal de base radial equivalente al modelo difuso. Por otra parte, el método de simplificación se transforma en un método más complejo para mejorar la transparencia y la compactitud y se define una nueva medida de similaridad. El algoritmo resultante es un algoritmo neuro-

evolutivo con el que se obtienen modelos más precisos, sin perder la interpretabilidad.

El algoritmo neuro-evolutivo puede ser utilizado también para calcular modelos aproximativos, sin tener en cuenta aspectos de interpretabilidad. En este caso, se obtienen modelos con mayor precisión, si bien estos modelos ya no resultan interpretables. Sin embargo, en algunos casos esta aproximación puede ser la más interesante.

Se han realizado experimentos con ambos algoritmos en diferentes problemas distintos y en todos los casos se han obtenido resultados similares o mejores a los obtenidos por otros autores. Los modelos difusos obtenidos cumplen ambos requisitos, son precisos y transparentes, aunque el algoritmo neuro-evolutivo consigue mejores valores de precisión gracias a la hibridación con la técnica de gradiente. Asimismo con el algoritmo neuro-evolutivo se han obtenido también modelos aproximativos sin tener en cuenta aspectos de interpretabilidad, pero con mayor precisión.

Conclusiones y Líneas Futuras

Conclusiones

Utilizando siempre la Computación Evolutiva como telón de fondo, este trabajo de tesis ha supuesto un recorrido dentro de dos líneas de actuación fundamentales de entre las que D.E. Goldberg denominó como Búsqueda, Optimización y Aprendizaje (*Genetic Algorithms in Search, Optimization, and Machine Learning, 1989*). Dentro del campo de la optimización nos hemos centrado en los aspectos de optimización multiobjetivo, tema éste de considerable actualidad al permitirnos abordar problemas complejos como los que encontramos en el mundo real y cuyas soluciones están acordes con la caracterización intrínseca de los algoritmos propios de la Computación Evolutiva. Por otro lado, en el ámbito del aprendizaje, la aplicación de la Computación Evolutiva a la generación de modelos difusos partiendo de datos del comportamiento de un sistema, ha proporcionado resultados muy satisfactorios en los campos de identificación de sistemas y de clasificadores difusos. Por último, la posibilidad de conectar la búsqueda de soluciones que sean a la vez precisas e interpretables a la hora de modelizar y generar sistemas difusos, con la utilización de optimización multiobjetivo, nos ha permitido establecer el marco idóneo de desarrollo de este trabajo de tesis, aplicando dichos resultados a diversos problemas test y de entornos reales.

Para alcanzar dichos objetivos se han diseñado y evaluado diversos algoritmos evolutivos multiobjetivo para optimización multiobjetivo y modelización difusa. Podemos diferenciar, por tanto, dos partes: (i) diseño y evaluación de nuevos algoritmos evolutivos para optimización multiobjetivo con restricciones, y (ii) diseño y evaluación de nuevos algoritmos evolutivos para modelización difusa. A continuación se realiza una descripción de la investigación realizada en cada una de estas partes de la tesis.

Diseño y evaluación de nuevos algoritmos evolutivos para optimización multiobjetivo con restricciones

Dentro de esta primera parte se ha comenzado realizando una revisión sobre algoritmos evolutivos y sobre técnicas evolutivas aplicadas a optimización multiobjetivo y con restricciones para, posteriormente, aplicar aquellas técnicas que más nos interesen para diseñar nuevos algoritmos.

Revisión de técnicas evolutivas y aplicación a optimización multiobjetivo.

Los algoritmos evolutivos, por sus características de robustez y versatilidad, han sido aplicados a una gran variedad de problemas. La optimización multiobjetivo es un área donde los algoritmos evolutivos han tenido una especial importancia, puesto que, al trabajar con poblaciones de individuos, son capaces de obtener en una sola ejecución del problema múltiples soluciones no dominadas según el concepto Pareto. En este trabajo se ha comenzado revisando las técnicas básicas utilizadas en los algoritmos evolutivos. Entre estas técnicas resultan de especial interés aquellas utilizadas para mantener la diversidad y el elitismo, dos puntos esenciales dentro del campo multiobjetivo. A continuación se han revisado los principales algoritmos evolutivos utilizados para optimización multiobjetivo y las técnicas más importantes para manejar restricciones. De este estudio se concluye que los aspectos de mayor relevancia dentro del campo multiobjetivo son el diseño basado en el concepto de dominación Pareto, las técnicas para mantener la diversidad y el elitismo.

Nuevos algoritmos evolutivos para optimización multiobjetivo. Dentro de este apartado se han propuesto nuevos algoritmos evolutivos para resolver problemas de optimización multiobjetivo. Todos estos algoritmos están basados en el concepto de dominación Pareto encontrando, en una sola ejecución, un conjunto de soluciones no dominadas para realizar un proceso de decisión *a posteriori*. Utilizan, además, diversas técnicas para mantener la diversidad y el elitismo.

El primer algoritmo que se propone en este trabajo resuelve problemas de optimización multiobjetivo sin restricciones. Utiliza la técnica de preselección simple para mantener la diversidad y el elitismo. Con este algoritmo se han obtenido buenos resultados en problemas de optimización sin restricciones y también se ha aplicado para resolver problemas con restricciones, realizando una transformación de los mis-

mos en problemas sin restricciones. Este es un algoritmo sencillo que da inicio a la investigación realizada y sirve como base para el diseño de los algoritmos posteriores.

En los sucesivos algoritmos propuestos (Algoritmo I, Algoritmo II y ENORA) se incorporan técnicas para el manejo de restricciones, por lo que resuelven directamente problemas de optimización multiobjetivo con restricciones sin necesidad de efectuar ninguna transformación de los mismos. Cada uno de estos algoritmos utiliza diferentes técnicas para mantener la diversidad.

El Algoritmo I utiliza de nuevo la técnica de preselección simple para mantener la diversidad. La ventaja de esta aproximación es que es un método sencillo y eficiente que funciona de forma adecuada en problemas donde no resulta especialmente difícil mantener la diversidad. Sin embargo, en ciertos problemas multiobjetivo, la preselección simple no es suficiente para mantener la diversidad, siendo necesario utilizar técnicas de diversidad explícita.

El Algoritmo II añade a la preselección simple una técnica de diversidad explícita que consiste en el uso de métricas de diversidad, tratando esta medida como un objetivo a cumplir junto con el resto de objetivos del problema. Esta técnica obtiene mejores resultados con respecto a la diversidad que la preselección simple, sin embargo, el evaluar la métrica de diversidad implica un coste añadido.

El último algoritmo propuesto, ENORA, realiza una partición del espacio de búsqueda en tramos radiales, lo que supone una técnica *ad hoc* para mantener la diversidad, puesto que fuerza a buscar entre soluciones que ya son diversas. Esta es una técnica de diversidad específica para optimización multiobjetivo. La ventaja que ofrece de esta técnica es su eficacia, el inconveniente es su mayor complejidad.

Tanto el Algoritmo II como ENORA obtienen soluciones dispersas al ser evaluados por algunos de los problemas test más importantes dentro del campo multiobjetivo, siendo ENORA el que consigue una mayor diversidad, mejorando los resultados obtenidos por algunos de los algoritmos evolutivos multiobjetivo más actuales. La elección de los problemas test utilizados ha sido de gran importancia, puesto que se han seleccionado problemas test difíciles de solucionar y sencillos de evaluar, lo que ha permitido una evaluación eficaz y clara de los algoritmos.

Diseño y evaluación de nuevos algoritmos evolutivos para modelización difusa

Esta segunda parte de la tesis se centra en el campo de la modelización difusa. Se comienza exponiendo algunos puntos sobre modelización difusa y posteriormente se utilizan los resultados de la investigación en optimización multiobjetivo general para diseñar nuevos algoritmos evolutivos que generen modelos difusos precisos e interpretables.

Aspectos de modelización difusa. La modelización difusa se ha convertido en una técnica fundamental para aplicaciones de todo tipo utilizándose con éxito los algoritmos evolutivos, en muchas ocasiones realizando hibridación con otras técnicas numéricas como clustering, gradiente, etc. Estos algoritmos obtienen modelos difusos aproximativos muy precisos, pero que no son interpretables. Sin embargo, la interpretabilidad es una de las características más interesantes de los modelos difusos. Resulta, por tanto, deseable obtener un modelo difuso preciso pero que, al mismo tiempo, sea interpretable, por lo que aparecen múltiples objetivos; y para resolver este problema pueden ser utilizadas algunas de las técnicas multiobjetivo estudiadas con anterioridad.

Nuevos algoritmos evolutivos para modelización difusa. Este trabajo termina utilizando las técnicas multiobjetivo investigadas en la primera parte de esta tesis para diseñar nuevos algoritmos evolutivos que sean capaces de obtener modelos difusos precisos e interpretables.

En primer lugar se propone un algoritmo para realizar selección de variables, lo que es un paso previo dentro del proceso de modelización difusa. Este algoritmo se aplica para resolver algunos problemas de selección de variables, siendo capaz, en todos los casos, de realizar dicha selección de forma correcta.

Posteriormente se proponen un par de algoritmos evolutivos multiobjetivo para generar modelos difusos. Ambos algoritmos se basan en el concepto de óptimo Pareto para calcular modelos con criterios de precisión y también de interpretabilidad, obteniendo en una sola ejecución del algoritmo múltiples soluciones no dominadas y realizando un proceso de decisión *a posteriori*. Se utiliza, por otra parte, una técnica de formación de nichos para mantener la diversidad respecto al número de reglas de

los modelos considerados. El primer algoritmo propuesto obtiene modelos difusos basados en funciones de pertenencia trapezoidales y utiliza una técnica para mejorar la transparencia de los conjuntos difusos. Este algoritmo es una primera aproximación para realizar modelos difusos precisos y transparentes. El segundo algoritmo se plantea como una mejora del anterior. Utiliza funciones de pertenencia gaussianas asimétricas como una representación alternativa y realiza, además, hibridación con una técnica de gradiente para obtener una mayor precisión en los modelos. Por otra parte, se perfecciona el método para mejorar la transparencia y se define una nueva medida de similaridad.

Los dos algoritmos anteriores se aplican a diversos problemas, encontrando modelos difusos precisos e interpretables en todos los casos. Sin embargo, el segundo de estos algoritmos consigue modelos que, siendo igualmente interpretables, tienen una mayor precisión. Este último algoritmo también se ha aplicado para calcular modelos difusos aproximativos, sin tener en cuenta la interpretabilidad, obteniendo en este caso modelos con una gran precisión. Se han utilizado problemas test minuciosamente seleccionados para permitir la evaluación y comparación de estos algoritmos.

Líneas Futuras

La gran versatilidad de los algoritmos evolutivos, junto a la diversidad existente de problemas de optimización multiobjetivo y modelización difusa, hace que el campo de investigación relacionado con dichos temas sea muy extenso. Partiendo de los resultados y experiencias de este trabajo de tesis, se abren diversas líneas futuras de investigación, que podemos clasificar dentro de dos apartados: (i) desarrollo de nuevos algoritmos y técnicas, y (ii) aplicación a distintos campos. A continuación se describen las líneas futuras de investigación dentro de cada uno de estos apartados.

Desarrollo de nuevos algoritmos y técnicas

En esta tesis se han diseñado y evaluado una serie de algoritmos evolutivos multiobjetivo para optimización y modelización difusa. Dentro de este grupo incluimos, por tanto, aquellas líneas futuras que consisten precisamente en el desarrollo de nuevas técnicas evolutivas multiobjetivo. Dicho desarrollo se plantea dentro de dos ámbitos diferentes: (i) optimización multiobjetivo, y (ii) modelización difusa.

Optimización multiobjetivo. El diseño de algoritmos para la resolución de problemas de optimización multiobjetivo es un tema de gran importancia en el desarrollo de esta tesis. La ampliación y extensión de dichos algoritmos resulta, por tanto, un tema de investigación futura de sumo interés. En concreto, se proponen como líneas de investigación futura la extensión de algoritmos a los siguientes campos:

1. Estrategias de evolución. El desarrollo de nuevos algoritmos tomando como base el esquema de las estrategias de evolución puede realizarse adaptando las técnicas ya desarrolladas en esta tesis al nuevo esquema, o el diseño de otras técnicas diferentes. El esquema de las estrategias de evolución resulta especialmente adecuado para optimización de parámetros, por lo que podría conseguirse una mejora en los resultados obtenidos al evaluar los nuevos algoritmos.

2. Optimización difusa. Otra línea futura de interés es la optimización difusa. Consideramos un problema de optimización general no lineal sujeto a restricciones difusas no lineales:

$$\begin{aligned} & \text{Minimizar} \quad f(\mathbf{x}) \\ & \text{sujeto a :} \quad g_j(\mathbf{x}) \lesssim b_j, \quad j = 1, \dots, m \end{aligned}$$

donde $\mathbf{x} = (x_1, \dots, x_p)$ es un vector de parámetros reales $x_k \in \mathfrak{R}$ pertenecientes a un dominio $[l_k, u_k]$, $k = 1, \dots, p$, $b_j \in \mathfrak{R}$, y $f(\mathbf{x})$, $g_j(\mathbf{x})$ son funciones arbitrarias lineales o no lineales. Consideramos la siguiente función de pertenencia no lineal asociada a cada restricción difusa:

$$\mu_j(\mathbf{x}) = \begin{cases} 0, & \text{si } g_j(\mathbf{x}) \geq b_j + d_j \\ \frac{b_j + d_j - g_j(\mathbf{x})}{d_j}, & \text{si } b_j \leq g_j(\mathbf{x}) \leq b_j + d_j \\ 1, & \text{si } g_j(\mathbf{x}) \leq b_j \end{cases}$$

que da el grado de satisfacción de $g_j(\mathbf{x})$ con respecto a la restricción j -ésima (no se toleran violaciones de las restricciones por encima del valor $b_j + d_j$, $j = 1, \dots, m$). El problema anterior puede ser transformado [Ver82] en un problema paramétrico de la forma:

$$\begin{aligned} & \text{Minimizar} \quad f(\mathbf{x}) \\ & \text{sujeto a :} \quad g_j(\mathbf{x}) \leq b_j + d_j(1 - \alpha), \quad j = 1, \dots, m \end{aligned}$$

donde $\alpha \in [0, 1]$. Un enfoque evolutivo a este problema consiste [Cad00] en resolver el problema para determinados valores del parámetro α obteniendo finalmente las

funciones difusas mediante aproximaciones lineales a partir de las soluciones puntuales. Otro enfoque evolutivo más cercano a la definición del problema paramétrico consiste en buscar directamente soluciones paramétricas, tema éste que proponemos como futuro trabajo.

Modelización difusa. Dentro del campo de modelización difusa también se pueden plantear nuevas técnicas. Es posible, por ejemplo, utilizar diferentes medidas de transparencia y similaridad: similaridad en el dominio de los consecuentes, en lugar de similaridad en el dominio de los antecedentes, o similaridad en el dominio de ambos. Si se utiliza un modelo de Mamdani con reglas de la forma:

$$R_i : \text{Si } x_1 \text{ es } A_{i1} \text{ y } \dots \text{ y } x_n \text{ es } A_{in} \text{ entonces } y \text{ es } B_i$$

donde $i = 1, \dots, M$, entonces la similaridad en el consecuente, S_C , puede expresarse de la forma:

$$S_C = \max_{\substack{i,j \\ B_i \neq B_j}} S(B_i, B_j)$$

donde $i, j = 1, \dots, M$. Otra opción puede ser combinar la similaridad en el consecuente, S_C , con la similaridad en el antecedente, S_A , para calcular un valor de similaridad total, S_T . Esto puede hacerse, por ejemplo, de la siguiente forma:

$$S_T = w_A S_A + w_C S_C$$

donde $w_A, w_C \in [0, 1]$, $w_A + w_C = 1.0$, son los pesos asignados a la similaridad en el antecedente y en el consecuente respectivamente, aplicando para ello los algoritmos evolutivos planteados en esta tesis.

Aplicación a distintos campos

En esta tesis, los algoritmos desarrollados han sido evaluados con un conjunto de problemas test, aplicándolos en algunos casos a problemas reales. Sin embargo, el campo de aplicación es mucho más amplio del cubierto en esta tesis. Dentro de este grupo se podrían incluir tantas líneas futuras como problemas reales en los que tuvieran aplicación los algoritmos diseñados. Para reducir dicho conjunto, nos centramos en un par de campos en los que, de alguna forma, ya se ha empezado a trabajar dentro de nuestro grupo de investigación. En concreto estos campos son los siguientes:

Asignación de recursos. El problema de asignación de recursos es muy común en el ámbito de la empresa, donde generalmente existe un conjunto de recursos limitados que hay que asignar de forma que el rendimiento final obtenido sea óptimo, evaluando dicho rendimiento normalmente en función de diversos parámetros; por ejemplo, minimizar el coste y, a la vez, minimizar el tiempo. Añadido a esto, no resulta extraño tampoco encontrar una serie de restricciones en este tipo de problemas. Resulta, por tanto, un problema de optimización multiobjetivo con restricciones.

La asignación de medios de transporte a chóferes es un problema de este tipo, donde los medios de transporte son los recursos que hay que asignar cumpliendo un conjunto de objetivos impuestos por la empresa y un conjunto de restricciones naturales marcadas por los clientes y los chóferes. En [Mir02] se utiliza el algoritmo evolutivo multiobjetivo propuesto por Jiménez et al. [Jim02] para resolver el problema anterior. Como línea futura de investigación se plantea dar continuidad a este trabajo utilizando los algoritmos y técnicas diseñadas en esta tesis para resolver el problema de la asignación de medios de transporte.

Robótica móvil. En el ámbito de la robótica móvil, las reglas de control difusas se han aplicado típicamente para el control de robots. Los controladores difusos utilizados requieren ser aprendidos o bien ajustados a partir de ejemplos de interacción con el entorno. Por otra parte, la interpretabilidad de los conjuntos de reglas difusas es un factor determinante en el ámbito de la robótica móvil, lo que justifica la aplicación de algoritmos evolutivos multiobjetivo para la obtención de modelos difusos precisos, transparentes y compactos.

Dentro del proyecto PANDORA¹ destaca el uso de la computación evolutiva como mecanismo principal para las tareas de optimización multiobjetivo. En concreto, la identificación de los modelos difusos utilizados en el control de robots. Se pretende, por tanto, la integración de los algoritmos y técnicas desarrolladas para modelización difusa en dicho proyecto.

¹PANDORA: Una Plataforma Distribuida Basada en Agentes para Auto-aprendizaje, Control y Teleoperación de Robots Móviles a través de Internet, CICYT-Feder, TIC2001-0245-302-01, Universidad de Murcia y Universidad de Alicante.

Apéndice A

Apendice: Conceptos básicos sobre Teoría de Conjuntos Difusos

En este apéndice se introducen las definiciones básicas y terminología sobre conjuntos difusos. Puesto que la investigación en este campo ha sido muy amplia, resulta imposible contemplar todos los aspectos actualmente desarrollados esta materia. Por tanto, en este apéndice simplemente se realizará una introducción breve a los conceptos más básicos que se han utilizado en esta tesis.

Tal como expone Zadeh en su trabajo seminal [Zad65], los conjuntos difusos se utilizan cuando se quiere realizar una evaluación cualitativa de alguna cantidad física. En los conjuntos tradicionales, un elemento pertenece o no pertenece a dicho conjunto; la diferencia de los conjuntos difusos es que se establece un *grado de pertenencia*, de forma que un elemento pertenece a un conjunto difuso con un cierto grado.

A.1 Conjuntos difusos

Un *conjunto difuso* A en el dominio \mathcal{X} se define mediante un conjunto de pares ordenados:

$$A = \{(x, \mu_A(x)) \mid x \in \mathcal{X}\}$$

donde $\mu_A(x)$ es la *función de pertenencia* para el conjunto difuso A :

$$\mu_A : \mathcal{X} \rightarrow [0, 1]$$

La función de pertenencia asigna a cada elemento $x \in \mathcal{X}$ un valor entre 0 y 1, dicho valor es el *grado de pertenencia* de x al conjunto A . Un conjunto difuso, por tanto, vendrá definido por su función de pertenencia. Normalmente se llama a \mathcal{X} *universo de discurso* o simplemente *universo* y puede consistir de objetos discretos o ser un espacio continuo.

La definición de conjunto difuso es, por tanto, una simple extensión de la definición de conjunto clásico, donde se permite que la función característica tenga cualquier valor entre 0 y 1. Si se restringe el valor de la función de pertenencia a 0 ó 1, entonces A es un conjunto clásico y $\mu_A(x)$ es la función característica de A .

A.2 Definiciones básicas

Soporte de un conjunto difuso

El *soporte* de un conjunto difuso A es el conjunto de todos los puntos $x \in \mathcal{X}$ tales que su función de pertenencia es mayor que cero:

$$\text{soporte}(A) = \{x \in \mathcal{X} | \mu_A(x) > 0\}$$

Núcleo de un conjunto difuso

El *núcleo* de un conjunto difuso A es el conjunto de todos los puntos $x \in \mathcal{X}$ tales que su función de pertenencia es igual a uno:

$$\text{núcleo}(A) = \{x \in \mathcal{X} | \mu_A(x) = 1\}$$

Normalidad

Un conjunto difuso A es *normal* si su núcleo es no vacío, es decir, si siempre podemos encontrar un punto $x \in \mathcal{X}$ tal que $\mu_A(x) = 1$.

Conjunto difuso singular

Se dice que A es un *conjunto difuso singular* si su soporte es un solo punto $x \in \mathcal{X}$ con $\mu_A(x) = 1$.

Conjunto difuso universal

El *conjunto difuso universal*, llamado \mathcal{U} , es el conjunto difuso en el cual todos los elementos del universo de discurso tienen grado de pertenencia igual a 1:

$$\mathcal{U} : \mu_{\mathcal{U}}(x) = 1, \forall x \in \mathcal{X}$$

Punto de cruce

Un *punto de cruce* de un conjunto difuso A es un punto $x \in \mathcal{X}$ tal que su función de pertenencia toma el valor 0.5:

$$cruce(A) = \{x \in \mathcal{X} | \mu_A(x) = 0.5\}$$

α -corte y α -corte estricto

El α -corte de un conjunto difuso A es el conjunto definido por:

$$A_{\alpha} = \{x \in \mathcal{X} | \mu_A(x) \geq \alpha\}$$

De forma similar se define α -corte estricto de un conjunto difuso A como:

$$A'_{\alpha} = \{x \in \mathcal{X} | \mu_A(x) > \alpha\}$$

Convexidad

Un conjunto difuso A es *convexo* si y solo si para todo $x_1, x_2 \in \mathcal{X}$ y para todo $\lambda \in [0, 1]$:

$$\mu_A(\lambda x_1 + (1 - \lambda)x_2) \geq \min\{\mu_A(x_1), \mu_A(x_2)\}$$

de forma alternativa, A es convexo si A_{α} es convexo, para todo $\alpha \in [0, 1]$.

Anchura

La *anchura* de un conjunto difuso convexo y normal se define como la distancia entre los dos únicos puntos de cruce:

$$anchura(A) = |x_2 - x_1|$$

donde $\mu_A(x_1) = \mu_A(x_2) = 0.5$.

A.3 Operaciones con conjuntos difusos

Los conjuntos difusos tienen operaciones similares a las de los conjuntos ordinarios, tales como la unión o la intersección. Zadeh en su trabajo seminal [Zad65] define inicialmente las operaciones sobre conjuntos difusos. En este apartado se discuten dos de las operaciones básicas sobre conjuntos difusos: la unión y la intersección difusa.

***T*-norma/Intersección difusa**

La intersección entre dos conjuntos A y B es otro conjunto difuso $C = A \cap B$ cuya función de pertenencia se calcula a partir de las funciones de pertenencia de A y B mediante una función $T : [0, 1] \times [0, 1] \rightarrow [0, 1]$ que realiza la agregación de dos grados de pertenencia de la siguiente forma:

$$\mu_C(x) = \mu_{A \cap B}(x) = T(\mu_A(x), \mu_B(x)) = \mu_A(x) \tilde{*} \mu_B(x)$$

donde $\tilde{*}$ es un operador binario para la función T . Esta clase de operadores de intersección difusa se llaman operadores T -norma (norma triangular) y son operadores binarios en el intervalo unidad que satisfacen, al menos, los siguientes axiomas para todo $a, b, c \in [0, 1]$:

condición de límite:	$T(0, 0) = 0, T(a, 1) = T(1, a) = a$
monótona:	$T(a, b) \leq T(c, d) \text{ si } a \leq c \text{ y } b \leq d$
conmutativa:	$T(a, b) = T(b, a)$
asociativa:	$T(a, T(b, c)) = T(T(a, b), c)$

Algunos de los operadores T -norma más frecuentes son:

mínimo (intersección estándar):	$T(a, b) = \min(a, b) = a \wedge b$
producto algebraico:	$T(a, b) = ab$
producto limitado:	$T(a, b) = \max(0, a + b - 1) = 0 \vee (a + b - 1)$

***T*-conorma/*S*-norma/Unión difusa**

La unión de dos conjuntos difusos A y B es otro conjunto difuso $C = A \cup B$ cuya función de pertenencia se calcula a partir de las funciones de pertenencia de A y B

mediante una función $S : [0, 1] \times [0, 1] \rightarrow [0, 1]$ que realiza la agregación de dos grados de pertenencia de la siguiente forma:

$$\mu_C(x) = \mu_{A \cup B}(x) = S(\mu_A(x), \mu_B(x)) = \mu_A(x) \tilde{+} \mu_B(x)$$

donde $\tilde{+}$ es un operador binario para la función S . Esta clase de operadores de unión difusa se llaman operadores T -conorma (o S -norma) y son operadores binarios en el intervalo unidad que satisfacen, al menos, los siguientes axiomas para todo $a, b, c \in [0, 1]$:

$$\begin{array}{ll} \text{condición de límite:} & S(1, 1) = 1, \quad S(0, a) = S(a, 0) = a \\ \text{monótona:} & S(a, b) \leq S(c, d) \quad \text{si } a \leq c \text{ y } b \leq d \\ \text{conmutativa:} & S(a, b) = S(b, a) \\ \text{asociativa:} & S(a, S(b, c)) = S(S(a, b), c) \end{array}$$

Algunos de los operadores T -conormas más frecuentes son:

$$\begin{array}{ll} \text{máximo (unión estándar):} & S(a, b) = \max(a, b) = a \vee b \\ \text{suma algebraica:} & S(a, b) = a + b - ab \\ \text{suma limitada:} & S(a, b) = \min(1, a + b) = 1 \wedge (a + b) \end{array}$$

A.4 Funciones de pertenencia

Existen muchas clases de funciones de pertenencia, a continuación describiremos algunas de las más utilizadas.

Función de pertenencia triangular

Esta función se define mediante tres parámetros $\{a, b, c\}$ de la siguiente forma:

$$\mu_{a,b,c}(x) = \begin{cases} 0, & \text{si } x \leq a \\ \frac{x-a}{b-a}, & \text{si } a \leq x \leq b \\ \frac{c-x}{c-b}, & \text{si } b \leq x \leq c \\ 0, & \text{si } c \leq x \end{cases}$$

Una forma de expresión alternativa es la siguiente:

$$\mu_{a,b,c}(x) = \max \left(\min \left(\frac{x-a}{b-a}, \frac{c-x}{c-b} \right), 0 \right)$$

Los parámetros $\{a, b, c\}$ determinan las coordenadas de las tres esquinas del triángulo definido por la función.

Función de pertenencia trapezoidal

Una función de pertenencia trapezoidal se especifica mediante cuatro parámetros $\{a, b, c, d\}$ de la forma:

$$\mu_{a,b,c,d}(x) = \begin{cases} 0, & \text{si } x \leq a \\ \frac{x-a}{b-a}, & \text{si } a \leq x \leq b \\ 1, & \text{si } b \leq x \leq c \\ \frac{d-x}{d-c}, & \text{si } c \leq x \leq d \\ 0, & \text{si } d \leq x \end{cases}$$

O expresado de forma más concisa:

$$\mu_{a,b,c,d}(x) = \max \left(\min \left(\frac{x-a}{b-a}, 1, \frac{d-x}{d-c} \right), 0 \right)$$

Los parámetros $\{a, b, c, d\}$ determinan las coordenadas de las cuatro esquinas del trapecio definido por la función.

Función de pertenencia gaussiana simétrica

La función de pertenencia gaussiana simétrica viene especificada por dos parámetros $\{c, \sigma\}$:

$$\mu_{c,\sigma}(x) = \exp \left[-\frac{1}{2} \left(\frac{x-c}{\sigma} \right)^2 \right]$$

donde el parámetro c representa el centro y σ la varianza de la gaussiana.

Función de pertenencia gaussiana asimétrica

La función de pertenencia gaussiana asimétrica viene especificada por tres parámetros $\{c, \sigma_l, \sigma_r\}$:

$$\mu_{c,\sigma^l,\sigma^r}(x) = \begin{cases} \exp \left[-\frac{1}{2} \left(\frac{x-c}{\sigma^l} \right)^2 \right], & \text{si } x \leq c \\ \exp \left[-\frac{1}{2} \left(\frac{x-c}{\sigma^r} \right)^2 \right], & \text{si } x \geq c \end{cases}$$

donde el parámetro c representa el centro y σ^l, σ^r la varianza izquierda y derecha de la gaussiana respectivamente.

Función de pertenencia gaussiana de dos lados

La función de pertenencia gaussiana de dos lados viene especificada por cuatro parámetros $\{c^l, \sigma^l, c^r, \sigma^r\}$:

$$\mu_{c^l, \sigma^l, c^r, \sigma^r}(x) = \begin{cases} \exp \left[-\frac{1}{2} \left(\frac{x-c^l}{\sigma^l} \right)^2 \right], & \text{si } x \leq c^l \\ 1, & \text{si } c^l < x < c^r \\ \exp \left[-\frac{1}{2} \left(\frac{x-c^r}{\sigma^r} \right)^2 \right], & \text{si } x \geq c^r \end{cases}$$

Función de pertenencia de campana o de Cauchy

Una función de pertenencia de campana se especifica mediante tres parámetros $\{a, b, c\}$ de la forma:

$$\mu_{a,b,c}(x) = \frac{1}{1 + \left| \frac{x-c}{a} \right|^{2b}}$$

donde el parámetro b es normalmente positivo (si fuera negativo, la forma de esta función sería la de una campana invertida). Se puede notar que esta función es una generalización directa de la distribución de Cauchy utilizada en teoría de probabilidad, por tanto, a esta función también se le llama de Cauchy.

Función de pertenencia sigmoideal

Una función de pertenencia sigmoideal se define mediante dos parámetros $\{a, c\}$ de la forma:

$$\mu_{a,c}(x) = \frac{1}{1 + \exp[-a(x-c)]}$$

donde a controla la pendiente en el punto de cruce $x = c$. Dependiendo del signo del parámetro a , la función sigmoideal es abierta a derecha o izquierda.

Función de pertenencia izquierda-derecha

Esta función viene especificada por tres parámetros $\{\alpha, \beta, c\}$:

$$\mu_{\alpha, \beta, c}(x) = \begin{cases} f_i \left(\frac{c-x}{\alpha} \right), & \text{si } x \leq c \\ f_d \left(\frac{x-c}{\beta} \right), & \text{si } x \geq c \end{cases}$$

donde $f_i(x)$ y $f_d(x)$ son funciones monótonamente decrecientes definidas en $[0, \infty)$ con $f_i(0) = f_d(0)$ y $\lim_{x \rightarrow \infty} f_i(x) = \lim_{x \rightarrow \infty} f_d(x) = 0$.

A.5 Defuzzificación

El proceso de defuzzificación permite asociar a un conjunto difuso un número no difuso. Esto se realiza para calcular el valor de salida de los modelos difusos. La defuzzificación puede realizarse de varias formas. A continuación se describen algunas.

Centro del área (COA)

Asocia el centro del área formada por el número difuso. Este es uno de los métodos más utilizados. Matemáticamente se expresa de la siguiente forma:

$$y = \frac{\int_{\mathcal{Y}} \mu(y)y dy}{\int_{\mathcal{Y}} \mu(y) dy}$$

Bisector del area (BOA)

Calcula el bisector del área formada por el número difuso, es decir, el valor obtenido y divide la región en dos partes con la misma área. Se expresa con la siguiente ecuación matemática.

$$y \text{ tal que } \int_{\alpha}^y \mu(y) dy = \int_y^{\beta} \mu(y) dy$$

donde α y β son los valores mínimos y máximos de la variable y respectivamente.

Media del máximo (MOM)

Realiza la media de los valores máximos del conjunto difuso. Matemáticamente se define de la siguiente forma:

$$y = \frac{\int_{\mathcal{Y}'} y dy}{\int_{\mathcal{Y}'} dy}$$

donde $\mathcal{Y}' = \{y \in \mathcal{Y} \mid \mu(y) = \mu^*\}$ y μ^* es el valor máximo de la función de pertenencia.

Bibliografía

- [Ade94] Adeli, H., Cheng, N.T. (1994). *Augmented lagrangian genetic algorithm for structural optimization*. Journal of Aerospace Engineering, 7(1):104-18.
- [Alle92] Allenson, R. (1992). *Genetic Algorithms with gender for multi-function optimisation*. Technical Report EPCC-SS92-01, Edinburgh Parallel Computing Centre, Edinburgh, Scotland.
- [Ant89] Antonisse, J.A. (1989). *A new interpretation of schema notation that overturns the binary encoding constraint*. Proceedings of the Third International Conference on Genetic Algorithm, J.D. Schaffer (ed.), Morgan Kaufmann Publishers, San Mateo, 86-91.
- [Ank91] Ankenbrandt, C.A. (1991). *An extension to the theory of convergence and a proof of the time complexity of genetic algorithms*. G.J.E. Rawlins (ed.), Foundations of Genetic Algorithms (San Mateo: Morgan Kauffman), 53-68.
- [Art70] Arthur, J.L., Ravindran, A. (1970). *PAGP: A Partitioning Algorithm for (Linear) Goal Programming Problems*. ACM Transactions on Mathematical Software, in press.
- [Bab94] Babuska, R., Verbruggen, H.B. (1994). *Applied fuzzy modeling*, IFAC Symposium on Artificial Intelligence in Real time Control, Spain.
- [Bab95] Babuska, R., Verbruggen, H.B. (1995). *A new identification method for linguistic fuzzy models*, Open Meeting of the FALCON working group, 1-8, Aachen, Germany.
- [Bac91] Bäck, T., Hoffmeister, F. (1991). *Extended selection mechanisms in genetic algorithms*. en R. Belew and L. Booker (eds.), Proceedings of the Fourth Inter-

- national Conference on Genetic Algorithms, Morgan Kaufmann Publishers, Los Altos, CA, 92-99.
- [Bak85] Baker, J.E. (1985). *Adaptive selection methods for genetic algorithms*, J.J. Grefenstette (ed.), Proceedings of the First International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Hillsdale, NJ, 101-111.
- [Bak89] Baker, J.E. (1987). *Reducing bias and inefficiency in the selection algorithm*. J.J. Grefenstette (ed.), Proceedings of the Second International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Hillsdale, NJ, 14-21.
- [Bea92] Bean, J.C., Hadj-Alouane, A.B. (1992). *A Dual Genetic Algorithm for Bounded Integer Programs*. Technical Report TR 92-53, Department of Industrial and Operations Engineering, The University of Michigan.
- [Bel97] Belur, S.V. (1997). *CORE: Constrained Optimization by Random Evolution*. In John R. Koza (ed.), Late Breaking Papers at the Genetic Programming 1997 Conference, Stanford University, California. Stanford Bookstore, 280-286.
- [Ber93] Bertoni, A., Dorigo, M. (1993). *Implicit parallelism in genetic algorithms*. Artificial Intelligence, 61:307-314.
- [Bez81] Bezdek, J.C. (1981). *Pattern Recognition with Fuzzy Objective Function*. Plenum Press, New York.
- [Bil95] Bilchev, G., Parmee, I.C. (1995). *The Ant Colony Metaphor for Searching Continuous Design Spaces*. In Terence C. Fogarty (ed.), Evolutionary Computing, Springer Verlag, Sheffield, UK, 25-39.
- [Bil96] Bilchev, G., Parmee, I.C. (1996). *Constrained and Multi-Modal Optimisation with an Ant Colony Search Model*. In Ian C. Parmee and M.J. Denham (eds.), Proceedings of 2nd International Conference on Adaptive Computing in Engineering Design and Control. University of Plymouth, Plymouth, UK.
- [Bin97] Binh, T.T., Korn, U. (1997). *MOBES: A multiobjective evolution strategy for constrained optimization problems*. In The Third International Conference on Genetic Algorithms (Mendel 97), 176-182.

- [Bla98] Blake, C.L., Merz, C.J. (1998). *(UCI) Repository of machine learning databases*. <http://www.ics.uci.edu/~mllearn/MLRepository.html>, University of California, Irvine, Dept. of Information and Computer Sciences.
- [Bri81] Brindle, A. (1981). *Genetic Algorithms for Function Optimization*. Doctoral Dissertation, University of Alberta, Edmonton.
- [Buc96] Buczak, A.L., Uhrig, R.E. (1996) *Hybrid Fuzzy-Genetic Technique for Multisensor Fusion*. Information Sciences, 93:265-281.
- [Cad00] Cadenas, J.M., Gómez-Skarmeta, A.F., Jiménez, F., Verdegay, J.L. (2000). *Evolutionary techniques for fuzzy optimization problems*. 8th International Conference on Information Processing and Management of Uncertainty in Knowledge-Base Systems (IPMU 2000), Madrid, Spain.
- [Cam97] Camponogara, E, Talukdar, S.N. (1997). *A Genetic Algorithm for Constrained and Multiobjective Optimization*. In Jarmo T. Alander (ed.), 3rd Nordic Workshop on Genetic Algorithms and Their Applications (3NWGA), Vaasa, Finland. University of Vaasa, 49-62.
- [Car] Carlson-Skalat, S., Shonkwiler, R., Babar, S., Aral, M. *Annealing a Genetic Algorithm over Constraints*. Available at <http://vlead.mech.virginia.edu/publications/shenkpaper/shenkpaper.html>.
- [Cha83] Chankong, V., Haimes, Y.Y. (1983). *Multiobjective Decision Making: Theory and Methodology*. North-Holland series in Systems Science and Engineering, Andrew P. Sage (ed.).
- [Char61] Charnes, A., Cooper, W.W. (1961). *Management Models and Industrial Applications of Linear Programming*, volume 1. John Wiley, New York.
- [Che94] Chen, Y.L., C. C. Liu. (1994). *Multiobjective VAR planning using the goal-attainment method*. IEEE Proceedings on Generation, Transmission and Distribution, 141(3):227-232.
- [Chu96] Chung, C., Reynolds, R.G. (1996). *A testbed for Solving Optimization Problems using Cultural Algorithms*. In Lawrence J. Fogel, Peter J. Angeline, and Thomas Bäck (eds.), *Evolutionary Programming V: Proceedings of the*

- Fifth Annual Conference on Evolutionary Programming, Cambridge, Massachusetts, MIT Press.
- [Coe96] Coello, C.A. (1996). *An Empirical Study of Evolutionary Techniques for Multiobjective Optimization in Engineering Design*. PhD thesis, Department of Computer Science, Tulane University, New Orleans, LA.
- [Coe97] Coello, C.A., Santos, F., Alonso, F. (1997). *Optimal design of reinforced concrete beams using genetic algorithms*. *Exper Systems with Applications: An International Journal*, 12(1).
- [Coe99] Coello, C.A., Christiansen, A.D. (1999). *MOSES: A multiobjective optimization tool for engineering design*. *Engineering Optimization* 31(3):269-308.
- [Coe00a] Coello, C.A. (2000). *Treating Constraints as Objectives for Single-Objective Evolutionary Optimization*. *Engineering Optimization* 32(3):275-308.
- [Coe00b] Coello, C.A., Toscano, G. (2000). *A micro-genetic algorithm for multiobjective optimization*. Technical Report Lania-RI-2000-06, Laboratorio Nacional de Informatica Avanzada, Xalapa, Veracruz, Mexico.
- [Coe02] Coello, C.A., Veldhuizen, D.V., Lamont, G.B. (2002). *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic/Plenum publishers, New York.
- [Coi96a] Coit, D.W., Smith, A.E. (1996). *Penalty guided genetic search for reliability design optimization*. *Computers and Industrial Engineering*, 30(4):895-904. Special Issue on Genetic Algorithms.
- [Coi96b] Coit, D.W., Smith, A.E., Tate, D.M. (1996). *Adaptive Penalty Methods for Genetic Optimization of Constrained Combinatorial Problems*. *INFORMS Journal on Computing*, 8(2):173-182.
- [Col91] Coloni, A., Dorigo, M., Maniezzo, V. (1991). *Distributed optimization by ant colonies*. In P. Bourguine and F. Varela (eds.), *Proceedings of the First European Conference on Artificial Life*, Cambridge, Massachusetts, MIT Press/Bradford Books.

- [Cord97] Cordon, O., Herrera, F. (1997). *A Three-Stage Evolutionary Process for Learning Descriptive and Approximate Fuzzy-Logic-Controller Knowledge Bases From Examples*. International Journal of Approximate Reasoning, 17:369-407.
- [Cord00] Cordon, O., Herrera, F. (2000). *A proposal for improving the accuracy of linguistic modeling*. IEEE Transactions on Fuzzy Systems, 8(3):335-344.
- [Corn00] Corne, D.W., Knowles, J.D., Oates, M.J. (2000). *The Pareto Envelope-based Selection Algorithm for Multiobjective Optimization*. In Marc Schoenauer, Kalyanmoy Deb, Günter Rudolph, Xin Yao, Evelyne Lutton, J.J. Merelo and Hans-Pau Schwefel (eds.), Proceedings of the Parallel Problem Solving from Nature VI Conference, Springer, 839-848.
- [Corn01] Corne, D.W., Jerram, N.R., Knowles, J.D., Oates, M.J. (2001). *PESA-II: Region-based Selection in Evolutionary Multiobjective Optimization*. In Lee Spector, Erik Goodman, Annie Wu, William B. Langdon Hans-Michael Voight, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke (eds.), Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001), Morgan Kaufmann Publishers, San Francisco, California, 283-290.
- [Cve98] Cvetković, D., Parmee, I., Webb, E. (1998). *Multi-objective Optimisation and Preliminary Airframe Design*.
- [Dar59] Darwin, C. (1959). *On the Origin of Species by Means of Natural Selection*. John Murray.
- [Das97] Dasgupta, D., Michalewicz, Z. (1997). *Evolutionary Algorithms in Engineering Applications*. Springer-Verlag, Berlin.
- [Dav91] Davis, L. (1991). *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York.
- [Deb89] Deb, K., D.E. Goldberg, D.E. (1989). *An investigation of niche and species formation in genetic function optimization*. Proceedings of the Third International Conference on Genetic Algorithms, Fairfax, VA, Morgan Kaufmann, 42-50.

- [Deb99] Deb, K. (1999). *Evolutionary algorithms for multi-criterion optimization in engineering design*. In K. Miettinen, P. Neittaanmäki, M.M. Mäkelä and J. Périaux (eds.), *Evolutionary Algorithms in Engineering and Computer Science*, Chichester, UK:Wiley, 135-161.
- [Deb00a] Deb, K. (2000). *An efficient constraint handling method for genetic algorithms*. *Computer Methods in Applied Mechanics and Engineering*, 186(2-4):311-338.
- [Deb00b] Deb, K., Agrawal, S., Pratap, A., Meyarivan, T. (2000). *A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II*. In *Proceedings of the Parallel Problem Solving from Nature VI (PPSN-VI)*, 849-858.
- [Deb01a] Deb, K., Pratap, A., Meyarivan, T. (2001). *Constrained test problems for multi-objective evolutionary optimization*. *Lectures Notes in Computer Science*, 1993:284-298.
- [Deb01b] Deb, K., Goel, T. (2001). *Controlled Elitist Non-dominated Sorting Genetic Algorithms for Better Convergence*. *Lectures Notes in Computer Science*, 1993:67-81.
- [Deb01c] Deb, K. (2001). *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley and Sons, LTD.
- [DeJ75] DeJong, K.A. (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. Doctoral Dissertation, University of Michigan.
- [Del95] Delgado, M., Gomez-Skarmeta, A.F., Martín, F. (1995). *Generating Fuzzy Rules Using Clustering Based Approach*, Third European Congress on Fuzzy and Intelligent Technologies and Soft Computing, Aachen, Germany, 810-814.
- [Del96] Delgado, M., González, A., (1996) *A frequency model in a fuzzy environment*. *Int. J. Approx. Reasoning* 11:159-174.
- [Dor89] Dorigo, M., Di Caro, G. (1989). *The Ant Colony Optimization Meta-Heuristic*. In D. Corne, M. Dorigo, and F. Glover (eds.), *New Ideas in Optimization*. McGraw-Hill.

- [Dor96] Dorigo, M., Maniezzo, V., Colorni, A. (1996). *The Ant System: Optimization by a Colony of Cooperating Agents*. IEEE Transactions on Systems, Man, and Cybernetics Part B, 26(1):29-41.
- [Dor97] Dorigo, M., Gambardella, L.M. (1997). *Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem*. IEEE Transactions on Evolutionary Computation, 1(1):53-66.
- [Dre98] Drechsler, M. (1998). *Evolutionary Algorithms for VLSI CAD*. Boston: Kluwer Academic Publishers.
- [Duc84] Duckstein, L. (1984). *Multiobjective Optimizacoin in Structural Design: The Model Choice Problem*. In E. Atrek, R. H. Gallagher, K. M. Ragsdell, and O. C. Zienkiewicz (eds.), *New Directions in Optimum Structural Design*, John Wiley and Sons, 459-481.
- [Dyc84] Dyckhoff, H., Pedrycz, W. (1984). *Generalized Means of Model of Compensative Connectives*. Fuzzy Sets and Systems, 4:143-154.
- [Eri01] Erickson, M., Mayer, A., Horn, J. (2001). *The Niche Pareto Genetic Algorithm 2 Applied to the Design of Groundwater Remediation Systems* Eckart Zitzler and Kalyanmoy Deb and Lothar Thiele and Carlos A. Coello Coello and David Corne (eds.). First International Conference on Evolutionary Multi-Criterion Optimization. Springer-Verlag. Lecture Notes in Computer Science, 1993:681-695.
- [Esh93] Eshelman, L.J., Schaffer, J.D. (1993). *Real-coded genetic algorithms and interval-schemata*. L.D. Whitley (ed.), *Foundations of Genetic Algorithms-2*, Morgan Kaufmann Publishers, San Mateo, 187-202.
- [Fog66] Fogel, L.J., Owens, A.J., Walsh, M.J. (1966). *Artificial Intelligence through Simulated Evolution*. Wiley, New York.
- [Fon93] Fonseca C.M., Fleming, P.J. (1993). *Genetic Algorithms for Multi-objective Optimization: Formulation, Discussion and Generalization*. In Stephanie Forrest (ed.), *Genetic Algorithms: Proceedings of the Fifth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, California, 416-423.

- [Fon98] Fonseca, C., Fleming, P.J. (1998). *Multiobjective Optimization and Multiple Constraint Handling with Evolutionary Algorithms - Part I: A Unified Formulation* IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, 28(1):26-37.
- [Fou85] Fourman, M.P. (1985). *Compaction of symbolic layout using genetic algorithms*. In Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms, Lawrence Erlbaum, 141-153.
- [For91] Forrest, S., Perelson, A.S. (1991). *Genetic algorithms and the immune system*. In Hans-Paul Schwefel and R. Männer, (eds.), Parallel Problem Solving from Nature, Lecture Notes in Computer Science, Springer-Verlag, Berlin, Germany, 320-325.
- [Gen96] Gen, M., Cheng, R. (1996). *Interval Programming using Genetic Algorithms*. In Proceedings of the Sixth International Symposium on Robotics and Manufacturing, Montpellier, France.
- [Gar90] Garis, H. (1990). *Genetic Programming: Building Artificial Nervous Systems using Genetically Programmed Neural Networks Modules* In R. Porter and B. Mooney (eds.), Proceedings of the 7th International Conference on Machine Learning, Morgan Kaufmann, 132-139.
- [Gey95] Geyer-Schulz, A. (1995). *Fuzzy Rule-Based Expert Systems and Genetic Machine Learning*. Physica-Verlag.
- [Gil85] Gillies, A.M. (1985). *Machine Learning Procedures for Generating Image Domain Feature Detectors, Doctoral Dissertation*. University of Michigan.
- [Glo77] Glover, F. (1977). *Heuristics for integer programming using surrogate constraints*. Decision Sciences, 8(1):156-166.
- [Glo95] Glover, F., Kochenberger, G. (1995). *Critical event tabu search for multi-dimensional knapsack problems*. In Proceedings of the International Conference on Metaheuristics for Optimization, Dordrecht, The Netherlands, Kluwer Publishing, 113-133.

- [Gol85] Goldberg, D.E. (1985). *Optimnal initial population size for binary-coded genetic algorithms*. TCGA Report No. 85001, The University of Alabama, Tuscaloosa, AL.
- [Gol87] Goldberg, D.E., Richardson, J.J. (1987). *Genetic algorithms with sharing for multimodal function optimization*. Genetic Algorithms and its Applications: Proceedings of the Second International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Hillsdale, NJ, 41-49.
- [Gol89] Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley publishing Company, Reading, Massachusetts.
- [Gol89b] Goldberg, D.E., Korb, B., Deb, K. (1989). *Messy genetic algorithms: motivation, analysis, and first results*. TCGA Report No. 89002 (Tuscaloosa: University of Alabama, The Clearingjouse for Genetic Algorithms).
- [Gol89c] Goldberg, D.E. (1989). *Sizing populations for serial and parallel genetic algorithms*. Proceedings of the Third International Conference on Genetic Algorithms, J.D. Schaffer (ed.), San Mateo, California, Morgan Kaufmann, Inc., 70-79.
- [Gol91] Goldberg, D.E., Deb, K. (1991). *A comparative analysis of selection schemes used in genetic algorithms*. en G.J.E. Rawlins (ed.) Foundations of Genetic Algorithms (San Mateo: Morgan Kauffman), 69-93.
- [Gol98] Goldberg, D.E., Wang, L. (1989). *Adaptive niching via coevolutionary sharing*. In D. Quagliarella, J. Périaux, C. Poloni, and G. Winter (eds.), Genetic Algorithms and evolution Strategies in Engineering and Computer Science, Chichester, UK: Wiley, 21-38.
- [Gom98] Gómez-Skarmeta, A.F., Jiménez, F., Ibáñez, J. (1998). *Pareto-optimality in fuzzy modeling*. Proceedings of the 6th European Congress on Intellingent Techniques and Soft Computing (EUFIT'98), Aachen, Alemania, 694-770.
- [Gom99a] Gómez-Skarmeta, A.F., Jiménez, F. (1999). *Fuzzy modeling with hybrid systems*. Fuzzy Sets and Systems, 104:199-208.

- [Gom99b] Gómez-Skarmeta, A.F., Jiménez, F., Ibáñez, J., Paredes, S. (1999). *Evolutionary variable identification*. Proceedings of the 7th European Congress on Intelligent Techniques and Soft Computing (EUFIT'99), Aachen, Alemania.
- [Gom01] Gómez-Skarmeta, A.F., Valdés, M., Jiménez, F., Marín-Blázquez, J.G. (2001). *Approximative fuzzy rules approaches for classification with hybrid-GA techniques*. Information Sciences, 136:193-214.
- [Gre84] Grefenstette, J.J. (1984). *GENESIS: A system for using genetic search procedures*. Proceedings of the 1984 Conference on Intelligent Systems and Machines, 161-165.
- [Gus79] Gustafson, D.E., Kessel, W.C. (1979). *Fuzzy clustering with a fuzzy covariance matrix*, IEEE International Conference on Fuzzy Systems, San Diego, 761-766.
- [Had92] Hadj-Alouane, A.B., Bean, J.C. (1992). *A Genetic Algorithm for the Multiple-Choice Integer Program*. Technical Report TR 92-50, Department of Industrial and Operations Engineering, The University of Michigan.
- [Haj92] Hajela, P., Lin, C.Y. (1992). *Genetic search strategies in multicriterion optimal design*. Structural Optimization, 4:99-107.
- [Haj95] Hajela, P., Lee, J. (1995). *Constrained Genetic Search via Schema Adaptation. An Immune Network Solution*. In Niels Olhoff and George I.N. Rozvany (eds.), Proceedings of the First World Congress of Structural and Multidisciplinary Optimization, Goslar, Germany. Pergamon, 915-920.
- [Haj96a] Hajela, P., Lee, J. (1996). *Constrained Genetic Search via Schema Adaptation. An Immune Network Solution*. Structural Optimization, 12:11-15.
- [Haj96b] Hajela, P., Yoo, J. (1996). *Constraint Handling in Genetic Search Using Expression Strategies*. AIAA Journal, 34(2):2414-2420.
- [Her95] Herrera, F., Lozano, M., Verdegay, J.L., *Tunning fuzzy controllers by genetic algorithms*. Approx. Reason. 12:299-314.
- [Hir99] Hiroyasu, T., Miki, M., Watanabe, S. (1999). *Distributed genetic algorithm with a new sharing approach in multiobjective optimization problems*. In Proceedings of the Congress on Evolutionary Computation (CEC-1999), 69-76.

- [Hof96] Hoffmeister, F., Sprave, J. (1996). *Problem-independent handling of constraints by use of metric penalty functions*. In Lawrence J. Fogel, Peter J. Angeline, and Thomas Bäck (eds.), *Proceedings of the Fifth Annual Conference on Evolutionary Programming (EP'96)*, San Diego, California, The MIT Press, 289-294.
- [Hol71] Hollstien, R.B. (1971). *Artificial Genetic Adaptation in Computer Control Systems*. Ph. D. Thesis, Ann Arbor, MI:University of Michigan. Dissertation Abstracts International, 32(3), 1510B. (University Microfilms no. 71-23,773).
- [Hol75] Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor.
- [Hol88] Holland, J.H. (1988). *The dynamics of searches directed by genetic algorithms*. Y.C. Lee (ed.) *Evolution, Learning and Cognition*, Word Scientific, 111-127.
- [Hom94] Homaifar, A., C.X. Qi, S.H. Lai. (1994). *Constrained optimization via genetic algorithms*. *Simulation* 62-4, 242-254.
- [Hor93] Horn, J., Nafpliotis, N. (1993). *Multiobjective Optimization Using the Niche Pareto Genetic Algorithm*. Technical Report IlliGAL Report No. 93005, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA.
- [Hwa98] Hwang, H.S. (1998). *Control strategy for optimal compromise between trip time and energy consumption in a high-speed railway*. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 28(6):791-802.
- [Ign76] Ignizio, J.P. (1976). *Goal Programming and Extensions*. Heath, Lexington, Massachusetts.
- [Iji65] Ijiri, Y. (1965). *Management Goals and Accounting for Control*. North-Holland, Amsterdam.
- [Ise77] Isermann, H. (1977). *The Enumeration of the Set of All Efficient Solutions for a Linear Multiple Objective Program*. *Operational Research Quarterly*, 28(3):711-725.
- [Ish97] Ishibuchi, H., Murata, T., Türksen, I. (1997). *Single-objective and two-objective genetic algorithms for selecting linguistic rules for pattern classification problems*. *Fuzzy Sets and Systems*, 89:135-150.

- [Ish99] Ishibuchi, H., Nakashima, T., Murata, T. (1999). *Performance evaluation of fuzzy classifier systems for multidimensional pattern classification problems*. IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics, 29(5):601-618.
- [Ish01] Ishibuchi, H., Takeuchi, D., Nakashima, T. (2001). *GA-based Approaches to Linguistic Modeling of Nonlinear Functions*. Proceedings of the Joint 9th IFSA World Congress and 20th NAFIPS International Conference, Vancouver, Canada, 1229-1234.
- [Jak92] Jakob, W., Gorges-Scheleuter, M., Blume, C. (1992). *Application of genetic algorithms to task planning and learning*. In R. Männer and B. Manderick (eds.), Parallel Problem Solving from Nature, 2nd Workshop, Lecture Notes in Computer Science, Amsterdam, North-Holland Publishing Company, 291-300.
- [Jagi99] Jagielska, I., Matthews, C., Whitfort, T. (1999). *An investigation into the application of neural networks, fuzzy logic, genetic algorithms, and rough sets to automated knowledge acquisition for classification problems*. Neurocomputing, 24:37-54.
- [Jan97] Jang, J.R., Sun, C., Mizutani, E., (1997). *Neuro-Fuzzy and Soft Computing. A Computational Approach to Learning and Machine Intelligence*. Prentice Hall.
- [Jim98] Jiménez, F., Verdegay, J.L. (1998). *Constrained Multiobjective Optimization by Evolutionary Algorithms*. Proceedings of the International ICSC Symposium on Engineering of Intelligent Systems (EIS'98), University of La Laguna, Tenerife, Spain, 266-271.
- [Jim99a] Jiménez, F., Verdegay, J.L. (1999). *Evolutionary techniques for constrained optimization problems*. In 7th European Congress on Intelligent Techniques and Soft Computing (EUFIT'99), Aachen, Germany. Springer-Verlag.
- [Jim99b] Jiménez, F., Verdegay, J.L., Gómez-Skarmeta, A.F. (1999). *Evolutionary techniques for constrained multiobjective optimization problems*. In Proceedings of the Workshop on Multi-Criterion Optimization Using Evolution-

- ary Methods held at Genetic and Evolutionary Computation Conference (GECCO-1999), Orlando, Florida, USA, 115-116.
- [Jim01] Jiménez, F., Gómez-Skarmeta, A.F., Roubos, H., Babuska, R. (2001). *Accurate, Transparent, and Compact Fuzzy Models for Function Approximation and Dynamic Modeling Through Multi-objective Evolutionary Optimization*. Lectures Notes in Computer Science, Evolutionary Multi-Criterion Optimization, 1993:653-667.
- [Jim02] Jiménez, F., Gómez-Skarmeta, A.F., Sánchez, G. (2002). *How Evolutionary Multi-objective Optimization can be used for Goal and Priority based Optimization*. Proceedings del Primer congreso iberoamericano de algoritmos evolutivos y bioinspirados, AEB'02, Mérida, 460-465.
- [Jin00] Jin, Y. (2000). *Fuzzy Modeling of High-Dimensional Systems*. Transactions on Fuzzy Systems: Complexity Reduction and Interpretability Improvement, 8:212-221.
- [Joh00] Johansen, T.A., Shorten, R., Murray-Smith, R. (2000). *On the interpretation and identification of dynamic Takagi-Sugeno fuzzy models*. IEEE Transactions on Fuzzy Systems, 8(3):297-313.
- [Joi94] Joines, J., Houck, C. (1994). *On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GAs*. In David Fogel (ed.), Proceedings of the first IEEE Conference on Evolutionary Computation, 579-584, Orlando, Florida. IEEE Press.
- [Jon93] Jones, G., Brown, R.D., Clark, D.E., Willett, Pl, Glen, R.C. (1993). *Searching Databases of Two-Dimensional and Three-Dimensional Chemical Structures using Genetic Algorithms*. In S. Forrest (ed.), Proceedings of the Fifth International Conference on Genetic Algorithms, San Mateo, California, Morgan Kaufmann, 597-602.
- [Jut67] Jutler, H. (1967). *Liniejnaja model z nieskolkimi celevymi funkzjami (linear model with several objective functions)*. Ekonomika i matematiceckije Metody, 3:397-406.

- [Kim97] Kim, J.H., Myung, H. (1997). *Evolutionary programming techniques for constrained optimization problems*. IEEE Transactions on Evolutionary Computation, 1:129-140.
- [Kit96] Kita, H., Yabumoto, Y., Mori, N., Nishihawa, Y. (1996). *Multi-objective optimization by means of thermodynamical genetic algorithm*. In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel (eds.), Proceedings of Parallel Problem Solving from Nature IV (PPSN-IV), Lectures Notes in Computer Science, Berlin, Germany, Springer-Verlag, 504-512.
- [Kno00] Knowles, J.D., Corne, D.W. (2000). *Approximating the non-dominated front using the Pareto archived evolution strategy*. Evolutionary Computation Journal, 8(2),149-172.
- [Kop97] Köppen, M., Teunis, M., Nicholay, B. (1997). *NESSY - an evolutionary learning neural network*. In Proceedings of the Second International ICSC Symposium on Soft Computing, SOCO'97, 243-248.
- [Kor79] Kornbluth, J.S.H. (1979). *Indifference Regions and Marginal Utility Weights in Multiple Objective Linear Fractional Programming*. Working Pap. 79-02-03. Department of Decision Sciences, The Wharton School, University of Pennsylvania.
- [Kow97] Kowalczyk, R. (1997). *Constraint Consistent Genetic Algorithms*. In Proceedings of the 1997 IEEE Conference on Evolutionary Computation, Indianapolis, USA, 343-348.
- [Koza92] Koza, J.R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press, Cambridge, Massachusetts.
- [Koz98] Koziel, S., Michalewicz, Z. (1998). *A Decoder-based Evolutionary Algorithm for Constrained Parameter Optimization Problems*. In T. Bäck, A.E. Eiben, M. Schoenauer, and H.P. Schwefel (eds.), Proceedings of the 5th Parallel Problem Solving from Nature (PPSN V), Amsterdam. Springer-Verlag, 231-240.

- [Koz99] Koziel, S., Michalewicz, Z. (1999). *Evolutionary Algorithms, Homomorphous Mappings, and Constrained Parameter Optimization*. Evolutionary Computation, 7(1):19-44.
- [Kri89] Krishnakumar, K. (1989). *Micro-genetic algorithms for stationary and non-stationary function optimization*. In SPIE Proceedings: Intelligent Control and Adaptive Systems, 289-296.
- [Kuri98] Kuri-Morales, A.F., Villegas-Quezada, C. (1998). *A Universal Eclectic Genetic Algorithm for Constrained Optimization*. In Proceedings 6th European Congress on Intelligent Techniques & Soft Computing, EIFIT'98, Aachen, Germany, Verlag Mainz, 518-522.
- [Kurs91] Kursawe, F. (1991). *A variant of evolution strategies for vector optimization*. In H.P. Schwefel and R. Männer (eds.), Parallel Problem Solving from Nature. 1st Workshop, PPSN I, Volume 496 of Lecture Notes in Computer Science, Berlin, Germany, Springer-Verlag, 193-197.
- [Lau98] Laumanns, M., Rudolph, G., Schwefel, H.P. (1998). *A sapatial predator-prey approach to multi-objective optimization: A preliminary study*. In Proceedings of the Parallel Problem Solving from Nature V (PPSN-V), 241-249.
- [Le95] Le, T. Van (1995). *A Fuzzy Evolutionary Approach to Constrained Optimization Problems*. In Proceedings of the Second IEEE Conference on Evolutionary Computation, Perth, 274-278.
- [Lee72] Lee, S.M. (1972). *Goal Programming for Decision Analysis*, Auerbach Publishers, Philadelphia, 126-157.
- [Leu98] Leung, K.S., Zhu, Z.Y., Leung, Y. (1998). *Multiobjective optimization using non-dominated sorting in annealing genetic algorithms*. Department of Geography and the Centre for Environmental Studies, Chinese University of Hong Kong, Hong Kong.
- [Lie90] Liepins, G.E., Vose, M.D. (1990). *Representational Issues in Genetic Optimization*. Journal of Experimental and Theoretical Computer Science, 2(2):4-30.

- [Lie91] Liepins, G.E., Potter, W.D. (1991). *A Genetic Algorithm Approach to Multiple-Fault Diagnosis*. In Lawrence Davis (ed.), *Handbook of Genetic Algorithms*, 17:237-250. Van Nostrand Reinhold, New York.
- [Lin76] Lin, J.G. (1976). *Maximal vectors and multi-objective optimization*. *Journal of Optimization Theory and Applications*, 18(1):41-64.
- [Lis96] Lis, J., Eiben, A.E. (1996). *A multi-Sexual Genetic Algorithm for Multi-objective Optimization*. In Toshio Fukuda and Takeshi Furuhashi (eds.), *Proceedings of the 1996 International Conference on Evolutionary Computation*, IEEE, Nagoya, Japan, 59-64.
- [Liu98] Liu, X., Begg, D.W., Fishwick, R.J. (1998). *Genetic approach to optimal topology/controller design of adaptive structures*. *International Journal for Numerical Methods in Engineering*, 41:815-830.
- [Lou97] Loughlin, D.H., Ranjithan, S. (1997). *The neighborhood constraint method: A multiobjective optimization technique*. In *Proceedings of the Seventh International Conference on Genetic Algorithms*, 666-673.
- [Maa92] Maa, C., Shanblatt, M. (1992). *A two-phase optimization neural network*. *IEEE Transactions on Neural Networks*, 3(6):1003-1009.
- [Mag96] Magdalena, L., Velasco, J.R. (1996). *Fuzzy Rule-Based Controllers that Learn by Evolving their Knowledge Base*. F. Herrera & J.L. Verdegay (eds). *Genetic Algorithms and Soft Computing*, Physica-Verlag, 172-201.
- [Mam75] Mamdani, E.H., Asilian, S. (1975). *An experiment in linguistic synthesis with a fuzzy logic controller*. *International Journal of Man-Machine Studies*, 7(1):1-13.
- [Mar00] Mariano, C.E., Morales, E.F. (2000). *Distributed reinforcement learning for multiple objective optimization problems*. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, 613-620.
- [Mic92] Michalewicz, Z. (1992). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Verlag, second edition.

- [Mic94] Michalewicz, Z., Attia, N. (1994). *Evolutionary Optimization of Constrained Problems*. In Proceedings of the 3rd Annual Conference on Evolutionary Programming, World Scientific, 98-108.
- [Mic95] Michalewicz, Z., Nazhiyath, G. (1995). *Genocop III: A co-evolutionary algorithm for numerical optimization with nonlinear constraints*. In David B. Fogel (ed.), Proceedings of the Second IEEE International Conference on Evolutionary Computation, Piscataway, New Jersey, IEEE Press, 647-651.
- [Mic95b] Michalewicz, A. (1995). *Genetic algorithms, numerical optimization and constraints*. In L.J. Eshelman (ed.), Proceedings of the Sixth International Conference on Genetic Algorithms, San Mateo, CA:Morgan Kaufmann, 151-158.
- [Mic96a] Michalewicz, Z., Dasgupta, D., Le Riche, R., Schoenauer, M. (1996). *Evolutionary algorithms for constrained engineering problems*. Computer & Industrial Engineering Journal, 30(4):851-870.
- [Mic96b] Michalewicz, Z., Schoenauer, M. (1996). *Evolutionary algorithms for constrained parameter optimization problems*. Evolutionary Computation Journal, 4(1):1-32.
- [Mir02] Miró, B., Lorente, M.L. (2002). *Análisis, Diseño e Implementación de un Sistema Inteligente para la Asignación de Recursos en Entornos Empresariales: Una Aplicación de la Computación Evolutiva*. Proyecto Informático, Facultad de Informática, Universidad de Murcia.
- [Mit78] Mitchell, T. (1978). *Version Spaces: An Approach to Concept Learning*. PhD thesis, Computer Science Department, Stanford University, Stanford, California.
- [Moo88] Moody, J., Darken, C. (1988). *Learning with localized receptive fields*. In D. Touretzky, G. Hinton, and T. Sejnowski (eds.), Proceedings of the 1988 Connectionist Models Summer School, San Mateo, CA. Carnegie Mellon University, Morgan Kaufmann.
- [Muh93] Mühlenbein, H., Schlierkamp-Voosen, D. (1993). *Predictive models for the breeder genetic algorithms I. Continuous parameter optimization*. Evolutionary Computation, 1:25-49.

- [Mur95] Murata, T., Ishibuchi, H. (1995). *MOGA: Multi-objective genetic algorithms*. In Proceedings of the Second IEEE International Conference on Evolutionary Computation, 289-294.
- [Myu95] Myung, H., J.H.Kim., Fogel, D.B. (1995). *Preliminary investigation into a two-stage method of evolutionary optimization on constrained problems*. In J.R. McDonnell, R.G. Reynolds, and D.B. Fogel (eds.), Proceedings of the Fourth Annual Conference on Evolutionary Programming, Cambridge, Massachusetts, MIT Press, 449-463.
- [Myu97] Myung, H., J.H.Kim. (1997). *Evolian: Evolutionary optimization based on Lagrangian with constraint scaling*. In P.J. Angeline, R.G. Reynolds, J.R. McDonnell, and R. Eberhart (eds.), Proceedings of the Sixth Annual Conference on Evolutionary Programming, Indianapolis, Springer-Verlag, 117-188.
- [Myu98] Myung, H., J.H.Kim. (1998). *Hybrid Interior-Lagrangian Penalty Based Evolutionary Optimization*. In V.W. Porto, N. Saravanan, D. Waagen, and A.E. Eiben (eds.), Proceedings of the Seventh Annual Conference on Evolutionary Programming, Springer-Verlag, 85-94.
- [Nak91] Nakano, R. (1991). *Conventional Genetic Algorithm for Job Shop Problems*. In R.K. Belew and L.B. Booker (eds.), Proceedings of the Fourth International Conference on Genetic Algorithms, San Mateo, California, Morgan Kaufmann Publishers, 474-479.
- [Nash50] Nash, J. (1950). *The bargaining problem*. *Econometrica* 18:155-162.
- [Nee99] Neef, M., Thierens, D., Arciszewski, H. (1999). *A case study of multiobjective recombinative genetic algorithm with coevolutionary sharing*. In Proceedings of the Congress on Evolutionary Computation (CEC-1999), 796-803.
- [Nel65] Nelder, A., Mead, R. (1965). *A Simplex Method for Function Minimization*. *Computer Journal*, 7:308-313.
- [Orv93] Orvosh, D., Davis, L. (1993). *Shall We Repair? Genetic Algorithms, Combinatorial Optimization and Feasibility Constraints*. In Stephanie Forrest (ed.), Proceedings of the Fifth International Conference on Genetic Algorithms, San Mateo, California, Morgan Kauffman Publishers, 650.

- [Orv94] Orvosh, D., Davis, L. (1994). *Using a Genetic Algorithm to Optimize Problems with Feasibility Constraints*. In Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE Press, 548-553.
- [Osy78] Osyczka, A. (1978). *An approach to multicriterion optimization for engineering desing*. Computer Methods in Applied Mechanics and Engineering, 15:309-333.
- [Osy81] Osyczka, A. (1981). *An approach to multicriterion optimization for structural design*. In Proceedings of International Symposium on Optimal Structural Design. University of Arizona.
- [Osy85] Osyczka, A. (1984). *Multicriteria optimization for engineering design*. John S. Gero (ed.), Design Optimization, Academic Press, 193-227.
- [Osy95] Osyczka, A., Kundu, S. (1995). *A new method to solve generalized multicriteria optimization problems using the simple genetic algorithm*. Structural Optimization, 10(2):94-99.
- [Par96] Pareto, V. (1896). *Cours D'Economie Politique, volume I and II*. F. Rouge, Lausanne.
- [Pare94] Paredis, J. (1994). *Co-evolutionary Constraint Satisfaction*. In Proceedings of the 3rd Conference on Parallel Problem Solving from Nature, New York, Springer Verlag, 46-55.
- [Parm94] Parmee, I.C., Purchase, G. (1994). *The development of a directed genetic search technique for heavily constrained design spaces*. In I.C. Parmee (ed.), Adaptive Computing in Engineering Design and Control-'94, Plymouth, U1, University of Plymouth, 97-102.
- [Per97] Périaux, J., Sefrioudi, M., Mantel, B. (1997). *GA Multiple Objective Optimization Strategies for Electromagnetic Backscattering*. In D. Quagliarella, J. Périaux, C. Poloni, and G. Winter (eds.), Genetic Algorithms and Evolution Strategies in Engineering and Computer Science. Recent Advances and Industrial Applications, John Wiley and Sons, West Sussex, England, 11:225-243.
- [Pom00] Pomares, H., Rojas, I., Ortega, J., Gonzalez, J., Prieto, A. (2000). *A systematic approach to a self-generating fuzzy rule-table for function approximation*.

- IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics, 30(3):431-447.
- [Pow93] Powell, D., Skolnick, M.M. (1993). *Using genetic algorithms in engineering design optimization with non-linear constraints*. In Stephanie Forrest (ed.), Proceedings of the Fifth International Conference on Genetic Algorithms, San Mateo, California, University of Illinois at Urbana-Champaign, Morgan Kaufmann Publishers, 424-431.
- [Rad91] Radcliffe, N.J. (1991). *Formal analysis and random respectful recombination*. Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, 222-229.
- [Rao87] Rao, S.S. (1987). *Game theory approach for multiobjective structural optimization*. Computers and Structures, 25(1):119-127.
- [Ray01] Ray, T., Tai, K., Seow, K.C. (2001). *An evolutionary algorithm for multiobjective optimization*. Engineering Optimization, 33(3):399-424.
- [Rec73] Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution (Evolutionary Strategy: Optimization of Technical Systems According to the Principles of Biological Evolution)*. Frommann-Holzboog, Verlag, Stuttgart.
- [Rey94] Reynolds, R.G. (1994). *An Introduction to Cultural Algorithms*. In A.V. Sebald, and L.J. Fogel (eds.), Proceedings of the Third Annual Conference on Evolutionary Programming, World Scientific, River Edge, New Jersey, 131-139.
- [Richa89] Richardson, J.T., Palmer, M.R., Liepins, G., Hilliard, M. (1989). *Some guidelines for genetic algorithms with penalty functions*. In J. David Shaffer (ed.), Proceedings of the Third International Conference on Genetic Algorithms, George Mason University, Morgan Kaufmann Publishers, 191-197.
- [Riche95] Le Riche, R.G., Knopf-Lenoir, C., Haftka, R.T. (1995). *A Segregated Genetic Algorithm for Constrained Structural Optimization*. In Larry J. Eshelman (ed.), Proceedings of the Sixth International Conference on Genetic Algorithms, San Mateo, California, University of Pittsburgh, Morgan Kaufmann Publishers, 558-565.

- [Rit94] Ritzel, B.J., Eheart, J.W., Ranjithan, S. (1994). *Using genetic algorithms to solve a multiple objective groundwater pollution containment problem*. Water Resources Research, 30(5):1589-1603.
- [Ros67] Rosenberg, R.S. (1967). *Simulation of genetic populations with biochemical properties*. PhD thesis, University of Michigan, Ann Harbor, Michigan.
- [Rou00] Roubos, J.A., Setnes, M. (2000). *Compact fuzzy models through complexity reduction and evolutionary optimization*. In Proceedings of FUZZ-IEEE-2000, San Antonio, Texas, USA, 762-767.
- [Rou01] Roubos, J.A., Setnes, M. (2001). *Compact and transparent fuzzy models and classifiers through iterative complexity reduction*. IEEE Transactions on Fuzzy Systems, 9(4):516-524.
- [Rus98] Russo, M. (1998). *FuGeNeSys - a Fuzzy Genetic Neural System for Fuzzy Modeling*. IEEE Transactions on Fuzzy Systems, 6(3):373-388.
- [Sch85] Schaffer, J.D. (1985). *Multiple objective optimization with vector evaluated genetic algorithms*. Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms, Lawrence Erlbaum, 93-100.
- [Sch87] Schaffer, J.D., Morishima, A. (1987). *An adaptive crossover distribution mechanism for genetic algorithms*. J.J. Grefenstete (ed.), Genetic Algorithms and their Applications, Hillsdale, New Jersey, Lawrence Erlbaum Associates, 36-40.
- [Sch89] Schaffer, J.D., Caruana, R.A., Eshelman, L.J., Das, R. (1989). *A study of control parameters affecting online performance of genetic algorithms for function optimization*. J.D. Schaffer (ed.), Proceedings of the Third International Conference on Genetic Algorithms, San Mateo, California, Morgan Kaufmann, Inc. 51-60.
- [Scho93] Shoenauer, M., Xanthakis, S. (1993). *Constrained GA Optimization*. In Stephanie Forrest (ed.), Proceedings of the Fifth International Conference on Genetic Algorithms, San Mateo, California, Morgan Kauffman Publishers, 573-580.

- [Schw81] Schwefel, H.P. (1981). *Numerical Optimization for Computer Models*. Wiley, Chichester, UK.
- [Set95] Setnes, M. (1995). *Fuzzy Rule Base Simplification Using Similarity Measures*. M.Sc. thesis, Delft University of Technology, Delft, the Netherlands.
- [Set98] Setnes, M., Babuška, R., Verbruggen, H.B. (1998). *Rule-Based Modeling: Precision and Transparency*. IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications & Reviews, 28:165-169.
- [Set99] Setnes, M., Roubos, J.A. (1999). *Transparent Fuzzy Modeling using Fuzzy Clustering and GA's*. 18th International Conference of the North American Fuzzy Information Processing Society, 198-202.
- [Smi92] Smith, A.E., Forrest, S., Perelson, A.S. (1992). *Searching for diverse, co-operative populations with genetic algorithms*. Technical Report TCGA No. 92002, University of Alabama, Tuscaloosa, Alabama.
- [Smi93a] Smith, A.E., Forrest, S., Perelson, A.S. (1993). *Population Diversity in an Immune System Model: Implications for Genetic Search*. In L. Darrell Whitley (ed.), *Foundations of Genetic Algorithms*, San Mateo, California, Morgan Kaufmann Publishers, 2:153-165.
- [Smi93b] Smith, A.E., Tate, D.M. (1993). *Genetic Optimization Using a Penalty Function*. In Stephanie Forrest (ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms*, San Mateo, California, University of Illinois at Urbana-Champaign, Morgan Kaufmann Publishers, 499-503.
- [Smi97] Smith, A.E., Coit, D.W. (1997). *Constraint Handling Techniques - Penalty Functions*. In Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz (eds.), *Handbook of Evolutionary Computation*, chapter 5.2., Oxford University Press and Institute of Physics Publishing.
- [Smi96] Smith, M., Allen, R.G., Pereira, L. (1996). *Revised FAO Methodology for Crop Water Requirements*. Proceedings of the International Conference on evapotranspiration and irrigation scheduling, San Antonio, Texas.

- [Sol69] Solich, R. (1969). *Zadanie programowania liniowego z wieloma funkcjami celu (Linear programming problem with several objective functions)*. Przegląd Statystyczny, 16:24-30.
- [Sri94] Srinivas N., Deb, K. (1994). *Multiobjective optimization using nondominated sorting in genetic algorithms*. Evolutionary Computation, 2(3):221-248.
- [Ste74] Steuer, R.E. (1974). *ADABASE: An adjacent Efficient Basis Algorithm for Solving Vector-Maximum and Interval Weighted-Sums Linear Programming Problems*. College of Business and Economics, University of Kentucky.
- [Sto97] Storn, R., Price, K. *Differential evolution, simple and efficient heuristic for global optimization over continuous spaces*. Global Optim, 11:314-359.
- [Sug88] Sugeno, M., Kang, G.T. (1998). *Structure identification of fuzzy model*. Fuzzy Sets and Systems, 28:15-33.
- [Sur95] Surry, P.D., Radcliffe, N.J., Boyd, I.D. (1995). *A Multi-Objective Approach to Constrained Optimization of Gas Supply Networks: The COMOGA Method*. In Terence C. Fogarty (ed.), Evolutionary Computing. AISB Workshop. Selected Papers, Lecture Notes in Computer Science, Springer-Verlag, Sheffield, U.K, 166-180.
- [Sys89] Syswerda, G. (1989). *Uniform crossover in genetic algorithms*. D. Schaffer (ed.), Proceedings of the Third International Conference on Genetic Algorithms, San Mateo, California, Morgan Kauffmann Publishers, Inc., 2-9.
- [Sys91a] Syswerda, G., Palmucci, J. (1991). *The Application of Genetic Algorithms to Resource Scheduling*. In R.K. Belew and L.B. Booker (eds.), Proceedings of the Fourth International Conference on Genetic Algorithms San Mateo, California, Morgan Kaufmann, 502-508.
- [Sys91b] Syswerda, G. (1991). *Schedule optimization using genetic algorithms*. L. Davis (ed.), Handbook of Genetic Algorithms, New York, Van Nostrand Reinhold, 332-349.
- [Tak85] Takagi, T., Sugeno, M. (1985). *Fuzzy identification of systems and its application to modeling and control*. IEEE Transactions on Systems, Man and Cybernetics, 15:116-132.

- [Tam95] Tamaki, H., Mori, M., Araki, M., Ogai, H. (1995). *Multicriteria optimization algorithms: a case of scheduling in hot rolling process*. In Proceedings of the 3rd APORS, 374-381.
- [Tam96] Tamaki, H., Kita, H., Kobayashi, S. (1996). *Multi-Objective Optimization by Genetic Algorithms: A Review*. In T. Fukuda and T. Furuhashi (eds.), Proceedings of the 1996 International Conference on Evolutionary Computation, Nagoya, Japan, IEEE, 517-522.
- [Tan95] Tanaka, M., Watanabe, H., Furukawa, Y., Tanino, T. (1995). *GA-Based Decision Support System for Multicriteria Optimization*. Proceedings of the International Conference on Systems, Man and Cybernetics, 2:1556-1561.
- [Tig75] Tigan, S. (1975). *Sur le probleme de la programmation vectorielle fractionnaire*. Mathematica - Revue d'analyse numerique et de theorie de l'approximation, 4(1):99-103.
- [Tse90] Tseng, C.H., Lu, T.W. (1990). *Minimax multiobjective optimization in structural design*. International Journal for Numerical Methods in Engineering, 30:1213-1228.
- [Val97] Valenzuela-Rendon, M., Uresti-Charre, E. (1997). *A Non-Generational Genetic Algorithm for Multiobjective Optimization*. In Thomas Bäck (ed.), Proceedings of the Seventh International Conference on Genetic Algorithms, San Mateo, California. Michigan State University, Morgan Kaufmann Publishers, 658-665.
- [Vale99] Valente de Oliveira, J. (1999). *Semantic constraints for membership function optimization*. IEEE Transactions on Fuzzy Systems, 19(1):128-138.
- [Vel99] Veldhuizen, D.V., Lamont, G.B. (1999). *Multiobjective evolutionary algorithms: Classifications, Analyses, and New Innovations*. Ph. D. Thesis Dayton, OH: Air Force Institute of Technology. Technical Report No. AFIT/DS/ENG/99-01.
- [Ver82] Verdegay, J.L. (1982). *Fuzzy mathematical programming*. M.M. Gupta and E. Sánchez (eds.), Fuzzy Information and Decision Processes, 231-237.

- [Wan98] Wang, C.H., Hong, T.P., Tseng, S.S. (1998). *Integrating fuzzy knowledge by genetic algorithms*. Fuzzy Sets and Systems, 2(4):138-149.
- [Wan99] Wang, L., Yen, J. (1999). *Extracting fuzzy rules for system modeling using a hybrid of genetic algorithms and Kalman filter*. Fuzzy Sets and Systems, 101:353-362.
- [Wall96] Wallace, D.R., et al. (1996). *Design Search Under Probabilistic Specifications Using Genetic Algorithms*. Computer-Aided Design, 28(5):405-421.
- [Whi89] Whitley, D. (1989). *The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best*. J. Schaffer (ed.) Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, Los Altos, CA, 116-121.
- [Wil93] Wilson, P.B., Macleod, M.D. (1993). *Low implementation cost IIR digital filter design using genetic algorithms*. In IEE/IEEE Workshop on Natural Algorithms in Signal Processing, Chelmsford, U.K., 4:1-8.
- [Woo95] Wooldridge, M., Jennings, N.R. (1995). *Intelligent Agentes: Theory and Practice* The Knowledge Engineering Review, 2(10):115-152.
- [Wri91] Wright, A.H. (1991). *Genetic algorithms for real parameter optimization*. en G.J.E. Rawlin (ed.), Foundations of Genetic Algorithms-1, Morgan Kaufmann, San Mateo, 205-218.
- [Yan94] Yang, X., Gen, M. (1994). *Evolution program for bicriteria transportation problem*. In M. Gend and T. Kobayashi (eds.), Proceedings of the 16th International Conference on Computer and Industrial Engineering (Ashikaga, Japan), 451-454.
- [Yag78] Yager, R.R. (1978). *Competitiveness and Compensation in Decision Making: A Fuzzy Set Based Interpretation*. Iona College Tech Rep. RRY 78-14, New Rochelle, N.Y.
- [Yen98] Yen, J., Wang, L. (1998). *Application of statistical information criteria for optimal fuzzy model construction*, IEEE Transactions on Fuzzy Systems, 6(3):362-371.

- [Yok95] Yokota, T., Gen, M., Ida, K., Taguchi, T. (1995). *Optimal Design of System Reliability by an Improved Genetic Algorithm*. Transactions of Institute of Electronics, Information and Computer Engineering, J78-A(6):702-709.
- [Yu76] Yu, P.L., Zeleny, M. (1976). *Linear Multiparametric Programming by Multi-criteria Simplex Method*. Management Science, 23(2):159-170.
- [Zad65] Zadeh, L.A. (1965). *Fuzzy sets*. Information and Control, 8:338-353.
- [Zel73] Zeleny, M. (1973). *Compromise Programming*. In J.L. Cochrane and M. Zeleny (eds.) Multiple Criteria Decision Making. University of South Carolina Press, Columbia, 262-301.
- [Zel76] Zeleny, M. (1974). *Linear Multiobjective Programming*. Springer-Verlag, New York, 197-220.
- [Zit98] Zitzler, E., Thiele, L. (1998). *An evolutionary algorithm for multiobjective optimization: The strength Pareto approach*. Technical Report 43, Zürich, Switzerland: Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH).
- [Zit01a] Zitzler, E., Deb, K., Thiele, L., Coello, C.A., Corne, D. (eds.) (2001). *Evolutionary Multi-Criterion Optimization*. Lecture Notes in Computer Science, 1993. Heidelberg: Springer
- [Zit01b] Zitzler, E., Laumanns, M., Thiele, L. (2001). *SPEA2: Improving the Strength Pareto Evolutionary Algorithm*. Technical Report 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, Switzerland.
- [Zyd01] Zydallis, J.B., Veldhuizen, D.V., Lamont, G.B. (2001). *A Statistical Comparison of Multiobjective Evolutionary Algorithms Including the MOMGA-II*. Eckart Zitzler and Kalyanmoy Deb and Lothar Thiele and Carlos A. Coello Coello and David Corne (eds.), First International Conference on Evolutionary Multi-Criterion Optimization, Springer-Verlag, Lecture Notes in Computer Science, 1993:226-240.