A Edurne

Contenido

Parte I

Introducción

Capítulo 1

Presentación

1.1 Objetivos y Metodología

Como su título indica, en este trabajo se ha pretendido aplicar los Algoritmos Genéticos (AAGG) a la Clasificación No Supervisada (CNS).

La CNS es un campo científico que se encuentra enmarcado dentro del Análisis de Datos. El objetivo de la CNS es, dado un conjunto de datos, descubrir la estructura de grupos subyacente al conjunto si es que esta existe. Desde muy antiguo la CNS ha sido utilizada por investigadores de ramas diversas de la ciencia: biología, geología, arqueología, etc., y dentro de cada ámbito se han realizado aportaciones muy diferentes, lo que la convierte en una rama muy interdisciplinal. Esta interdisciplinariedad ha provocado la creación de diferentes tipos de estructuras clasificatorias dentro de la CNS. Así entre otras se pueden distinguir:

- Clasificación Particional
- Clasificación Jerárquica
- Clasificación Piramidal

La clasificación particional, dado un conjunto de datos, trata de dividir dicho conjunto en grupos de manera que, datos que están en el mismo grupo sean lo más parecidos posible entre sí, a la vez que lo más diferentes posible de datos de otros grupos.

Por su parte la clasificación jerárquica, crea, a partir de un conjunto de datos, una sucesión de grupos encajados, cuya estructura puede representarse por medio de un árbol.

La clasificación piramidal, finalmente, crea una sucesión de recubrimientos encajados, de tal forma, que dicha estructura puede representarse gráficamente por medio de un grafo, que cumple unas propiedades especiales.

Los AAGG pueden considerarse como una técnica de optimización estocástica, aparecida dentro del ámbito de la Inteligencia Artificial (IA). Esta técnica de optimización, al igual que muchas provenientes de la IA, se fundamenta en la naturaleza. En particular en simular los principios de selección natural y supervivencia de los más adaptados enunciados por Darwin.

Estos algoritmos tienen su origen en los trabajos de Holland [77], quien estableció sus principios básicos en 1975. Sin embargo, han sido los últimos años de los ochenta y principio de los noventa cuando los AAGG se han extendido dentro de la comunidad de la IA.

El objetivo de la tesis consistía en estudiar la forma en la que dos ramas, relativamente separadas de la ciencia -los AAGG y la CNS- podían unirse para obtener mejores resultados.

En particular los objetivos buscados en cada uno de los diferentes tipos de CNS fueron los siguientes:

1) En *clasificación particional* se pretendió utilizar los AAGG, para llevar a cabo, no sólo la distribución de un conjunto de datos en grupos, de manera óptima, sino que al mismo tiempo se pretendía que el propio algoritmo hallase el número de grupos óptimo.

2) En el caso de la *clasificación jerárquica* el objetivo fué intentar hallar, la estructura de jerarquía indexada que mejor representaba la estructura del conjunto de datos. Esta búsqueda se llevo a cabo en el conjunto de las disimilaridades ultramétricas.

3) Al igual que en la clasificación jerárquica, en la *clasificación piramidal* se intentó hallar la estructura de pirámide indexada que mejor representaba al conjunto de datos.

Al mismo tiempo, el estudio de los AAGG nos ha llevado a proponer un nuevo algoritmo, el cual puede verse como un híbrido entre el

algoritmo de Enfriamiento Estadístico (EE) y los AAGG, o como una versión probabilista de los AAGG elististas. Para dicho algoritmo se da un modelo matemático con el que se concluye su convergencia hacia el óptimo.

1.2 Organización de la Tesis

La tesis está dividida en cinco partes:

Parte I: Introducción

Parte II: Algoritmos Genéticos

Parte III: Clasificación No Supervisada

Parte IV: Algoritmos Genéticos aplicados a la Clasificación No Supervisada

Parte V: Conclusiones

La parte II está dedicada, por un lado a introducir los AAGG, y por otro a describir en detalle el algoritmo novedoso propuesto y la demostración de su convergencia. En el primer capítulo de la misma se introduce el algoritmo genético (AG) simple, utilizando un ejemplo para su ilustración. Se introducen también en este capítulo los mecanismo básicos del AG simple, dando además algunas extensiones de los operadores básicos, selección, cruce y mutación, así como algunas guías para el ajuste de los parámetros. Por último se introduce en detalle el AG particular utilizado para la consecución de los experimentos.

El capítulo 3 comienza describiendo un modelo matemático del AG simple. Se continúa introduciendo el algoritmo de EE, describiendo su modelado mediante cadenas de Markov y estableciendo sus condiciones de convergencia. Basados en lo anterior revisamos la bibliografía en la que se relacionan los AAGG y el EE, para posteriormente proponer un nuevo algoritmo, y tras hallar los elementos de la matriz de transición de la cadena de Markov no homogenea que lo modela se demuestra su convergencia hacia el óptimo.

La parte III, está dedicada a introducir de manera muy general los problemas que aparecen en la CNS, y en mayor detalle los tres tipos de clasificación en los que estamos interesados. Las diferentes tareas se exponen en el mismo orden, en el que habría que hacerlo, al enfrentarse con un problema real. Comenzamos estudiando, los tipos y las formas en las que pueden venir representados los datos de entrada a cualquier algoritmo de CNS, para pasar a introducir algunas técnicas de visualización. A continuación se revisan de forma bastante detallada los tres tipos de clasificación: particional, jerárquica y piramidal, terminando comentando algunos conceptos de validación.

La penúltima parte, consta de tres capítulos que junto con el tercero forman el núcleo de las aportaciones novedosas de esta memoria.

El capítulo 6 dedicado a la aplicación de los AAGG a la clasificación particional, comienza realizando un análisis bibliográfico de la aplicación de los AAGG en este tipo de clasificación. La aproximación propuesta es dada a continuación, primero para conjuntos en $\mathbb{R}^2$, extendiéndola, a la vista de los resultados, a conjuntos p-dimensionales ($p > 2$). Terminamos viendo como la realización de una selección de casos, utilizando de nuevo los AAGG, puede acelerar el proceso sin afectar apenas a los resultados finales. Se analiza estadísticamente el comportamiento del AG en función de los distintos operadores.

La aplicación de los AAGG a la clasificación jerárquica se expone en el capítulo 6. Comenzamos analizando los trabajos previos dedicados a la búsqueda de la clasificación jerárquica óptima. La propuesta hecha en esta memoria, se describe a continuación. Se comprueba experimentalmente que esta aproximación puede ser usada no sólo con conjuntos pequeños sino también con grandes. Nuevamente tras los experimentos se realiza un análisis estadístico de los operadores.

El último capítulo de esta parte es donde se usan los AAGG para intentar alcanzar una representación piramidal óptima de un conjunto de datos. El problema es dividido en dos partes. Por una parte se trata de hallar un orden óptimo en un conjunto de datos y por otro la resolución de un problema de programación cuadrática concreto. La resolución de este último problema, nos conduce a plantear algunos resultados teóricos sobre la estructura piramidal óptima. Nuevamente son realizados experimentos para constatar la validez de la aproximación, analizándolos además estadísticamente.

La última parte contiene un único capítulo dedicado a recoger las conclusiones de este trabajo y el trabajo futuro.

1.3 Guía de Lectura

Se ha tratado la difícil tarea de hacer esta memoria autocontenida. En cualquier caso, la mejor forma de leerla depende de los conocimientos del lector.

Para un conocedor de temas relacionados con CNS le sería suficiente con leer el capítulo 2, para entender la parte de aplicación de aplicación de los AAGG a la CNS. Si por el contrario el lector es una persona conocedora de los principios de los AAGG, podría obviar la lectura del capítulo 2, excepto quizás los apartados 2.4 y 2.5., pasando directamente a la lectura del capítulo 4.

Finalmente para entender el AG propuesto es necesario la lectura completa del capítulo 3 y del capítulo si no conoce nada sobre AAGG.

Parte II

Algoritmos Genéticos

Capítulo 2

Algoritmos Genéticos

2.1 Introducción

Este tema está dedicado a introducir los AAGG así como a describir en detalle, el algoritmo concreto utilizado en la realización de los experimentos en los diferentes tipos de CNS.

Para situar a los AAGG de forma adecuada, deberíamos empezar por tener en cuenta una cantidad de técnicas que han aparecido dentro de la IA más o menos recientemente y dentro de cuyos objetivos se encuentra la optimización. En este conjunto de técnicas existen tres que son las más extendidas y utilizadas, éstas son:

- Enfriamiento Estadístico (Van Laarhoven y Aarts [92])
- Algoritmos Evolutivos (Bäck [13], Rudolph [140])
- Búsqueda Tabú (Glover [62], [63]).

Casi todos estos algoritmos están basados en simular principios naturales o procesos que se producen en la naturaleza. Así, el Enfriamiento Estadístico intenta simular el proceso de "annealing" de un sólido (en inglés se denomina "simulated annealing"), es decir, el proceso por el cual un sólido es calentado hasta altas temperaturas, para dejarlo enfriar poco a poco de manera que alcance el equilibrio. Los Algoritmos Evolutivos por su parte, tratan de simular la evolución de las especies de acuerdo con los postulados de Darwin. La Búsqueda

Tabú quizás sea la que más se aleje de esta simulación de la naturaleza, aunque sigue teniendo ciertas conexiones. Esta técnica de optimización hace uso de memoria como principal herramienta a la hora de llevar a cabo la búsqueda.

Es posible encontrar textos en la literatura como por ejemplo Reeves [133], Reeves y col. [134] y Díaz [47] que recogen de manera relativamente profunda este tipo de técnicas.

Dentro de los Algoritmos Evolutivos se pueden distinguir principalmente tres tipos de técnicas:

- Estrategias Evolutivas (Rechenberg [132], Shwefel [143])
- Programación Evolutiva (Fogel [59], Fogel y col. [60])
- Algoritmos Genéticos (Holland [77]).

Las Estrategias Evolutivas, han sido usadas principalmente en el campo de la optimización numérica. Esta es la principal diferencia con los AAGG, los cuales, fueron aplicados inicialmente para resolver problemas de optimización combinatoria.

Por su parte la Programación Evolutiva ha sido utilizada mayormente en problemas de optimización combinatoria y su principal diferencia con los AAGG es la utilización únicamente del operador de mutación para conseguir la nueva población, dejando de lado el operador de cruce.

Los AAGG son la técnica que desarrollaremos en este capítulo y la utilizada en el resto de la memoria para llevar a cabo la tarea de la CNS. Esta técnica de optimización tiene como precursor a Holland [77], aunque quien principalmente ha contribuido a su extensión y divulgación ha sido Goldberg con su libro (Goldberg [64]): "Genetic algorithms in search, optimization, and machine learning". Desde la aparición del libro de Goldberg la literatura en el campo de los AAGG se ha incrementado de manera espectacular, existiendo en la actualidad gran cantidad de libros, Davis [44], Michalewicz [115], Mitchell [121] donde se exponen los principios de este algoritmo y gran parte de sus aplicaciones.

Los AAGG tratan de simular el comportamiento de las especies a lo largo del tiempo. Así como las poblaciones evolucionan por medio de

la reproducción y por algunos cambios aleatorios que se producen en su material genético (mutación), de la misma forma funcionan los AAGG. Estos algoritmos mantienen en cada paso de los mismos un conjunto de soluciones a un problema dado, las cuales se mezclan entre ellas (reproducción) dando lugar a nuevas soluciones. Las nuevas soluciones obtenidas son modificas de manera aleatoria, de tal forma que éstas tras la modificación pasan a formar parte de una nueva población. Otro de los principios de Darwin: "la supervivencia de los mejores adaptados" también se aplica en los AAGG. De la misma manera que en el entorno natural los individuos compiten por una cantidad de recursos escasos, y únicamente los más adaptados sobreviven y son capaces de reproducirse o lo hacen en mayor medida, así en los AAGG, son las mejores soluciones mantenidas en una iteración del algoritmo, las que se mezclan para producir las nuevas soluciones.

Este paralelismo con la evolución de las especies hace que la nomenclatura utilizada en el campo de los AAGG sea muy parecida a la utilizada en el campo de la evolución. Por ejemplo, un conjunto de soluciones será llamado *población*, una solución en particular se llamará *individuo* o a veces *cromosoma*, las componentes que forman los individuos se denominarán *genes*, etc.

Los AAGG, desde su aparición, han sido aplicados a gran cantidad de problemas prácticos de optimización donde se ha demostrado su eficacia. La principal cualidad de los AAGG es su robustez. Son capaces de obtener buenas soluciones en tiempos de computación competitivos con el resto de algoritmos de optimización. Aunque para aquellos problemas para los que existan algoritmos específicos, normalmente estos funcionan mejor que los AAGG, es posible encontrar en la literatura situaciones, donde la creación de nuevos algoritmos híbridos entre AAGG y los algoritmos específicos, ha conducido a mejorar los resultados existentes hasta el momento. Ésto unido a la facilidad con la que es posible implementarlos ha llevado a la existencia de un crecimiento exponencial en las aplicaciones de los mismos.

En lo que sigue se comenzará exponiendo el AG simple o canónico. A continuación desarrollaremos en más profundidad algunos aspectos de los AAGG que quedan fuera del AG simple, a la vez que algunas extensiones de los componentes básicos. La siguiente sección será dedicada al estudio del esquema general del AG utilizado en la realización

de los experimentos. En una última sección se expondrán los operadores de cruce y mutación, que se utilizaron en el AG que se aplicó a los distintos tipos de CNS.

2.2 El Algoritmo Genético Simple

Daremos en esta sección el AG simple o canónico, el cual, es el tipo más básico de AG a la vez que el punto de partida para las secciones que vienen a continuación. Estudiaremos los mecanismos básicos que envuelven los AAGG en detalle, para pasar en la siguiente sección a analizar las variaciones y extensiones de estos mecanismos.

El AG simple mantiene en cada paso del algoritmo un conjunto de soluciones a un problema al que llamaremos población. Cada elemento de la población, es decir, cada posible solución al problema será llamado individuo. En el caso del AG simple, las posibles soluciones del problema que se quiere optimizar vienen codificadas como vectores 0-1. Es decir, cada individuo de la población es un vector 0-1 de longitud l que codifica una solución. Evidentemente existe una función F, llamada función objetivo o función de adaptación, que mide el valor que tiene cada individuo de cara a resolver el problema. En términos evolutivos la función F mide el grado de adaptación del individuo al entorno. Si nos encontramos ante un problema de optimización normalmente la función F suele ser la propia función a optimizar.

Cada iteración del AG simple consiste en la realización de tres operaciones básicas: selección, reproducción y mutación.

La primera de estas tres operaciones realiza una selección de dos individuos de la población. Esta selección suele ser llevada a cabo de manera aleatoria pero proporcional al valor que la función objetivo F toma en cada individuo.

La segunda operación, es decir la reproducción, se efectúa con cierta probabilidad. Esta operación consiste en la aplicación de un operador de cruce. Este operador de cruce trata de mezclar las componentes de los dos individuos seleccionados. En el caso del AG simple se suele utilizar lo que de aquí en adelante llamaremos *operador de cruce basado en un punto*. Este operador trata de simular la mezcla genética que se produce en la reproducción en el mundo natural. En el caso de que

obtenga que los individuos no se cruzan estos pasan directamente a formar parte de la siguiente población.

La última de las operaciones consiste en aplicar un operador de mutación. En este caso el operador modifica el valor de cada gen de un individuo de acuerdo con una probabilidad.

Un paso del algoritmo consiste en la aplicación de estas tres operaciones hasta conseguir una nueva población.

En la figura 2.1 se puede ver un pseudo-código para el AG simple.

```
Inicializar probabilidad de cruce p_c
Inicializar probabilidad de mutación p_m
Hallar la población inicial P_0
Hasta condición_parada = TRUE hacer
  begin
  hacer n/2 veces
    begin
    Seleccionar dos individuos de la población P_k
    Cruzar los dos individuos con probablidad p_c
    Mutar los dos individuos resultantes con probabilidad p_m
    Introducir los dos nuevos individuos en la población P_{k+1}
    end
  end
Devolver la mejor solución
```

Figura 2.1: Pseudo-código para el AG simple

Analizaremos cada una de las componentes por separado.

Selección

En el AG simple para realizar la selección es necesaria la asignación de una probabilidad de selección a cada individuo de la población. Esta probabilidad de selección es proporcional al valor que la función objetivo toma en los individuos. Si denotamos por $I_1, I_2, \ldots, I_n$ los individuos de la población entonces la probabilidad de elegir el individuo I_i

sería:

$$\frac{F(I_i)}{\sum_{j=1}^{n} F(I_j)}.$$

De esta forma se logra que los individuos mejor adaptados (con mayor valor de función objetivo) tengan mayor probabilidad de ser elegidos para la reproducción.

El muestreo de estas distribuciones de probabilidad que se crean para la selección se suele hacer mediante el método de "la ruleta sesgada". Este método consiste en dividir una circunferencia en arcos de manera que cada trozo de arco se corresponda con un individuo de la población. El tamaño de los arcos es proporcional al valor de la función objetivo. De esta forma, si estamos en un problema de maximización, cuanto más grande sea el valor de la función objetivo más grande será el tamaño del arco correspondiente a dicho individuo. La figura 2.2 muestra un ejemplo de lo anterior.

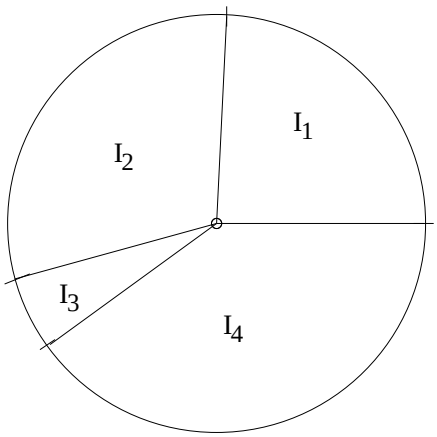

Figura 2.2: Ejemplo de la aplicación de la ruleta sesgada

Una vez que se tiene dividida la ruleta se muestrea una distribución uniforme en el intervalo $[0, 2\pi]$. El número obtenido se hace corresponder con un ángulo expresado en radianes y dicho ángulo se corresponde con un trozo de arco de la circunferencia y por lo tanto con un individuo.

Cruce

El operador de reproducción en el caso del AG simple consiste en el cruce basado en un punto. El operador tiene como entrada dos individuos. Comienza obteniendo un número aleatorio entre 1 y $l-1$ (suponemos que l es el tamaño del individuo) que será el punto de cruce. Los dos nuevos individuos se forman de la siguiente forma: el primero con la parte del primer individuo antes del punto de cruce y la parte del segundo individuo que se encuentra después del punto de cruce; el segundo individuo nuevo se forma justo de manera contraria al primero. En la figura 2.3 se puede ver un esquema de aplicación del operador de cruce basado en 1 punto.

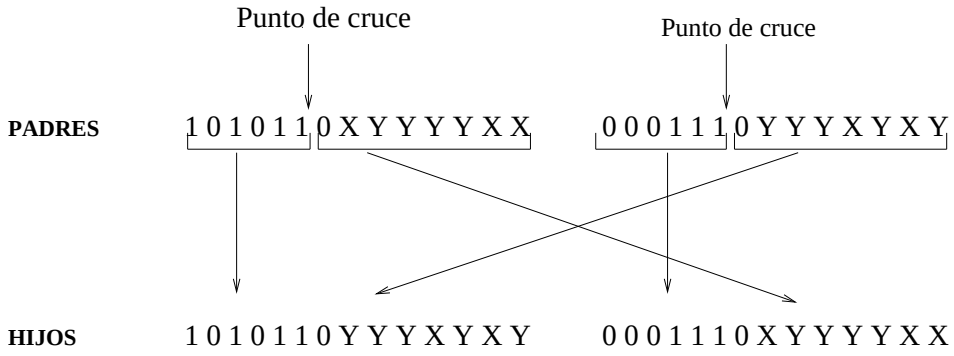

Figura 2.3: Ejemplo del cruce en 1 punto

Mutación

El último operador que se aplica para obtener los nuevos individuos que forman la nueva población es el operador de mutación. Este operador consiste en cambiar el valor de cada gen de los individuos provenientes del cruce con cierta probabilidad p_m.

Para ejecutar este operador, se suele muestrear una distribución uniforme en el intervalo $[0, 1]$. Si dicho valor es menor que p_m, entonces

si el gen es mutado. Si éste tenía un valor de 1 pasa a tener un valor de 0 y viceversa (ver figura 2.4).

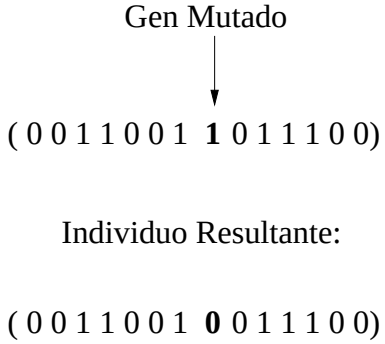

Figura 2.4: Esquema de aplicación de la mutación

Ejemplo

Para ilustrar cada una de las componentes y para entender en mayor medida el funcionamiento del AG simple, veremos a continuación un pequeño ejemplo. El ejemplo ha sido adaptado de Reeves [133]. Suponemos que queremos maximizar la siguiente función:

$$F(x) = x^3 - 60x^2 + 900x + 100.$$

Esta función vamos a tratar de optimizarla en el conjunto de números $\{0, 1, 2, \ldots, 31\}$. Evidentemente maximizar esta función es muy fácil en un espacio tan pequeño, basta proceder por enumeración. Se pretende, por lo tanto, que el valor del ejemplo sea meramente ilustrativo. Dada la magnitud del espacio de búsqueda, podemos suponer que cada posible solución al problema puede ser codificada mediante un vector 0-1 de longitud 5, por ejemplo (0 1 1 0 1). Supondremos también que la población tiene cuatro individuos y que la inicial ha sido generada aleatoriamente. Para simplificar aún más el problema asumiremos que la probabilidad de cruce es $p_c = 1$. Esta suposición es habitual cuando se aplican los AG en la práctica. La tabla 2.1 recoge los datos referentes a la población inicial.

Nº.	Individuo	x	$F(x)$	Prob. de selección
1	(1 0 0 1 1)	19	2399	.2441
2	(0 0 1 0 1)	5	3225	.3282
3	(1 1 0 1 0)	26	516	.0525
4	(0 1 1 1 0)	14	3684	.3750
Valor medio de F en la población: 2456				

Tabla 2.1: Datos de la población inicial

Los resultados de seleccionar dos individuos, cruzarlos y mutarlos se pueden ver en la tabla 2.2. Al ser el tamaño de la población tan pequeño las probabilidades de selección y de mutación quedan muy poco reflejadas Así, por ejemplo, ningún individuo ha sido mutado.

Padre 1	Padre 2	P. Cruce	Mut.	Hijo 1	Hijo 2
(0 0 1 0 1)	(0 1 1 1 0)	2	No	(0 0 1 1 0)	(0 1 1 0 1)
(1 0 0 1 1)	(0 1 1 1 0)	3	No	(1 0 0 1 0)	(0 1 1 1 1)

Tabla 2.2: Población en el tiempo 1 tras aplicar una iteración del AG simple a la población de la tabla 2.1

Por último los valores que los individuos de la nueva población toman en la función objetivo quedan reflejados en la tabla 2.3, donde se puede comprobar además, como ha mejorado el valor medio de la función objetivo en la nueva población, en relación a este valor medio en la población inicial.

Nº.	Individuo	x	$F(x)$	Prob. de selección
1	(0 0 1 1 0)	6	3556	.2615
2	(0 1 1 0 1)	13	3875	.2849
3	(1 0 0 1 0)	18	2692	.1979
4	(0 1 1 1 1)	15	3475	.2555
Valor medio de F en la población: 3399.5				

Tabla 2.3: Datos de la población tras una iteración del AG simple

Una vez expuesto el funcionamiento del AG simple, es evidente que hay muchos aspectos que pueden ser modificados. Por ejemplo, el tipo de operadores a utilizar, parámetros para los que se necesita establecer que valores son los más apropiados, tamaño de la población, probabilidad de cruce, probabilidad de mutación. Al mismo tiempo, se intuye también el hecho, de que se podrían llevar a cabo una gran cantidad de modificaciones dentro del AG simple, que podrían conducirnos a realizar una mejor búsqueda del óptimo. Analizaremos en la siguiente sección todos estos aspectos.

2.3 Modificaciones y Extensiones del AG simple

Aunque muchos de los aspectos que veremos a continuación están interrelacionados unos con otros por ejemplo, el tamaño de los individuos suele estar relacionado con la probabilidad de mutación, etc. intentaremos analizar cada componente por separado, apuntando en cada caso este tipo de relaciones. Comenzaremos con aspectos que tienen que ver con la población.

Tamaño de la población

Evidentemente el tamaño de la población parece un aspecto crucial en el AG. Si por ejemplo utilizamos poblaciones pequeñas corremos el riesgo de no cubrir todo el espacio de búsqueda, sin embargo, utilizar grandes poblaciones puede suponer un esfuerzo computacional excesivo. Existen en la literatura bastantes trabajos dedicados a proponer valores que supongan un compromiso entre la cantidad de espacio de búsqueda que se cubre y el gasto computacional en que incurre.

Goldberg [65] indica que el tamaño de la población óptima con individuos binarios crece, desde un punto de vista teórico, exponencialmente con la longitud del individuo. Sin embargo, esto sería imposible ponerlo en práctica en problemas reales. Posteriormente utilizando diferentes argumentos afirma que en el caso de la aplicación práctica de los AAGG, un tamaño de población relativamente pequeño es suficiente. Otros estudios experimentales realizados por Alander [4] sugieren que

un tamaño de población entre l y $2l$ es suficiente en la mayoría de las aplicaciones prácticas.

Por otro lado, Reeves [135] trata el problema de la utilización de AAGG con poblaciones pequeñas, dando un valor mínimo que deben tener estas poblaciones para cumplir ciertas propiedades. Se ve también en dicho trabajo que el uso de codificaciones no binarias hace que dicho valor mínimo aumente de forma importante en relación con el necesario para individuos binarios.

Otros autores, Arabas y col. [10], proponen que el uso de tamaños de población variables es beneficioso para la búsqueda. Sin embargo, esto conlleva otro problema, esto es, en cada momento de la búsqueda, ¿cuál es el tamaño más adecuado?

Población Inicial

En el AG simple hemos visto que la población inicial era generada de forma aleatoria, sin embargo, es fácil pensar en otras posibilidades.

Reeves [134] expone una forma de inicializar la población que es más conveniente que la aleatoria en el caso de tener que trabajar con poblaciones pequeñas.

No es difícil encontrar trabajos dentro de la bibliografía, por ejemplo Reeves [135], donde se indican utilizaciones de los AAGG donde la población inicial está formada por soluciones obtenidas mediante la aplicación de otro optimizador al problema. Aunque esta forma de inicializar la población puede ayudar al algoritmo a conseguir mejores soluciones en menor tiempo, también tiene el problema de que esta utilización de muy buenos individuos iniciales puede conducir a la convergencia prematura. Este problema consiste en que la existencia de un individuo cuyo valor de la función objetivo sea mucho mejor que los demás (un superindividuo), el cual en muchos casos es un óptimo local, conduce en pocas iteraciones hacia una población en la que todos los individuos son iguales al superindividuo.

Selección

En el algoritmo propuesto por Holland [77] la selección se llevaba a cabo de manera proporcional al valor que cada individuo tomaba en la

función objetivo. De esta forma si denotamos por I_i el i-ésimo individuo de la población, la probabilidad de seleccionar dicho individuo viene dada por:

$$p_i = \frac{F(I_i)}{\sum_{j=1}^{n} F(I_j)}.$$

Evidentemente este mecanismo ha sido modificado y muchos otros tipos de selección pueden encontrarse en la literatura. Expondremos a continuación algunos de ellos.

La principal crítica recibida por la selección utilizada por el AG simple es que este tipo de selección conduce al problema de la convergencia prematura. Para subsanar este problema surgió la selección basada en el rango. Esta selección consiste en que la probabilidad de seleccionar un individuo no dependa directamente del valor que toma la función objetivo en él, sino del lugar (rango) que ocupa dicho valor en el conjunto de la población. De esta forma si definimos $rg(F(I_i))$ como el rango de la función objetivo cuando los individuos de la población han sido ordenados de mayor a menor teniendo en cuenta su valor de función objetivo, (el de menor función objetivo sería el primero y el de mayor el último, suponiendo que estoy maximizando), entonces la probabilidad de selección es en este caso:

$$p_i = \frac{rg(F(I_i))}{n(n-1)/2}.$$

Utilizando las anteriores dos técnicas de selección ocurre en muchos casos que el resultado real de la selección está muy lejos del resultado esperado. Para solucionar este problema se ideó la *selección de valor esperado* De Jong [85]. En esta técnica de selección se intenta forzar a que los individuos sean seleccionados tantas veces como su valor esperado. Esto se hace asignando a cada individuo un contador inicializado a $F(I_i)/\overline{F}$ donde $\overline{F}$ representa la media de la función objetivo en la población. Una vez que un individuo ha sido elegido, dicho contador decrece una cantidad c ($c \in (0.5, 1)$). El individuo no podrá ser seleccionado más veces cuando su contador tome un valor menor o igual que cero.

Otra técnica parecida a la anterior Brindle [29] consiste en seleccio-

nar cada individuo el número de veces esperado, es decir,

$$\left[\frac{F(I_i)}{\overline{F}}\right]$$

veces, este es, la parte entera del número esperado de ocurrencias. Los individuos que faltan hasta completar la selección de los n, son elegidos aleatoriamente con una probabilidad proporcional a $\frac{F(I_i)}{\overline{F}} - \left[\frac{F(I_i)}{\overline{F}}\right]$.

Baker [14] propone otra técnica, llamada *muestro universal estocástico*, que está en la misma línea que las anteriores. El autor tiene en cuenta la ruleta sesgada, al igual que en el primer método, pero en lugar de realizar varios giros, Baker utiliza un único giro para seleccionar todos los individuos. Para ello dispone de un sistema de marcadores igualmente espaciados cuya posición se elige aleatoriamente. En la figura 2.5 se puede ver un ejemplo de ello. En este caso el individuo I_1 ha sido seleccionado una vez, el individuo I_2 otra, I_3 ninguna, mientras que I_4 ha sido seleccionado dos veces.

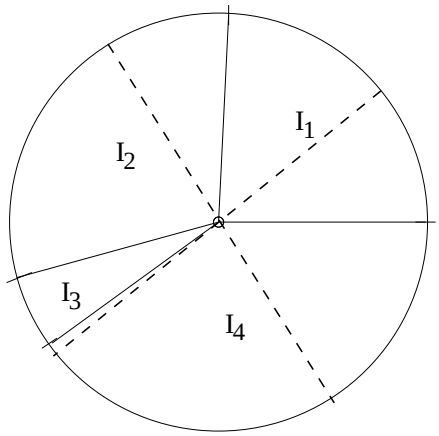

Figura 2.5: Ejemplo de la aplicación del muestreo universal estocástico

Por último cabe destacar la *selección por torneo*. Esta técnica de selección consiste en escoger de manera aleatoria un conjunto de individuos de la población y efectuar dentro de ellos una competición decidiendo finalmente que individuo es seleccionado. En esta competición

por supuesto influye el valor de la función objetivo. Habitualmente el número de individuos elegidos para llevar a cabo el torneo es dos. En este caso la selección más habitual utilizada es la selección por torneo basada en la distribución de Boltzmann Mahfoud [112], y que más tarde comentaremos en mayor detalle.

Comparaciones entre diferentes tipos de selección pueden encontrarse en Goldberg y Deb [67] y en Bäck [12]. En el primer trabajo los autores apuntan que la selección basada en el rango y la selección por torneo obtienen mejores resultados que la selección proporcional. En el segundo trabajo, Bäck, analiza distintos criterios de selección en cuanto a su presión selectiva sobre la población. Esta presión selectiva la mide en función del número de iteraciones de la selección necesarias para obtener un población compuesta por todos los individuos iguales al mejor individuo de la población inicial. Sus resultados son que la presión selectiva aumenta en el siguiente orden: selección proporcional, selección basada en el rango, selección por torneo de Boltzmann.

Operadores de cruce

En el AG simple hemos visto el operador de cruce más básico, esto es, el operador de cruce basado en un punto. Este operador se puede generalizar fácilmente.

Una primera generalización de este operador consiste en el operador de cruce en múltiples puntos. En este caso se eligen varios puntos dentro de los cromosomas y se intercambian las subcadenas entre dichos puntos. En un primer trabajo realizado por De Jong [85] se demostró que el operador de cruce basado en dos puntos obtenía mejores resultados que operadores de cruce basados en más puntos. Hemos encontrado en la bibliografía otro trabajo, Booker [27], donde se compara empíricamente el operador de cruce en un punto frente al operador de cruce en dos puntos concluyendo el autor, que es precisamente este último el que mejores resultados obtiene.

Otro operador de cruce que ha recibido mucha atención en los últimos años es el operador de cruce uniforme (Syswerda [149]). En este caso el cruce es realizado a través de una máscara. Un esquema de aplicación del cruce uniforme puede verse en la figura 2.6. La máscara de cruce está indicando el cromosoma del que se tomará cada gen para

formar los hijos. De esta forma, si en la máscara aparece un 0 tomaré el gen correspondiente del primer individuo y si aparece un 1 realizaré lo contrario.

0	1	0	1	1	1	0	1	1	0	1	1	0	Máscara de cruce
1	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	Padre 1
0	1	0	1	0	1	0	0	0	0	1	1	0	Padre 2
1	1	**1**	1	0	1	**0**	0	0	**0**	1	1	**0**	Hijo

Figura 2.6: Ejemplo de cruce uniforme

Una manera de generalizar este operador de cruce es no generando la máscara de forma aleatoria, sino que esta dependa del valor de la función objetivo del par de individuos a cruzar. De esta forma se posibilitaría que el individuo hijo tuviera más genes provenientes del padre mejor adaptado Larrañaga y col. [97].

Codificación de las soluciones

La codificación de las soluciones es otra cuestión importante en los AAGG. En el AG simple cada individuo está codificado como un vector 0-1, sin embargo, es fácil pensar en otro tipo de codificaciones.

Las formas más habituales de codificar una solución a la hora de utilizar los AAGG son las siguientes:

- vectores donde cada gen toma un valor entre un rango finito dado,
- vectores formados por permutaciones de un conjunto de números, y
- vectores formados por genes que toman valores reales.

Cada uno de este tipo de codificaciones posee sus operadores específicos ya que, en la mayoría de los casos, los operadores expuestos en el apartado anterior, no sirven para este tipo de codificaciones.

La discusión en cuanto al tipo de codificación que es más adecuada a un problema dado, ha traído durante mucho tiempo de cabeza a los investigadores y no hay conclusiones categóricas en este sentido.

Probabilidades

Dentro de los parámetros que utiliza el AG simple se encuentran las probabilidades de cruce p_c y de mutación p_m.

El trato dado por los investigadores a cada uno de estos parámetros ha sido muy diferente. Mientras que la probabilidad de cruce no ha recibido ninguna atención por parte de los mismos, y casi siempre suele ser 1 en aplicaciones prácticas, se ha dedicado gran esfuerzo, tanto desde un punto de vista práctico como teórico a establecer el valor óptimo de la probabilidad de mutación. Este hecho puede estar motivado por algunos trabajos como el de Shaffer y col. [142], donde se afirma que, establecer un valor óptimo para la probabilidad de mutación es mucho más importante para la búsqueda que hacerlo para la probabilidad de cruce.

De Jong [85] en su tesis doctoral recomienda una probabilidad de mutación que sea inversamente proporcional al tamaño del individuo, es decir, $1/l^{-1}$. Shaffer y col. [142] por su parte proponen un valor, $1/n^{0.9318}l^{0.4535}$, que depende de la longitud del individuo y del tamaño de la población. Otros resultados en este sentido, algunos de ellos más recientes se pueden encontrar en, Hesser y Manner [76], Suzuki [147], Bäck [13] y Rudolph [138].

Otros trabajos relacionados con la probabilidad de mutación se basan en modificar esta a medida que se lleva a cabo la búsqueda: Ackley [1], Davis [43], Fogarty [56], Revees [135], Michalewicz y Janikov [116]. Los autores de estos trabajos se basan en que la probabilidad de mutación debería tener en cuenta la diversidad de la población. De esta forma la probabilidad sería más grande si la diversidad en la población fuera pequeña y viceversa.

Elitismo

En el AG simple, la nueva población compuesta por los hijos, sustituía por completo a la población anterior. Esto no se suele realizar de esta

forma en la práctica. Habitualmente se utiliza lo que se ha venido a llamar *elitismo*.

El primer autor en proponer el elitismo fué De Jong [85] en su tesis doctoral. Para el autor un AG era elitista si el mejor individuo de la población en el tiempo k era introducido en la población en el tiempo $k+1$.

Por otro lado Eiben y col. [52] introdujeron otro tipo de elitismo mediante operadores de reducción. Los operadores de reducción se basan en utilizar tanto la población de los hijos como la de los padres para obtener la siguiente población de los AAGG. Un tipo de operador de reducción muy común es el operador de reducción *elitista de grado 1*. Este operador consiste en introducir en el tiempo $k+1$ el mejor individuo del conjunto formado por la población en el tiempo k y los hijos. El operador puede claramente, ser generalizado Larrañaga y col. [94], para tener algoritmos elitistas de grado n (siendo n el tamaño de la población).

Otra cuestión no comentada hasta el momento es el hecho de que en cada iteración del AG simple se generan n individuos. Este hecho ha sido modificado por algunos investigadores, que en vez de generar en cada paso n individuos, generan menos, de tal forma que la población en el siguiente paso del algoritmo está formada por algunos de los individuos que estaban en ella en el tiempo k y algunos de los hijos generados.

2.4 El AG Aplicado en CNS

El AG concreto utilizado para llevar a cabo la tarea de la CNS es el algoritmo llamado GENITOR. Este algoritmo fué desarrollado por Whitley [152], Whitley y Kauth [153] y Whitley y Starkweather [154].

En la figura 2.7 se puede ver un pseudo-código para el algoritmo en el caso de un problema de maximización.

Este algoritmo se diferencia del AG simple principalmente en dos características:

- la selección está basada en el rango, y

Inicializar probabilidad de cruce p_c
Inicializar probabilidad de mutación p_m
Hallar la población inicial P_0
Hasta condición_parada = TRUE **hacer**
  begin
    Seleccionar, basándose en el rango, dos individuos de la población P_k
    Cruzar los dos individuos, obteniendo I
    Mutar el individuo I con probabilidad p_m
    si $F(I) > F(I^k_{peor})$ **entonces**
      P_{k+1}=$(P_k \cup \{I\})\backslash\{I^k_{peor}\}$
    sino
      $P_{k+1} = P_k$
  end
Devolver la mejor solución

Figura 2.7: Pseudo-código para el algoritmo GENITOR

- en cada iteración del algoritmo únicamente se genera un individuo.

La segunda de estas diferencias es quizás las más importante. El algoritmo en vez de evolucionar población a población, como lo hace el AG simple, evoluciona individuo a individuo. Este tipo de algoritmos se denominan "steady-state" en inglés. En cada paso se eligen, al igual que en el AG simple, dos individuos para el cruce (esta vez con selección basada en el rango), sin embargo, se genera un único individuo hijo. Este individuo hijo generado se introduce en la población si su valor objetivo es mejor que el peor valor de los individuos de la población. En el caso de introducir dicho individuo, la nueva población se forma eliminando el peor individuo de la población. En caso contrario la nueva población es igual a la anterior.

La otra diferencia es la forma de llevar a cabo la selección. El tipo de selección utilizada por el algoritmo es una generalización de la selección basada en el rango. Las probabilidades de selección dependen de los

rangos de los valores que la función objetivo toma en los individuos y de un parámetro β (habitualmente $\beta \in (0, 2]$) que es introducido por el usuario y que representa el número de veces que es más probable elegir el mejor individuo de la población en relación al peor. En la figura 2.8 se puede ver un ejemplo de la asignación de estas probabilidades. El mejor individuo de la población es el I_1 mientras que el peor es el I_n. La probabilidad de elegir el mejor es $\beta = 2$ veces mayor que la probabilidad de elegir el peor. Los demás individuos tienen asociada una probabilidad que viene dada por la recta r.

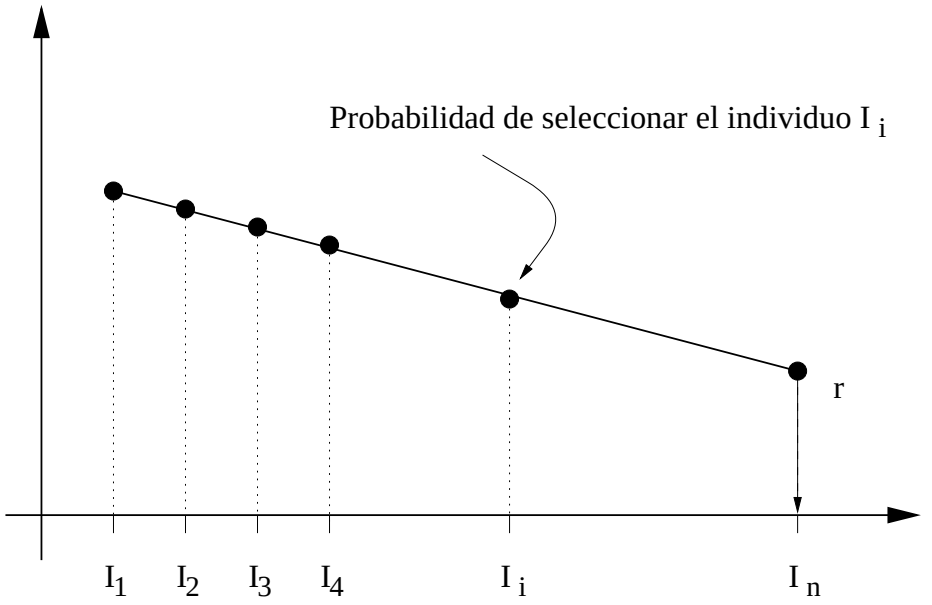

Figura 2.8: Esquema de asignación de probabilidades de la selección en el algoritmo GENITOR

La elección del algoritmo GENITOR en vez de cualquier otro tipo de AG, ha estado motivada principalmente por dos cuestiones. La primera es que, como hemos visto anteriormente, no existen dentro de la comunidad, criterios claros acerca de cuales son los parámetros óptimos o el tipo de algoritmo más adecuado. Claramente esto depende del problema. Por lo tanto nos decidimos a utilizar GENITOR ya que la mayoría de los artículos de investigación dedicados a estudiar

algoritmos poblacionales (que evolucionan población a población) frente a los algoritmos que evolucionan individuo a individuo apuntan que los resultados de estos últimos son mejores que el de los primeros.

La segunda razón es la existencia de un software para este algoritmo que proporciona las rutinas básicas necesarias en cualquier AAGG, como por ejemplo: rutinas de selección, rutinas con algunos operadores de cruce, introducción y extracción de individuos en/de la población, generación de la población inicial, etc. Esto conlleva que el programador esté únicamente concentrado en diseñar de forma conveniente la rutina que implementa la función objetivo.

Dado que el tipo de individuos utilizados en la consecución de la CNS es en todos los casos el mismo: permutaciones de números, estudiaremos a continuación el tipo de operadores que se han venido utilizando en este tipo de cromosomas.

2.5 Operadores de Cruce para individuos de tipo Permutación

El tipo de operadores diseñados para cromosomas formados por permutaciones de un conjunto de números proviene principalmente de la aplicación de los AAGG al *problema del agente viajero.*

El problema del agente viajero consiste en lo siguiente: un viajante tiene que visitar n ciudades y volver a la ciudad inicial, se trata de hallar el recorrido que, pasando por todas las ciudades una sola vez, minimiza la distancia recorrida por el viajante. Este problema es un problema NP-completo, es decir, no se conoce ningún algoritmo que sea capaz de resolver el problema, de manera exacta, en un tiempo de computación polinomial en el número de ciudades.

Cualquier solución al problema, es decir, cualquier ruta a través de las n ciudades puede expresarse como una permutación π:

$$(\pi_1 \ \pi_2 \ \ldots \pi_n)$$

donde cada término π_i representa la ciudad que se visita en el lugar i-ésimo.

Es posible representar las soluciones para este problema de forma diferente. Un resumen bastante completo de la aplicación de los AAGG

al problema del agente viajero puede encontrarse en Larrañaga y col. [95].

Es evidente que los operadores clásicos de los AAGG, es decir, los operadores estudiados en la sección 2.3 no pueden ser aplicados directamente a este tipo de individuos, ya que podrían conducirnos a recorridos no válidos: se repiten ciudades, no se pasa por todas las ciudades, etc. Esto conlleva la creación de operadores específicos para este tipo de cromosomas.

A continuación expondremos los operadores que han sido utilizados, en los AAGG diseñados en esta memoria para efectuar la CNS. Es posible, y somos conscientes de ello, encontrar en la bibliografía más operadores, sin embargo, nosotros nos hemos restringido a estos por dos razones. En primer lugar el objetivo principal de la memoria era comprobar de que manera se podían aplicar los AAGG en CNS y que beneficios, si es que existían, se podían sacar de dicha aplicación. Es decir, nuestro objetivo no era afinar hasta el punto de decir que operador es el más adecuado para llevar a cabo la tarea de la CNS. Esto no es óbice que hayamos efectuado test de hipótesis dentro de los operadores empleados para evaluar cual de ellos obtiene los mejores resultados en menor tiempo.

En segundo lugar, haber realizado experimentos con todos los operadores existentes para este tipo de cromosomas, hubiera representado tal gasto computacional que, probablemente este trabajo hubiese necesitado mucho más tiempo para completarse.

2.5.1 Operadores de Cruce para Permutaciones

Para ilustrar los operadores de cruce de manera más conveniente supondremos en todos los casos que el cruce es llevado a cabo entre los siguientes dos individuos:

Padre 1: (2 4 6 7 1 8 5 3 9)
Padre 2: (1 5 9 3 4 7 8 2 6).

Operador de cruce basado en una correspondencia parcial (PMX)

El operador de cruce PMX fué introducido por Goldberg y Lingle [68]. Este operador tomados dos individuos padres, devuelve dos hijos. Se basa en seleccionar dos subrutas que se intercambian entre los padres, el resto de la información se intercambia a continuación.

Para llevar a cabo este cruce, dados dos individuos, la selección del trozo de ruta que se intercambia se realiza estableciendo dos puntos de corte. Los puntos de corte son elegidos aleatoriamente dentro de los posibles puntos. En el ejemplo anterior supongamos que estos dos puntos han sido el tercero y el séptimo:

Padre 1: (2 4 6 |7 1 8 5 |3 9)
Padre 2: (1 5 9 |3 4 7 8 |2 6).

Para crear los hijos se intercambian inicialmente los trozos de subrutas, de manera que la subruta del primer padre pasa al segundo descendiente y al revés:

Hijo 1: (∗ ∗ ∗ |3 4 7 8 | ∗ ∗)
Hijo 2: (∗ ∗ ∗ |7 1 8 5 | ∗ ∗).

Para terminar de generar las rutas se rellena cada descendiente con las ciudades restantes del padre correspondiente. Si existe una ciudad que ya se encuentra en el hijo, dicha ciudad se sustituye a partir de una correspondencia que se establece entre las subrutas elegidas. En este caso la correspondencia es: 7 y 3, 1 y 4, 8 y 7, 5 y 8. De esta forma vamos a rellenar el primer hijo. Comenzamos poniendo la primera ciudad del padre, la 2, la siguiente sería la 4, pero, esta ciudad ya está puesta luego, buscando en la lista de correspondencias, terminamos poniendo la ciudad que se corresponde con la 4 esto es, la 1, si dicho número hubiese estado también puesto, seguiríamos buscando dentro de la lista de correspondencias. Así proseguiríamos hasta el final. Los descendientes que se obtienen son:

Hijo 1: (2 1 6 |3 4 7 8 |5 9)
Hijo 2: (4 3 9 |7 1 8 5 |2 6).

Operador de cruce basado en ciclos (CX)

Este operador diseñado por Oliver y col. [125], a partir de dos individuos padres crea un sólo hijo. Se trata de rellenar las posiciones del cromosoma hijo, de forma alternativa, con el valor que en dicha posición tiene uno de los padres teniendo en cuenta algunos ciclos.

Considerando los padres del ejemplo:

$$\text{Padre 1: } (2\ 4\ 6\ 7\ 1\ 8\ 5\ 3\ 9)$$
$$\text{Padre 2: } (1\ 5\ 9\ 3\ 4\ 7\ 8\ 2\ 6)$$

comenzaríamos rellenando la primera posición del cromosoma hijo eligiendo aleatoriamente uno de los dos padres para copiar su posición, supongamos que elegimos el segundo padre, tenemos entonces:

$$\text{Hijo : } (1\ *\ *\ *\ *\ *\ *\ *\ *)$$

una vez elegido el 1 en el segundo padre, esto significa que el uno no puede volver a ser elegido, luego en la quinta posición necesariamente se elegirá el 4:

$$\text{Hijo : } (1\ *\ *\ *\ 4\ *\ *\ *\ *).$$

Esto significa que en la segunda posición estamos obligados a elegir la ciudad del segundo individuo padre, es decir, la 5:

$$\text{Hijo : } (1\ 5\ *\ *\ 4\ *\ *\ *\ *).$$

Podríamos continuar rellenando el individuo hijo de esta forma y obtendríamos:

$$\text{Hijo : } (1\ 5\ *\ 3\ 4\ 7\ 8\ 2\ *).$$

Una vez completado un ciclo, la primera posición no rellena hasta el momento sería completada nuevamente eligiendo al azar entre ambos padres. Suponiendo que elijo en este caso el segundo, se obtiene como hijo:

$$\text{Hijo : } (1\ 5\ 6\ 3\ 4\ 7\ 8\ 2\ 9).$$

Operador de cruce basado en el orden (OX1)

Davis [42] propuso el siguiente operador de cruce. Los nuevos individuos son construidos utilizando una subruta de uno de los padres y el orden de los individuos del otro padre.

Si consideramos que los individuos padres, una vez seleccionadas las subrutas, son:

Padre 1: (2 4 6 |7 1 8 |5 3 9)
Padre 2: (1 5 9 |3 4 7 |8 2 6).

Copiamos para empezar las subrutas en los individuos hijos:

Hijo 1: (∗ ∗ ∗ |7 1 8 | ∗ ∗ ∗)
Hijo 2: (∗ ∗ ∗ |3 4 7 | ∗ ∗ ∗).

Para completar los individuos hijos comenzamos eligiendo el primero, por ejemplo. Para completar el individuo se comienza tras el segundo punto de corte. Los ciudades son puestas, en el orden en que estas están en el segundo padre, a partir de, también, el segundo punto de corte. La primera ciudad que está, a partir del segundo punto de corte en el padre 2 es la 8, sin embargo, ésta ya está puesta en el Hijo 1, por lo tanto vamos a la siguiente ciudad, esta es la 2. Ésta si que se sitúa, obteniendo:

Hijo 1: (∗ ∗ ∗ |7 1 8 |2 ∗ ∗).

La siguiente ciudad es la 6, poniéndola en el octavo lugar del Hijo 1. El orden en el Padre 1 se continua comenzando por el primer gen. Finalmente, obtengo los individuos:

Hijo 1: (5 9 3 |7 1 8 |2 6 5)
Hijo 2: (6 1 8 |3 4 7 |5 9 2).

Operador de cruce basado en la alternancia de posiciones (AP)

El operador de cruce basado en la alternancia de posiciones aparece por primera vez en los trabajos de Larrañaga y col. [96].

El operador AP tomados dos individuos padre devuelve dos hijos. La forma de conseguir cada uno de los hijos consiste en rellenar las

posiciones de los mismos de manera alternativa, eligiendo primero la primera ciudad de un padre, luego la primera del otro padre, luego la segunda del primer padre, la segunda del segundo padre, y así sucesivamente, pero sin que se repitan.

Teniendo en cuenta nuevamente los individuos del ejemplo:

Padre 1: (2 4 6 7 1 8 5 3 9)
Padre 2: (1 5 9 3 4 7 8 2 6)

si comenzamos eligiendo primero del primer padre obtendríamos el hijo:

Hijo 1: (2 1 4 5 6 9 7 3 8).

y eligiendo la primera ciudad del segundo individuo se obtendría:

Hijo 2: (1 2 5 4 9 6 3 7 8).

2.5.2 Operadores de Mutación para Permutaciones

Operador de mutación basado en el desplazamiento (DM)

Este operador debido a Michalewicz [115], se basa en elegir una subruta dentro del individuo la cual se inserta de forma aleatoria en otro lugar del individuo.

Si consideramos el siguiente individuo:

(1 2 5 4 9 6 3 7 8)

para elegir la subruta se eligen dos puntos, dentro del individuo de manera aleatoria, supongamos que son el segundo y el quinto:

(1 2 |5 4 9 |6 3 7 8).

Dicha subruta se introduce en otro lugar del individuo. Este lugar se determina eligiendo de forma aleatoria una posición dentro de las ciudades restantes. Si la posición elegida es la cuarta:

(1 2 6 3 |7 8)

se obtiene el individuo:

(1 2 6 3 5 4 9 7 8).

Operador de mutación basado en cambios (EM)

El operador EM (Banzhaf [18]) es el más obvio de todos. Consiste en seleccionar al azar dos ciudades de la permutación e intercambiarlas.

Considerando nuevamente el individuo:

$$(1\ 2\ 5\ 4\ 9\ 6\ 3\ 7\ 8)$$

si elegimos al azar la cuarta y la sexta posición para modificar, tras la mutación se obtiene el individuo:

$$(1\ 2\ 5\ 6\ 9\ 4\ 3\ 7\ 8).$$

Operador de mutación basado en la inserción (ISM)

El operador ISM fué propuesto independientemente por Fogel [57] y Michalewicz [115]. Este operador de mutación se basa en escoger aleatoriamente una ciudad de la permutación para a continuación insertarla en otro lugar elegido al azar.

Nuevamente si partimos de:

$$(1\ 2\ 5\ 4\ 9\ 6\ 3\ 7\ 8)$$

y elegimos al azar la ciudad 9, para más tarde elegir el cuarto lugar para situarla, obtenemos el individuo:

$$(1\ 2\ 5\ 9\ 4\ 6\ 3\ 7\ 8).$$

Operador de mutación basado en la inversion simple (SIM)

El operador SIM (Holland [77] y Grenfesttete y col. [72]) realiza una selección de dos lugares en la permutación para luego invertir el orden de las ciudades que se encuentran entre los dos lugares elegidos.

Si el individuo y los puntos elegidos son:

$$(1\ 2\ 5\ |4\ 9\ 6\ 3\ |7\ 8)$$

el individuo que se obtiene es:

$$(1\ 2\ 5\ |3\ 6\ 9\ 4\ |7\ 8).$$

Operador de mutación basado en la inversión (IVM)

Fogel [58] es el creador de este operador. Este operador es una mezcla del operador DM y del operador anterior. Se elige una subruta y luego esta subruta se inserta en un lugar elegido aleatoriamente, pero en orden contrario.

Si el individuo es:

$$(1\ 2\ 5\ |4\ 9\ 6\ 3\ |7\ 8)$$

y la subruta elegida es la que viene acotada por | entonces eligiendo el primer lugar para insertar la subruta obtendríamos el individuo:

$$(1\ 3\ 6\ 9\ 4\ 2\ 5\ 7\ 8).$$

Operador de mutación basado en el cambio (SM)

El operador SM fué introducido por Syswerda [149]. Este operador se basa en seleccionar una subruta dentro de cada individuo y cambiar de forma aleatoria el orden de las ciudades dentro de la subruta.

Si el individuo y la subruta elegida son:

$$(1\ 2\ 5\ |4\ 9\ 6\ 3\ |7\ 8),$$

entonces un posible individuo resultante después de la aplicación del operador de mutación es:

$$(1\ 2\ 5\ |9\ 6\ 3\ 4\ |7\ 8).$$

Capítulo 3

Modelado Matemático de los Algoritmos Genéticos

3.1 Introducción

En este capítulo trataremos de introducir el modelado matemático de los AAGG. También se expondrá un nuevo algoritmo de optimización basado en los AAGG y en el algoritmo de enfriamiento estadístico (EE). Se demostrará además la convergencia hacia el óptimo del mismo.

El estudio teórico de los AAGG comienza en el trabajo de Holland [77]. En él se exponía lo que ha sido durante mucho tiempo el resultado fundamental para justificar el funcionamiento de los AAGG, aunque dicho teorema únicamente se aplicaba al AG simple. Este resultado se resumía en lo que se ha venido a llamar el *Teorema de los Esquemas*. La idea fundamental de este teorema es considerar que lo que hacen los AAGG en cada paso es muestrear hiperplanos del espacio de búsqueda. El teorema de los esquemas trata en este caso, de aproximar, conocido el número de individuos que muestrean un hiperplano en la población en el tiempo k, el número de individuos esperados que muestrearán dicho hiperplano en la población en el tiempo $k+1$. Conocida esta fórmula es posible ver que si el conjunto de individuos que muestreaban un hiperplano en particular, en la población en el tiempo k, eran tales que el valor medio que tomaba la función objetivo en ellos estaba por encima del valor medio de la población, entonces el número de in-

dividuos esperados en la población en el tiempo $k+1$ muestreando este hiperplano aumentaba exponencialmente. De esta manera siempre se mejoraba la búsqueda. El resultado expuesto por Holland ha sido afinado de forma considerable, mejorando la cota (el teorema únicamente da una cota inferior para el número de individuos esperados), Whitley [155], y generalizándolo a poblaciones donde los individuos no tenían porque ser necesariamente vectores 0-1 (Antonisse [9]). A pesar de ello, este resultado ha recibido gran cantidad de críticas (consultar Peck y Dhawan [126] por ejemplo), principalmente de investigadores que no piensan que el teorema de los esquemas, pueda utilizarse para justificar el hecho, de la gran evidencia empírica del buen funcionamiento de los AAGG. En cualquier caso este resultado en ningún momento habla sobre convergencia, velocidad de convergencia, o probabilidad de alcanzar el óptimo de una función, número de generaciones esperadas antes de alcanzar el óptimo etc.; cuestiones todas ellas interesantes a la hora de aplicar los AAGG en un problema práctico de optimización.

Los primeros trabajos rigurosos de sistematización, modelado y análisis de AAGG son debidos principalmente a Eiben y col. [52], Nix y Vose [124], Davis y Príncipe [45], Suzuki [147] y Rudolph [138].

En Eiben y col. [52] se plantea por primera vez de forma seria el problema de la convergencia de los AAGG. Hasta dicho momento los trabajos sobre convergencia (Ankenbrand [8] y Chakraborty y Dastidar [37]) se habían basado principalmente en el concepto de uniformidad de la población. La cuestión era conocer en que generación se daba el hecho de que la mayoría de los miembros de la población eran casi todos iguales. Eiben y col. [52] modelan los AAGG utilizando cadenas de Markov. El resultado al que llegan es que para garantizar que la distribución estacionaria de la cadena de Markov que modela el AG asigne probabilidad distinta de cero únicamente a las poblaciones que contienen individuos que son el óptimo global del problema, es suficiente que el algoritmo sea elitista de grado 1, esto es que guarde al menos el mejor individuo de la población conjunta de padres e hijos, de generación en generación. A pesar de lo importante del resultado, el modelo propuesto por Eiben y col. [52] es un modelo tan general que no tiene la capacidad de analizar en detalle los componentes del AG, no se detalla el valor de los elementos de la matriz de transición de estados. Dicho modelo fué generalizado más tarde por Larrañaga y

col. [94], donde se enmarcó al algoritmo de EE como un caso particular de un AG abstracto.

Nix y Vose [124] y en particular Vose con trabajos que veremos más adelante, son los que más han contribuido al estudio teórico de los AAGG. Los autores modelan el AG simple, al igual que Eiben y col. [52] utilizando cadenas de Markov. Sin embargo, a diferencia del estudio de Eiben y col. [52], en su modelo se ve de que manera contribuye cada operador a dicha cadena. De esta forma los autores dan una expresión cerrada para los elementos de la cadena de Markov que modela el AG simple. Esta expresión cerrada tiene la ventaja de que es posible ver en élla la influencia de cada componente del AG. En esta expresión están reflejados tanto la probabilidad de cruce como la de mutación, además del tipo de operador de cruce o mutación que se esté utilizando. Su trabajo, no obstante, no sólo está basado en investigar la convergencia del AG, sino que sus objetivos son más científicos, en el sentido en que los autores están preocupados por conocer el comportamiento del algoritmo a medida que este evoluciona en el tiempo. El artículo investiga cuales son las trayectorias que sigue el AG en su ejecución, dentro del espacio de las poblaciones.

Davis y Príncipe [45] desarrollan, al igual que los anteriores autores, un modelo matemático de los AAGG mediante cadenas de Markov. Sin embargo, su análisis fué motivado por el hecho de investigar si el "annealing" de la probabilidad de mutación podía conducir al AG hacia el óptimo. Aunque la respuesta fué negativa, el trabajo realizado ha contribuido de forma importante al conocimiento de los AAGG, y particularmente a la demostración de convergencia del algoritmo propuesto por el autor de esta memoria.

Suzuki [147] utilizando en parte el modelo dado por Nix y Vose [124] y Davis y Príncipe [45], comenzó demostrando la convergencia del algoritmo elitista en el sentido de De Jong [85], es decir, cuando el mejor individuo de la población en el tiempo k se introduce en la población en el tiempo $k+1$. El autor trató además de hallar, desde un punto de vista más práctico, la probabilidad de que cierta generación del AG contuviese al individuo óptimo del problema en el algoritmo elitista anterior. En el trabajo se demostró que para mejorar esta probabilidad el valor de la probabilidad de mutación había que establecerlo en función de un autovalor de una submatriz de la matriz de transición de estados

de la cadena de Markov.

Por su parte Rudolph [138] no utilizó el mismo modelo que los anteriores autores. El autor demostró que si en un AG simple nos vamos quedando con el mejor individuo obtenido hasta el momento entonces el AG converge, lo cual parece un resultado lógico siempre que la probabilidad de alcanzar dicho individuo no tienda a 0 a medida que se ejecuta el algoritmo, hecho que ocurre en el AG simple. Algo similar se podría decir de un algoritmo de búsqueda aleatoria.

En la siguiente sección de este capítulo se describirá el modelado de los AAGG mediante cadenas de Markov. Recogeremos también un breve resumen con resultados referentes a los mismos que pueden ser obtenidos a partir de este modelo. La siguiente sección estará dedicada a una breve exposición del algoritmo de EE, la cual es necesaria para entender el trabajo posterior. El algoritmo propuesto en esta memoria junto con el análisis de su convergencia se expondrá en la cuarta sección, terminando el capítulo con una sección dedicada a las conclusiones.

3.2 Modelado del AG simple mediante cadenas de Markov

El modelo que se expone a continuación sigue principalmente las ideas expuestas en Nix y Vose [124] y Vose [150]. La clave es modelar el AG simple mediante una cadena de Markov, donde los diferentes estados de la cadena de Markov son las posibles diferentes poblaciones por las que puede pasar el AG. Introduciremos antes de nada la notación.

Suponemos que queremos hallar el máximo (el problema de minimización se trataría de manera similar) de una función:

$$F : \Omega \longrightarrow I\!R^{+}.$$

Supondremos que la función es positiva simplemente por simplificar el aparato matemático. Esto no impone ninguna restricción en el desarrollo posterior. El espacio de búsqueda es $\Omega = \{0,1\}^{l}$, es decir, el conjunto de vectores 0-1 de longitud l. Denotaremos el cardinal de este conjunto por $|\Omega| = N$. Una población del algoritmo va a ser un subconjunto (siempre que hablemos de aquí en adelante de conjuntos nos

estaremos refiriendo en realidad a multiconjuntos ya que en las poblaciones del AG pueden existir individuos que se repitan) de tamaño n del conjunto Ω. Cada población puede ser representada mediante un vector $Z_s = (z_{0s}, z_{1s}, \ldots, z_{N-1s})$ donde z_{is} representa el número de individuos de tipo i que hay en la población Z_s, por supuesto se cumple que:

$$\sum_{i=0}^{N-1} z_{is} = n.$$

Para verlo con un ejemplo supongamos que el tamaño de los individuos es $l = 3$. El espacio de búsqueda será por tanto:

$$\Omega = \{(0\ 0\ 0), (0\ 0\ 1), (0\ 1\ 0), (0\ 1\ 1), (1\ 0\ 0), (1\ 0\ 1), (1\ 1\ 0), (1\ 1\ 1)\}$$

y si el tamaño de la población es $n = 4$, entonces la siguiente población:

$$(0\ 0\ 1)\ \ (1\ 0\ 0)\ \ (0\ 0\ 1)\ \ (1\ 0\ 1)$$

se representaría por medio del siguiente vector:

$$Z_s = (0, 2, 0, 0, 1, 1, 0, 0).$$

El número de posibles poblaciones viene dado por tanto, por el número de combinaciones con repetición de n elementos de un total de N, dicho número es:

$$r = \begin{pmatrix} n + N - 1 \\ N - 1 \end{pmatrix}.$$

Por lo tanto el conjunto de poblaciones podemos representarlo mediante una matriz de tamaño $r \times N$:

$$\begin{pmatrix} z_{01} & z_{11} & \ldots & z_{N-11} \\ z_{02} & z_{12} & \ldots & z_{N-12} \\ \vdots & \vdots & \vdots & \vdots \\ z_{0s} & z_{1s} & \ldots & z_{N-1s} \\ \vdots & \vdots & \vdots & \vdots \\ z_{0r} & z_{1r} & \ldots & z_{N-1r} \end{pmatrix}.$$

La razón para modelar el AG mediante una cadena de Markov es que la población en el tiempo k sólo depende de la población en el tiempo

$k-1$, lo que claramente nos conduce hacia una cadena de Markov donde cada estado de la cadena de Markov va a ser una posible población del AG.

Antes de continuar con el modelo, es esencial establecer un orden entre las diferentes poblaciones. Para establecer un orden en las poblaciones estableceremos antes, un orden entre los elementos de Ω. El orden en los elementos de Ω viene impuesto por la función objetivo F. De esta forma, si un individuo i tiene un valor de F superior a otro individuo j entonces el individuo i se encuentra antes que el j. En caso de empate en la función objetivo, se puede utilizar cualquier criterio que sirva para desempatar, por ejemplo, su orden canónico como números binarios. Si ahora realizamos una permutación de los elementos de los vectores que representan las poblaciones Z_s, en concordancia con el orden dado en los individuos, entonces estos pueden ser considerados como números $n+1$-arios, lo cual impone un orden implícito en los mismos.

Si en el ejemplo anterior tenemos que los siguientes valores de función objetivo:

$$\begin{array}{lll} F((0\ 0\ 0)) = 15 & F((0\ 0\ 1)) = 13 & F((0\ 1\ 0)) = 7 \\ F((0\ 1\ 1)) = 7 & F((1\ 0\ 0)) = 15 & F((1\ 0\ 1)) = 8 \\ F((1\ 1\ 0)) = 12 & F((1\ 1\ 1)) = 9. & \end{array}$$

entonces los individuos de Ω escritos en orden quedan como sigue:

$$(1\ 0\ 0)\ (0\ 0\ 0)\ (0\ 0\ 1)\ (1\ 1\ 0)\ (1\ 1\ 1)\ (1\ 0\ 1)\ (0\ 1\ 1)\ (0\ 1\ 0).$$

Bajo este orden las primeras poblaciones son aquellas que contienen los mejores individuos y las últimas los peores. En particular la primera población estará formada por n copias de un individuo óptimo.

A continuación trataremos de hallar los elementos de la matriz de dicha cadena de Markov. Estamos buscando los elementos de la matriz $Q = [q_{st}]_{s,t=1,2,\ldots,r}$, donde q_{st} representa la probabilidad de estando en la población Z_s obtener mediante una iteración del AG simple la población Z_t. Evidentemente esta probabilidad va a depender de varios factores: las poblaciones Z_s y Z_t, la probabilidad de mutación p_m, la probabilidad de cruce p_c, y el valor que la función objetivo toma en los elementos de la población Z_s.

Si denotamos por $p_s(i)$ la probabilidad de obtener el individuo i a partir de la población Z_s entonces claramente se puede decir que la probabilidad que estamos buscando sigue una distribución multinomial con parámetros $p_s(0), p_s(1), \ldots, p_s(i), \ldots, p_s(N-1)$, es decir, el número que estamos buscando se puede escribir como:

$$q_{st} = \frac{n!}{z_{0t}! z_{1t}! \ldots z_{N-1t}!} \prod_{i=0}^{N-1} p_s(i)^{z_{it}}.$$

Por lo tanto únicamente resta hallar las probabilidades $p_s(i)$. Estas probabilidades vienen expuestas en detalle en el trabajo Vose y Liepins [151]. La probabilidad de hallar un individuo i a partir de una población Z_s se puede expresar como:

$$p_s(i) = \sum_{j=0}^{N-1} \sum_{k=0}^{N-1} p^s_{sel}(j) p^s_{sel}(k) p_{jk}(i)$$

donde $p^s_{sel}(j)$ (resp. $p^s_{sel}(k)$) es la probabilidad de obtener el individuo j (resp. k) a partir de la población Z_s utilizando la selección, y $p_{jk}(i)$ es la probabilidad de obtener el individuo i mediante cruce y mutación a partir de los individuos j y k.

Si suponemos que la selección es la del AG simple, explicada en el capítulo anterior, la probabilidad $p^s_{sel}(j)$ viene expresada como sigue:

$$p^s_{sel}(j) = \frac{z_{js} F(j)}{\sum_{i=0}^{N-1} z_{is} F(i)}$$

obteniéndose $p^s_{sel}(k)$ de manera similar.

Para hallar la probabilidad de obtener el individuo i a partir de los individuos j y k, vamos a hallar primero la probabilidad de obtener el individuo 0 (aquel en el que todos sus genes toman el valor 0) a partir de j y k y luego veremos que la probabilidad que buscamos puede ser derivada a partir de esta. La probabilidad $p_{jk}(0)$ está deducida en Vose y Liepins [151] y se expresa como:

$$p_{jk}(0) = \frac{(1-p_m)^l}{2}\left\{\eta^{|j|}\left(1-p_c+\frac{p_c}{l-1}\sum_{w=1}^{l-1}\eta^{-\Delta_{jkw}}\right)+\right.$$

$$\left.+\eta^{|k|}\left(1-p_c+\frac{p_c}{l-1}\sum_{w=1}^{l-1}\eta^{-\Delta_{jkw}}\right)\right\}$$

donde $\eta = p_m/(1-p_m)$, y $|j|$ es el número de unos que posee el individuo j. Por último:

$$\Delta_{jkw} = |(2^w-1)\otimes j| - |(2^w-1)\otimes k|$$

siendo $\otimes$ la multiplicación término a término. El operador $\otimes$ podríamos también considerarlo como el operador lógico *and*.

Finalmente si definimos $\oplus$ como la suma módulo 2 entonces se puede ver que:

$$p_{jk}(i) = p_{jk}(i\oplus 0) = p_{j\oplus ik\oplus i}(0).$$

De esta forma quedan hallados los elementos de la matriz Q. A partir de ellos se pueden realizar varias deducciones. Claramente todos los elementos son distintos de 0, por lo tanto la cadena de Markov que modela el AG simple es una cadena irreducible y aperiódica (Feller [53]). De aquí se deduce la existencia de una distribución estacionaria. Esta distribución estacionaria es tal que asigna a todo estado de la cadena de Markov, es decir, a toda posible población del AG, una probabilidad distinta de 0. Se puede decir por lo tanto, que la cadena de Markov visita cada estado infinitamente a menudo. Desde este punto de vista el AG no es un algoritmo convergente, entendiendo por convergencia, que cuando el número de iteraciones tiende a infinito, la cadena de Markov asocia probabilidad distinta de 0 únicamente, a aquellas poblaciones que contienen individuos óptimos. La definición de convergencia utilizada por Rudolph [138] no me parece válida en este contexto ya que también podría ser utilizada para decir que una búsqueda aleatoria que asigne una probabilidad distinta de 0 a cada población sería convergente.

Evidentemente, a pesar de que el AG simple no es convergente, sería muy interesante conocer la distribución de probabilidad estacionaria

de la cadena de Markov. Conocer dicha distribución de probabilidad nos llevaría a valorar de otra manera el AG simple, ya que se podría saber cual es la masa de probabilidad asignada a las poblaciones que contienen algún individuo óptimo, cual es la asignada a las no óptimas y su relación. Al mismo tiempo se podría estudiar como depende esta distribución estacionaria de las probabilidades p_c, p_m y los valores de la función objetivo F.

Se han realizado algunos intentos, De Jong y col. [86], de hallar dicha distribución estacionaria. Sin embargo, esto es muy costoso de para para $l > 3$, debido a la explosión combinatoria del número de poblaciones.

Para alcanzar la convergencia en el sentido expuesto anteriormente es, en principio, suficiente que algoritmo sea elitista (en cualquiera de ambos sentidos). Es decir, es suficiente tener un algoritmo que pase el mejor individuo de la población en el tiempo t a la población en el tiempo $t+1$ o que el mejor de la población conjunta de padres e hijos pase a formar parte de la siguiente población. Más tarde veremos que una versión probabilista del elitismo nos conducirá también a unos resultados de convergencia. En este caso, si denotamos por $Q^e = [q^e_{st}]_{s,t=0,1,\ldots,N-1}$ la matriz de transición de estados de la cadena de Markov que modela el AG elitista (en el sentido de De Jong), entonces dichos elementos pueden expresarse de la siguiente forma:

$$q^e_{st} = \begin{cases} 0 & \text{si } \exists\, i \in Z_s \text{ tal que } F(i) > F(j)\ \forall\ j \in Z_t \\ q_{st} & \text{en otro caso} \end{cases}$$

En el algoritmo elitista se puede ver que la matriz Q^e puede considerarse como una matriz triangular inferior por cajas. Cada caja se corresponde con aquellas poblaciones que tienen como mejor individuo al mismo. Así la primera caja está compuesta por todas las poblaciones que contienen algún individuo óptimo. Una representación para esta

matriz es la siguiente:

$$Q^e = \begin{pmatrix} \boxed{Q^e(1)} & & & \\ & \boxed{Q^e(2)} & & 0 \\ & & \ddots & \\ & & & \boxed{Q^e(a)} \end{pmatrix}$$

siendo a el número de diferentes valores que los elementos de Ω toman en la función F.

Claramente esta matriz no es irreducible. Es más, se puede ver que todos los estados, excepto los que se corresponden con la matriz cuadrada $Q^e(1)$ son estados transitorios. Los únicos estados recurrentes son los que están en $Q^e(1)$ que a la vez son estados absorbentes. Esto significa que la distribución de probabilidad estacionaria asigna probabilidad distinta de 0 únicamente a los estados que contienen el individuo óptimo.

3.3 Algoritmo de Enfriamiento Estadístico

En esta sección se describirá el algoritmo de EE. La inclusión de este algoritmo en este capítulo está motivada, por el hecho de que el algoritmo que se expondrá en la siguiente sección está basado en buena parte en ideas y resultados que han sido dados para el EE.

El nombre del algoritmo se ha tomado traduciendo uno de los artículos donde se describe (Storer y col. [146]). Aunque su nombre en inglés es "simulated annealing", su traducción directa al castellano, utilizada, por ejemplo en Díaz [47], "recocido simulado" no nos parece muy acertada.

El algoritmo de EE, (Kirpatrick y col. [91], Van Laarhoven y Aarts [92]) tiene su origen en los trabajos de Metrópolis y col. [114]. El EE es un algoritmo cuyo objetivo es la optimización. Aunque suele ser interpretado de dos formas diferentes, su origen, sin embargo se encuentra en la simulación del proceso de "annealing" de un sólido. Este

proceso consiste en calentar un sólido hasta altas temperaturas, para más tarde dejar enfriarlo lentamente de manera que éste alcance estados de energía estables, es decir, estados donde los electrones alcancen los niveles más bajos de energía. Metropolis y col. [114], idearon un procedimiento que intentaba simular este proceso, por medio un algoritmo probabilista basado en la distribución de Boltzmann. Kirpatrick y col. [91] observaron la similitud que existía entre el "annealing" de un sólido y la resolución de un problema de optimización combinatoria.

La otra forma de interpretar el algoritmo, es considerarlo como una modificación de los algoritmos de búsqueda local. Los algoritmos de búsqueda local se basan en elegir en cada paso una solución dentro del entorno de la solución actual, de manera que se mejore el valor de la primera. Si tenemos un problema combinatorio, es decir, se quiere optimizar una función:

$$F : D \longrightarrow I\!R$$

donde D es un conjunto finito, entonces un sistema de vecinos o entornos para este problema viene dado por:

$$\mathcal{N} : D \longrightarrow \mathcal{P}(D)$$

siendo $\mathcal{P}(D)$ el conjunto de las partes de D. En la figura 3.1 se puede ver un posible código para un algoritmo de búsqueda local en un problema de minimización.

```
Elegir una solución inicial x_0
hacer
   Buscar una solución x_l en N(x_i)
   si F(x_l) < F(x_i) entonces
      x_{i+1} = x_l
   sino
      x_{i+1} = x_i
hasta que tras cierto número de evaluaciones no se
   haya mejorado el valor de la función objetivo
```

Figura 3.1: Pseudo-código de un optimizador local

Este tipo de algoritmos tiene principalmente tres desventajas:

- son algoritmos que terminan en un óptimo local y no hay manera de saber cuan alejado está este óptimo local del óptimo global

- el óptimo local en el que termina el algoritmo depende totalmente de la solución inicial que se tome, y, en general, no existe ninguna regla que nos de información sobre como elegir esta solución inicial

- en un problema cualquiera no hay forma de conocer el tiempo de computación de estos algoritmos.

El algoritmo de EE nace como una alternativa que intenta solucionar todos estos problemas. Para ello, utiliza una variación de los métodos de búsqueda local. Esta variación consiste en aceptar movimientos que empeoran el valor de la función objetivo. De esta manera lo que se está propiciando es la posibilidad de salir de óptimos locales. Esta aceptación de movimientos es llevada a cabo con cierta probabilidad, ya que en caso contrario nos encontraríamos ante un algoritmo de búsqueda aleatoria. A medida que el algoritmo se va ejecutando la probabilidad de aceptación de un movimiento que empeore decrece, de manera que ésta es cercana a 0 cuando el algoritmo termina.

Un posible pseudo-código para el EE básico para un problema de minimización puede consultarser en 3.2.

Como se puede ver el algoritmo está controlado por varios parámetros, siendo el más importante de todos es el c_k. Dicho parámetro, al que llamaremos temperatura por analogía con el proceso de "annealing", es el que controla el grado de aceptación de soluciones que empeoran el valor de la función objetivo. Si el parámetro c_k tiene un valor muy alto se aceptarán casi todas las soluciones aunque empeoren mucho el valor de la función objetivo. Si por el contrario dicho parámetro tiene un valor muy pequeño entonces el algoritmo estará muy cercano a un algoritmo de búsqueda local, ya que apenas si se aceptarán transiciones a soluciones peores.

Al igual que los AAGG, este algoritmo puede ser modelado mediante cadenas de Markov. La solución en el paso k del algoritmo, únicamente depende de la solución en el paso $k-1$ y no de las anteriores. Se han construido dos modelos matemáticos basados en cadenas de Markov para demostrar la convergencia.

```
Seleccionar una solución inicial x_0 de D
Seleccionar una temperatura inicial c_0
Seleccionar un valor de modificación de la temperatura α
Seleccionar un tamaño de cadena tam_max
Inicializar iteraciones num_iter=0
repetir
  tam_cadena=0
  repetir
    Elegir de forma aleatoria una solución x_l en N(x_k)
    si F(x_l) < F(x_k) entonces
      x_{k+1} = x_l
    sino
    elegir un número aleatorio aleat en [0,1]
      si e^{-(F(x_l)-F(x_k))/c_k} > aleat
        x_{k+1} = x_l
      sino
        x_{k+1} = x_k
    tam_cadena=tam_cadena+1
  hasta tam_cadena > tam_max
  c_{k+1} = α * c_k
  num_iter=num_iter+1
hasta num_iter > iter_max
```

Figura 3.2: Pseudo-código para un EE

El primero de ellos se basa en utilizar cadenas de Markov homogéneas. Para cada c_k se tiene en cuenta una cadena de Markov y se demuestra bajo ciertas condiciones, que cuando la temperatura tiende a 0 de manera "suficientemente" lenta, entonces es posible ver que la sucesión de distribuciones estacionarias de las cadenas de Markov, para cada c_k, convergen hacia una distribución de probabilidad que sólo asigna probabilidad distinta de 0 a las soluciones óptimas. Es más esta distribución es uniforme sobre el conjunto de las soluciones óptimas. Este modelo, que tiene la ventaja de ser fácil de entender, no aporta nada desde un punto de vista práctico ya que supone que para

cadena de Markov homogénea es necesario llegar hasta la distribución estacionaria. Es importante tener en mente el hecho de que fijado un valor de c_k la cadena de Markov tendrá como distribución estacionaria la distribución de Boltzmann sobre el conjunto de soluciones.

El segundo modelo se basa en cadenas de Markov inhomogéneas. Estas son cadenas de Markov en las que la matriz de transición de estados puede modificar su valor a cada paso, de hecho en nuestro caso depende del parámetro temperatura c_k.

Veamos primero como son los elementos de la matriz de la cadena de Markov que modela el EE. Si denotamos por $P_{ij}(c_k)$ la probabilidad de estando en la solución i en el paso $k-1$ pasar, tras una iteración del algoritmo a la solución j, entonces para una generalización del algoritmo de EE dado en la figura 3.2, los elementos pueden expresarse mediante la siguiente ecuación:

$$P_{ij}(c_k) = \begin{cases} G_{ij}(c_k)A_{ij}(c_k) & \forall\ j \neq i \\ 1 - \sum_{l=1, l\neq i}^{|D|} G_{ij}(c_k)A_{il}(c_k) & j = i \end{cases}$$

donde $G_{ij}(c_k)$ representa la probabilidad de, estando en la solución i elegir como solución candidata la j. Habitualmente esta probabilidad es independiente de c_k y suele estar uniformemente distribuida sobre el conjunto de vecinos de la solución i como es el caso del algoritmo de la figura 3.2. $A_{ij}(c_k)$ es la probabilidad de siendo la solución actual la i aceptar como nueva solución la solución candidata j. En nuestro caso esta probabilidad viene expresada como:

$$A_{ij}(c_k) = \begin{cases} 1 & \text{si } F(j) < F(i) \\ e^{-(F(j)-F(i))/c_k} & \text{en otro caso.} \end{cases}$$

Para demostrar que una cadena de Markov inhomogénea es convergente, hay que demostrar que es débilmente y fuertemente ergódica.

En Van Laarhoven y Aarts [92] se puede ver que las condiciones que debe cumplir la cadena de Markov inhomogénea para ser convergente son:

i) $\forall\ i, j \in D,\ \ \exists\ p \geq 1$ y $\exists\ l_0, l_1, \ldots, l_p \in D\ \ (l_0 = i$ y $l_p = j)$ tal que $G_{l_k l_{k+1}}(c_k) > 0\ \ \forall\ k = 0, 1, \ldots, p-1$

ii) $\forall\ c_k > 0\ \ \exists\ i, j \in D$ tal que $A_{ij}(c_k) < 1$ y $G_{ij}(c_k) > 0$

iii) $\forall\ k\ c_k \leq c_{k+1}$ y $\lim_{k\to\infty} c_k = 0$

iv) $\forall\ k\ \ c_k \geq \frac{\Gamma}{\log k}$.

Obsérvese que las dos primeras condiciones son condiciones que se imponen en las matrices G y A y que son condiciones relativamente lógicas y fáciles de cumplir. Si A y G son como hemos comentado anteriormente se cumplen trivialmente. La condición **iii)** dice simplemente que la secuencia de temperaturas $\{c_k\}_{k=0,1,...}$ debe ser no decreciente y además su límite debe ser 0. La idea es que en el límite el algoritmo haya alcanzado el óptimo global y que apenas si se acepten soluciones que empeoren el valor de la función objetivo. Por último la última condición da una cota para el decrecimiento del valor de c_k. El valor Γ es una constante que depende del problema. En cualquier caso este es el mayor handicap del algoritmo de EE. Esta cota en el decrecimiento hace que el algoritmo sea muy lento si se quiere tener la seguridad de alcanzar el óptimo global.

3.4 Relajando el Elitismo

3.4.1 Introducción

Expondremos en esta sección la extensión de los AAGG con un operador de reducción basado en la distribución de Boltzmann y al mismo tiempo demostraremos su convergencia hacia el óptimo. Este algoritmo puede verse de dos formas diferentes, como un híbrido entre los AAGG y el EE o como un AG en el que se ha introducido un modelo relajado del elitismo. El parámetro "temperatura", c_k, permite controlar los elementos de la matriz de transición de la cadena de Markov que modela el algoritmo. En el límite, es decir cuando $c_k \to 0$, el operador de reducción se convierte en un tipo de elitismo "fuerte". Además la convergencia hacia el óptimo será demostrada bajo condiciones muy débiles en la secuencia $\{c_k\}_{k=0,1,...}$. Este trabajo está recogido de Lozano y col. [102].

3.4.2 Trabajo Previo

El primer trabajo en el que se mencionaron los operadores de reducción fué el efectuado por Eiben y col. [52], y ya comentado anteriormente. Un operador de reducción es un operador que dada la población en el tiempo k y sus descendientes se obtiene, a partir de ambos conjuntos la población en el tiempo $k+1$. Como se dijo anteriormente la condición suficiente de convergencia enunciada por Eiben y col. [52] fue el elitismo. Sin embargo, el elitismo rompe en algún sentido con la idea de los AAGG como algoritmos "estocásticos" de optimización. Como veremos la condición suficiente de convergencia impuesta por Eiben y col. [52] puede ser relajada mediante la introducción de un operador de reducción probabilista basado en la distribución de Boltzmann.

Por otro lado, como hemos visto en la sección anterior el algoritmo de EE posee unas propiedades de convergencia muy satisfactorias las cuales sería muy interesante exportar a los AAGG. Estas dos ideas son las que han motivado el desarrollo del algoritmo.

Es posible encontrar en la literatura gran cantidad de trabajos (Davis y Príncipe [45], Bilbro y col. [26], Mahfoud y Goldberg [113], Suzuki [148], Rudolph [139], Lin y col. [100]) en los que se intentan crear híbridos entre los AAGG y el EE y de esta manera llevar las propiedades de convergencia del EE al campo de los AAGG. Comentaremos los trabajos que están más cercanos al presentado en la memoria.

Davis y Prıncipe [45] intentaron utilizar la probabilidad de mutación p_m como el parámetro c_k del EE. Los autores probaron que cuando la probabilidad de mutación tiende a 0 la cadena de Markov converge hacia una distribución de probabilidad estacionaria, en la cual, sólo los estados (es decir poblaciones) cuyos individuos eran todos iguales (poblaciones uniformes) tenían asignada probabilidad distinta de 0. Claramente este resultado está lejos de la convergencia hacia el óptimo. En el artículo se realizaron varios experimentos, donde se observaba que si la población era suficientemente grande, los estados uniformes cuyo individuo era el óptimo tenían asignados mucha más probabilidad que los demás.

Suzuki [148] utilizando la aproximación y los resultados dados por Davis y Príncipe diseñó otro híbrido entre AAGG y EE. Su aproxi-

mación no sólo está basada en llevar la probabilidad de mutación a 0 sino también la de cruce y una relación basada en la selección. El autor demostró que su algoritmo converge hacia una población uniforme formada por el individuo óptimo, al igual que nosotros haremos más adelante. Aunque este trabajo tiene bastante similaridades con el nuestro, se diferencia en aspectos básicos. Dentro de las similaridades está el hecho de que el autor utiliza un operador de selección probabilista que en el caso límite se comporta como un operador elitista al igual que nuestro operador de reducción. La principal diferencia es que para alcanzar la convergencia Suzuki necesita mantener la misma cota en la secuencia de temperaturas que Davis y Príncipe [45], (los autores necesitaban para garantizar la convergencia una cota parecida a la que garantiza la convergencia en el EE) mientras que nuestra aproximación no necesita de ninguna cota.

Bilbro y col. [26] desarrollan un algoritmo llamado "genetic-annealing algorithm" el cual es básicamente un EE pero, en vez de mantener en cada paso del algoritmo una única solución, mantienen un conjunto de soluciones. Bilbro y col. [26] demostraron que su algoritmo conservaba las mismas propiedades de convergencia que el EE. Sin embargo, este algoritmo tiene realmente poco que ver con el AG.

En Mahfoud y Goldberg [113] se puede encontrar un buen resumen de artículos dedicados a exponer algoritmos híbridos. Además los autores proponen un nuevo algoritmo llamado "Parallel Recombinative Simulated Annealing" (PRSA). El algoritmo PRSA puede ser interpretado como un AG en el que para elegir los individuos que pasan a la siguiente población se lleva a cabo una competición entre los padres y los hijos utilizando una función de tipo Boltzmann. Los autores dieron una demostración de convergencia para una variante de su algoritmo. Las principales diferencias con nuestra aproximación son las siguientes: la probabilidad de selección en su algoritmo es uniforme, su algoritmo puede verse como un "steady-state" en vez de un algoritmo generacional, su operador de reducción es local y finalmente nuestros resultados de convergencia son más generales que los dados en este trabajo.

A parte de los trabajos dedicados a construir híbridos entre AAGG y EE, la función de Boltzmann ha sido utilizada de forma extensa en los AAGG en el operador de selección. Goldberg [66] y Mahfoud [112]

han estudiado en profundidad esta utilización dando lugar al operador de "Selección por Torneo de Boltzmann" (STB), ya comentado en el capítulo anterior. Como claramente es imposible demostrar la convergencia hacia el óptimo con la mera utilización del STB, su objetivo consistía en demostrar que el operador STB conservaba la diversidad en la población. Esto significa que el algoritmo cae menos frecuentemente en poblaciones uniformes formadas por individuos subóptimos. El parámetro clave para conservar esta diversidad es, por supuesto, la temperatura c_k.

3.4.3 El Algoritmo

En esta sección daremos el algoritmo. Nuestro híbrido entre AAGG y EE se basa en dar una forma concreta al operador de reducción probabilista: un operador de reducción Boltzmann. Pensamos que el algoritmo que proponemos a continuación es un buen híbrido entre los AAGG y el EE ya que preservamos los conceptos más importantes que hay detrás de cada uno de estos algoritmos. Por un lado se mantiene el uso de poblaciones que evolucionan en el tiempo para llevar a cabo la búsqueda como lo hace el AAGG, y por otro mantenemos el control de la convergencia mediante el uso de un parámetro c_k al igual que en el EE.

Además demostraremos que para cualquier sucesión $\{c_k\}_{k=0,1,...}$ el algoritmo converge hacia el óptimo cuando $c_k \to 0$, sin la imposición de ninguna cota que restrinja la velocidad de convergencia.

Veremos que el algoritmo puede ser modelado mediante una cadena de Markov inhomogénea y que esta cadena converge hacia una distribución de probabilidad estacionaria que asigna probabilidad distinta de 0 tan sólo a las poblaciónes que posean únicamente individuos óptimos.

Más aún, el algoritmo al ser controlado por el parámetro c_k nos da la posibilidad de controlar la diversidad en la población al igual que se hacia con el operador STB, y en algún sentido también el grado de elitismo. El operador de reducción Boltzmann puede ser interpretado de dos formas diferentes. Como una versión relajada de un elitismo fuerte o como un modelo probabilista para un tipo de elitismo fuerte.

El algoritmo que se propone está compuesto por las operaciones

básicas del AG simple. A este algoritmo le hemos añadido, el operador de reducción. La forma concreta de este operador es lo que nos diferencia de, por ejemplo, el trabajo de Eiben y col. [52]. Un pseudo-código para el algoritmo puede verse en la figura 3.3.

```
Inicializar_Población(P_0)
Inicializar_Temperatura(c_0)
k = 0
mientras no stop hacer
  begin
  hacer n/2 veces
    begin
    Seleccionar dos padres de P_k
    Generar dos hijos utilizando un operador de cruce
    Mutar los dos hijos
    Introducir los hijos en la población de hijos CH_k
    end
  Crear la población extendida P'_k = P_k ∪ CH_k
  Utilizando el operador de reducción reducir al población
    extendida P'_k a tamaño original obteniéndose P_{k+1}
  Modificar_Temperatura (c_{k+1})
  k := k + 1
  end
```

Figura 3.3: Pseudo-código para el AG con el operador de reducción

Para modelar el algoritmo supondremos que la selección es la basada en la ruleta sesgada, que el cruce es el basado en un punto y que la mutación es la del AG simple. A pesar de lo anterior los resultados de convergencia no se pierden si en vez de estos operadores se utilizasen otros, como operadores de cruce en múltiples puntos o operador de cruce uniforme.

El operador de reducción consiste en muestrear una distribución de probabilidad de Boltzmann en la población extendida $P_k^{'}$ (la unión de la población de los padres con la de los hijos). La masa de probabilidad que esta distribución asigna a cada individuo depende del valor que la

función objetivo toma en el propio individuo y de la población extendida. De esta manera a cada individuo i (padre o hijo) se le asignará una probabilidad igual a:

$$\frac{1}{R(c_k)} \exp\left(\frac{F(i)-\overline{F}}{c_k} \right) \tag{3.1}$$

de ser elegido para formar parte de la siguiente población, donde $R(c_k)$ es la constante de normalización y $\overline{F}$ es el valor medio de la función objetivo F en la población de los padres. Por lo tanto para muestrear la distribución de Boltzmann el operador de reducción hace lo siguiente (este pseudo-código es expansión de la línea de código de la figura 3.3: "Utilizando el operador de reducción reducir la población extendida $P_k^{'}$ a tamaño original obteniéndose P_{k+1}"):

```
mientras no se hayan seleccionado n individuos hacer
  begin
    elegir aleatoriamente un individuo i de la población extendida
    si F(i) > F̄ entonces
      seleccionar i para la siguiente población
    sino
      seleccionar el individuo i con probabilidad igual a la ec. 3.1
  end
```

Figura 3.4: Pseudo-código para el operador de reducción

Por tanto, se elige un individuo i de la población extendida aleatoriamente, si el individuo es tal que $F(i) > \overline{F}$, es decir el valor que toma F en dicho individuo es mayor que el valor medio de F en la población de los padres. Dicho individuo pasa directamente a formar parte de la siguiente generación. En caso contrario el individuo pasa a la siguiente población con una probabilidad dada por la ecuación 3.1. El ciclo se repite hasta que se seleccionan n individuos.

En el caso límite, cuando $c_k \longrightarrow 0$, el operador de reducción probabilista se comporta como un operador elitista fuerte en el sentido de que, únicamente pasan a la siguiente población individuos cuyo valor de función objetivo está por encima del valor medio que la función objetivo toma en la población de los padres.

3.4.4 Modelado

En este apartado veremos como se modela el algoritmo mediante cadenas de Markov. Antes de nada hay que darse cuenta de que el algoritmo, una vez más, puede ser modelado por una cadena de Markov. Nuevamente la población en el tiempo k únicamente depende de la población en el tiempo $k-1$. Nuestro primer objetivo por lo tanto es hallar la matriz de transición de estados de la cadena de Markov. Denotaremos esta matriz por $Q(c_k) = (q_{st}(c_k))_{s,t=1,\ldots,r}$. Es importante darse cuenta que la matriz de transición depende del parámetro c_k, es decir, en general existe una matriz diferente para cada iteración del algoritmo y por lo tanto la cadena de Markov, es una *cadena de Markov inhomogénea*. Abusando del lenguaje utilizaremos la misma notación para la matriz de la cadena de Markov dada para el AG simple en la segunda sección que para la matriz de la cadena de Markov de nuestro algoritmo. También utilizaremos más tarde los elementos de la matriz de la cadena de Markov del AG simple denotándolos en este caso como p_{st}.

Supondremos de aquí en adelante que la función F es inyectiva. Esta suposición está hecha simplemente de cara a simplificar la notación y a que los resultados sean más claros de interpretar. Los mismos resultados de convergencia son también válidos para funciones no inyectivas.

El elemento (s,t) de la cadena de Markov $Q(c_k)$ que representa la probabilidad de estando en el paso k en la población Z_s ir mediante la aplicación del AG a la población Z_t en el paso $k+1$, se puede expresar como:

$$q_{st}(c_k) = \sum_{v \in C_s^t} p_{sv} PR_{s \cup vt}(c_k)$$

donde

- p_{sv} es la probabilidad de ir de la población Z_s a la población Z_v mediante la aplicación de la selección, el cruce y la mutación (es decir, el AG simple). Dicha probabilidad fué hallada en la segunda sección del capítulo. Es importante darse cuenta de que esta probabilidad es independiente del tiempo.

- $C_s^t = \{Z_v \mid \forall i \in Z_t \Rightarrow i \in Z_s \cup Z_v\}$ donde $\cup$ es considerado como unión e intersección en multiconjuntos. Es decir, C_s^t está

representando el conjunto de todas las posibles poblaciones Z_v que pueden ser generadas a partir de Z_s y tal que la aplicación del operador de reducción a $Z_s \cup Z_v$ pueda obtener Z_t.

- $PR_{s\cup v,t}(c_k)$ es la probabilidad de obtener, mediante la aplicación del operador de reducción, la población Z_t a partir de la población extendida $Z_s \cup Z_v$.

El nuevo término en la expresión anterior, esto es, la probabilidad de reducción PR, sigue una distribución multinomial donde los parámetros vienen dados por la ecuación 3.1. Estas probabilidades pueden expresarse como sigue:

$$PR_{s\cup vt}(c_k) = \frac{n!}{z_{0t}!z_{1t}!\ldots z_{N-1t}!}\prod_{i=0}^{N-1}\left(\frac{z_{is\cup v}}{R_s^v(c_k)}\exp\left(\frac{F(i)-\overline{F}^s}{c_k}\right)\right)^{z_{it}} =$$

$$= \frac{n!}{z_{0t}!z_{1t}!\ldots z_{N-1t}!}\frac{\prod_{i=0}^{N-1}(z_{is\cup v})^{z_{it}}}{(R_s^v(c_k))^n}\exp\left(\frac{(\sum_{i=0}^{N-1} z_{it}F(i))-n\overline{F}^s}{c_k}\right)$$

donde la expresión $\overline{F}^s$ es el valor medio que la función objetivo F toma en los individuos de Z_s, y $R_s^v(c_k)$ es la constante de normalización que puede ser expresada como:

$$R_s^v(c_k) = \sum_{i=0}^{N-1} z_{is\cup v}\exp\left(\frac{F(i)-\overline{F}^s}{c_k}\right)$$

representando $z_{is\cup v}$ el número de individuos de tipo i en la población extendida $Z_s \cup Z_v$ que posee $2n$ elementos.

La estructura de la matriz está controlada por el parámetro c_k. En el caso límite, es decir, cuando $c_k \to 0$, la matriz $Q(c_k)$ llega a ser una matriz triangular inferior por cajas, siendo la mejor población Z_0 el único estado absorbente de la cadena de Markov. El caso límite es muy parecido al elitismo determinista en el que pasarían a la siguiente población los n mejores individuos entre los padres y los hijos. En este sentido se dijo al principio que el operador de reducción de Boltzmann podía verse como un modelo probabilista de un elitismo fuerte.

3.4.5 Convergencia

Se examinará a continuación la convergencia del algoritmo. Comenzaremos introduciendo la notación que se utilizará más adelante.

Si denotamos por $q(c_k)$ la distribución de probabilidad de las diferentes poblaciones tras k pasos del algoritmo, es decir:

$$q(c_k) = q(c_0)Q(c_0)Q(c_1)\dots Q(c_{k-1})$$

y si $Q(c_{m,k})$, para $m < k$, está representando la siguiente matriz estocástica:

$$Q(c_{m,k}) = Q(c_m)Q(c_{m+1})\dots Q(c_{k-1})$$

entonces demostraremos en este apartado que existe una distribución q que asigna probabilidad 1 a la población uniforme conteniendo el óptimo global (la llamaremos población óptima), y tal que para todo $m \in I\!N$:

$$\lim_{k\to\infty} \sup_{q(c_0)} ||q(c_0)Q(c_{m,k}) - q|| = 0.$$

Es decir, la sucesión converge en norma hacia la distribución q, independientemente de la distribución inicial $q(c_0)$ y de los pasos iniciales de la cadena. Esto, en términos del algoritmo, significa que éste converge hacia la población óptima. Los resultados serán dados para la norma L_1.

Las únicas condiciones que hay que imponer en la sucesión $\{c_k\}_{k=0,1,\dots}$ es que cumpla las siguientes propiedades:

- $\exists\ k^*$ tal que $\forall\ k > k^*\ \ c_{k+1} \leq c_k$
- $\lim_{k\to\infty} c_k = 0$

Para probar la convergencia hacia la población óptima anterior, es necesario, como vimos en la sección dedicada al EE que la cadena de Markov sea débilmente y fuertemente ergódica. Una vez haya sido demostrada la ergodicidad débil y la ergodicidad fuerte habremos asegurado que la cadena de Markov converge hacia una distribución de probabilidad. Además esta distribución de probabilidad coincide con la distribución límite de la sucesión de distribuciones de probabilidad $\{\pi(c_k)\}_{k=0,1,\dots}$. Siendo cada

$$\pi(c_k) = (\pi_1(c_k), \pi_2(c_k), \dots, \pi_s(c_k), \dots, \pi_r(c_k))$$

la distribución estacionaria de la cadena de Markov homogénea $Q(c_k)$ con c_k fijo. Por lo tanto, únicamente quedará por demostrar que dicha sucesión converge hacia una distribución de probabilidad que asigna probabilidad 1 a la población óptima.

Definiremos primero las nociones de ergodicidad débil y fuerte.

Definición 1 *Una cadena de Markov $Q(c_k)$ es débilmente ergódica si para todo m se cumple:*

$$\lim_{k\to\infty} \sup_{q(c_0),p(c_0)} ||q(c_0)Q(c_{m,k}) - p(c_0)Q(c_{m,k})|| = 0.$$

En términos un poco más intuitivos, el que una cadena de Markov sea débilmente ergódica significa que el efecto de la distribución inicial se pierde con el tiempo, es decir, que el resultado final es independiente de la distribución inicial.

Definición 2 *Una cadena de Markov $Q(c_k)$ es fuertemente ergódica si existe un vector $q = (q_1, q_2, \ldots, q_r)$ con $||q|| = 1$ y $q_i \geq 0$ para todo $i = 1, 2, \ldots, r$ tal que para todo m se cumple:*

$$\lim_{k\to\infty} \sup_{q(c_0)} ||q(c_0)Q(c_{m,k}) - q|| = 0.$$

Una cadena fuertemente ergódica es una cadena en la que la sucesión de distribuciones de probabilidad $\{q(c_k)\}_{k=0,1,\ldots}$ converge en norma hacia una distribución q.

La ergodicidad débil y la fuerte serán demostradas utilizando dos teoremas clásicos que aparecen en Isaacson and Madsen [79], pag. 151 y 160. Daremos a continuación el enunciado de estos teoremas, adaptados a la notación que estamos utilizando. Pero antes necesitamos definir el coeficiente ergódico de una matriz.

Definición 3 *Dada una matriz estocástica $Q = [q_{st}]_{s,t=1,2,\ldots,r}$ se define su coeficiente ergódico $\alpha(Q)$ como el número:*

$$\alpha(Q) = \min_{(s,v)} \sum_{t=1}^{r} \min(q_{st}, q_{vt}).$$

Teorema 1 *Una cadena de Markov inhomogénea es débilmente ergódica si existe una sucesión de enteros $k_1, k_2, \ldots, k_l, \ldots$ tal que:*

$$\sum_{l=1}^{\infty} \alpha(Q(c_{k_l,k_{l+1}})) = \infty$$

Teorema 2 *Una cadena de Markov es fuertemente ergódica si es débilmente ergódica y además para todo k existe un vector $\pi(c_k)$ que es un autovector por la izquierda de $Q(c_k)$ y cumple:*

- $\sum_{s=1}^{r} \pi_s(c_k) = 1$
- $\sum_{k=0}^{\infty} \sum_{s=1}^{r} \mid \pi_s(c_k) - \pi_s(c_{k+1}) \mid < \infty.$

Mas aún, si $\pi = \lim_{k\to\infty} \pi(c_k)$ entonces π satisface:

$$\lim_{k\to\infty} q_{st}(c_{m,k}) = \pi_t$$

donde $q_{st}(c_{m,k})$ el elemento (s,t) de la matriz $Q(c_{m,k})$.

Ergodicidad Débil

Para probar la ergodicidad débil vamos a analizar la primera columna de las matrices $Q(c_k)$, es decir, las probabilidades de estando en cualquier población ir a la población óptima (la denotaremos por **1**). Demostraremos que estos números son todos mayores que 0 para cualquier valor de c_k y que además están acotados por un número mayor que cero. Esto es suficiente para implicar que la suma de la serie $\sum_{k=0}^{\infty}(\alpha(Q(c_k))$ es ∞ (como sucesión de números k_l del teorema tomaremos la sucesión $0, 1, 2, \ldots$).

Los elementos de la primera columna de las matrices $Q(C_k)$ tienen la forma:

$$q_{s\mathbf{1}}(c_k) = \sum_{t\in C_s^{\mathbf{1}}} p_{st} PR_{s\cup t\mathbf{1}}(c_k) = \tag{3.2}$$

$$= \sum_{t\in C_s^{\mathbf{1}}} p_{st} \frac{n!(z_{0s\cup t})^{z_{0\mathbf{1}}}(z_{1s\cup t})^{z_{1\mathbf{1}}} \ldots (z_{N-1s\cup t})^{z_{N-1\mathbf{1}}}}{z_{0\mathbf{1}}! z_{1\mathbf{1}}! \ldots z_{N-1\mathbf{1}}!} \frac{\exp\left(\frac{n(F^* - \overline{F}^s)}{c_k}\right)}{(R_s^t(c_k))^n}$$

donde

$$R_s^t(c_k) = \sum_{i=0}^{N-1} z_{is\cup t} \exp\left(\frac{F(i)-\overline{F}^s}{c_k} \right).$$

Evidentemente debe existir un único $i^* \in \{0, 1, \ldots, N-1\}$ tal que $F(i^*) = F^*$, es decir, el individuo i^* es el individuo óptimo. Por lo tanto si sólo tenemos en cuenta los términos que dependen de c_k tendremos que:

$$\lim_{c_k \to 0} \frac{\exp(\frac{n(F^*-\overline{F}^s)}{c_k})}{(R_s^t(c_k))^n} = \frac{1}{(z_{i^* s\cup t})^n}$$

y finalmente obtenemos que:

$$\lim_{c_k \to 0} q_{s\mathbf{1}}(c_k) = \sum_{t \in C_s^{\mathbf{1}}} p_{st}.$$

Tenemos entonces que los números $q_{s\mathbf{1}}(c_k)$ son todos diferentes de 0 (basta comprobar la definición de los mismos) para cualquier población Z_s, y además por lo anterior su límite es distinto de 0. Se puede entonces establecer que:

$$\sum_{k=0}^{\infty} (\alpha(Q(c_k)) = \sum_{k=0}^{\infty} \min_{s,v} \sum_{t=1}^{r} \min(q_{st}(c_k), q_{vt}(c_k)) \geq \sum_{k=1}^{\infty} \delta = \infty$$

donde $\delta = \min_{k,s}(q_{s\mathbf{1}}(c_k))$. Esto completa la demostración.

Es importante darse cuenta del hecho de que para probar la ergodicidad débil no hemos tenido que imponer ninguna condición en la sucesión de valores del parámetro temperatura $\{c_k\}_{k=0,1,2,\ldots}$. Lo anterior implica que el algoritmo será extremadamente robusto en el sentido de casi cualquier sucesión $\{c_k\}_{k=0,1,\ldots}$ dará buenas aproximaciones. Por otro lado esto contrasta con la cota, $c_k \geq \frac{\Gamma}{\log k}$, que es necesario imponer en el EE o en los trabajos posteriores de Davis y Príncipe [45] y Suzuki [148] para garantizar la convergencia.

El hecho de que no sea necesario imponer ningún tipo de cota en la sucesión $\{c_k\}_{k=0,1,ldots}$ hace posible que este parámetro sea ajustado a nuestro antojo, garantizándose la convergencia siempre que exista un k^* tal que para todo $k > k^*$ la sucesión $\{c_k\}_{k=0,1,\ldots}$ sea no creciente y tienda a 0.

Ergodicidad Fuerte

Para demostrar la ergodicidad fuerte utilizando el teorema 2, antes de nada es necesario dar una expresión explícita para las distribuciones estacionarias de las cadenas de Markov para cada valor c_k (o dicho de otra forma para los autovectores por la izquierda con autovalor 1 de las matrices $Q(c_k)$). Estas distribuciones existen porque, si fijamos el valor de c_k, la cadena de Markov es homogénea y como todos los elementos son distintos de cero, siguiendo lo expuesto en la sección 1, la cadena es irreducible y aperiódica.

Anily y Ferdergruen [7] dieron la siguiente expresión para las distribuciones estacionarias:

$$\pi_s(c_k) = \frac{\mid Q_s(c_k) - I \mid}{\sum_{t=1}^{r} \mid Q_t(c_k) - I \mid} \tag{3.3}$$

donde I es la matriz identidad, la expresión $\mid\ \mid$ significa el determinante y $Q_s(c_k)$ es la matriz $Q(c_k)$ pero con la fila s cambiada por ceros.

Utilizando el apartado (e) del teorema de Perron-Frobenious (ver apéndice), se puede ver que dado que $Q_s(c_k) \leq Q(c_k)$ entonces 1 no puede ser un autovalor de $Q_s(c_k)$, por lo tanto todos los determinantes anteriores son diferentes de 0 y además tienen el mismo signo.

Por otro lado teniendo en cuenta que los elementos de las matrices de las cadenas de Markov $q_{st}(c_k)$ son funciones racionales compuestas por funciones exponenciales en el numerador y en el denominador, entonces estas expresiones son continuas y su primera derivada también es continua en c_k, para todo valor de c_k distinto de 0, considerando este como un parámetro continuo. Aplicando entonces el teorema del valor medio se obtiene:

$$\pi_s(c_{k+1}) - \pi_s(c_k) = \left.\frac{d\pi_s(c)}{dc}\right|_{c=c_*^k} (c_{k+1} - c_k)$$

donde c_*^k es un número real en el intervalo (c_{k+1}, c_k).

Entonces la expresión que queremos acotar puede escribirse como sigue:

$$\sum_{k=0}^{\infty}\sum_{s=1}^{r} \mid \pi_s(c_{k+1}) - \pi_s(c_k) \mid = \sum_{k=0}^{\infty}\sum_{s=1}^{r} \left| \left.\frac{d\pi_s(c)}{dc}\right|_{c=c_*^k} \right| |c_{k+1} - c_k|$$

y es posible construir un nueva función continua Θ extendiendo $\frac{d\pi}{dc}$ por continuidad en el 0:

$$\Theta(c) = \begin{cases} \frac{d\pi}{dc} & \text{si } c \neq 0 \\ \lim_{c\to 0} \frac{d\pi}{dc} & \text{si } c = 0 \end{cases}$$

que será utilizada para acotar la serie anterior. La función Θ puede ser acotada en $[0, 1]$ ya que es continua en este intervalo. Si llamamos Δ a su máximo, entonces:

$$\mid \pi_s(c_{k+1}) - \pi_s(c_k) \mid \leq \Delta |c_{k+1} - c_k|$$

y por tanto

$$\sum_{k=0}^{\infty} \sum_{s=1}^{r} \mid \pi_s(c_{k+1}) - \pi_s(c_k) \mid \leq \sum_{s=1}^{r} \Delta c_0 \leq r \Delta c_o < \infty$$

con lo que queda demostrada la ergodicidad fuerte.

Es importante darse cuenta que en la demostración anterior la sucesión $\{\pi(c_k)\}_{k=0,1,\ldots}$ de distribuciones estacionarias de las cadenas de Markov homogéneas con c_k fijo, son sólo una herramienta para demostrar la convergencia de la cadena inhomogénea. Es decir, no es necesario ejecutar todas las cadenas de Markov estacionarias, de hecho, el parámetro c_k puede cambiar en cada paso.

Convergencia de la sucesión de distribuciones estacionarias

Para terminar la demostración de convergencia hacia el óptimo, resta ver que la sucesión $\{\pi(c_k)\}_{k=0,1,\ldots}$ converge hacia una distribución de probabilidad que únicamente asigna probabilidad distinta de 0 a las poblaciones que contienen el individuo óptimo. Sin embargo, se demostrará un resultado más fuerte, esto es, la cadena de Markov únicamente asignará probabilidad distinta de cero a la población óptima. En el caso de funciones no inyectivas la distribución asignaría probabilidad distinta de 0 a aquellas poblaciones cuyos individuos fueran todos individuos óptimos.

Ahora bien, esto es evidente si tenemos en cuenta que:

$$\lim_{k\to\infty} q_{\mathbf{11}}(c_k) = 1.$$

El límite anterior puede ser deducido fácilmente utilizando la ecuación 3.2. Esto implica que:

$$\lim_{k\to\infty} \pi_s(c_k) = \lim_{c_k\to 0} \frac{\mid Q_s(c_k) - I \mid}{\sum_{t=1}^{r} \mid Q_t(c_k) - I \mid} = \begin{cases} 0 & \text{if } s \neq 1 \\ 1 & \text{if } s = 1. \end{cases}$$

Por lo tanto la convergencia hacia el óptimo ha sido demostrada.

Como comentario final a toda la demostración es interesante resaltar que es posible pensar en otras expresiones para la distribución de probabilidad del operador de reducción que no cambiarían los resultados de convergencia. Por ejemplo la asignación de probabilidad de acuerdo con la exponencial de la diferencia $F(i) - F_{max}$ (siendo F_{max} el valor máximo de la función objetivo en la población de los padres). Sin embargo, en la ecuación 3.1, $\overline{F}$ es el mínimo valor que asegura la convergencia hacia el óptimo.

3.5 Conclusiones

Hemos dado en este capítulo una introducción al modelado matemático de los AAGG mediante el uso de cadenas de Markov. Esta forma de modelarlo ha conducido a modelos exactos para el algoritmo, con los cuales se puede describir en algunos casos el comportamiento del mismo.

Como aportación novedosa, se ha introducido un nuevo algoritmo a la vez que se ha dado una demostración de su convergencia hacia el óptimo, bajo unas condiciones sorprendentemente simples. El algoritmo puede interpretarse como un híbrido entre los AAGG y el EE o como una versión relajada de un algoritmo elitista fuerte. Tiene la propiedad de que su convergencia está guiada por un parámetro que permite además controlar la diversidad en la población y el grado de elitismo del algoritmo.

En cualquier caso se puede decir, que el desarrollo del análisis matemático de los AAGG está todavía en sus primeras etapas. Aunque desde un punto de vista teórico se han conseguido resultados importantes, estos tienen poca repercusión a nivel práctico. Es necesario dedicar un gran esfuerzo a determinar los siguientes aspectos:

- Problemas en los que la utilización de los AAGG sea particularmente ventajosa.

- Operadores adecuados a cada tipo de problema.
- Probabilidades tanto de cruce como de mutación que aceleren el proceso de búsqueda.
- Tiempos medios esperados para que un individuo óptimo aparezca por primera vez en la población.

El primer aspecto apuntado anteriormente es quizás el más importante. A partir del trabajo de Wolpert y Macready [158] donde se dice que siempre existe un conjunto de problemas donde un algoritmo se comporta mejor que otro y al contrario. El detectar aquellos problemas donde el AG es capaz de hallar mejores soluciones que los demás algoritmos de optimización parece fundamental.

3.6 Apéndice

Definición 4 *Dada un matriz estocástica P, diremos que es primitiva, si existe un entero positivo k tal que $P^k > 0$.*

Un cadena de Markov irreducible y aperiódica posee una matriz de transición de estados que es primitiva.

El teorema de Perron-Frobenius dice lo siguiente:

Teorema 3 *Sea P estocástica primitiva. Entonces existe un autovalor ρ de P tal que:*

(a) $\rho = 1$.

(b) *El autovalor $\rho = 1$ tiene asociados autovectores por la izquierda y por la derecha estrictamente positivos.*

(c) $\rho = 1 > |\lambda|$ *para cualquier otro autovalor* $\lambda \neq \rho$.

(d) *Los autovectores asociados con $\rho = 1$ son únicos, excepto por multiplicación por una constante.*

(e) *Si* $0 \leq B \leq P$ *y* β *es un autovalor de* B, *entonces* $|\beta| \leq \rho$. *Además,* $|\beta| = \rho = 1$ *implica* $B = P$.

Parte III

Clasificación No Supervisada

Capítulo 4

Clasificación No Supervisada

4.1 Introducción

En este capítulo se tratará de situar de forma adecuada el ámbito donde nos vamos a mover, así como de introducir los diferentes tipos de CNS a los que se les va a aplicar los AAGG.

La CNS la podemos situar dentro de un conjunto de técnicas más amplias denominadas en IA *clasificación*. La clasificación alberga dentro de sí dos tipos de técnicas diferentes, una es la ya citada CNS y la otra es la llamada *clasificación supervisada* (CS). Esta última técnica se conoce en Estadística con el nombre de reconocimiento de patrones. Existen bastantes diferencias entre ambas técnicas. Las dos más importantes son: el objetivo de la clasificación y la información de la que se dispone para llevar ésta a cabo.

El objetivo de la CNS, como hemos apuntado anteriormente consiste en, dado un conjunto de objetos, intentar establecer la estructura de grupos subyacente a dicho conjunto. Esto puede realizarse por varias razones, Gordon [69], Mirkin [120]: simplificación de un conjunto de datos, análisis de los mismos, predicción, extracción de conocimiento, etc. Por el contrario, la CS pretende, utilizando los objetos del conjunto, construir un sistema que asigne los propios objetos del conjunto y objetos venideros a clases cuyo número viene prefijado de antemano.

En cuanto a la información que posee cada tipo de clasificación, se tiene que la CNS únicamente dispone de los datos. Por el contrario la

CS además de lo anterior, dispone del número de clases y también de la clase a la que cada dato pertenece.

Kendall [89] recoge una buena comparación entre ambas técnicas.

Dentro de la CNS podríamos distinguir dos tipos de clasificaciones si nos atenemos a si se utilizan números o medidas para describir los objetos del conjunto de datos, o si se utilizan atributos simbólicos. La CNS en la que los objetos son descritos por atributos simbólicos y éstos son utilizados en la clasificación suele ser denominada *clasificación conceptual*. Los primeros trabajos en CNS conceptual tiene como autores a Michalsky y Stepp [118], y Fisher [54]. Las diferencias entre la CNS que llamaremos numérica y la CNS conceptual pueden encontrarse en Michalsky y Stepp [117].

Otra posible distinción dentro de la CNS se refiere a si los modelos que se construyen o los métodos que se utilizan para construirlos, están basados en la probabilidad o no. La aproximación al problema que se expondrá en esta memoria, aunque teniendo en cuenta en algunos momentos la teoría de la probabilidad, podríamos fácilmente incluirla en la aproximación no basada en modelos. Las técnicas de CNS basadas en modelos probabilistas Bandfield y Raftery [17], Duda y Hart [50] y Hanson y col. [73] suponen que los objetos del conjunto de datos han sido obtenidos muestreando una distribución de probabilidad cuya función de densidad suele expresarse como suma de funciones de densidad de distribuciones conocidas. Estas distribuciones conocidas suelen ser habitualmente normales Celeux y Govaert [35] o uniformes Hardy y Rasson [74]. Este tipo de modelos se denominan modelos mixtos. Los algoritmos de clasificación utilizados en los modelos mixtos están basados en hallar los parámetros desconocidos de las funciones de densidad anteriormente citadas. Esto suele realizarse mediante el algoritmo EM, Dempster y col. [46] u otras técnicas del mismo estilo Bandfield y col. [16].

Centrándonos en la CNS no basada en modelos probabilísticos podemos dividir los tipos de clasificación existentes en función de diferentes parámetros o factores. De esta forma, podríamos distinguir dentro de la CNS entre clasificaciones en las que se obtienen particiones del conjunto de datos (por ejemplo, las clasificaciones particional o jerárquica), y clasificaciones en las que se obtienen recubrimientos, es decir, las clases no están en general separadas, sino que comparten objetos (por ejemplo,

la clasificación piramidal). Otro tipo de clasificación que difícilmente puede introducirse dentro de estos dos grupos es la CNS fuzzy, Bezdeck [22], Bezdeck y Pal [23]. En este tipo de clasificación no se produce directamente una partición del conjunto de datos sino que a cada dato se le asigna un conjunto de números, uno por cada clase. Dicho número indica el grado de pertenencia del objeto a la clase.

El trabajo en esta memoria se centra en CNS en la que los objetos vienen descritos por números o por distancias entre ellos, los modelos que vamos a utilizar no están basados en la probabilidad, y estos pueden producir particiones del conjunto de datos o recubrimientos. En particular nos centraremos en los métodos más clásicos que producen particiones, esto es: la *clasificación particional* y la *clasificación jerárquica*, y en uno más novedoso que produce recubrimientos o clases solapadas: la *clasificación piramidal*.

A la hora de hallar la estructura inherente a un conjunto de datos se suelen seguir varios pasos de los cuales la aplicación del algoritmo de clasificación suele ser el más importante. Los pasos están descritos en la figura 4.1. Esta figura representa un diagrama sín fin ya que

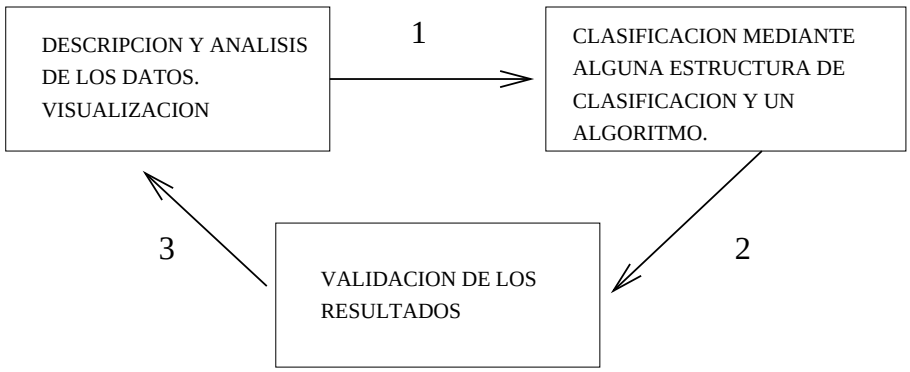

Figura 4.1: Proceso de clasificación

se supone que la información que se gana después de realizar todos los pasos puede ser utilizada para, nuevamente, intentar obtener una mejor clasificación.

Se dará a continuación una pequeña introducción al análisis de los datos y a la visualización, para en lo que sigue, meternos de lleno en los

algoritmos de clasificación, terminando con unas breves nociones sobre validación.

4.2 Análisis previo de los datos

Si el objetivo de la CNS es analizar datos, habrá que describir las formas en que estos datos pueden aparecer o se pueden representar y el tipo de datos que pueden ser utilizados para realizar dicho análisis.

Los datos pueden venir representados de dos formas diferentes: como una *matriz de medidas*, o como una *matriz de proximidad*. Utilizaremos en esta memoria una forma u otra dependiendo del tipo de algoritmo.

En la matriz de medidas cada uno de los datos viene descrito por un número finito de variables o medidas y podría ser representado por un vector de medidas. Así todos los datos podrían ser representados por una matriz de medidas. Si suponemos que cada dato $w_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$ viene expresado por p variables y que disponemos de n datos, la matriz de medidas $\mathcal{A}$ puede representarse de la siguiente forma:

$$\mathcal{A} = \begin{pmatrix} x_{11} & x_{12} & \ldots & x_{1p} \\ x_{21} & x_{22} & \ldots & x_{2p} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & \ldots & x_{np} \end{pmatrix}.$$

La matriz de proximidad es una matriz de tamaño $n \times n$ donde el elemento (i, j) representa, de alguna manera, la proximidad de los datos w_i y w_j. Las medidas de proximidad pueden ser de dos tipos: de *similaridad* y de *disimilaridad*. Las medidas de similaridad son aquellas que dan un mayor valor cuanto más parecidos o cercanos sean dos datos. Por el contrario, las medidas de disimilaridad dan un valor bajo a aquellos datos u objetos que sean muy parecidos. Normalmente se usan más las medidas de disimilaridad que las de similaridad, esto es debido, principalmente, a que nuestra intuición esta más acostumbrada a utilizar estas últimas (la propia distancia Euclidea es una medida de disimilaridad). Dado que muchos algoritmos sólo trabajan con uno de las dos tipos de proximidades es interesante encontrar algoritmos que transformen disimilaridades en similaridades y al contrario. Algunos de dichos procedimientos pueden encontrarse en el libro de Gordon [69].

Los tipos de variables o medidas que pueden aparecer en los datos se clasifican (siguiendo a Gordon [69]) principalmente en tres: *numéricos, ordinales* y *nominales*, y *binarios*.

Las variables numéricas son variables cuantitativas y toman valores reales. Éstas se suelen dividir en dos tipos: variables de tipo *rango* y variables de tipo *intervalo*. En las variables de tipo rango existe un cero absoluto, por contra en las variables de tipo intervalo no hay definido o no tiene sentido definir un cero absoluto. Un ejemplo de variable numérica de tipo intervalo es la temperatura de un paciente, aquí no tiene ningún sentido hablar de temperatura cero. Por contra una variable numérica de tipo rango puede ser la longitud de una hoja.

Las variables ordinales y nominales toman una cantidad numerable de valores, de tal manera que cada característica toma su valor en uno de estos valores. La diferencia entre los dos tipos de variables es que mientras que en la primera existe un orden (imaginemos la variable "goles conseguidos por un jugador en la liga de futbol"), en la segunda no tiene ningún sentido (un buen ejemplo es el "color de los ojos", que podría tomar tres valores: azul, marrón y verde).

Por último las variables binarias son variables que únicamente toman dos valores. Aunque podrían ser incluidas dentro de las nominales u ordinales, debido a que estas aparecen frecuentemente, se les da un tratamiento especial.

Otra clasificación de tipos de variables se puede encontrar en Jain y Dube [80] y en [6]. Estos autores distinguen dos aspectos diferentes, el *tipo* (hace referencia al grado de cuantización de los datos) y la *escala* (hace referencia al significado relativo de los números), e introducen dentro de esta clasificación a los propios números de la matriz de proximidad. El esquema de la Figura 4.2 tomado de Jain y Dubes [80] es un buen resumen de lo expuesto anteriormente.

Dado que muchos algoritmos trabajan con matrices de proximidad será primordial poder obtener a partir de una matriz de medidas una matriz de proximidad. Esto no es una tarea nada fácil si se quiere que en dicho paso se conserve la mayoría de la información que está recogida en los datos. Para realizar esta tarea se definen *índices de proximidad*.

Los índices de proximidad tienen que cumplir ciertas propiedades. Si representamos la proximidad entre el dato w_i y el dato w_j por $d(i,j)$, las propiedades que deben cumplir son las siguientes:

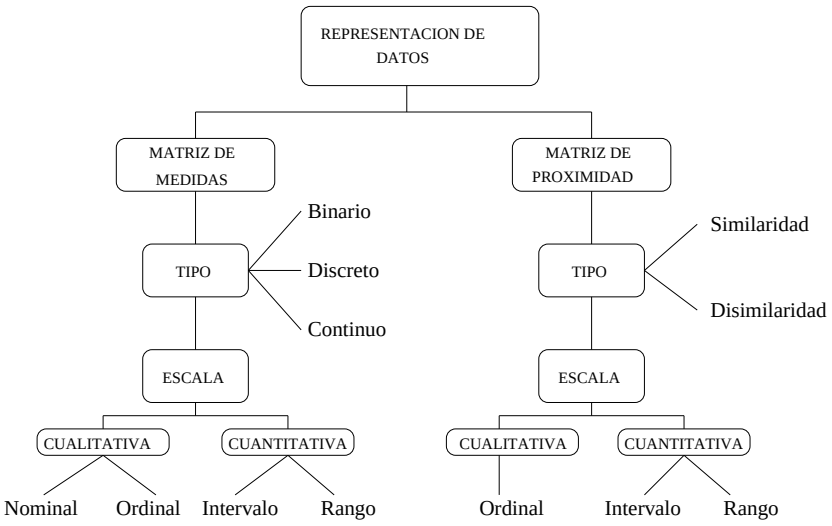

Figura 4.2: Esquema de la representación de los datos

i) (a) Para disimilaridad: $d(i,i) = 0 \ \ \forall \ i \in \{1, 2, \ldots, n\}$

i) (b) Para similaridad: $d(i,i) \geq \max_j d(i,j) \ \forall \ i \in \{1, 2, \ldots, n\}$

ii) $d(i,j) = d(j,i) \ \ \forall \ (i,j)$ con $\ i, j \in \{1, 2, \ldots, n\}$

iii) $d(i,j) \geq 0 \ \ \forall \ (i,j)$ con $\ i, j \in \{1, 2, \ldots, n\}$.

Tal y como se han planteado las propiedades anteriores los índices de proximidad deben de ser simétricos. Sin embargo, en algunas ocasiones puede ser interesante trabajar con índices no simétricos. Ejemplos de ellos se pueden ver en Jain y Dubes [80]. A lo largo de este trabajo se tratará siempre con índices de proximidad simétricos y con medidas de disimilaridad.

Existen en la literatura gran cantidad de índices de proximidad diferentes, dependientes del tipo de las variables que se estén considerando. Para las medidas numéricas de tipo rango los más habituales son los derivados de la distancia de Minkowski, expresándose la distancia entre w_i y w_k como:

$$d(i,k) = (\sum_{j=1}^{p} |x_{ij} - x_{kj}|^r)^{1/r}.$$

Dándole valores concretos a r se pueden conseguir, entre otros, la distancia Euclidea (r=2) que se representará por L_2, la distancia de Manhattan ($r = 1$) que será representada por L_1, y la distancia infinito ($r \to \infty$). Todas ellas cumplen una propiedad adicional llamada desigualdad triangular:

iv) $d(i,j) \leq d(i,k) + d(k,j) \;\; \forall\, i,j,k \in \{1,2,\ldots,n\}$.

A la hora de decidir un valor apropiado para r hay que tener en cuenta que a mayores valores de r mayor énfasis se da a grandes diferencias $|x_{ij} - x_{kj}|$.

Otra distancia empleada habitualmente es la distancia de Mahalanobis que puede ser expresada de la siguiente forma:

$$d(i,j) = (w_i - w_j)\mathcal{S}(w_i - w_j)^T$$

donde $\mathcal{S}$ es una matriz cuadrada $p \times p$. Se obtendrán distancias diferentes en función de los diferentes valores que tome $\mathcal{S}$.

Las dos proximidades binarias más utilizadas son el *coeficiente de macheado simple* y el *coeficiente de Jaccard*, ambos son índices de similaridad. Si consideramos la tabla 4.1

		w_i	
		0	1
w_j	0	a_{00}	a_{01}
	1	a_{10}	a_{11}

Tabla 4.1: Tabla utilizada para hallar los índices de proximidad en medidas binarias

donde a_{00} representa el número de variables en los que los datos w_i y w_j toman el valor 0, a_{11} el número de variables donde ambos datos toman el valor 1 y así sucesivamente, estas medidas son:

- Coeficiente de macheado simple: $d(i,j) = \frac{a_{00}+a_{11}}{p}$
- Coeficiente de Jaccard: $d(i,j) = \frac{a_{11}}{a_{01}+a_{10}+a_{11}}$.

La diferencia entre los dos índices anteriores radica en que en el primero se supone que la cantidad de veces que coinciden en el valor 1 tiene la misma importancia que la cantidad de veces en que coincidan en el valor 0. Sin embargo, en la segunda similaridad esto no es así. Más similaridades y disimilaridades para características binarias se pueden encontrar en Kaufman y Rousseeuw [87]. Un buen resumen de medidas de proximidad tanto para variables cuantitativas como binarias se puede encontrar en Cox y Cox [40].

A veces, en vez de querer hacer una clasificación de datos se pretende hacer clasificación de variables, para, por ejemplo, simplificar un conjunto de datos y utilizar sólo las variables que son necesarias. En este caso la disimilaridad entre variables se suele construir a partir de coeficientes de correlación. En Kaufman y Rousseeuw [87] pueden verse medidas de similaridad y disimilaridad para variables utilizando el coeficiente de correlación de Pearson y el de Spearman.

En la creación de matrices de proximidad, además del problema comentado anteriormente de la conservación de la mayor parte de información posible, aparecen otros problemas de tipo más técnico. Algunos de ellos son:

- variables con unidades de medida diferentes
- datos con diferentes tipos de variables (binarias, discretas continuas)
- datos con información incompleta o valores perdidos
- variables con más importancia unas que otras, etc.

El problema de las diferentes unidades de medida proviene del siguiente hecho. Imaginemos que utilizamos la distancia Euclidea para hallar la matriz de proximidad. El hecho de que una medida esté en cm. o en m. influye de forma importante en el resultado final. Este problema suele resolverse por medio de la normalización, es decir, si denotamos por:

$$m_j = (1/n)\sum_{i=1}^{n} x_{ij} \qquad S_j = (1/n)\sum_{i=1}^{n}(x_{ij} - m_j)^2$$

la media y la varianza de los datos en la variable j, cada medida de cada dato es reemplazada por uno de los siguientes valores:

- $xn_{ij} = x_{ij} - m_j$ siendo xn_{ij} el valor de la variable normalizada. Con esto se consigue que la media muestral en todas las variables sea 0.

- $xs_{ij} = \frac{x_{ij}-m_j}{S_j}$. Con esto se consigue que todas las variables tengan media muestral 0 y varianza muestral 1.

Existen por supuesto más tipos de normalizaciones algunas de las cuales tratan de evitar el problema de los denominados "outliers", es decir, datos que por alguna razón han sido mal medidos y que influyen de forma negativa en la clasificación, y particularmente en la normalización (Kaufman y Rousseeuw [87]).

Sin embargo, el hecho de normalizar cada variable por separado hace que esta normalización pueda distorsionar las relaciones entre las diferentes variables. Ejemplos del efecto nocivo de la normalización en algunos casos pueden encontrarse en Jain y Dubes [80]. Esto se puede solucionar utilizando la distancia de Mahalanobis tomando como $\mathcal{S}$ la matriz de varianzas-covarianzas de los datos.

En el caso de la existencia de diferentes tipos de variables, una posible solución es calcular la disimilaridad de cada variable de manera diferente, cada una con una distancia acorde con el tipo de variable. De esta forma, para cada par de objetos se calcularía la diferencia entre cada variable y la suma de estas diferencias se convertiría en la disimilaridad entre los objetos.

Otro de los problemas que uno se encuentra cuando intenta analizar datos es el hecho de que éstos muchas veces no son completos, hay algunas medidas que faltan porque no se pudieron recoger o porque se perdieron. Las formas de tratar con datos perdidos pueden ser muy diversas, algunas de éllas se tratan de recoger a continuación:

- Se borran aquellos datos que no tengan todas sus medidas.

- Se reemplaza el valor perdido en la i-ésima variable del dato por la media de los valores que en dicha variable toman los k datos más cercanos al que posee el valor perdido.

- Se mide la distancia entre dos datos sin tener en cuenta la variable que tenía un valor perdido.
- Se mide la distancia media de todos los individuos en cada variable. La distancia entre cada par de datos en la variable i-ésima será igual a la media anterior si alguno de los datos tiene un valor perdido en dicha variable.

El problema del peso de las variables es quizás el más difícil de resolver ya que depende del criterio que se esté utilizando para realizar la clasificación y de la información que posea la persona que está realizando ésta. El pesar las variables es equivalente, en realidad, al problema de las diferentes medidas en los datos que producen un cambio de escala, ya que dar un peso a una variable a la hora de calcular la disimilaridad es equivalente a cambiar el tipo de medida de esa variable.

4.3 Visualización

El ojo humano es el mejor mecanismo del que disponemos para hallar clases en datos. Sin embargo, el mecanismo ojo-cerebro sólo es capaz de descubrir clases en una, dos o tres dimensiones. De esto se deducen dos cosas: la primera es la necesidad del uso de algoritmos para hallar clases de datos en dimensiones mayores que tres, y la otra es la gran cantidad de esfuerzo dedicado al desarrollo de técnicas que intentan representar datos p-dimensionales ($p > 3$) en 1, 2 o 3 dimensiones. Si esto último pudiera llevarse a cabo conservando toda la información inherente a los datos, la clasificación podría realizarse a partir de una representación de los objetos en dicho subespacio de 1, 2 o 3 dimensiones. Describiremos a continuación algunas de las aproximaciones que se han desarrollado en lo que llamaremos *reducción de la dimensionalidad.*

Comenzaremos con las *proyecciones lineales.* La idea es intentar reducir la dimensionalidad de los datos, pasar de p a m ($p >> m$) dimensiones mediante una transformación lineal conservando la mayor parte de la información inherente a los datos en términos de varianza. Es decir, queremos hallar una matriz $\mathcal{H}$ de tamaño $m \times d$ tal que $u = \mathcal{H}w$, siendo u el nuevo dato. La forma más común de realizar esta transformación es utilizar la matriz compuesta por los autovectores

de la matriz de varianzas-covarianzas de los datos. Estos existen, ya que la matriz de varianzas-covarianzas es definida positiva. Utilizando los m autovectores asociados a los m autovalores mayores de la matriz de varianzas-covarianzas se consigue que la transformación tenga las siguientes propiedades:

- los ejes del espacio m-dimensional están en las direcciones de los autovectores asociados a los mayores autovalores
- las variables en el nuevo espacio son no correladas, es decir, tienen coeficientes de correlación nulos
- la suma de los m primeros autovalores es la varianza retenida en el nuevo espacio (si $p = m$ la suma de los autovalores es la varianza de los datos iniciales)
- es posible obtener la proyección anterior de manera que se minimize cierta diferencia cuadrática.

La proyección lineal expuesta anteriormente no da buenos resultados en todos los casos. Un ejemplo de esto se puede encontrar en Jain y Dubes [80]. Esto es debido a dos razones: por un lado puede que los autovectores que se estén cogiendo para hallar la matriz $\mathcal{H}$ no sean los más adecuados de cara a la clasificación, y por otro puede que los datos tengan una estructura tan complicada que la cantidad de información perdida con la proyección lineal sea demasiado grande.

Para solucionar el segundo problema aparecen las técnicas denominadas de *proyección no lineal*. Estas técnicas tienen como objetivo el mismo que las anteriores, es decir, representar datos en dimensión 1, 2 o 3 perdiendo la mínima cantidad de información. Las proyecciónes no lineales en vez de tratar de buscar una matriz y proyectar linealmente los puntos, lo que hacen es intentar minimizar una función no lineal que de alguna manera mide la pérdida de información en que se incurre cuando se realiza la proyección. Un ejemplo de este tipo de proyección es el siguiente: dados n puntos p-dimensionales ($p > 3$), encontrar n puntos en dos dimensiones de forma que estos n puntos conserven en lo posible las distancias entre los primeros n puntos. Una de las funciones

que se ha utilizado para medir la pérdida de información es la siguiente:

$$E = \sqrt{\frac{\sum_{i<j}(d(i,j) - D(i,j))^2}{\sum_{i<j} d(i,j)^2}}$$

siendo $d(i,j)$ la distancia entre los datos w_i y w_j en el espacio (inicial) p-dimensional y $D(i,j)$ la distancia entre los datos correspondientes en el espacio 2-dimensional.

Las proyecciones no lineales tienen principalmente dos problemas. Uno es que la función E esta llena de mínimos locales, luego es muy difícil hallar el mínimo global de la misma. El otro es la gran cantidad de parámetros de la que esta depende, en nuestro caso $2 * n$. Recientemente se ha leído una tesis doctoral que recoge la aplicación de los AAGG a dicho problema, Ngouenet [123].

Hasta ahora hemos tratado el problema de la proyección lineal o no lineal de un conjunto de puntos p-dimensional en un espacio de dimensión menor, pero como anteriormente dijimos, los datos no sólo vienen representados en forma de una matriz de medidas, sino que también pueden venir representados en forma de matrices de proximidad. En este último caso también se han desarrollado técnicas de representación que se conocen con el nombre de *escalado multidimensional* y que en el fondo son bastante similares a las expuestas anteriormente. La mayor parte del esfuerzo desarrollado en este último problema ha sido en el caso en que la matriz de disimilaridad esté compuesta de datos ordinales, parte de este estudio puede verse en Jain y Dubes [80].

4.4 Clasificación particional

La CNS particional es las más habitual y la más extendida. Esta técnica trata de descubrir la posible estructura de grupos que pueda poseer un conjunto de datos. Es decir, dado un conjunto de datos Ω, se trata de dividir dicho conjunto en grupos o clases $\{C_1, C_2, \ldots, C_k\}$ de manera que cada clase posea objetos que son parecidos entre sí, a la vez que diferentes de los objetos de las demás clases. Para el concepto de clase

se pueden dar muchas definiciones diferentes, algunas de ellas se recogen en Jain y Dubes [80].

Aunque aparentemente con lo escrito anteriomente el problema está definido de forma clara, queda al menos un punto de incertidumbre: ¿que significa la palabra "parecidos"? Evidentemente el dar significado preciso a la palabra "parecidos" en la definición anterior implica de alguna manera cuantificar dicha noción. Así, casi todos los métodos de clasificación particional se basan en optimizar una función matemática, F, que depende de la partición del conjunto en clases:

$$F : \mathcal{P}_k(\Omega) \longrightarrow I\!R$$

siendo en este caso $\mathcal{P}_k(\Omega)$ el conjunto de las particiones del conjunto Ω en k subconjuntos. A dicha función F la llamaremos *criterio de clasificación.*

Como puede deducirse de la definición, el hallar la estructura de clases subyacentes al conjunto no es una tarea trivial y envuelve principalmente dos tipos de problemas:

1. hallar el número de clases que existen en el conjunto
2. dado el número de clases anterior, repartir los objetos del conjunto en cada clase de forma que se optimice la función F.

Para cuantificar en mayor medida la dificultad del problema, es suficiente ver que el número de posibles divisiones de n objetos en k subconjuntos no vacíos viene dado por un número de Stirling de segundo tipo:

$$\frac{1}{k!}\sum_{j=1}^{k}(-1)^{k-j}\left(\begin{array}{c} k \\ j \end{array}\right) j^n$$

y si tenemos en cuenta que k puede variar de 1 a n entonces dicho número pasa a ser:

$$\sum_{k=1}^{n}\frac{1}{k!}\sum_{j=1}^{k}(-1)^{k-j}\left(\begin{array}{c} k \\ j \end{array}\right) j^n.$$

Los dos problemas han sido resueltos principalmente con dos tipos de estrategias diferentes. La primera de dichas estrategias consiste en

utilizar en una fase inicial un algoritmo que halle el número de grupos en el conjunto, para en una fase posterior utilizar otro algoritmo que dado el número de grupos asigne cada objeto a la clase más adecuada. La segunda estrategia consiste en utilizar un algoritmo que realice ambas cosas al mismo tiempo.

Entraremos en primer lugar a estudiar de manera somera los algoritmos existentes para hallar el número de grupos en un conjunto de datos.

4.4.1 Algoritmos para hallar el número de grupos en un conjunto

La motivación para este tipo de técnicas radica en el hecho de que la mayoría de los algoritmos de clasificación particional existentes son únicamente capaces de hallar las clases si tienen como parámetro de entrada su número. Por lo tanto, una persona que se tuviera que enfrentar a un conjunto de datos y no dispusiera del número de clases, tendría que aplicar un algoritmo varias veces, cada vez con un número de clases diferentes, para terminar, a la vista de los resultados, decidiendo cual es la mejor clasificación sin ningún criterio objetivo en que basarse.

Nuevamente se pueden distinguir varios tipos de algoritmos para hallar el número de clases en un conjunto. Existen algoritmos que se basan en algún resultado de *tipo heurístico*, otros que se basan en realizar *test de hipótesis*, y finalmente algunos que plantean utilizar *modelos probabilistas*. Un buen resumen exhaustivo de estas técnicas a la vez que una comparación de las mismas se pueden encontrar en Milligan y Cooper [119], exhaustiva bibliografí en Gordon [71]. Daremos aquí como ejemplo una técnica de cada clase.

Técnica heurística para hallar el número de clases

Comenzaremos con la técnica heurística más conocida. Esta técnica consiste en construir una gráfica en la que el número de clases se representa en las abcisas y los posibles valores que puede alcanzar un algoritmo de clasificación en la función F se representan en las ordenadas. A partir de esto se dibuja una gráfica en la que aparecen los

valores conseguidos por un algoritmo de clasificación particional en F para cada número de clases concreto k. El número de clases óptimas será aquel para el que exista una discontinuidad en la gráfica. Evidentemente este criterio es totalmente heurístico, por ello, aunque ha sido un criterio muy extendido hasta hace poco tiempo, ha recibido últimamente muchas críticas (Gordon [69]).

Test de hipótesis para hallar el número de clases

Un ejemplo en el que se utiliza un test de hipótesis para hallar el número de clases en un conjunto de datos puede encontrase en Rasson y Kubushishi [131]. En este trabajo los autores suponen que los datos a clasificar provienen de un proceso de Poisson estacionario en el dominio C, siendo C la unión de k dominios compactos y convexos disjuntos, formando cada dominio un grupo diferente. El test se basa en medir el espacio existente entre las clases de una partición con un número de clases k.

Supongamos que tenemos un conjunto de puntos:

$$\{w_1, w_2, \ldots, w_{n_1}, w_{n_1+1}, \ldots, w_{n_1+n_2}\}$$

provenientes de un proceso de Poisson en $C = C_1 \cup C_2$. Se puede suponer que los primeros n_1 números están en C_1 y los siguientes en C_2. El test de hipótesis que se plantea es entonces:

H_o : hay $n = n_1 + n_2$ en $C_1 \cup C_2$

H_1 : hay n_1 puntos en C_1 y n_2 en C_2 con $C_1 \cap C_2 = \emptyset$

Si llamamos m_1, m_2, m_D a las medidas de las envolventes convexas de cada uno de los conjuntos de puntos, la formulación de la razón de la verosimilitud es:

$$Q(w) = \frac{L_{H_0}(w)}{L_{H_1(w)}} = \frac{m_D^n}{m_1^n m_2^n}.$$

Por el lema de Neyman-Pearson la región crítica es:

$$\{x | m' \leq U_\alpha\}$$

donde $m^{'} = m_D - m_1 - m_2$, es decir, el estadístico del test es precisamente el espacio entre clases: $m^{'}$. La estrategia del test es negar H_0 si el espacio entre las clases C_1 y C_2 es muy pequeño.

Utilización de un modelo mixto para hallar el número de clases

Por último veamos un método en el que se utilizan los modelos mixtos pero no un test de hipótesis. En el modelo mixto se supone que cada dato w proviene de una distribución de probabilidad con función de densidad:

$$\theta(x) = \sum_{j=1}^{k} p_j f(x, a_j)$$

los parámetros p_j y a_j son, en principio, desconocidos. A los primeros se les denomina proporciones de la mezcla. La función f es una distribución de probabilidad, habitualmente se suele tomar la distribución normal multivariante, donde a_j representa los parámetros de dicha distribución, en el caso de anterior $a_j = (\mu_j, \Sigma_j)$. El neperiano de la función de verosimilitud asociado a la muestra toma la forma:

$$L(k) = \sum_{i=1}^{n} \ln[\sum_{j=1}^{k} p_j f(x_i, a_j)]$$

Existe un criterio de clasificación que se basa en la maximización del neperiano de la función de verosimilitud suponiendo el número de clases k fijo. Sin embargo, esta función no puede utilizarse para hallar el número de grupos en un conjunto de datos ya que la verosimilitud aumenta con el número de grupos k, luego siempre obtendríamos como número de grupos el número de datos.

La técnica para hallar el número de clases en el conjunto consiste en penalizar al neperiano de la función de verosimilitud. Esta penalización se suele llevar a cabo en función del número de parámetros desconocidos del sistema. La idea es penalizar modelos que sean demasiado complejos. En Celeux y Soromenho [36] se recogen algunos de los criterios expuestos a continuación.

Si denotamos por $\nu(K)$ el número de parámetros libres en el modelo, varios criterios para hallar el número de clases son los siguientes:

- Criterio de información de Akaike:

  $$AIC(K) = -2L(K) + 2\nu(K).$$

  Este criterio es orden inconsistente (no tiende a 1 la probabilidad de acertar el orden del modelo, cuando el tamaño del conjunto de datos aumenta) y tiende a sobreestimar los modelos.

- Criterio de información Bayesiano:

$$BIC(K) = -2L(K) + 2\nu(K)\ln n$$

es orden consistente y se supone que debería dar una respuesta a la sobreparametrización del AIC, sin embargo, no hay evidencia de ello.

- Peso aproximado de evidencia:

$$AWE(K) = -2L(K) + 2\nu(K)(\frac{3}{2} + \ln n).$$

4.4.2 Algoritmos para hallar los grupos en un conjunto dado el número de clases

Siguiendo con la primera aproximación al problema de la clasificación particional, pasamos ahora a ver un compendio de los tipos de algoritmos más utilizados para dividir un conjunto de datos en clases siendo el número de éstas conocido de antemano.

Antes de nada hay que tener en cuenta que casi todos estos algoritmos utilizan el mismo criterio de clasificación. Suponen que cada objeto del conjunto está definido por p variables numéricas. De esta forma es posible considerar los objetos del conjunto como vectores en un espacio $\mathbb{R}^p$.

El criterio de clasificación utilizado por estos algoritmos consiste en una función, que dada una partición $\{C_1, C_2, \ldots, C_k\}$, halla para cada clase, C_i, el punto medio (baricentro) de los objetos que forman dicha clase:

$$\overline{w}_i = \frac{1}{n_i}\sum_{j=1}^{n_i} w_{ij} \quad i = 1, 2, \ldots, k$$

donde n_i es el número de objetos que se encuentran en la clase C_i. A dichos puntos les llamaremos *centroides* de aquí en adelante. El criterio de clasificación (lo denotamos por B), consiste en las sumas al cuadrado de las distancias entre los elementos de cada clase y su centroide, esto es:

$$B(\{C_1, C_2, \ldots, C_k\}) = \sum_{i=1}^{k}\sum_{j=1}^{n_i} \parallel w_{ij} - \overline{w}_i\parallel^2$$

donde $\| \ \|$ está representando la norma Euclidea.

Los dos primeros algoritmos que expondremos a continuación tratan de minimizar la función B. Los métodos que utilizan este criterio de clasificación se les suele llamar *métodos de minimización de la varianza.* A pesar de lo dicho anteriormente estos métodos también pueden utilizarse con otro tipo de distancia entre los elementos del conjunto, como la distancia de Mahalanobis o cualquier otra distancia de Minkowsky que no sea la L_2.

Un resultado importante relacionado con esto es el dado por Huygens:

Teorema 4 *Dado un conjunto Ω, se cumple que para cualquier partición $\{C_1, C_2, \ldots, C_k\}$ del conjunto la suma:*

$$B + W = T$$

permanece constante. Donde B es la función definida anteriormente, W es la distancia ponderada entre los centroides de las clases, $\overline{w}_i$, y el centroide del conjunto de datos $\overline{w}$:

$$W = \sum_{i=1}^{k} n_i \parallel \overline{w}_i - \overline{w} \parallel^2$$

y por último T es la distancia entre cada objeto del conjunto de datos y el centroide del mismo:

$$T = \sum_{i=1}^{n} \parallel w_i - \overline{w} \parallel^2$$

Cada uno de estos valores representa una noción diferente, así a la función B se la suele denominar la varianza intra-clase, W la varianza entre-clases y T la varianza total.

Este resultado es importante, entre otras cosas, ya que nos dice que el criterio anterior no puede ser utilizado para comparar clasificaciones con diferente número de clases.

Pasando ya a los algoritmos tenemos que los dos algoritmos de clasificación particional más conocidos son el "k-medias" propuesto por MacQueen en 1967 [111] y el "Forgy" que toma el nombre del autor

Forgy [61]. Ambos algoritmos son parecidos y se basan en lo siguiente: dada una partición inicial en k clases realizan cambios de objetos de un grupo a otro en caso de que dicho cambio reduzca el valor de la función B.

Un pseudocódigo para el k-medias es el siguiente:

1. Dar una partición inicial
2. Siguiendo el orden de los datos realizar:
   i) Asignar el objeto w a la clase C que más reduzca el valor de la función criterio B
   ii) Modificar el centroide de la clase C
3. Si algún objeto ha cambiado de clase ir al paso 2. En caso contrario terminar dando la clasificación obtenida.

El algoritmo propuesto por Forgy es similar al anterior, pero en vez de modificar el valor del centroide una vez reasignado el objeto w, éste permanece constante hasta el siguiente paso del algoritmo, esto es:

1. Dar una partición inicial.
2. Siguiendo el orden de los datos asignar cada objeto w a la clase C que más reduzca el valor de B.
3. Recalcular los centroides de cada clase.
4. Si algún objeto ha cambiado de clase ir al paso 2. En caso contrario terminar dando la clasificación obtenida.

Aunque más adelante comentaremos con más detalle los problemas o carencias generales de estos algoritmos, una de las más importantes es la necesidad de que los datos vengan descritos por medio de una matriz de medidas. Esto ha motivado el nacimiento de métodos parecidos pero más generales, donde los datos de entrada al algoritmo son simplemente una matriz proximidad entre los objetos.

En particular el método que damos a continuación, tomado de Kaufman y Rousseeuw [87], utiliza como entrada una matriz de disimilaridad. Este método en vez de utilizar la noción de centroide (no puede

hacerlo ya que en general los datos no vienen descritos por variables numéricas), utiliza la noción de *medoide*. El medoide es el punto de cada clase cuya disimilaridad media a los demás elementos de la clase es mínima. Uno de los argumentos utilizados por los autores para usar este concepto en vez del de centroide, aún en el caso de disponer de datos descritos por variables numéricas, es que de esta forma, siempre se posee un representante de cada clase, que pertenece al propio conjunto de datos. Por otro lado, el criterio de clasificación utilizado por este algoritmo, consiste en minimizar la suma de la disimilaridades de cada objeto al medoide de la clase en la que se encuentra. Los autores argumentan que esta función es más robusta a outliers que la función B.

El algoritmo es como sigue:

1. Hallar una partición inicial.

2. Para cada par de objetos $(w, w^{'})$ siendo w no medoide y $w^{'}$ medoide, realizar el cambio de uno por otro almacenando el cambio en valor del criterio de clasificación.

3. Si no se produce ninguna mejora en el criterio parar. En caso contrario elegir el par $(w, w^{'})$ que mayor mejora produzca. El objeto w pasa a ser medoide y el $w^{'}$ deja de serlo. Recalcular todas las clases asignando cada objeto a la clase con el medoide más cercano. Volver al paso 2.

Los autores utilizan otro algoritmo para hallar la partición inicial. Dicho algoritmo elige k medoides para a partir de ellos calcular las clases. Los autores comienzan eligiendo el punto central como medoide para, a partir de él, elegir aquel objeto que más minimiza el valor del criterio, y así sucesivamente hasta elegir los k medoides iniciales.

Es importante darse cuenta que el algoritmo anterior lo que trata es de buscar los k medoides mejores, ya que dados estos, las clases quedan completamente determinadas. Es decir, se trata de realizar una búsqueda en un espacio de tamaño:

$$\left(\begin{array}{c} n \\ k \end{array} \right).$$

En cualquier caso, parece evidente que el tiempo de ejecución de este algoritmo va a ser mayor que el de los dos anteriores ya que comprueba todos los pares medoide-no medoide ($(n-k)k$ en cada iteración).

En el propio libro de Kaufman y Rousseeuw [87] se puede ver como este problema puede ser planteado como un problema de programación 0-1, y algunas referencias con técnicas "ramificar y acotar" para resolverlo de forma óptima. Sin embargo, evidentemente esto sólo puede hacerse para conjuntos de datos muy pequeños (número de datos menor que 50).

Existen en la literatura muchos más algoritmos, sin embargo, no es el objetivo de esta memoria el recogerlos todos, sino el de dar un repaso a los más conocidos para, a la vez de introducirnos en el área, motivar el trabajo posterior.

Un pequeño análisis de los algoritmos anteriores nos lleva a concluir que los tres adolecen de los siguientes problemas:

- En el k-medias y en el Forgy la construcción de las clases se realiza de forma local, es decir, en cada paso del algoritmo se elige la mejor opción, teniendo en cuenta la estructura de vecinos que ambos algoritmos imponen en el espacio de búsqueda, los dos conducen a un mínimo local de B.

- En el k-medias y el resultado del algoritmo depende del orden de introducción de los datos.

- El resultado de los tres algoritmos depende de la clasificación inicial de la que se parte ya que son algoritmos de optimización local.

- En el Forgy puede ocurrir que nos devuelva una partición con menos clases de las especificadas de antemano, k.

4.4.3 Algoritmos sin conocer el número de clases

Dentro de los algoritmos que hacen todo a la vez, es decir, hallan el número de clases a la vez que clasifican los objetos del conjunto en clases, destaca el ISODATA, (Ball y Hall [15]. Este ha sido el algoritmo más utilizado para hacer ambas cosas durante mucho tiempo. Un pseudocódigo para el mismo puede verse a continuación:

1. Selecciona una partición inicial en k clases
2. Realiza el algoritmo Forgy
3. Si se cumple algún criterio de parada, parar el algoritmo. En caso contrario, ajustar el número de clases mezclando o separando clases. Borrar clases muy pequeñas y outliers.

Estudiando el código anterior puede verse que este algoritmo necesita de muchos parámetros y decisiones que tienen que ser tomadas de antemano por parte del usuario: el criterio de parada, cuando se juntan dos clases, cuando se separan, que es una clase pequeña, que es un outlier, etc. Por supuesto, las claves del algoritmo radican en estos parámetros. Dos clases suelen ser unidas si sus centroides están muy cercanos, mientras que, una clase suele separarse en dos si contiene muchos objetos y su varianza en alguna variable es extremadamente grande.

Como se ve por lo dicho anteriormente, este algoritmo no sólo arrastra los problemas que tenían tanto el k-medias como el Forgy, sino que además tenemos un conjunto de parámetros que tienen que ser elegidos por el usuario de manera más o menos arbitraria.

4.5 Clasificación jerárquica

La clasificación no supervisada jerárquica es otra técnica de CNS. Esta técnica, en vez de crear una única partición en el conjunto de datos, crea una sucesión encajada de particiones cuya estructura puede ser representada por medio de un árbol. Un ejemplo de una clasificación jerárquica puede verse en la figura 4.3. En la misma se clasifica un conjunto con cinco objetos. Las hojas del árbol representan clases conteniendo los objetos a clasificar, cada nodo interno representa un clase que contiene a todos los objetos que se encuentran en las hojas de dicho nodo. La raíz representa a la clase que contiene a todos los objetos del conjunto. Al lado de cada nodo se encuentra un número que representa el "grado" al que se han juntado dichas clases y será llamado *altura*.

Como puede verse en la figura 4.4, es posible cortar el árbol de forma que se obtenga una clasificación para cada número k ($k = 1, 2, \ldots, n$)

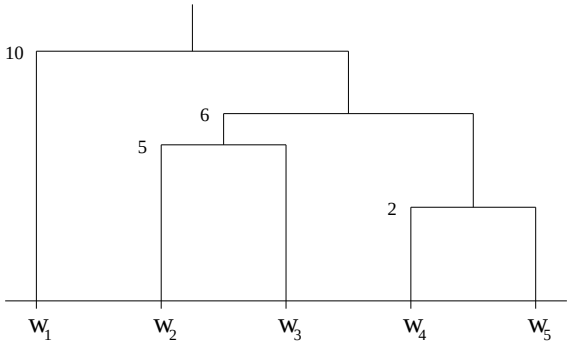

Figura 4.3: Ejemplo de una clasificación jerárquica

de clases diferentes. Así, si cortásemos el árbol a un tercer nivel, como en la figura 4.3, obtendríamos las siguientes clases:

$$C_1 = \{w_1\},\ C_2 = \{w_2, w_3\},\ C_3 = \{w_4, w_5\}.$$

Antes de pasar a dar una introducción de los tipos de algoritmos que

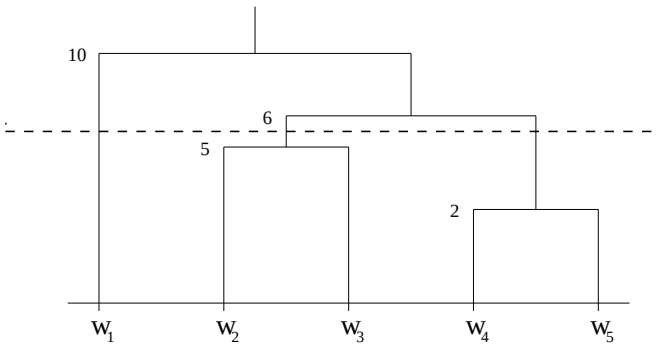

Figura 4.4: Corte en el árbol para obtener un número de clases concreto

se utilizan para hallar una clasificación jerárquica de un conjunto de datos, vamos a pasar a definir todos los conceptos anteriores de manera formal.

La formalización la hemos tomado principalmente de Gordon [70], donde además puede encontrarse una buena revisión de la clasificación jerárquica.

Definición 5 *Dado un conjunto de datos Ω, un árbol sobre Ω es un subconjunto T de $\mathcal{P}(\Omega)$ tal que :*

i) $\Omega \in T$

ii) $\emptyset \notin T$

iii) $\forall\, \omega \in \Omega \Longrightarrow \{\omega\} \in T$

iv) $\forall\, A, B \in T \implies A \cap B \in \{\emptyset, A, B\}$.

Definición 6 *Una clasificación jerárquica o dendograma sobre Ω es un par (T, h) formado por un árbol T sobre Ω junto con una aplicación h, que llamaremos altura, definida en los nodos de T de manera que se cumple:*

i) $h(A) = 0 \iff A = \{\omega\}$ *para algún* $\omega \in \Omega$

ii) *Si* $A \cap B \neq \emptyset$ *entonces* $A \subset B \iff h(A) \leq h(B)$.

Pasaremos a continuación a dar una breve descripción de los tipos de algoritmos utilizados para hallar clasificaciones jerárquicas. Estos algoritmos son de tres tipos diferentes: *algoritmos aglomerativos*, *algoritmos divisivos* y *algoritmos directos*. Estos últimos, debido a su cercanía con el trabajo desarrollado en esta memoria, serán expuestos en el capítulo dedicado a la aplicación de los AAGG al problema de la clasificación jerárquica.

4.5.1 Algoritmos aglomerativos

Los algoritmos aglomerativos suelen tener como entrada -Mirkin [120]- una matriz de proximidad aunque, como veremos más tarde, algunos algoritmos necesitan de una matriz de medidas. Se caracterizan por ir construyendo el árbol de abajo hacia arriba.

Se parte de una situación en la que cada objeto del conjunto de datos se encuentra en una sola clase. En cada paso del algoritmo se unen dos clases, de acuerdo a algún criterio, para formar una nueva clase. Se procede de esta forma hasta que todos los objetos se encuentran en una única clase.

La forma de elegir las clases que se unen en cada paso del algoritmo es lo que conduce a diferentes clasificaciones. Habitualmente se suelen unir aquellas clases que se encuentran más cercanas, de acuerdo a alguna medida o disimilaridad definida entre clases.

La mayoría de las distancias se pueden ver como un caso particular de la siguiente fórmula dada por Lance y Williams [93]:

$$d(C_i \cup C_j, C_k) = \alpha_i d(C_i, C_k) + \alpha_j d(C_j, C_k) + \beta d(C_i, C_j) + \\ + \gamma \mid d(C_i, C_k) - d(C_j, C_k) \mid$$

siendo C_i y C_j las clases que se acaban de unir.

Dando diferentes valores a los parámetros $\alpha_i, \alpha_j, \beta$ y γ obtendremos diferentes medidas de disimilaridad. Las disimilaridades más utilizadas se recogen a continuación en la Tabla 4.2.

Nombre	α_i	β	γ
Mínimo	$\frac{1}{2}$	0	$-\frac{1}{2}$
Máximo	$\frac{1}{2}$	0	$\frac{1}{2}$
UPGMA	$\frac{p_i}{p_i+p_j}$	0	0
Centroide	$\frac{p_i}{p_i+p_j}$	$\frac{-p_i p_j}{(p_i+p_j)^2}$	0
Ward	$\frac{p_i+p_k}{p_+}$	$\frac{-p_k}{p_+}$	0
Mediana	$\frac{1}{2}$	$\frac{1}{4}$	0
Media	$\frac{1}{2}$	0	0
Flexible	$\frac{1}{2}(1-\beta)$	$\beta(<1)$	0

Tabla 4.2: Disimilaridades utilizadas en clasificación jerárquica aglomerativa

En la tabla anterior p_i es el peso asociado a la clase C_i y en la mayoría de los casos está representando el número de objetos que hay

en dicha clase, mientras que p_+ es $p_i + p_j + p_k$. El valor β en el método Fexible es un parámetro del cual depende el método.

Método del mínimo

El método del mínimo es el más simple y a la vez que el más utilizado y estudiado. Se caracteriza por definir la distancia entre dos clases C_i y C_j como la mínima distancia entre los objetos de una clase y los de la otra, es decir:

$$d(C_i, C_j) = \min\{d(w_i, w_j) \mid w_i \in C_i \text{ y } w_j \in C_j\}.$$

Abusamos en este caso de la notación utilizando la letra d tanto para la distancia entre objetos, como para la distancia entre clases.

El principal reproche que se le ha asociado a este método es el hecho de que, cuando existan dos clases perfectamente separadas pero que posean dos objetos cercanos el método les unirá. Esto es lo que ha sido denominado el efecto cadena. Por lo tanto el método del mínimo dará lugar a clases con aspecto más o menos estirado o incluso lineal, lejos del aspecto de hiperesfera intuitivamente esperado.

Método del máximo

El método del máximo puede ser considerado justo como el contrario al método anterior. En él la distancia entre dos clases se define como la máxima distancia entre los elementos de ambas clases:

$$d(C_i, C_j) = \max\{d(w_i, w_j) \mid w_i \in C_i \text{ y } w_j \in C_j\}.$$

Evidentemente este método padece de los problemas opuestos al anterior. Con el método del máximo se producen muchas clases muy compactas. Así dos clases que poseen únicamente dos puntos muy alejados difícilmente serán unidas.

UPGMA

Este método se encuentra situado en medio de los dos anteriores. En él la disimilaridad entre dos clases está definida como la media de las

disimilaridades entre los elementos de ambas clases:

$$d(C_i, C_j) = \frac{1}{p_i p_j} \sum_{w_i \in C_i \ w_j \in C_j} d(w_i, w_j).$$

Este método es el que más se utiliza en la actualidad. Una implementación así como gran información acerca de él puede encontrarse en Kaufman y Rousseeuw [87].

Método del centroide

El método del centroide sólo es válido para datos que vengan descritos, no por una matriz de disimilaridad como los anteriores, sino por una matriz de medidas con variables numéricas.

Para cada clase, el método considera su centroide $\overline{x}(C_i)$ y se define la distancia entre dos clases como la distancia Euclidea entre sus centroides:

$$d(C_i, C_j) = \parallel \overline{x}(C_i) - \overline{x}(C_j) \parallel .$$

Uno de los principales problemas de este método radica en el hecho de no producir un dendograma. Se pueden dar casos en los que el método no es monótono, es decir, la disimilaridad asociada a la unión de dos clases podría ser estrictamente menor a la disimilaridad asociada a dos clases unidas previamente.

Método de Ward

Este método nuevamente hace uso del valor de las variables que describen los objetos del conjunto de datos, y supone que éstas son reales.

Para hallar la disimilaridad entre dos clases hace uso también de la distancia Euclidea entre los centroides, pero en este caso multiplicada por un factor:

$$d(C_i, C_j) = \left(\frac{2 p_i p_j}{p_i + p_j} \parallel \overline{x}(C_i) - \overline{x}(C_j) \parallel^2 \right)^{1/2} .$$

El método trata de unir aquellas clases que hacen que la suma de las distancias de cada objeto de una clase a su centroide sea mínima en el conjunto de todas las clases.

Método de la mediana

Este método es una variante del método del centroide. Al igual que él necesita que los objetos vengan descritos por sus variables. La idea de este método radica en no tener en cuenta el número de objetos que tiene cada clase que es unida a la hora de formar la nueva clase.

Para ello se define un nuevo valor llamado el centro de una clase C, $c(C)$, que en el caso de un sólo objeto es él mismo, y para las demás clases utiliza la siguiente fórmula de modificación:

$$c(C_i \cup C_j) = \frac{1}{2}c(C_i) + \frac{1}{2}c(C_j).$$

La disimilaridad entre dos clases es entonces definida como:

$$d(C_i, C_j) = \parallel c(C_i) - c(C_j) \parallel .$$

Uno de los problemas de este método es que se puede llegar a contradicciones debido a que los valores de los centros dependen del orden en el que éstos se hallan. Así se podrían dar situaciones en que la altura asociada a una misma clase podría ser diferente dependiendo del orden en el que han venido los datos y por lo tanto el orden en que se han calculado los centroides.

Otro problema es que, de la misma forma que el método del centroide, este método no nos conduce necesariamente a un dendograma.

En los métodos estudiados hasta el momento era posible definir una distancia entre clases, en los que vienen a continuación ésto es imposible. Estos métodos se caracterizan por poseer únicamente una fórmula de modificación. Es decir, la altura asociada a una nueva clase está en función de la altura asociada a las dos clases que la han formado.

Método de la media

El método de la media es una variante del método UPGMA. Se parte de la disimilaridad entre los datos y la fórmula que nos da la altura de la nueva clase es:

$$d(C_i \cup C_j, C_k) = \frac{1}{2}d(C_i, C_k) + \frac{1}{2}d(C_j, C_k).$$

Al igual que en el método anterior pueden ocurrir contradicciones, en este caso si se presentan empates. Así se podrían dar situaciones en que la altura asociada a una misma clase podría ser diferente dependiendo de la clase que se eligió para deshacer el empate.

Método flexible

Este algoritmo depende de un parámetro β que tiene que ser especificado por el usuario. El método utiliza la siguiente ecuación de modificación:

$$d(C_i \cup C_j, C_k) = \beta d(C_i, C_k) + \beta d(C_j, C_k) + (1 - 2\beta) d(C_i, C_j)$$

donde β es una constante estrictamente positiva.

En este método también se pueden dar contradicciones. Ejemplos de contradicciones para los tres últimos métodos se pueden encontrar en Kaufman y Rousseuw [87]. El comportamiento del método, está totalmente influenciado por el valor del parámetro β.

La mayoría de los algoritmos jerárquicos aglomerativos se caracterizan por ser computacionalmente no muy caros y por no necesitar mucha memoria. Las fórmulas de modificación citadas en la tabla 4.2 para hallar la altura a la cual se forma una nueva clase, hacen que los cálculos se simplifiquen de forma importante. Además, es posible guardar los datos que el algoritmo necesita según va progresando éste, en la propia matriz de disimilaridad de los datos, si es que éstos venían descritos de esta manera.

4.5.2 Algoritmos divisivos

Los algoritmos divisivos se caracterizan por construir el árbol de manera contraria a la utilizada por los aglomerativos. Comienzan con una sola clase en la que se encuentran todos los objetos del conjunto, y en cada paso del algoritmo dividen una clase en dos, hasta que cada clase consta de un sólo objeto.

Al igual que con los aglomerativos, es posible dar diferentes clasificaciones, en este caso, atendiéndonos a la forma de seleccionar la clase que va a ser dividida y la forma de dividir dicha clase.

Los algoritmos divisivos han sido mucho menos utilizados que los algoritmos aglomerativos debido a que, en principio, su complejidad computacional y su demanda de recursos informáticos es mucho mayor. Por ejemplo, en el primer paso de un algoritmo aglomerativo hay que comprobar que par de objetos se deben unir lo que hace un total de:

$$\left(\begin{array}{c} n \\ 2 \end{array} \right) = \frac{n(n-1)}{2}$$

comprobaciones. Sin embargo, en un primer paso de un algoritmo divisivo habría que mirar cuales son las posibles particiones del conjunto en dos, lo que hace un total de:

$$2^{n-1} - 1$$

comprobaciones. Claramente ésto es imposible realizarlo para conjuntos de un tamaño relativamente pequeño.

Por esta razón los algoritmos divisivos han recogido menos estudios e interés por parte de los investigadores. Daremos, sin embargo, algunos de los algoritmos utilizados, a la vez que algunas situaciones donde puede ser de mucho interés su utilización.

Dentro de los algoritmos existentes en la literatura, el primero y el más obvio consiste en inspeccionar todas las posibles particiones de una clase en dos. Este algoritmo es debido a Edwards y Cavalli-Sforza [51]. En él se supone que los datos vienen en forma de matriz de medidas con variables numéricas. La división de la clase se lleva a cabo de manera que las nuevas clases que se forman minimicen las sumas de las distancias al centroide de cada clase. Es decir, C es dividido en C_1 y C_2 de manera que se minimiza:

$$B(C_1) + B(C_2).$$

El método debido a Macnaughton-Smith y col. [110] es probablemente el más usado en problemas generales. En este método se utiliza el diámetro de la clase:

$$diam(C) = \max_{w,w^{'} \in C} d(w, w^{'})$$

para elegir la clase a dividir. Aquella clase con mayor diámetro será la elegida.

Para realizar la división se van pasando objetos de la clase a dividir a la nueva clase que se crea. La nueva clase comienza con el objeto más disimilar a todos los de la clase, y continúan uniéndose objetos siempre que su disimilaridad con los objetos que restan en la clase inicial sea mayor que la disimilaridad con la nueva clase. Para hallar que objeto es más disimilar a los de la clase, se utiliza, al igual que en los algoritmos aglomerativos, una distancia entre un objeto y una clase. Este método utiliza, la misma distancia entre un objeto y una clase que el método aglomerativo del mínimo. Con esta forma de elegir las clases que se dividen y de dividirlas, el algoritmo incurre en unos gastos computacionales, que nunca van más allá del doble utilizado por cualquier método aglomerativo.

Una de las críticas que se suelen objetar al método es la forma de tratar las clases. Mientras en la división hay una nueva clase que se crea a partir del primer objeto, la otra clase parece que es una simple colección con los objetos que quedan de la clase inicial. Una posibilidad de solucionar esto, Hubert [78], consiste en construir los dos grupos partiendo de los dos datos más alejados, para luego asignar el resto, en orden, al más cercano.

Donde si han sido utilizados estos métodos con bastante asiduidad es en la clasificación jerárquica donde los datos vienen descritos por variables binarias. El tipo de algoritmos desarrollados se basan en utilizar en cada paso del algoritmo una única variable para dividir cada clase. Estos algoritmos suelen denominarse *monotéticos*, frentes a los vistos hasta el momento que se llaman *politéticos* y que utilizan todas las variables.

Estos métodos proceden eligiendo la variable que se va a utilizar para dividir la clase e introduciendo en una de las nuevas clases todos los objetos que tienen un cero para esa variable, y en la otra los que tienen un uno. El proceso continúa eligiendo en cada una de las clases una nueva variable, la cual no tiene porque ser la misma en ambas clases, hasta que hay un solo elemento en cada clase, o los objetos de una clase no pueden ser más discriminados por las variables restantes.

Evidentemente los algoritmos se diferencian en el criterio que utilizan en cada paso, para decidir cual es la variable por la que se divide la clase.

La idea en general va a ser elegir la variable más central, es decir,

aquella cuya suma de similaridades con respecto a las demás variables sea lo más grande posible. Para realizar esto lo primero que hay que hacer es definir medidas de similaridad entre variables que toman valores 0-1.

		f	
		0	1
g	0	a_{00}	a_{01}
	1	a_{10}	a_{11}

Tabla 4.3: Tabla utilizada para hallar los índices de proximidad entre variables binarias

Si consideramos la tabla de contingencia 4.3, donde a_{00} está representando el número de objetos en los que ambas variables f y g toman el valor 0, a_{11} el número de objetos en los que ambas variables toman el valor 1 etc.

La medida más conocido utiliza como similaridad entre dos variables el siguiente valor:

$$K_{fg} = \frac{(a_{00} \times a_{11} - a_{10} \times a_{01})^2 n}{(a_{00} + a_{10})(a_{00} + a_{01})(a_{10} + a_{11})(a_{01} + a_{11})}.$$

Se sabe que para una tabla de contingencia el estadístico anterior tiene aproximadamente una distribución chi-cuadrado, y por lo tanto puede ser utilizado para medir hasta que punto dos variables son independientes.

El método, debido a Williams y Lambert [156], consiste en calcular para cada variable f el número:

$$K_f = \sum_{g \neq f} K_{fg}$$

siendo K_{fg} el valor de la medida de similaridad entre las variables f y g. El método elige la variable con mayor valor de K_f, es decir aquella variable que se encuentra más "centrada" en el conjunto de variables.

A la vista de los métodos de clasificación jerárquica, tanto aglomerativos como divisivos, se puede decir que todos tienen una carencia:

son métodos de optimización local. En cada paso del algoritmo se elige la mejor opción, lo que significa que la construcción del árbol nos va a llevar hacia, probablemente, un óptimo local para el criterio que se esté utilizando.

4.6 Clasificación piramidal

La clasificación piramidal, Bertrand y Daday[28] y Bertrand [20], es el tercer tipo de clasificación al que vamos a aplicar los AAGG. Es la clasificación más novedosa y al mismo tiempo la menos conocida. Este tipo de clasificación se diferencia de las anteriores principalmente en que, en vez de producir una partición del conjunto de datos, como la clasificación particional o jerárquica , produce un conjunto de recubrimientos. Es decir, la intersección de dos clases puede no ser vacía, pueden existir objetos del conjunto que pertenezcan a más de una clase. Esta aproximación a la CNS tiene la ventaja de ser más flexible, de manera que un objeto que se encuentre en dos clases a la vez puede compartir propiedades de los objetos de ambas clases.

Por otro lado, la clasificación piramidal, como más adelante se demostrará, puede verse como una generalización de la clasificación jerárquica. Mientras que en la clasificación jerárquica se construía una sucesión encajada de particiones, en este caso se construye una sucesión encajada de recubrimientos. Esta sucesión encajada de recubrimientos suele ser representada gráficamente mediante una estructura también en forma de jerarquía, que llamaremos *pirámide*.

En la figura 4.5 puede verse un ejemplo de una clasificación piramidal. Se observa que la estructura es bastante similar a los árboles construidos con la clasificación jerárquica. Cada hoja de la estructura contiene una clase formada por un solo objeto, y cada nodo interno contiene una clase. Si un nodo es padre de otro significa que la clase representada por el nodo padre contiene a la clase representada por el nodo hijo. Una partición de la pirámide como la dada en la figura 4.6, a un cierto nivel, produce un conjunto de recubrimientos del conjunto de objetos. En nuestro ejemplo las clases que se obtienen son:

$$C_1 = \{w_1, w_2, w_3\},\ C_2 = \{w_3, w_4, w_5\},\ C_3 = \{w_5\},\ C_4 = \{w_6\}.$$

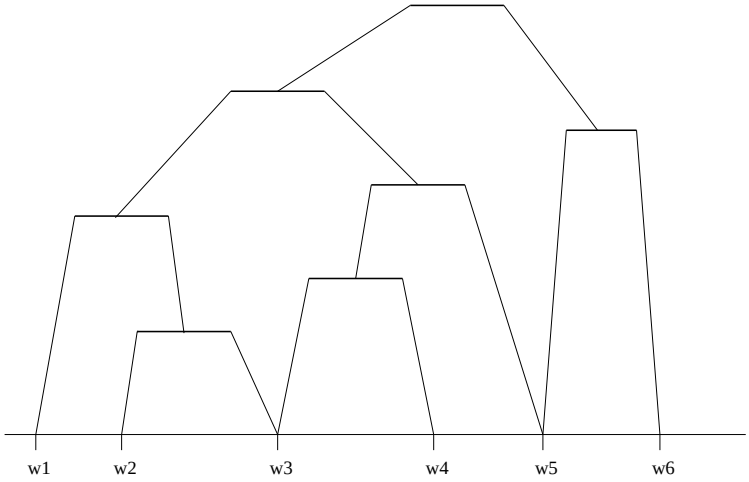

Figura 4.5: Ejemplo de una clasificación piramidal

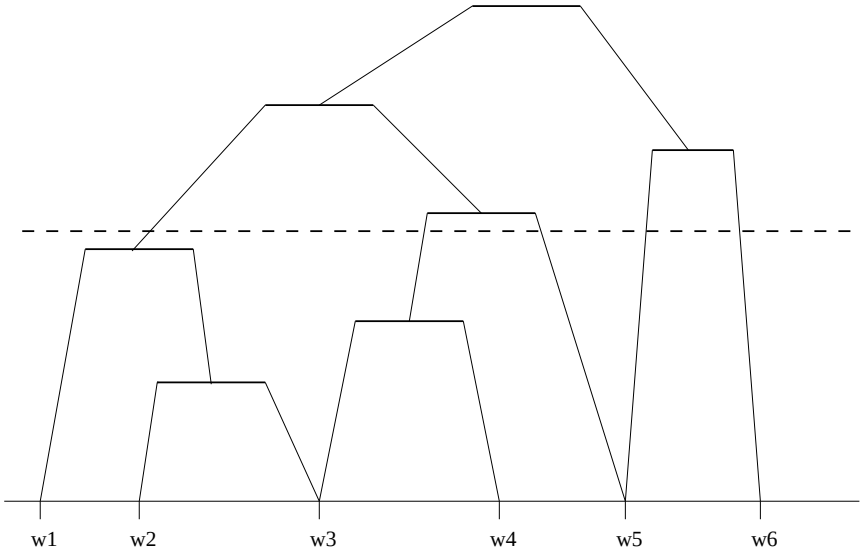

Figura 4.6: Corte de una clasificación piramidal

Fácilmente se observa que las clases que se obtienen son solapadas.

Pasaremos a continuación a definir de manera formal lo que es una

pirámide, para al mismo tiempo, dar los principales resultados teóricos que hemos encontrado en la literatura. Esta formalización y los resultados que enunciamos, los hemos tomado principalmente de Diday [49] y Capdevila y Arcas [31]. Terminaremos la sección viendo los algoritmos existentes para la construcción de pirámides.

La definición de una pirámide está basada en la compatibilidad entre una disimilaridad y un orden. La compatibilidad entre un orden y una disimilaridad puede definirse de muchas maneras diferentes Diday [48]. La que nosotros pensamos que es la más intuitiva y la usada en el contexto de las pirámides es la siguiente:

Definición 7 *Dado un conjunto Ω, una disimilaridad d y un orden θ sobre dicho conjunto, diremos que θ es compatible con d si:*

$$w\ \theta\ w^{'}\ \theta\ w^{''} \Rightarrow d(w, w^{''}) \geq \max\{d(w, w^{'}), d(w^{'}, w^{''})\}.$$

Intuitivamente esto significa que la distancia entre dos elementos ordenados siempre es mayor que la distancia entre éstos y cualquier elemento que se encuentre entre ambos de acuerdo con el orden.

Definición 8 *Sea Ω un conjunto de objetos y θ un orden definido en Ω. Un subconjunto $R \subset \Omega$ es conexo en relación con θ si:*

$$\omega \in R \Leftrightarrow \omega_{min}\ \theta\ \omega\ \theta\ \omega_{max}$$

donde w_{min} y w_{max} son respectivamente el elemento máximo y el elemento mínimo de R en relación a θ.

El significado que tiene la definición anterior es que un conjunto es conexo en relación al orden si dicho conjunto es un intervalo para dicho orden.

Definición 9 *Un orden θ es compatible con un subconjunto $P \subset \mathcal{P}(\Omega)$, de partes de Ω, si todo elemento de P es conexo en relación con θ.*

A partir de los dos conceptos anteriores daremos la definición de pirámide:

Definición 10 *Sea Ω un conjunto finito y $P \subset \mathcal{P}(\Omega)$ un subconjunto no vacío de partes de Ω. P es una pirámide si:*

i) $\Omega \in P$

ii) $\forall\ \omega \in \Omega\ \Rightarrow \{\omega\} \in P$

iii) $\forall\ R, R^{'} \in P$, *se tiene* $R \cap R^{'} = \emptyset$ *o* $R \cap R^{'} \in P$

iv) *Existe un orden θ compatible con P.*

Como puede verse hay grandes similitudes entre la definición de árbol dada en la sección anterior y la definición de piramide. La principal diferencia radica en el hecho de que la condición **iv)** de la definición de árbol, es decir, que la intersección de dos conjuntos sea el conjunto vacío o uno de ellos, queda relajada, así como la aparición del concepto de compatibilidad entre un orden y una y un subconjunto P de $\mathcal{P}(\Omega)$.

El siguiente resultado asegura que la estructura piramidal generaliza la estructura jerárquica:

Teorema 5 *El conjunto de árboles está incluido en el conjunto de pirámides.*

Definición 11 *Dados dos elementos R y $R^{'}$ de una pirámide P, con $R \subset R'$, diremos que $R^{'}$ es predecesor de R si no existe ningún elemento $R^{''}$ en la pirámide tal que $R \subset R^{''} \subset R^{'}$ estrictamente.*

La definición anterior nos permite enunciar el siguiente teorema que aporta mucha luz a la hora de describir la estructura de una pirámide. De la misma forma esta propiedad es básica de cara a dibujar la pirámide.

Teorema 6 *Todo elemento de una pirámide P tiene a lo sumo dos predecesores.*

Al igual que un dendograma estaba compuesto por un par (T, h) siendo T la estructura, es decir, un árbol y h una función definida en los elementos de T, definiremos la noción de pirámide indexada asociando un valor a cada elemento de la pirámide.

Definición 12 *Una pirámide indexada es un par (P, f) donde P es una pirámide y f es una función real definida en P y tal que:*

i) $f(R) = 0 \iff R$ *está compuesto por un solo objeto*

ii) $\forall\ R, R^{'} \in P$ *tal que* $R \subset R^{'}$ *se tiene que* $f(R) \leq f(R^{'})$.

Definición 13 *Una pirámide indexada (P, f) se dirá indexada en sentido amplio si $\forall R, R' \in P$ tal que $R \subset R^{'}$ estrictamente y $f(R) = f(R^{'})$ se tiene que existen $R_1, R_2 \in P$ distintos de R tal que $R = R_1 \cap R_2$.*

Definición 14 *Una pirámide indexada (P, f) se dirá indexada en sentido estricto si $\forall R, R' \in P$ tal que $R \subset R^{'}$ estrictamente implica que $f(R) < f(R^{'})$.*

Pasaremos a ver dos algoritmos que se han utilizado para construir pirámides indexadas. La forma que tienen estos métodos de construir la pirámide es muy similar a la de los algoritmos de clasificación jerárquica aglomerativos. Se comienza en una situación en la que cada elemento está en una clase y en cada paso del algoritmo se unen las dos clases más cercanas. La diferencia radica en que, mientras que en los algoritmos de clasificación jerárquica cada clase unicamente se unía una vez, en este caso cada clase puede unirse hasta dos veces.

El primer método lo hemos tomado de Diday [49]:

1. Inicialmente todo elemento de Ω está en una clase.

2. Se unen los dos grupos más cercanos que no han sido unidos dos veces anteriormente y que cumplen:

   i) la unión de los grupos es conexa,

   ii) si w y $w^{'}$ son los objetos extremos de una parte conexa de Ω asociada con una clase C, entonces ningún grupo que no contenga w o $w^{'}$ puede ser conectado a un grupo incluido en C.

3. Cada vez que dos clases son unidas se asocia un orden entre los elementos de las clases.

4. Volvemos a 2 hasta que se forma una clase conteniendo a Ω.

Evidentemente para poder aplicar este algoritmo se necesita precisar como medir cuales son las clases más cercanas. Esto se hace, al igual que en los métodos jerárquicos aglomerativos, mediante la definición de medidas o distancias entre clases. En este caso se pueden utilizar todas las distancias que fueron definidas para los métodos jerárquicos aglomerativos.

El otro algoritmo es debido a Capdevila [30] y es una modificación del anterior:

1. Inicialmente cada elemento de Ω está en una clase.

2. Se unen los dos grupos más cercanos que no han sido unidos dos veces anteriormente.

3. Cada vez que dos clases son unidas se asocia un orden entre los elementos de las clases.

4. Volvemos a 2 hasta que se forma una clase conteniendo a Ω.

En este proceso se tendrán en cuenta las siguientes condiciones:

i) Si $C, C^{'}$ y $C^{''}$ son tal que $C^{'} \subset C^{''}$ y $d(C, C^{'}) = d(C, C^{''})$ y esta distancia es la mínima entonces se une $C^{'}$ con C.

ii) Si definimos el índice de una clase C que se formó a partir de $C^{'}$ y $C^{''}$ como:

$$i(C) = \begin{cases} 0 & \text{si } C = \{w\} \text{ para algún } w \in \Omega \\ d(C^{'}, C^{''}) & \text{si } C = C^{'} \cup C^{''} \end{cases} .$$

entonces si en un cierto nivel existen dos clases $C, C^{'}$ tal que $C \subset C^{'}$ e $i(C) = i(C^{'})$ se elimina la clase C del recubrimiento en el paso k.

iii) Si C es una clase del recubrimiento en el paso $k-1$, y $C^{'}$ es la nueva clase formada cumpliéndose $C \subset C^{'}$ y C no contiene ninguno de los extremos de $C^{'}$ entonces C no puede ser unido más veces.

El autor de este último algoritmo propone una técnica por la cual puede eliminar grupos sobrantes en la pirámide (grupos que no aportan ninguna información), de forma que al final obtiene una pirámide indexada en sentido estricto.

Al igual que hicimos en las secciones anteriores conviene ver cuales son las carencias de estos métodos para así motivar el trabajo posterior.

De la misma forma que ocurría con la clasificación jerárquica, la construcción de la pirámide se realiza de forma local, eligiendo en cada paso la mejor opción. Esto nos conduce en la mayoría de los casos hacia un óptimo local no pudiéndose alcanzar la mejor estructura piramidal.

4.7 Validación

El objetivo de la validación es proporcionar un marco formal en el cual apoyarse para validar los resultados de una clasificación. Como vimos anteriormente, cada criterio de clasificación está imponiendo de manera implícita una estructura en el conjunto de datos o en las clases que se descubren. Esto puede llevar fácilmente a conclusiones erróneas: descubrir clases que no existen en realidad, partir verdaderas clases en dos, etc.

Las herramientas fundamentales a la hora de llevar a cabo una validación son tres: los test de hipótesis, las simulaciones de Monte Carlo y el "bootstrapping".

El primer análisis que suele llevarse a cabo en un conjunto de datos es realizar un test estadístico para conocer si los datos pueden estar distribuidos de forma aleatoria o no, es decir, ¿es posible que exista alguna estructura en el conjunto de datos o no? Una vez aplicado un tipo de clasificación vendría otro tipo de validaciones: validación de una jerarquía, validación de una partición, o validación de una clase en particular.

El objetivo que se persigue es ver hasta que punto, por ejemplo, una jerarquía se adecúa de forma inusual a los datos.

El proceso sería el siguiente:

1. Se establece una hipótesis nula que recoja nuestra idea de aleatoriedad teniendo en cuenta el tipo de dato.

2. Se selecciona un estadístico o un índice sensible a la estructura de los datos y se obtiene su distribución de acuerdo con la hipótesis nula anterior.

3. Se selecciona un umbral para el estadístico.

4. Se realiza el test.

Los tipos de criterios que se suelen tener en cuenta para llevar a cabo una validación son tres:

1. Criterio externo. Una forma de validar los resultados de una clasificación es compararlos con una estructura dada a priori para los mismos, por ejemplo, por un experto o por una clasificación dada de antemano.

2. Criterio interno. El resultado de una clasificación se valida utilizando los propios datos con los que se ha llevado a cabo la clasificación.

3. Criterio relativo. Dadas dos clasificaciones se intenta decidir cual de las dos estructuras es más adecuada a los datos en algún sentido.

Parte IV

Algoritmos Genéticos Aplicados a la Clasificación No Supervisada

Capítulo 5

Algoritmos Genéticos Aplicados a la CNS Particional

5.1 Introducción

En este capítulo se estudiará la aplicación de los AAGG al problema de la clasificación particional. Las razones de la aplicación de esta técnica de optimización combinatoria a dicho problema fueron expuestas, en mayor parte, en el capítulo anterior. Los algoritmos de clasificación particional clásicos son algoritmos de optimización local y además su resultado final depende de la partición inicial, además, en su mayoría necesitan de antemano el conocimiento del número de clases. A todo lo anterior hay que añadir la imposibilidad de inspeccionar todo el espacio de búsqueda debido a su enorme tamaño.

Los problemas enumerados anteriormente han motivado la búsqueda de otro tipo de técnicas que pudieran subsanar las limitaciones de los algoritmos clásicos. Dado que el problema se puede considerar como un problema de optimización en un espacio finito de gran tamaño, es decir, un problema de optimización combinatoria NP-completo, la utilización de las nuevas técnicas estocásticas de optimización provenientes de la IA parece totalmente justificada.

Además de los AAGG, se han aplicado al problema de la clasifi-

cación particional otras técnicas de optimización combinatoria novedosas como el EE (Klein y Dubes [88]), o la Búsqueda Tabú (Al-Sultan [3]). Sin embargo, en todas estas aplicaciones se supone al igual que en los algoritmos clásicos que el número de clases es conocido con antelación.

En esta memoria se propone la utilización de los AAGG como base para un algoritmo de clasificación que no sólo recupere las clases conocido su número, sino que además halle dicho número.

La organización del capítulo es como sigue: se comenzará haciendo un repaso exhaustivo de las aplicaciones previas de los AAGG al problema de la clasificación particional. En la siguiente sección se expondrá la aproximación novedosa propuesta en esta memoria, terminando con una sección dedicada a la selección de casos.

5.2 Revisión de aplicaciones de los AAGG a la clasificación particional

Daremos a continuación un resumen de las diferentes aproximaciones que se han encontrado en la literatura de la aplicación de los AAGG al problema de la clasificación particional. A la hora de realizar esta revisión, se han tenido en cuenta también artículos que no son exactamente de clasificación particional. No tratan de encontrar los posibles grupos subyacentes a un conjunto de datos, sino más bien, de dividir un conjunto de datos en un número de grupos dado, de manera que se optimice una determinada función. Es evidente que en muchas ocasiones, la forma de llevar a cabo la resolución del problema de la clasificación particional, se reduce al caso anterior.

La literatura sobre este tema es abundante y bastante variada. A pesar de ello, hay bastantes aspectos que se repiten de unos artículos a otros. Intentaremos aquí dar un amplio resumen que recoja todas las diferentes alternativas que se han planteado para este problema.

La primera cuestión que suele aparecer en el uso de los AAGG en cualquier problema es la codificación de las posibles soluciones. Es decir, como vamos a codificar cada posible solución de forma que estas representaciones puedan convertirse en individuos de nuestro AG.

Los problemas que se plantean a la hora de buscar una codificación en el ámbito de la clasificación particional vienen expuestos en Bhuyan y col. [25]. Existen dos posibilidades. Una de ellas consiste en tratar de buscar una codificación clásica de manera que cada individuo se codifique como un vector 0-1 y de esta forma aprovechar los operadores de cruce y mutación tradicionales, los cuales han sido profundamente estudiados y analizados. El problema de esta aproximación radica en que la codificación y decodificación de un individuo suele hacerse demasiado complicada.

La segunda opción para abordar el problema consiste en utilizar codificaciones no estándar, que se adecuen de manera simple a las soluciones del problema. La parte negativa de esta aproximación radica en la necesidad de diseñar operadores de cruce y mutación específicos, de forma que sean capaces de identificar trozos de los individuos asociados con una buena adaptación y puedan mezclarlos de manera conveniente. Como veremos a continuación, no hay una estrategia que se haya impuesto sobre la otra, los artículos se reparten casi por igual entre ambas estrategias.

Para aclarar los diferentes conceptos se considerará un pequeño ejemplo. Se supondrá que se dispone de un conjunto de seis elementos $\{A, B, C, D, E, F\}$ para los cuales vamos a intentar dar una clasificación en tres grupos. Consideraremos en todos los ejemplos la clasificación: $\{A\}, \{B, E\}, \{C, D, F\}$.

Comenzaremos dando un resumen de artículos más clásicos, la mayoría de los cuales contienen muchas similitudes, para dar más tarde un par de aproximaciones al problema totalmente diferentes, terminando esta sección con trabajos donde uno de los objetivos era también hallar el número de clases.

5.2.1 Aproximaciones clásicas

Dado que hay bastantes codificaciones que aparecen en más de un artículo, daremos un resumen de ellas inicialmente, para mas tarde pasar a comentar alguno de los artículos en particular.

Para describir la mayoría de las codificaciones supondremos que tenemos un problema en el que se trata de dividir un conjunto de n objetos en k grupos diferentes.

Codificaciones 0-1

Las codificaciones que han utilizado la primera opción de las anteriores vienen recogidas a continuación.

1) Bhuyan y col. [25] proponen una codificación basada en teoría de grafos. Suponen que cada clasificación puede representarse mediante un grafo, donde cada objeto del conjunto de datos está representado por un vértice del grafo, y entre dos vértices existe una arista si los objetos se encuentran en la misma clase. Para los autores un individuo de la clasificación es un vector 0-1 de tamaño $\frac{n(n-1)}{2}$ (número máximo de aristas en un grafo no dirigido con n vértices), con cada gen representando una arista del grafo. Si un gen tiene un valor de 1 significa que dicha arista está en el grafo, y sino lo contrario. Nuestra partición ejemplo podría ser representada por el siguiente individuo: (0 0 0 0 0 0 0 1 0 1 0 1 0 1 0). Una idea similar a la anterior es utilizada por Jones y Beltramo [84] pero los autores codifican tan solo las aristas del grafo que forman parte de la partición.

2) Alippi y Cucchiara [5] proponen una codificación 0-1 en la que cada individuo es una vector de tamaño $n * k$. Cada individuo está dividido en k genes. Si el j-ésimo bit del i-ésimo gen toma un valor de 1, esto significa que el j-ésimo objeto está en la i-ésima clase. Al mismo tiempo se tiene que dar que el j-ésimo bit debe estar a 0 en los demás genes. La misma codificación es utilizada por Bezdeck y col. [24], y por Kettaf y Asselin de Beauville [90], pero como veremos más adelante con objetivos muy diferentes. Nuestro ejemplo quedaría codificado como: (100000 010010 001101).

3) La última codificación 0-1 ha sido propuesta por Cucchiara [41]. La autora codifica cada partición mediante un vector de longitud $n*([\log_2 k]+1)$. El individuo está dividido en n genes cada uno de longitud $[\log_2 k]+1$, cada uno de los cuales representa la clase a la que pertenece dicho objeto. La partición ejemplo se representaría como: (00 01 10 10 01 10).

A pesar de los comentarios hechos anteriormente, y aunque en este caso, todas las codificaciones son 0-1, es imposible aplicar directamente

los operadores clásicos para codificaciones 0-1. La razón radica en el hecho de que su aplicación podría devolver clasificaciones con menor o mayor número de clases del deseado, o devolver individuos sin sentido. Para resolver estos problemas, los autores diseñan operadores de cruce y mutación específicos para su codificación y/o modifican los individuos provenientes de la aplicación de los operadores clásicos.

Codificaciones no 0-1

Las codificaciones no 0-1 que se han propuesto para el problema de la clasificación particional son las siguientes:

4) Bhuyan y col. [25] proponen una codificación donde cada individuo es un vector de longitud n y cada gen puede tomar un valor entre 1 y k. El valor que posee el individuo en el lugar i-ésimo representa la clase a la que pertenece el objeto i-ésimo. La partición ejemplo estaría representada mediante: (1 2 3 3 2 3). Esta misma codificación fué usada por Jones y Beltramo [83] en su segundo trabajo.

5) Otra codificación diferente fué utilizada por Bhuyan y col. [25]. Para los autores un individuo es una permutación de los números $\{1, 2, 3, \ldots, n\}$. En el artículo se da un significado de proximidad a la permutación. Si tenemos los números 2 3 5 en este orden, esto significa que el objeto 2 está más cercano del 3 que del 5, y por lo tanto 2 y 5 únicamente pueden estar en la misma clase si el objeto 3 lo está. Es evidente que dado un individuo es posible obtener de esta forma:

$$\left(\begin{array}{c} n-2 \\ k-1 \end{array} \right)$$

diferentes particiones. Los autores utilizan un algoritmo de programación dinámica para hallar la mejor partición que se puede obtener a partir de un individuo, y de esta forma evaluarlo. Nuestro ejemplo admitiría varias posibles codificaciones, una de ellas es (1 2 5 3 4 6). Al igual que con la codificación anterior, también esta fué utilizada por Jones y Beltramo [83].

6) Por último Jones y Beltramo [83] y Bhuyan y col. [25] proponen una codificación en la que cada individuo es una permutación de los números $\{1, 2, \ldots, n + k - 1\}$. Los números del 1 al n representan los objetos a clasificar y los números del $n + 1$ al $n+k-1$ representan separadores entre clases. De esta forma nuestra partición ejemplo se representaría por medio del individuo : (1 7 2 5 8 3 4 6), donde los números 7 y 8 hacen de separadores entre diferentes particiones. Uno de los problemas de esta codificación es que los separadores no pueden aparecer en la posición inicial o final del individuo o dos juntos, ya que en este caso, tendríamos una clasificación con un número de clases menor que k.

Al igual que en el caso anterior, codificaciones 0-1, los operadores clásicos no pueden ser usados con estas formas de representar los individuos.

5.2.2 Otras aproximaciones diferentes

Daremos en esta sección una pequeña revisión de dos artículos que difieren de los anteriores en aspectos importantes.

Lucasius y col. [107] proponen una aproximación totalmente diferente al problema de la clasificación particional. Para construir una partición, los autores, se basan en las ideas de Kaufman y Rousseeuw [87] y buscan los k medoides mejores. Por lo tanto, convierten el problema de clasificación particional en uno de buscar un subconjunto óptimo de medoides. Para Lucasius y col.[107], un individuo es un vector de longitud k. Cada gen toma un valor diferente de los números $\{1, 2, \ldots, n\}$ el cual representa el medoide elegido. Cada individuo representa por tanto un conjunto de medoides. Al igual que en el algoritmo visto en la sección 4.4.2 y propuesto por Kaufman y Rousseeuw [87] el criterio de clasificación utiliza la distancia L_1 en vez de la Euclidea.

Por su parte, Babu y Murty [11] en lugar de utilizar AAGG emplean estrategias evolutivas. Para ellos un individuo es un vector de longitud $k * p$ siendo p la dimensión de los puntos. El individuo está dividido en k genes cada uno de los cuales representa el centroide de una clase.

5.2.3 Hallando el número de clases con AAGG

Terminamos esta sección revisando cuatro trabajos en los que se han utilizado los AAGG para hallar el número de clases a la vez que la configuración de las mismas.

El primer trabajo es debido a Lunchian y col. [109]. Los autores utilizan un tipo de AAGG con codificación real. La aproximación que proponen es únicamente para datos de $I\!R^2$. Para Lunchian y col. [109], un individuo de su población es un vector de longitud variable, donde cada gen es un par de números reales representando un centroide. El principal problema de este tipo de aproximaciones, esto es, la definición de la función objetivo, lo resuelven utilizando el teorema de Huygens (dado en la sección 4.4.2). Si k representa el número de clases su función objetivo es:

$$(B/T)^k$$

y lo que pretenden con el AG es minimizar dicha función. Los autores llevaron a cabo varios experimentos con datos generados a partir de una distribución normal bivariante, y con distintos números de clases. Aunque los resultados son buenos, como los propios autores aseveran, la función objetivo necesita de un estudio más profundo.

El segundo artículo que trata el problema de hallar el número de clases utilizando AAGG tiene como autores a Moraczewski y col. [122]. El trabajo realizado por los autores difiere de los anteriores, por un lado, en que parten de la matriz de proximidad en vez de hacerlo a partir de una matriz de medidas. Al igual que Kaufman y Rousseeuw [87] y Lucasius y col. [107] los autores tratan de seleccionar los k medoides mejores. De esta forma sus individuos son vectores 0-1 de tamaño n, donde un 1 en el gen i significa que dicho objeto es elegido como medoide. La otra cuestión en la que difiere este trabajo de los anteriores y precisamente lo más novedoso de su aproximación, es la función objetivo que utilizan. Esta función consiste en comparar dos matrices, mediante la siguiente fórmula:

$$s(D,\Delta) = \sqrt{\sum_{1\leq i<j\leq n} \frac{d_{ij}-\delta_{ij}}{d_{ij}^2}}.$$

La matriz $D = [d_{ij}]_{i,j=1,2,\ldots,n}$ es la matriz de disimilaridad inicial de los datos, y la otra matriz $\Delta = (\delta_{ij})_{i,j=1,2,\ldots,n}$ es construida a partir de una

clasificación producida por un individuo de la siguiente forma:

$$\delta_{ij} = \begin{cases} d_C & \text{si los objetos } w_i \text{ y } w_j \text{ pertenecen a la misma clase } C \\ d_{ext} & \text{en otro caso} \end{cases}$$

donde

$$d_C = \frac{1}{|C|} \sum_{w_i, w_j \in C} d(w_i, w_j)$$

es la distancia media de los puntos en la clase C y

$$d_{ext} = \frac{1}{|C||C'|} \sum_{w \in C} \sum_{w' \in C'} d(w, w')$$

es la distancia media entre puntos en dos clases C y C' diferentes.

Uno de los trabajos mas novedosos es el realizado por Kettaf y Asselin de Beauville [90]. Los autores utilizan la segunda codificación expuesta en la sección 5.2.1 pero con individuos de longitud variable. Como función objetivo proponen el criterio de Davis-Boulding (Jain y Dubes [80]).

Un trabajo reciente en clasificación particional con algoritmos evolutivos, es debido a Sarkar y col. [141]. En este trabajo, al igual que en el trabajo anterior, se intenta minimizar el criterio de Davis-Boulding, pero al mismo tiempo tratan de minimizar la función B. En lugar de utilizar AAGG los autores proponen utilizar estrategias evolutivas.

5.3 Nuestra propuesta de utilización de los AAGG en clasificación particional

A continuación se recoge el trabajo original realizado por el autor de esta memoria en la aplicación de los AAGG al problema de la clasificación particional. Este trabajo está recogido en Lozano y col. [103] y Lozano y Larrañaga [106].

Antes de comenzar a explicar la aproximación al problema, es importante reseñar que nuestro algoritmo sólo es válido para conjuntos de datos que estén descritos por medio de una matriz de medidas.

5.3.1 Formulación del problema como un problema de optimización combinatoria

El primer paso que se va a seguir en la resolución del problema de la clasificación particional consiste en intentar formular el problema como un problema de optimización combinatoria, para de esta forma poder aplicar los AAGG.

Se trata de buscar una función F de tal forma que si denotamos por $\Omega = \{w_1, w_2, \ldots, w_n\}$ un conjunto de datos p dimensionales y $\mathcal{P}_k(\Omega)$ el conjunto de las particiones de Ω en k clases, entonces la función:

$$F : \mathcal{P}_k(\Omega) \rightarrow I\!R^+$$

debe de alcanzar el óptimo cuando las clases y su número son las más "naturales". En todo momento entiendo por más naturales aquellas que el ojo humano elegiría.

La definición de una buena función, dejando a parte el algoritmo, es la cuestión más importante y complicada en un problema de clasificación particional. La función dada en esta memoria, y que pasaremos a exponer a continuación, a la vez de ser muy intuitiva, está en algún sentido basada en ideas de Rasson y Kubushishi [131].

En un problema de clasificación la mayoría de las clases se pueden caracterizar por no poseer dentro de ellas grandes espacios vacíos. Es decir, por no poseer grandes espacios sin puntos. Por lo tanto, siguiendo esta idea, lo que se propone en esta memoria es un criterio de clasificación que de alguna forma mida el espacio vacío, (sin puntos) dentro de las clases de una clasificación.

Para llevar a cabo esta medición del espacio vacío dentro de cada clase, vamos a dividir el espacio ocupado por el conjunto de puntos en hipercubos. En cada hipercubo comprobaremos si este posee puntos dentro de él o no. Si un hipercubo posee al menos un punto dentro de él le asignaremos un valor de 0, en caso contrario se le asignará un valor de 1. La figura 5.1 muestra un ejemplo de lo anterior en $I\!R^2$ con los datos de Ruspini.

A la hora de calcular el valor de la función objetivo para una clasificación $\{C_1, C_2, \ldots, C_k\}$ concreta hacemos lo siguiente. Para cada clase C, hallamos su envolvente convexa $H(C)$, esto es, el mínimo conjunto convexo que la contiene. Una vez hallada la envolvente convexa se

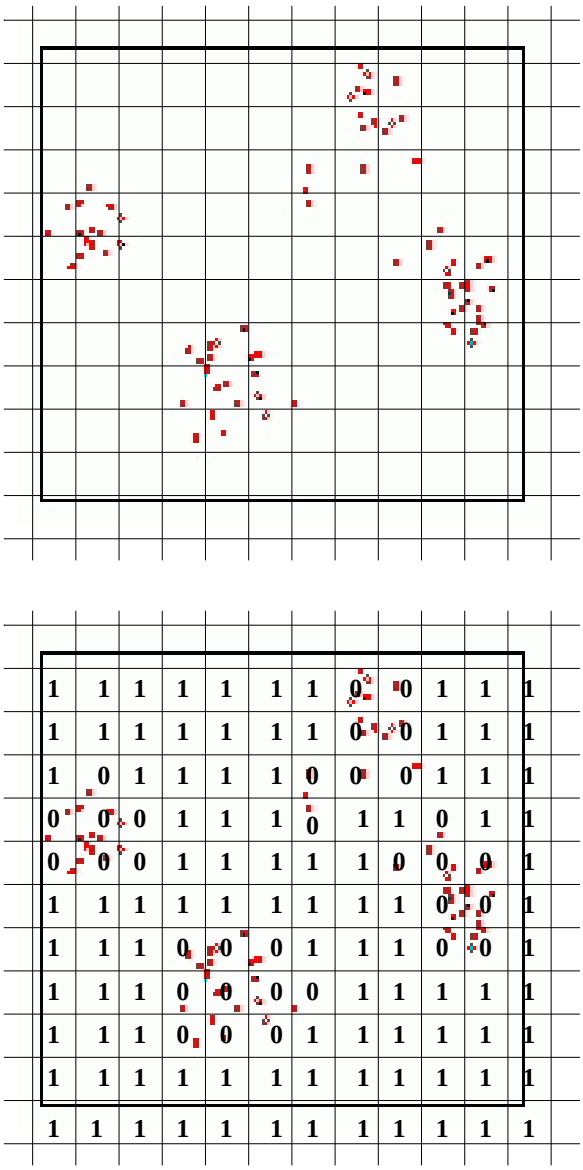

Figura 5.1: Valor dado a los hipercubos en $I\!R^2$ con los datos Ruspini

comprueba que centros de hipercubos se encuentran dentro de dicha envolvente. Se suma el valor asociado a cada hipercubo cuyo centro se encuentra dentro de la envolvente convexa de dicha clase, y este será el valor asociado a la misma. Para calcular finalmente el valor asociado a una clasificación concreta basta sumar el valor asociado a cada clase.

Matemáticamente lo anterior puede escribirse como sigue:

$$F^*(\{C_1, C_2, \ldots, C_k\}) = \sum_{i=1}^{k} \sum_{\mathbf{x} \in H(C_i) \cap CT} V(\mathbf{x})$$

(se utiliza F^* ya que la función no es definitiva), donde CT es el conjunto de centros de hipercubos, $H(C_i)$ es la envolvente convexa de los puntos que forman C_i y por último $V(\mathbf{x})$ es el valor asociado al hipercubo con centro en $\mathbf{x}$.

Sin embargo, esta función aun tiene un problema. Supongamos que se está trabajando con un conjunto de datos que únicamente posee una clase y además el valor óptimo de la función objetivo fuese 0 (no existe en la envolvente convexa ningún hipercubo vacío). Es muy fácil encontrar divisiones de dicha clase en dos o tres nuevas clases, de manera que ninguna de las nuevas envolventes convexas posean ningún centro de hipercubo vacío, y por lo tanto la función también asignaría a esta partición el valor de 0.

Para evitar esta situación lo que se propone es penalizar a la función objetivo con un valor que depende del número de clases. De esta manera el criterio de clasificación se puede expresar como:

$$F(\{C_1, C_2, \ldots, C_k\}) = F^*(\{C_1, C_2, \ldots, C_k\}) + \alpha * k.$$

En relación al valor de α, la única restricción que debe cumplir es $\alpha < 1$. La razón es que si $\alpha \geq 1$ la función podría asignar un valor mayor a una partición que tuviese un hipercubo vacío dentro, que a otra con una clase más pero sin hipercubos vacíos dentro de ellas, siendo ésta última la más natural.

Por último, hay una cuestión que está aún sin resolver y que es la clave de la aproximación presentada en esta memoria, esto es: ¿cúal es el tamaño de los hipercubos? Para resolver este problema se ha utilizado un método que aunque simple y muy intuitivo, da unos resultados, como veremos más adelante, bastante buenos.

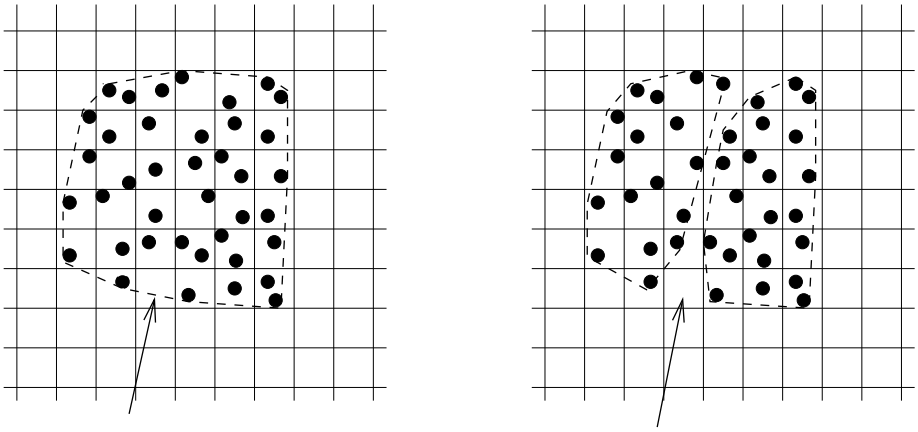

Figura 5.2: Justificación del tamño del hipercubo t

Supongamos que tenemos un conjunto de datos del cual no tenemos ninguna información. Si dicho conjunto de datos tuviese una única clase y nosotros quisiéramos descubrirla, tendría que ocurrir que su envolvente convexa no contuviese ningún hipercubo vacío. De no ocurrir esto, podría darse el caso de poder dividir el conjunto de datos en dos o tres clases de manera que no contuvieran ningún hipercubo vacío y por lo tanto obtuviesen menor valor de función objetivo. En la figura 5.2 se puede ver una situación en la que se produce lo anterior. El cuadrado marcado con una flecha tiene un valor de 1. La partición en una única clase tiene entonces un valor de $1 + \alpha * 1$, mientras la partición en dos clases no posee ningún cuadrado vacío y el valor que toma la función objetivo en ella es $0 + \alpha * 2$.

Como no se posee ninguna información acerca de los datos se va a suponer que han sido generados aleatoriamente muestreando una distribución uniforme. El tamaño del hipercubo se tomará, para que se cumpla que la probabilidad de encontrar un hipercubo vacío en el caso anterior, sea menor que cierto valor pequeño. Tomaremos en este caso 0.001 como cota superior de dicha probabilidad. De esta forma nos queda:

$$\left(1 - \frac{t^p}{S}\right)^n \leq 0.001$$

donde t es el tamaño del lado del hipercubo, S es el volumen de la envolvente convexa formada por el conjunto de datos, n es el número de puntos y p la dimensión de los mismos. En todos los experimentos se utilizó el valor de t más pequeño que cumplía la desigualdad anterior.

5.3.2 La codificación

En nuestro algoritmo cada individuo de la población es una permutación de los números del 1 al n (tamaño del conjunto de datos). Cada individuo es posible decodificarlo para cualquier número de clases. El proceso de decodificación llevado a cabo en los experimentos se realizó únicamente para un número de clases $k = 1, 2 \ldots, 15$. Se podría haber realizado una búsqueda para un número de clases mayor, pero en la mayoría de los problemas prácticos no tiene sentido hacerlo. Para cada k se obtiene un valor para la función de evaluación. El valor asociado al individuo será el menor valor de los quince. Desde este punto de vista nuestro AG puede verse como un algoritmo híbrido, en el que para decodificar un individuo se lleva a cabo una optimización. Dado un individuo I su valor será:

$$F(I) = \min_{k=1,2,\ldots,15} F(\{C_1, C_2, \ldots, C_k\})$$

Para decodificar el individuo, dado un número de clases concreto k, se toman los primeros k objetos de la permutación como centros de clases, y los demás objetos de la permutación son asignados en orden, a la clase cuyo centroide sea el más cercano. Una vez que un objeto ha sido asignado a una clase el centroide de la clase se modifica.

Para verlo de forma más clara, vamos a suponer que tenemos que decodificar el siguiente individuo (2 1 3 4 6 5) para dos clases. Los objetos que representa cada gen están representados en la Figura 5.3. Como primer elemento de la primera clase se toma el objeto 2, por lo tanto $C_1 = \{2\}$, y como primer elemento de la segunda clase el objeto 1, es decir, $C_2 = \{1\}$. Ahora el siguiente objeto, en nuestro caso el objeto 3, es asignado a la clase que tiene el centroide más cercano a él. Como ambas clases están formadas por un único objeto, dicho objeto es el propio centroide de la clase. Claramente $d(3,1) < d(3,2)$ por lo que el objeto 3 será asignado a la clase 1, siendo ahora ésta

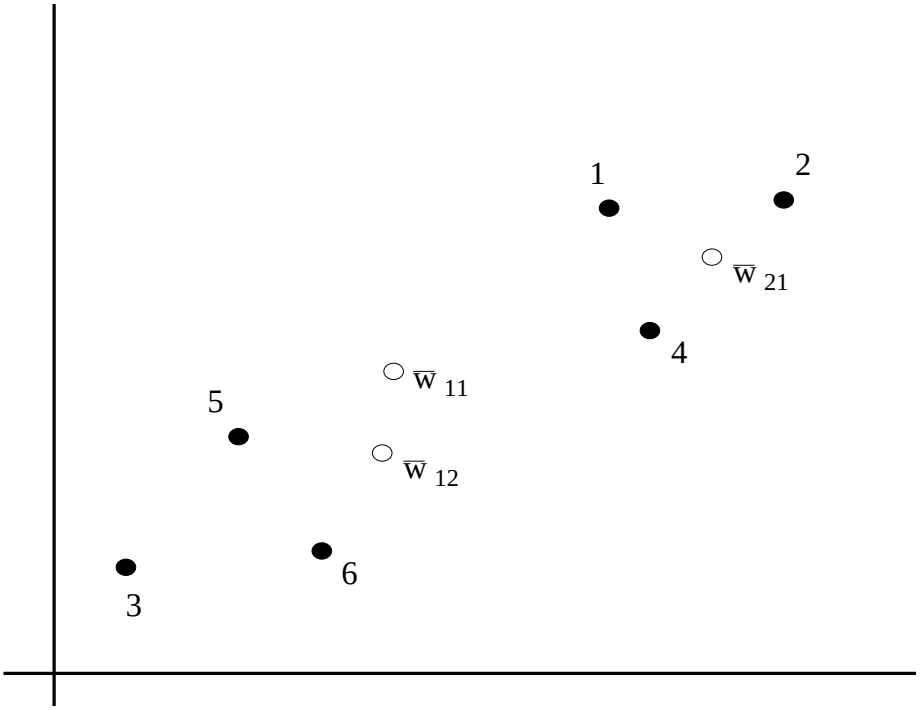

Figura 5.3: Ejemplo de decodificación de un individuo

$C_1 = \{1, 3\}$, y el nuevo centroide pasa a ser $\overline{w}_{11}$. Ahora se procesa el punto 4, claramente este objeto está más cerca del centroide de la segunda clase que del centroide de la primera, por lo tanto es asignado a la segunda, pasando ésta a ser $C_2 = \{1, 4\}$ y su centroide $\overline{w}_{21}$. El punto 6 es asignado claramente a la clase C_1 siendo el nuevo centroide $\overline{w}_{12}$, y finalmente el objeto 5 es asignado nuevamente a la clase C_1. Al final las clases que se obtienen son: $C_1 = \{1, 3, 5, 6\}$ y $C_2 = \{2, 4\}$.

5.3.3 Aplicación del algoritmo de clasificación en problemas en $\mathbb{R}^2$

Como es bien sabido, no existe un algoritmo de clasificación que se adecue de forma precisa a cualquier tipo de datos y de clases. Existen algoritmos que descubren fácilmente clases alargadas o lineales, por

ejemplo el algoritmo de clasificación jerárquica aglomerativo que utiliza la disimilaridad del mínimo, mientras que otros descubren de forma más conveniente clases en forma de hiperesfera. Para saber a que tipo de clases se adecua de forma conveniente nuestro algoritmo, se han llevado a cabo unos experimentos iniciales con conjuntos de datos en $I\!R^2$, para más tarde, estudiados los resultados obtenidos en dichos conjuntos pasar a generalizar el algoritmo para poder resolver situaciones en $I\!R^p$ con $p > 2$.

Los conjuntos con los que hemos llevado a cabo los experimentos iniciales son 6: el primero es el conjunto de datos de Ruspini, los otros cinco son conjuntos de datos generados de forma artificial. La tabla 5.1 recoge las caracterísitisticas generales de los conjuntos generados de forma artificial. La figura 5.4 recoge una representación de cada uno

datos	puntos	clases	distribución subyacente
1	1000	1	uniforme
2	400	1	uniforme
3	800	3	uniforme
4	1000	5	uniforme
5	1000	5	normal

Tabla 5.1: Características de los conjuntos de datos artificiales

de los conjuntos.

El algoritmo y los operadores utilizados para realizar los experimentos son los expuestos en el capítulo dedicado a los AAGG. Los parámetros concretos son los siguientes:

$$\lambda = 20, \quad p_m = 0.01, \quad p_c = 1.$$

Por otro lado el algoritmo finalizaba si el valor medio de la población no se modificaba en 20 generaciones. Los valores de los parámetros fueron establecidos cuando se llevaron a cabo los experimentos con el conjunto de datos de Ruspini y el primer conjunto de datos. Aunque aparentemente el tamaño de la población pueda parecer muy pequeño, este valor es válido en la mayoría de las ocasiones. Cuando analicemos los experimentos veremos que sólo en uno de los conjuntos un mayor

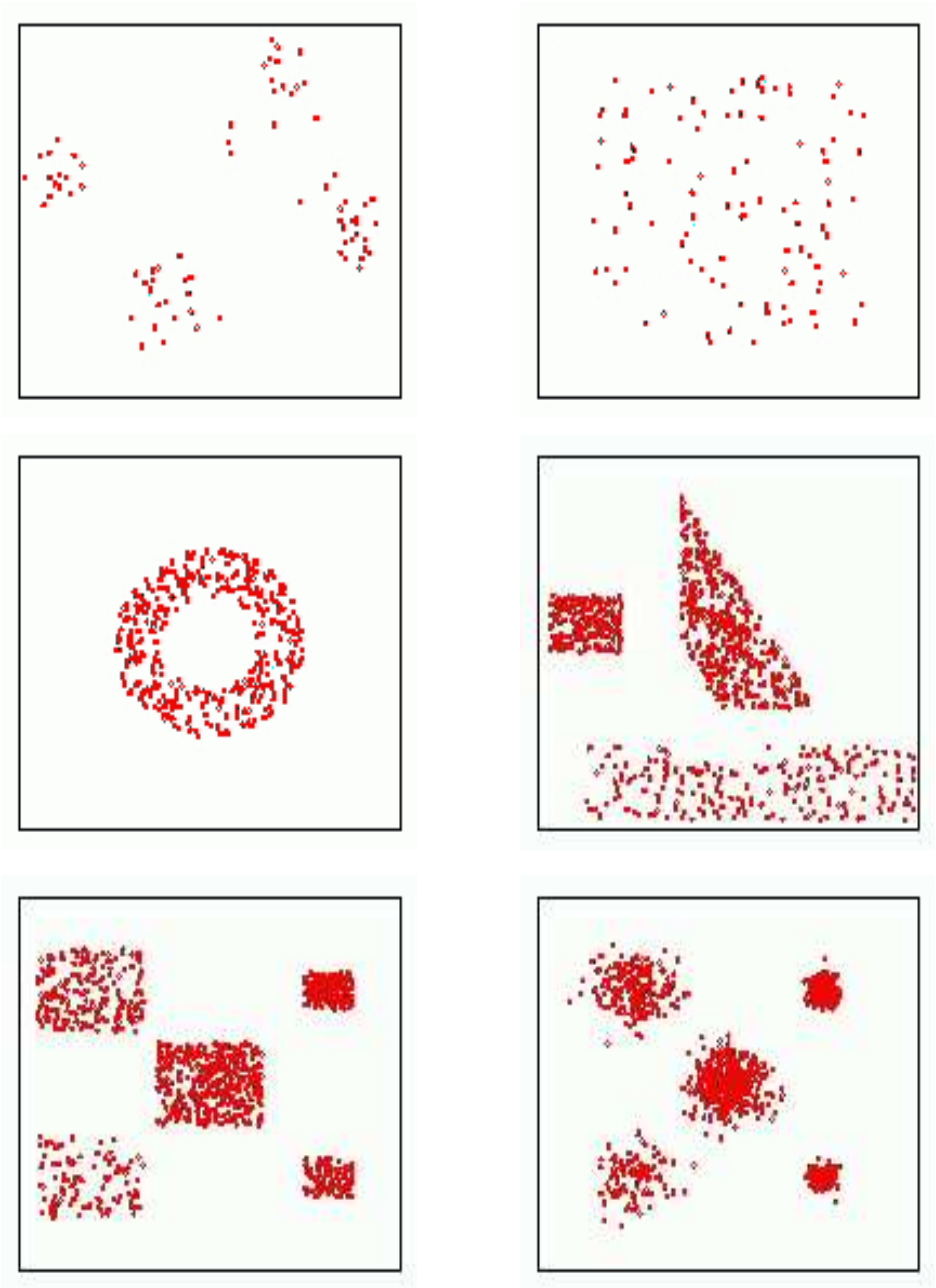

Figura 5.4: Representaciones gráficas de los conjuntos bidimensionales con los que se han llevado a cabo los experimentos

valor en el tamaño de la población y el criterio de parada hubiera posiblemente mejorado de forma sensible los resultados finales.

Para cada conjunto de datos y cada par de operadores de cruce y mutación, se llevaron a cabo 10 experimentos, es decir, en cada conjunto se hicieron 60 experimentos con cada operador de cruce y 40 con cada operador de mutación, lo que da un total de 240 experimentos por conjunto.

Antes de pasar a describir y comentar los resultados, conviene ver cual es el tamaño de los espacios de búsqueda en cada caso, para de esta forma valorar en mejor medida el trabajo realizado por el AG. Los tamaños de los espacios de búsqueda para los conjuntos Ruspini, conjunto 2, conjunto 3 y conjuntos 1, 4 y 5 son respectivamente:

$$1.178 \times 10^{80}, \quad 2.755 \times 10^{393}, \quad 2.755 \times 10^{793}, \quad 2.755 \times 10^{993}$$

La tabla 5.2 recoge los resultados conseguidos. En élla se puede ver el número de veces que se ha obtenido cierto número de clases para cada operador y cada conjunto. Por ejemplo, con el operador CX en el conjunto de datos 3 se obtuvieron 4 clases en 4 ejecuciones y 5 clases en las restantes 56 ejecuciones no obteniendo en ninguna ejecución 6 clases.

	Ruspini		Dat. 1	Dat. 2	Datos 3			Datos 4			Datos 5				
Clases	4	5	1	4	4	5	6	5	6	7	5	6	7	8	9
CX	60	0	60	60	4	56	0	59	1	0	0	56	4	0	0
AP	60	0	60	60	4	56	0	55	5	0	3	36	21	0	0
OX1	60	0	60	60	4	56	0	59	1	0	5	41	14	0	0
PMX	59	1	60	60	4	54	2	46	11	3	8	31	17	3	1
SM	40	0	40	40	2	37	1	36	3	1	4	30	5	0	1
SIM	39	1	40	40	5	34	1	36	4	0	0	27	13	0	0
ISM	40	0	40	40	0	40	0	35	4	1	1	31	8	0	0
IVM	40	0	40	40	5	35	0	36	4	0	3	27	10	0	0
EM	40	0	40	40	1	39	0	38	2	0	4	23	12	1	0
DM	40	0	40	40	3	45	0	38	1	1	4	26	8	2	0

Tabla 5.2: Resultados de los experimentos

Para evaluar la calidad de cada uno de los operadores a la hora de llevar a cabo la tarea de optimización en clasificación particional hemos realizado un test de Kruskal-Wallis. Este test se lleva a cabo en relación

con el número de evaluaciones necesarias para alcanzar la convergencia y con el número de clases obtenido.

Los resultados de la aplicación del test así como el análisis de los resultados obtenidos de cara a la clasificación son los siguientes:

Ruspini. El óptimo es alcanzado por cualquier operador sin ningún problema. En relación con los test estadísticos se vio que había una diferencia significativa entre los operadores de cruce en relación al número de evaluaciones siendo el operador PMX el más rápido (84.6 ejecuciones de media frente a las 154.5 del peor OX1).

Datos 1. Nuevamente este es un problema simple para nuestro planteamiento. En todas las ocasiones se consiguió el número óptimo de clases. El operador PMX continúa siendo el más rápido.

Datos 2. Es un problema muy difícil de resolver con nuestro planteamiento. Hay una forma de hacerlo, elegir t de manera que sólo exista un único cuadrado dentro del círculo y tomar el parámetro α mayor que 0.5. De esta manera se puede alcanzar el óptimo, sin embargo, evidentemente es una forma muy artificial de hacerlo.

Datos 3. Los resultados no son aparentemente muy buenos, sin embargo, esto es debido al tamaño de los parámetros utilizados en el AAGG. Es posible ver que en este problema la función objetivo alcanza el óptimo cuando se tienen las clases más naturales. La forma de decodificar el individuo tampoco ayuda en este caso a obtener el resultado deseado. Además de los experimentos que se plasman en la tabla anterior hemos llevado a cabo más experimentos con este conjunto, donde el óptimo fué alcanzado. En cuanto a los operadores se tiene nuevamente que el PMX es el más rápido siendo el más lento el CX.

Datos 4 Este es el primer conjunto de datos en el que se encuentran diferencias significativas entre los operadores en relación con los resultados. El operador CX es el mejor, siendo capaz de encontrar el número de clases óptimo más veces que los demás operadores. No se registró ninguna variación en relación con la velocidad de lo operadores.

Datos 5 Nuevamente los resultados no parecen muy buenos. Sin embargo, una pequeña variación en el valor de t nos hubiese conducido hasta el óptimo en casi todas las ejecuciones. El mejor operador en relación con el número de clases alcanzado es de nuevo el CX. Tampoco hubo variaciones en relación con la rapidez, siendo el PMX (119.7 ejecuciones) el mejor, y el OX1 el peor (227.1).

A la vista de los resultados anteriores, se pueden deducir varias conclusiones. Primero, la forma de enfocar el problema hace que el algoritmo no tenga mayores problemas en descubrir clases en las que los puntos estén distribuidos más o menos de manera uniforme. En el caso en el que los puntos estén distribuidos mediante una distribución normal parece interesante utilizar un valor de t algo más grande. En ningún caso puede utilizarse nuestro algoritmo para descubrir clases parecidas a las del conjunto de datos 2.

5.3.4 Aplicación del algoritmo de clasificación en problemas en $\mathbb{R}^p$

A la vista de los resultados obtenidos en la sección anterior, se intentó generalizar el algoritmo para problemas en $\mathbb{R}^p$. El principal problema en esta generalización consistía en encontrar un algoritmo que hallase la envolvente convexa de n puntos en $\mathbb{R}^p$. Mientras que realizar esto en $\mathbb{R}^2$ es relativamente sencillo, llevarlo a cabo en una dimensión cualquiera es bastante complicado. De hecho, este problema consiste en sí mismo una línea de investigación actual.

Afortunadamente encontramos en la bibliografía los resultados suficientes como para resolver esta cuestión. Barber y col. [19] han desarrollado un algoritmo denominado "quickhull" que es capaz de hallar, en tiempos no prohibitivos, envolventes convexas de conjuntos p-dimensionales.

En este caso, los conjuntos con los que hemos llevado a cabo experimentos son tres. Uno de ellos es el conocido conjunto de datos Iris, y los otros dos han sido generados de forma artificial. La tabla 5.3 recoge las características de los conjuntos.

Datos	puntos	dimensión	distribución subyacente	clases
1	1000	5	uniforme	3
2	600	6	normal	4
Iris	150	4	——	3

Tabla 5.3: Características de los conjuntos de datos en dimensión $p > 2$

Para realizar estos experimentos se tuvo en cuenta el mismo algoritmo y los mismos parámetros que en el caso anterior. Nuevamente se realizaron 240 ejecuciones con cada conjunto de datos, 60 con cada operador de cruce y 40 con cada operador de mutación, es decir, 10 experimentos con cada combinación de operador de cruce y operador de mutación.

Hay que tener en cuenta también que en todos los conjuntos únicamente buscamos como mucho hasta 15 clases.

En este caso para evaluar los resultados no sólo se tuvo en cuenta el número de clases que conseguía el algoritmo, sino que además recogimos el número de objetos que eran asignados a la clase correcta. Esto es debido a que el algoritmo no sólo es capaz de hallar el número de clases en un conjunto de datos sino que además, realiza al mismo tiempo la tarea de asignar cada objeto a una clase. Esto último tiene la ventaja de convertir al algoritmo en un algoritmo muy versátil, ya que incluso podría utilizarse para hallar las clases conocido su número.

La tabla 5.4, recoge los resultados de la aplicación del algoritmo a los conjuntos de datos anteriores.

Para interpretar la tabla anterior basta poner un ejemplo. Con el operador OX1 y el conjunto de datos Iris se alcanzaron tres clases en 24 ejecuciones, mientras que en las otras ejecuciones se alcanzaron 4 clases. El porcentaje medio de bien clasificados fué del 89.61%. Para calcular este porcentaje de bien clasificados en el caso de obtener más clases de las adecuadas se entendían por mal clasificados aquellos objetos que se encontraban en las clases más pequeña.

Como puede verse el algoritmo alcanzó resultados bastante buenos y únicamente tiene problemas con el conjunto de datos Iris. Para evaluar la validez de los distintos operadores, aplicamos nuevamente un test de Kruskal-Wallis ($\alpha = 0.05$) en relación con el número de clases

	Conjunto 1		Conjunto 2		Conjunto Iris		
	n. clases	% bien clas.	n. clases	% bien clas.	num. clases		% bien clas.
Clases	3		4		3	4	
CX	60	99.97	60	99.59	25	35	89.85
AP	60	99.93	60	99.61	17	43	88.82
OX1	60	99.96	60	99.51	24	36	89.61
PMX	60	99.98	60	99.66	29	31	89.75
SM	40	99.95	40	99.71	17	23	89.66
SIM	39	99.97	40	99.52	16	24	89.31
ISM	40	99.95	40	99.49	18	22	89.90
IVM	40	99.96	40	99.55	16	24	89.81
EM	40	99.95	40	99.69	15	25	89.28
DM	40	99.98	40	99.62	13	27	89.08

Tabla 5.4: Resultado de los experimentos en $I\!R^p$

devueltas por el algoritmo, el número de bien clasificados y el número de ejecuciones necesarios para alcanzar la convergencia. Las diferencias no fueron estadísticamente significativas en relación con los dos primeros aspectos, pero si en relación con la velocidad de convergencia. El operador PMX es al igual que en los experimentos primeros, el más rápido.

Aunque pueda parecer que los resultados obtenidos por el algoritmo en los datos Iris son muy malos en relación con el número de clases descubierto, hay una manera de explicarlo. Analizando detenidamente los resultados al completo era posible ver que aquellas ejecuciones que habían conseguido tres clases necesitaban mucho más tiempo para alcanzar la convergencia. Esto motivó el siguiente estudio.

Realizamos un nuevo test de hipótesis de Kruskal-Wallis para descubrir si había relación entre el número de clases alcanzado por el algoritmo y el número de evaluaciones. El resultado es que hay una diferencia estadísticamente significativa entre el número de evaluaciones cuando el algoritmo descubrió tres clases en comparación a cuando éste descubrió cuatro clases. Esto sugiere que si se hubiese aumentado el tamaño de población y el criterio de parada (probablemente esto último hubiese sido suficiente), se hubiese encontrado tres clases en casi todas las ejecuciones. Esto podrá verse más tarde.

A pesar de que los resultados obtenidos son, en general, bastante buenos, el algoritmo es, desde un punto de vista computacional muy

caro. Este coste computacional es debido principalmente al cálculo de la envolvente convexa. Este cálculo depende evidentemente de la cantidad de puntos y de la dimensión de los mismos. Por ejemplo, para hallar la envolvente convexa del segundo de los conjuntos de datos (600 puntos en dimensión 6) el algoritmo necesitó 20.97 segundos, mientras que para hallar la envolvente convexa de un 15 % de los puntos necesitó únicamente 2.6 segundos. Por lo tanto, si fuese posible trabajar con un subconjunto del conjunto de datos de manera que dicho subconjunto conservase las propiedades estructurales del primer conjunto, el ahorro en tiempo computacional sería muy grande. Este será el siguiente paso que vamos a llevar a cabo.

5.4 Selección de casos

5.4.1 Trabajo previo con AAGG en selección de casos

Dado que vamos a intentar realizar una selección de casos en los conjuntos con los que hemos llevado a cabo los experimentos anteriores, antes de nada, conviene realizar un repaso en la utilización de los AAGG en problemas de selección de conjuntos.

El uso de los AAGG en el problema de la selección de casos no ha sido muy común. Solamente se han encontrado 4 trabajos dedicados a ellos. El primero debido a Radcliffe [128], trata de generalizar uno de los conceptos más importantes en AAGG, el concepto de esquema. Una vez generalizado este concepto el autor trata de enumerar una serie de propiedades, que deberían poseer los operadores de cruce. En su segundo trabajo (Radcliffe [129]) el autor diseña un operador para el problema de la selección de subconjuntos. Este operador puede ser modificado de manera que cumpla con las propiedades definidas en su primer trabajo. En el último trabajo, Radcliffe y George [130], comparan las aproximaciones más directas al problema de la selección de conjuntos con su propia aproximación al problema.

El trabajo realizado por Lucasius y Kateman [108] es bastante similar al realizado por Radcliffe. Los autores definen una serie de buenas propiedades que debe cumplir un operador, y definen operadores para el

problema de la selección de conjuntos de acuerdo con estas propiedades.

La codificación de los individuos coincide en todos los artículos anteriores. Si se está buscando un subconjunto de tamaño s en un conjunto de tamaño n, un individuo es un vector de longitud s, donde cada gen del individuo puede tomar un valor del conjunto $\{1, 2, \ldots, n\}$, siendo evidentemente todos los valores diferentes. Por ejemplo, supongamos que tenemos el conjunto $\{1, 2, 3, 4, 5, 6, 7, 8\}$ y queremos realizar una selección de un subconjunto de tamaño 3, un individuo podría ser (245) lo que significaría que se ha elegido el subconjunto $\{2, 4, 5\}$.

5.4.2 La función objetivo

Como se dijo anteriormente en la motivación de la selección de casos, el objetivo es obtener un subconjunto de puntos que mantenga las propiedades estructurales del conjunto total. El punto crucial y probablemente el más complicado es elegir una función objetivo que sea capaz de valorar en que medida el subconjunto elegido está manteniendo estas propiedades estructurales.

Para elegir la función objetivo se tomó la decisión de representar cada conjunto mediante la matriz de correlación de Pearson. El elemento (i, j) de esta matriz, denotado por r_{ij} se define como:

$$r_{ij} = \frac{S_{ij}}{S_i S_j} = \frac{\frac{1}{n}\sum_{l=1}^{n}(x_{il} - \overline{x}_i)(x_{jl} - \overline{x}_j)}{\sqrt{\frac{1}{n}\sum_{l=1}^{n}(x_{il} - \overline{x}_i)^2}\sqrt{\frac{1}{n}\sum_{l=1}^{n}(x_{jl} - \overline{x}_j)^2}}$$

donde $\overline{x}_i$ y $\overline{x}_j$ son los valores medios de los datos en las variables i y j respectivamente $(i, j = 1, 2, \ldots, p)$.

Aunque es conocido que representar un conjunto de datos mediante esta matriz no es conveniente en algunos casos, un ejemplo de ello puede verse en el libro de Jain y Dubes [80] pag. 254, en esta memoria se adoptó esta manera de recoger la estructura del conjunto porque se piensa que en la mayoría de los casos es una buena representación.

Finalmente, la función objetivo trata de medir la cercanía entre la matriz de correlación de Pearson del subconjunto seleccionado con respecto de la matriz de correlación de Pearson del conjunto de datos total. Para un subconjunto Ω_s la función objetivo puede expresarse

como:

$$F(\Omega_s) = \sum_{i,j=1}^{p} |r_{ij} - r_{ij}^s|^2$$

donde r_{ij} es el coeficiente de correlación del conjunto inicial en las variables i y j, y r_{ij}^s es el coeficiente de correlación en el subconjunto elegido.

5.4.3 La codificación

Los individuos del AG utilizado en la selección de casos son vectores 0-1 de longitud:

$$[\log_2 \left(\begin{array}{c} n \\ s \end{array} \right)] + 1.$$

Esto es debido a que para codificar cada subconjunto utilizamos un método que asigna a cada uno de ellos un número, y el número de posibles subconjuntos es precisamente $\left(\begin{array}{c} n \\ s \end{array} \right)$. La numeración de los subconjuntos es llevada a cabo de manera que a los subconjuntos que poseen los primeros objetos del conjunto se les asignan los primeros números y así de forma sucesiva.

Dado un individuo, primero se calcula su valor decimal, y dicho número se utiliza para, mediante una estructura de árbol, obtener el subconjunto que tiene asignado dicho número. En cada nodo del árbol se decide si un elemento del conjunto se encuentra en el subconjunto o no.

Se dará a continuación un ejemplo en el que se clarifican los conceptos anteriores. Supongamos que disponemos de un conjunto con 6 objetos $\{A, B, C, D, E, F\}$ y se quiere seleccionar un subconjunto de tamaño 4. Inicialmente se enumeran los posibles subconjuntos de tamaño 4. Esto se hace de la siguiente manera:

$$
\begin{array}{llll}
1 & \longrightarrow & \{A,B,C,D\} & 2 \longrightarrow \{A,B,C,E\} \\
3 & \longrightarrow & \{A,B,C,F\} & 4 \longrightarrow \{A,B,D,E\} \\
5 & \longrightarrow & \{A,B,D,F\} & 6 \longrightarrow \{A,B,E,F\} \\
7 & \longrightarrow & \{A,C,D,E\} & 8 \longrightarrow \{A,C,D,F\} \\
9 & \longrightarrow & \{A,C,E,F\} & 10 \longrightarrow \{A,D,E,F\} \\
11 & \longrightarrow & \{B,C,D,E\} & 12 \longrightarrow \{B,C,D,F\} \\
13 & \longrightarrow & \{B,C,E,F\} & 14 \longrightarrow \{B,D,E,F\} \\
15 & \longrightarrow & \{C,D,E,F\}. &
\end{array}
$$

Una vez numerados todos los posibles subconjuntos se construye el árbol de clasificación. A cada nodo del árbol se le asocia un número que servirá para discriminar si un objeto en concreto se encuentra en el subconjunto o no. A la raíz, por ejemplo, se le asocia el número de subconjuntos que poseen el primer objeto del conjunto inicial, esto es, $\left(\begin{array}{c} 5 \\ 3 \end{array} \right) = 10$. El árbol que se genera para el ejemplo puede verse en la figura 5.5.

Si ahora decodificásemos un individuo y se obtuviera un número menor o igual que 10, esto significaría que el primer objeto está en el conjunto, si el número fuese mayor significaría que no está. En caso afirmativo, pasaríamos al nodo de la izquierda, donde se analizaría si el segundo objeto está en el conjunto, supuesto que el primer ya está en el mismo. En este caso, dicho nodo tiene asociado el número $\left(\begin{array}{c} 4 \\ 2 \end{array} \right) = 6$, que coincide con el número de subconjuntos que poseen el primer y el segundo objeto del conjunto. La decodificación continuaría de esta manera hasta obtener un subconjunto completo.

5.4.4 Operadores y parámetros

Los operadores utilizados en este caso son los operadores clásicos de AAGG para codificaciones 0-1. Como operador de cruce hemos utilizado el cruce en dos puntos y el operador de mutación consiste en cambiar el valor de un gen con cierta probabilidad.

Los parámetros utilizados por el algoritmo son los siguientes:

$$\lambda = 200, \quad p_m = 0.005, \quad p_c = 1.$$

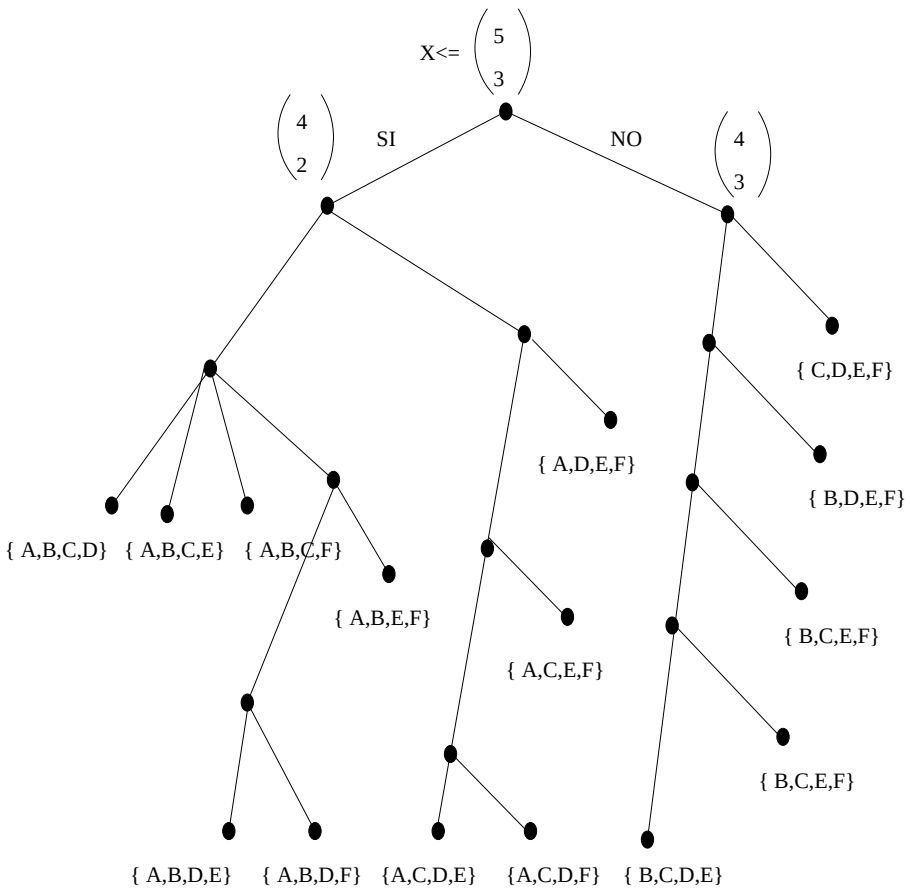

Figura 5.5: Árbol generado para la decodificación de un individuo en la selección de casos

En cuanto al criterio de parada, el algoritmo terminaba si el valor medio de la función objetivo de la población no se modificaba en 1000 generaciones.

Una vez utilizado el AG para hallar el subconjunto seleccionado, se utilizaba el algoritmo de clasificación particional expuesto anteriormente para clasificar dicho subconjunto. Los operadores utilizados en este caso para calcular la clasificación son los siguientes:

$$\lambda = 40, \quad p_m = 0.01, \quad p_c = 1$$

y el criterio de parada establece que el algoritmo termina si el valor medio de la función objetivo no varia en 40 iteraciones.

Para conseguir una clasificación del conjunto de datos total se utiliza, obviamente, la clasificación del subconjunto. Para cada clase obtenida en la clasificación del subconjunto seleccionado se halla su centroide. Supongamos que estos son $\{\overline{w}_1, \overline{w}_2, \ldots, \overline{w}_k\}$. Cada objeto del conjunto de datos no seleccionado se asigna a la clase cuyo centroide es el más cercano. Supongamos que dichos objetos se pueden representar por $\{w_{s,1}, w_{s,2}, \ldots, w_{s,n-s}\}$, entonces lo anterior puede expresarse como:

$$w_{s,i} \in C_j \Longleftrightarrow d(w_{s,i}, \overline{w}_j) = \min_{t=1,2,\ldots,k} d(w_{s,i}, \overline{w}_t)$$

$$\forall\ i = 1, 2, \ldots, n-s \ \ \forall\ j = 1, 2, \ldots, k$$

donde C_j es el conjunto formado por los objetos de la clase j.

Para obtener la clasificación del subconjunto de objetos seleccionado únicamente fue utilizado el operador PMX, ya que como se vio era el más rápido y no perdía efectividad con respecto a los otros. Como operador de mutación se eligió el SM.

5.4.5 Resultados experimentales

Se llevaron a cabo varios experimentos con los conjuntos anteriores para medir la validez de la selección de casos. En cada conjunto se eligieron 10 subconjuntos diferentes utilizando el AG. En todos los experimentos fué elegido un subconjunto con tamaño del 15 % del conjunto completo. En la Tabla 5.5 se recogen los resultados medios de bien clasificados, tanto con la selección de casos como sin ella.

datos	% bien clas. con selección	% bien clas. sin selección
1	99.85	99.96
2	98.7	99.59
Iris	89.8	89.51

Tabla 5.5: Porcentaje de bien clasificados con y sin selección de casos

Como se puede ver los resultados de aplicar el algoritmo al conjunto completo y al conjunto de datos seleccionados no produce gran variación en los resultados, lo que justifica nuestra aproximación.

Es importante destacar el hecho de que los resultados obtenidos con la selección de casos en el conjunto de datos Iris son mejores que los obtenidos sin ella. Esto es debido al aumento en los parámetros del algoritmo de clasificación que se utilizó para clasificar el subconjunto seleccionado. De alguna manera este resultado corrobora la afirmación que, acerca de los resultados en el conjunto de datos Iris, se hizo en la sección anterior.

Capítulo 6

Algoritmos Genéticos Aplicados a la CNS Jerárquica

6.1 Introducción

Este capítulo está dedicado, a exponer la forma en la que se han aplicado los AAGG al problema de la clasificación jerárquica. Como vimos en el capítulo 4, los algoritmos de clasificación jerárquica, tanto los aglomerativos como los divisivos, padecían del mismo problema que los algoritmos particionales, es decir, son algoritmos de optimización locales. Esto entonces, motiva el uso de los AAGG u otra técnica de optimización combinatoria.

Sin embargo, la resolución de este problema con cualquiera de estas técnicas no es algo directo y envuelve dentro de sí varias cuestiones. La primera de todas consiste en idear un método para comparar de manera cuantitativa diferentes clasificaciones jerárquicas de un conjunto de datos. Además tiene que ser posible tener un *marco de referencia* con el cual poder comparar cualquier clasificación jerárquica. La segunda cuestión consiste en la manera de codificar una clasificación jerárquica. Debido a que esta encierra en sí dos cosas totalmente diferentes, por un lado, una estructura de árbol, y por otro una función que asocia un número a cada nodo, no parece una tarea fácil llevar a cabo dicha

codificación.

Este trabajo está tomado en gran parte de Lozano y Larrañaga [104] y Lozano y Larrañaga [105].

El capítulo se estructura de la siguiente manera. En la próxima sección se plantea el problema de la búsqueda de la mejor clasificación jerárquica de manera formal, de tal forma que éste puede resolverse mediante el uso de un algoritmo de optimización. La tercera sección expondrá las aproximaciones previas a este problema. La cuarta está dedicada a la descripción de la metodología desarrollada en la aplicación de los AAGG al problema de la clasificación jerárquica propiamente dicha. Se termina con una sección donde se exponen los resultados experimentales y se validan las aproximaciones.

6.2 Planteamiento del problema

Trataremos en esta sección de plantear el problema de manera que pueda ser resuelto mediante un algoritmo de optimización. Comenzaremos proponiendo maneras de medir la bondad de una clasificación jerárquica.

Conviene, en todo momento, tener presente la definición formal de una clasificación jerárquica. Esto es, un par (T, h) donde T es un árbol y h es una función definida sobre los nodos de T.

Por otro lado supondremos de aquí en adelante que los datos de entrada al algoritmo de clasificación jerárquica vienen expresados en forma de matriz de medidas.

La siguiente idea es fundamental en la comparación de clasificaciones jerárquicas. Dada una clasificación jerárquica cualquiera (T, h), es posible, a partir de élla, obtener en el conjunto de datos una nueva disimilaridad de la siguiente forma:

$$U(\omega, \omega^{'}) = \min_{A \in T}\{h(A) \mid \omega, \omega^{'} \in A\}.$$

Si consideramos la clasificación jerárquica dada en la figura 4.3 del capítulo 4 la disimilaridad que se crea mediante el proceso anterior se puede ver en la figura 6.1.

$$
\begin{array}{c} \\ w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \end{array}
\begin{array}{c}
\begin{array}{ccccc} w_1 & w_2 & w_3 & w_4 & w_5 \end{array} \\
\left(\begin{array}{ccccc}
0 & 10 & 10 & 10 & 10 \\
 & 0 & 5 & 6 & 6 \\
 & & 0 & 6 & 6 \\
 & & & 0 & 2 \\
 & & & & 0
\end{array} \right)
\end{array}
$$

Tabla 6.1: Disimilaridad ultramétrica asociada a la clasificación dada en la figura 4.3

Únicamente hemos representado la parte triangular superior ya que la matriz es simétrica. Este hecho implica que en lo que sigue únicamente trataremos con la parte triangular superior de la matriz.

Esta disimilaridad, posee, además de las propiedades que, en el capítulo 3, definían una disimilaridad, otra nueva propiedad llamada *propiedad ultramétrica*. Esta propiedad se expresa matemáticamente como sigue:

$$\forall\ \omega, \omega^{'}, \omega^{''} \in \Omega \Longrightarrow U(\omega, \omega^{''}) \leq \max\{U(\omega, \omega^{'}), U(\omega^{'}, \omega^{''})\}.$$

El nombre que recibe esta propiedad es el que toman las disimilaridades que la cumplen, esto es, *disimilaridades ultramétricas*.

Las disimilaridades ultramétricas, definen espacios en los que todos los triángulos que se forman, al unir tres puntos distintos de los mismos, deben ser isósceles. Dados tres objetos del conjunto de datos w_i, w_j y w_k, se debe cumplir la propiedad ultramétrica y esto implica que de los tres números: $d(w_i, w_j), d(w_i, w_k)$ y $d(w_j, w_k)$ los dos más grandes deben ser iguales.

Otra propiedad interesante de las disimilaridades ultramétricas, y que se tendrá en cuenta más adelante, es que el máximo número de términos diferentes en la matriz de disimilaridad es $n - 1$, siendo n el tamaño del conjunto de datos.

El resultado que se presenta a continuación descubierto independientemente por Hartigan [75], Jardine y col. [81], y Johnson [82] es la clave para llevar a cabo nuestro trabajo posterior:

Teorema 7 *Existe una biyección entre las clasificaciones jerárquicas y las disimilaridades ultramétricas.*

Utilizando el teorema anterior es posible medir la bondad de una clasificación jerárquica. Para ello, basta con hallar la matriz de disimilaridad ultramétrica asociada a dicha clasificación y compararla con la matriz de disimilaridad inicial de los datos. Por lo tanto únicamente queda pendiente determinar la medida que se va a utilizar para comparar la disimilaridad ultramétrica asociada a la clasificación y la disimilaridad inicial de los datos.

Es posible encontrar en la literatura un gran cantidad de medidas para realizar la tarea anterior. Así por ejemplo, puede consultarse Caroll y Pruzansky [33] y Sneath y Sokal [144]. En nuestro trabajo al igual que la mayoría de los autores, utilizaremos como medida, de nuevo la distancia Euclidea, esto es L_2.

Por lo tanto el problema puede ser planteado como:

> Dada un conjunto de datos descrito por una matriz de disimilaridad, encontrar, en norma L_2, la matriz de disimilaridad ultramétrica más cercana a la matriz de disimilaridad de los datos.

Una vez encontrada la disimilaridad ultramétrica que cumple lo anterior, su clasificación jerárquica asociada será la mejor posible para el conjunto de datos.

Aparte de utilizar el teorema anterior para cuantificar la bondad de una clasificación jerárquica, éste puede ser usado también en la búsqueda de la mejor clasificación. El teorema permite buscar en el conjunto de disimilaridades ultramétricas en lugar de hacerlo en el de las clasificaciones jerárquicas.

6.3 Aproximaciones previas al problema

Hartigan [75] fué el primero en establecer el problema de hallar la mejor clasificación jerárquica como un problema de buscar la disimilaridad ultramétrica más cercana a la disimilaridad de los datos. El autor intentó llevar a cabo esta tarea de un manera local. Para ello definió una serie

de operaciones locales sobre un árbol y su objetivo consistió en hallar el mejor árbol para este conjunto de operaciones locales. El autor cuestionó por primera vez la idea de realizar una búsqueda en el espacio completo de disimilaridades ultramétricas, viendo que esto era imposible debido al enorme tamaño del conjunto de estas disimilaridades.

El siguiente trabajo que abordó este problema es debido a Chandon y col. [38]. En este trabajo se desarrolló un algoritmo de "ramificar y acotar" que era capaz de hallar la mejor clasificación jerárquica para un conjunto de datos. El problema que posee su algoritmo es la gran cantidad de recursos computacionales que requiere su ejecución. Es por ello que el algoritmo solamente puede ser aplicado a conjuntos de datos de tamaño menor que 14. Sin embargo, su trabajo recoge una propiedad muy interesante que será utilizada en la aplicación de los AAGG al problema, y que se expone a continuación.

Si denotamos por $S = \{(i,j) \mid \omega_i \neq \omega_j\}$ el conjunto de pares no ordenados de elementos diferentes del conjunto de datos, entonces cualquier disimilaridad ultramétrica $U = [\delta_{ij}]_{i,j=1,2,\ldots,n}$ puede ser descompuesta en una relación de preorden en S denotada por Γ:

$$\Gamma_1 < \Gamma_2 < \ldots < \Gamma_k$$

y una sucesión de números:

$$\delta_1 < \delta_2 < \ldots < \delta_k$$

tal que:

$$\forall\ (i,j) \in \Gamma_r \Longrightarrow \delta_{ij} = \delta_r \ \ \forall\ r \in \{1,2,\ldots,k\}.$$

En el caso de la disimilaridad ultramétrica dada en la figura 6.1, tendríamos la siguiente relación de preorden en S:

$$\Gamma_1 = \{(4,5)\} \ < \ \Gamma_2 = \{(2,3)\} \ < \ \Gamma_3 = \{(2,4),(2,5),(3,4),(3,5)\} \ <$$

$$< \ \Gamma_4 = \{(1,2),(1,3),(1,4),(1,5)\}$$

siendo la secuencia de números:

$$\delta_1 = 2 \ < \ \delta_2 = 5 \ < \ \delta_3 = 6 \ < \ \delta_4 = 10.$$

La propiedad anterior afirma además que si Γ es solución del problema, es decir, es la disimilaridad ultramétrica más cercana a los datos, entonces:

$$\delta_r = (1/n_r) \sum_{(i,j)\in\Gamma_r} d_{ij} \quad \forall\ r \in \{1, 2, \ldots, k\}$$

donde d_{ij} denota la disimilaridad inicial entre los objetos w_i y w_j, y n_r representa el número de pares no ordenados en el conjunto Γ_r.

De Soete [145], siguiendo las ideas de Caroll y Pruzansky [34] plantea el problema de una forma totalmente diferente. El autor formaliza el problema de manera que en su aproximación este se convierte en un problema de programación matemática, en particular de programación cuadrática. La función objetivo de este problema de programación cuadrática está compuesta de dos términos. Uno de ellos mide la diferencia entre la disimilaridad de los datos y la disimilaridad propuesta, el otro mide (o penaliza) en cuanto viola la disimilaridad propuesta la condición de ultramétrica. La influencia de esta última parte depende de un parámetro ρ. Este valor aumenta a medida que lo hace el algoritmo, de manera que la influencia de esta última parte es mucho mayor al final del algoritmo. Esta estrategia, se lleva a cabo, de cara a conseguir que la disimilaridad final propuesta esté lo más cercana posible de ser ultramétrica.

La primera parte de la función puede escribirse como:

$$L(U) = \sum_{i<j} (d_{ij} - \delta_{ij})^2,$$

la segunda toma la expresión:

$$P(U) = \sum_{\Lambda} (\delta_{ik} - \delta_{jk})^2$$

donde Λ es:

$$\Lambda = \{(i, j, k) \mid \delta_{ij} \leq \min(\delta_{ik}, \delta_{jk})\}$$

es decir, el conjunto de las tripletas que no cumplen la propiedad de ultramétrica. Finalmente la función a minimizar queda de la siguiente forma:

$$\Phi(U, \rho) = L(U) + \rho P(U)\ (\rho > 0).$$

El algoritmo funciona de manera que en cada paso del mismo se resuelve un problema de programación cuadrática sin restricciones. Una vez que esta resolución se ha llevado a cabo se aumenta el valor de ρ y se resuelve un nuevo problema.

Esta forma de plantear la búsqueda de la mejor clasificación jerárquica, aunque es más rápida que la propuesta por Chandon y col. [38], también padece del consumo de recursos computacionales del que padecía el trabajo anterior. Además de este problema, hay que añadir la imposibilidad de asegurar un óptimo global. Esto es debido a que la forma en la que es resuelto dependiendo del parámetro ρ puede conducir a un mínimo local. En la práctica los autores aplican el algoritmo varias veces para intentar asegurar la obtención del óptimo global.

Chandon y De Soete [39] llevan a cabo una comparación entre sus métodos. En dicho trabajo se ve que únicamente la aproximación dada por De Soete [145] puede ser aplicada a conjuntos medianos (tamaño 18). Por otro lado en conjuntos pequeños el tiempo empleado por la aproximación de De Soete [145] es incluso mayor que el usado por la aproximación de Chandon y col. [38]. El algoritmo de De Soete [145], alcanza el óptimo en bastantes ejecuciones, en los conjuntos con los que se llevan a cabo los experimentos. Sin embargo, no existen resultados acerca de su aplicación a conjuntos de datos más grandes (tamaño mayor que 50).

Por último cabe destacar un trabajo desarrollado por Agarwala y col. [2], en el cual se da un algoritmo polinomial para resolver el problema que estamos tratando, pero cuando la norma es la L_1.

6.4 AAGG en la búsqueda de la mejor clasificación jerárquica

Planteamos a continuación dos formas diferentes de utilizar los AAGG en la búsqueda de la mejor clasificación jerárquica de un conjunto de datos. La primera de ellas es completamente novedosa y la llamaremos aproximación basada en el orden. La segunda esta basada en la utilización de los AAGG en la optimización de la función $\Phi(U, \rho)$ definida anteriormente con $\rho = 1$, la llamaremos aproximación basada en la

penalización.

6.4.1 Aproximación basada en el orden

La resolución del problema mediante esta aproximación se apoya principalmente en dos resultados diferentes.

La primera, enunciada en Lerman [99], afirma que dada una disimilaridad ultramétrica en un conjunto de datos Ω, es posible encontrar un orden θ para los objetos del conjunto de datos, de tal forma que si se ordenan las filas y columnas de U en dicho orden, los elementos de U cumplen las siguientes propiedades :

i) $\forall\ i \leq j \Longrightarrow \delta_{ij} \leq \delta_{ij+1}$

ii) $\forall\, k \in \{1, 2, \ldots, n\}$ si $\delta_{kk+1} = \delta_{kk+2} = \ldots = \delta_{kk+s+1} < \delta_{kk+s+2}$ entonces:

- $\delta_{k+1j} \leq \delta_{kj}\ \forall\ k+1 < j \leq k+s+1$
- $\delta_{k+1j} = \delta_{kj}\ \forall\ j > k+s+1.$

Es decir, los elementos de la matriz no decrecen al salir de la diagonal, y si en la fila k-ésima dos elementos seguidos son diferentes, a partir de la columna donde ocurra esto, los elementos de la fila k-ésima y los de la $k+1$-ésima coinciden. El orden θ no es necesariamente único. Para entender mejor la segunda propiedad consideremos la siguiente matriz:

$$\begin{pmatrix} 0 & 3 & 3 & 3 & 4 & 5 & 7 \\ & 0 & 2 & 2 & 4 & 5 & 7 \\ & & 0 & 1 & 4 & 5 & 7 \\ & & & 0 & 2 & 5 & 7 \\ & & & & & 1 & 7 \\ & & & & & & 6 \\ & & & & & & 0 \end{pmatrix}.$$

La matriz representa una matriz de disimilaridad ultramétrica cuyas filas y columnas están puestas en el orden θ. Se puede ver que en la fila 1 los valores entre la columna cuarta y quinta son diferentes. Entonces en los valores de la fila 2 a partir de la columna cuarta son iguales a los de la fila 1.

El segundo resultado en el que se apoya esta aproximación es el dado por Chandon y col. [38] y ya comentado anteriormente.

Dado un orden θ para los datos y una estructura de árbol T, es fácil determinar la matriz de disimilaridad ultramétrica más cercana a la disimilaridad de los datos en dicho orden y para dicho árbol T. La estructura de árbol establece que valores de la matriz de disimilaridad ultramétrica son iguales, para de esta manera obtener como mucho $n-1$ valores diferentes. De esta forma el problema puede ser planteado como un problema de programación cuadrática con restricciones, en $n-1$ variables.

Veamos un ejemplo de esto último. Supongamos que tenemos un conjunto de datos Ω con 5 objetos, cuya matriz de disimilaridad en el orden θ es:

$$\begin{pmatrix} 0 & 4 & 3 & 5 & 1 \\ & 0 & 2 & 8 & 3 \\ & & 0 & 6 & 2 \\ & & & 0 & 9 \\ & & & & 0 \end{pmatrix}.$$

La estructura de árbol es la dada por la figura 4.3 del capítulo 4. Esta estructura de árbol impone la siguiente estructura en la matriz de disimilaridad ultramétrica buscada:

$$\begin{pmatrix} 0 & \delta_1 & \delta_1 & \delta_1 & \delta_1 \\ & 0 & \delta_2 & \delta_3 & \delta_3 \\ & & 0 & \delta_3 & \delta_3 \\ & & & 0 & \delta_4 \\ & & & & 0 \end{pmatrix}.$$

De esta manera el problema de programación cuadrática puede plantearse de la siguiente forma:

$$\min (\delta_1 - \frac{(4+3+5+1)}{4})^2 + (\delta_2 - 2)^2 + (\delta_3 - \frac{(8+3+6+2)}{4})^2 + (\delta_4 - 9)^2$$

sujeto a las siguientes restricciones:

$$\begin{array}{c} \delta_3 - \delta_1 \leq 0 \\ \delta_2 - \delta_3 \leq 0 \\ \delta_4 - \delta_3 \leq 0 \\ \delta_i \in \mathbb{R}^+ \quad i = 1,2,3,4. \end{array} \tag{6.1}$$

Para resolver este problema es posible encontrar amplia bibliografía y una variedad importante de métodos dedicados a ello. En el siguiente capítulo se dará información acerca de ello.

Codificación

Vamos a intentar dar una codificación para el problema siguiendo las ideas del punto anterior. Ya que resolver el problema, dado un orden θ y una estructura de árbol T es algo relativamente sencillo, nos planteamos codificar en cada individuo del AAGG al mismo tiempo un orden para los datos y una estructura de árbol.

Un individuo del AG va a ser entonces una permutación de los números del 1 al $n + (n - 2)$. Los números del 1 al n codifican un orden para los objetos del conjunto de datos, cada número representa un objeto. Por otro lado los números del $n + 1$ al $n + (n - 2)$ codifican la estructura de árbol.

Es importante recalcar el hecho de que la codificación no es única, esto es, existen diferentes pares (*orden*, *estructura de árbol*) que conducen a la misma matriz de disimilaridad ultramétrica. Es más, sería posible establecer una relación de equivalencia en el conjunto de pares (*orden*, *estructura de árbol*). Esto es lo que se denomina en AAGG redundancia en el código. La influencia de la redundancia según Reeves [136] no es perjudicial, incluso en algunos casos puede llegar a ser favorable en la tarea de búsqueda.

Función Objetivo

Dado un individuo, hallaremos una matriz de disimilaridad para el mismo, una vez hallada ésta, mediremos su diferencia cuadrática con la matriz de disimilaridad inicial de los datos en el orden θ que codificaba el individuo.

Para obtener una matriz de disimilaridad ultramétrica a partir del individuo, una opción es resolver el problema cuadrático planteado anteriormente. Dado que esto es relativamente caro en recursos computacionales, en esta memoria se ha tomado otra opción que consiste en aproximar la solución del problema cuadrático.

El valor de cada variable de la matriz ultramétrica buscada, es

aproximado inicialmente, mediante la media de los valores correspondientes en la matriz de disimilaridad inicial de los datos en el orden θ. En el ejemplo propuesto las variables tomarían los siguientes valores:

$$\delta_1 = \frac{(4+3+5+1)}{4} = 3.25, \qquad \delta_2 = 2$$
$$\delta_3 = \frac{(8+3+6+2)}{4} = 4.75, \qquad \delta_4 = 9.$$

Ahora bien, al igual que ocurre en este ejemplo, es posible que se dé el caso de que la disimilaridad que se obtenga de esta manera no sea una disimilaridad ultramétrica. Para conseguir finalmente una disimilaridad ultramétrica lo que hacemos es fijar el valor de la variable que supuestamente debe ser la más grande, en nuestro caso δ_1. El valor de las demás variables es modificado, si es necesario, de manera que se cumplan con las restricciones del programa cuadrático 6.1, y por lo tanto las condiciones de ultramétrica. Los valores finales que se obtienen para los elementos de la matriz de disimilaridad ultramétrica son:

$$\delta_1 = 3.25, \qquad \delta_2 = 2$$
$$\delta_3 = 3.25, \qquad \delta_4 = 3.25.$$

Esta forma de hallar la matriz de disimilaridad ultramétrica a partir del individuo tiene principalmente dos ventajas. Una primera es la rapidez. El gasto computacional de hallar dicha matriz es muy pequeño comparado con el que supone resolver en cada paso, un programa cuadrático con restricciones. Como segunda ventaja tenemos que, siguiendo con la propiedad dada por Chandon y col. [38], si un individuo contuviese un orden θ y una estructura de árbol T óptima entonces esta forma de decodificar el individuo nos conduciría a la disimilaridad ultramétrica óptima.

6.4.2 Aproximación basada en la penalización

Como dijimos anteriormente esta aproximación se basa en utilizar los AAGG para optimizar la función que incorpora la penalización propuesta por De Soete [145] con $\rho = 1$. Para realizar esta tarea, llevaremos a cabo, una clasificación particional, de los elementos de la parte triangular superior, de la matriz de disimilaridad inicial de los datos. Dicha partición se llevara a cabo en un número de clases $k \leq n-1$, ya

que como vimos, el máximo número de términos diferentes que puede tomar una matriz de disimilaridad ultramétrica es precisamente $n-1$.

Codificación

Dado que nuestro objetivo es realizar una clasificación particional de los elementos de la matriz de disimilaridad inicial de los datos, lo lógico sería utilizar el algoritmo dado en el capítulo anterior. Sin embargo, dicho algoritmo solamente podía utilizarse cuando los datos venían descritos mediante una matriz de medidas y este no es el caso. Por lo tanto se planteará una aproximación diferente.

Un individuo de la aproximación basada en la penalización va a ser una permutación de los números del 1 al $(n(n-1)/2)+(n-2)$. Los números del 1 al $(n(n-1)/2)$ representan los términos de la parte triangular superior de la matriz de disimilaridad inicial de los datos. Por otro lado los números del $n(n-1)/2+1$ al $n(n-1)/2+n-2$ están haciendo el papel de separadores entre clases. De esta forma si dos de estos números están juntos o uno de ellos se encuentra como primero o último de la permutación obtendremos una clasificación con menos de $n-1$ clases.

Considerando el ejemplo expuesto durante este capítulo y suponiendo que tenemos el individuo siguiente:

$$(2\ 4\ 5\ \mathbf{11}\ 10\ \mathbf{12}\ 8\ 1\ 9\ \mathbf{13}\ 3\ 6)$$

obtendríamos al decodificarlo que tenemos cuatro clases, esto es:

$$C_1=\{2,4,5\}\ \ C_2=\{10\}\ \ C_3=\{8,1,9\}\ \ C_4=\{3,6\}.$$

Función objetivo

A la hora de hallar la disimilaridad asociada a un individuo cualquiera se hace lo siguiente. Para cada clase se calcula el valor medio que los términos de esa clase tienen en la matriz de disimilaridad inicial de los datos. Dicho valor sustituye a cada término que forma la clase.

Siguiendo con el ejemplo, tenemos que los valores asociados a cada clase son:

$$\delta_1=\tfrac{3+1+2}{3}=2 \qquad \delta_2=\tfrac{9}{1}=9$$
$$\delta_3=\tfrac{6+4+2}{3}=4 \qquad \delta_4=\tfrac{5+8}{2}=6.5$$

y por lo tanto la matriz de disimilaridad que se obtiene es:

$$\begin{pmatrix} 0 & 4 & 2 & 6.5 & 2 \\ & 0 & 2 & 6.5 & 3 \\ & & 0 & 4 & 4 \\ & & & 0 & 9 \\ & & & & 0 \end{pmatrix}.$$

6.5 Los experimentos

Para comprobar la validez de ambas aproximaciones hemos llevado a cabo varios experimentos sobre dos conjuntos de datos. El primero se obtuvo del libro de Kaufman y Rousseeuw [87] pag. 57. Dicho conjunto consta de 18 objetos, cada uno de los cuales representa un tipo de flor de jardín. El segundo conjunto es el conjunto de datos Ruspini, ya utilizado previamente en esta memoria. En estos datos a partir de la distancia Euclidea se obtuvo la matriz de disimilaridad. Como se ve hemos escogido dos conjuntos con diferentes tamaños para comprobar nuestra aproximación no sólo en conjuntos pequeños.

Al igual que en la aproximación particional, en cada conjunto se llevaron a cabo 10 experimentos para cada combinación de operadores de cruce y mutación. Con la aproximación basada en el orden, se hicieron por lo tanto 240 experimentos en cada conjunto de datos. Con la aproximación basada en la penalización también se hicieron 240 experimentos en el primer conjunto de datos, sin embargo fue imposible aplicarla al segundo conjunto de datos, debido al tamaño del mismo y a los recursos computacionales requeridos por esta aproximación.

Los parámetros del algoritmo fueron establecidos en función del tamaño del individuo l:

$$\lambda = 3 * l, \quad p_m = 1/l, \quad p_c = 1.$$

Se utilizaron dos criterios de parada diferentes. El primero consistía en parar el algoritmo si el valor medio de la función objetivo no se modificaba en 50.000 generaciones sucesivas. El otro consistía en un tope máximo de iteraciones. Este número máximo fué de 150.000 y 300.000 para la aproximación basada en el orden en el conjunto de

las flores y en los datos Ruspini respectivamente, y de 50.000 para la aproximación basada en la penalización en el primer conjunto.

	basada en el orden		basada en la penalización	
	Media	Mínimo	Media	Mínimo
PMX	2.266.480	1.948.069	3.290.859	2.887.105
CX	1.978.757	1.942.537	3.215.737	3.020.156
OX1	2.136.466	1.948.155	3.191.071	2.956.487
AP	2.403.217	1.970.875	3.394.806	2.905.927
SM	2.236.766	1.942.917	3.368.590	2.956.487
SIM	2.232.747	1.948.240	3.229.441	2.924.306
ISM	2.272.982	1.945.578	3.365.626	3.128.738
IVM	2.151.611	1.942.537	3.229.217	2.996.016
EM	2.115.374	1.942.537	3.220.158	3.029.918
DM	2.167.900	1.945.492	3.225.678	2.887.105

Tabla 6.2: Resultado de los experimentos con el primer conjunto de datos

En la tabla 6.2 se pueden ver los resultados conseguidos por las dos aproximaciones en el primer conjunto de datos. La tabla recoge para cada operador el valor medio obtenido de todas las ejecuciones, más el menor valor obtenido en las 60 evaluaciones de cada operador de cruce y las 40 de cada operador de mutación, referido a la diferencia cuadrática entre la matriz de disimilaridad inicial de los datos, y la mejor matriz de disimilaridad ultramétrica dada por el algoritmo. En la segunda aproximación, a dicho valor hay que añadirle la penalización si la disimilaridad obtenida no es ultramétrica.

Como puede verse los resultados obtenidos por la primera aproximación son mucho mejores que los obtenidos con la segunda.

Para comparar nuestros resultados, hemos aplicado un subconjunto de métodos jerárquicos aglomerativos de clasificación, de los dados en el primer capítulo 4, al conjunto de las flores y los datos Ruspini. Para ello hemos utilizado los algoritmos implementados en el el paquete SAS que devolvían una clasificación jerárquica (sin cruces). Para cada uno de estos métodos se ha calculado la diferencia cuadrática entre la matriz

de disimilaridad ultramétrica obtenida a partir de las clasificaciones y la matriz de disimilaridad inicial de los datos. Los resultados pueden verse en la tabla 6.3.

Métodos	Mínimo	UPGMA	Máximo
Resultados	51.644.604	49.918.336	51.972.852
Métodos	Flexible	Media	Ward
Resultados	64.569.380	42.249.532	50.365.488

Tabla 6.3: Resultados obtenidos por los métodos de SAS en el primer conjunto de datos

Claramente cualquiera de las aproximaciones presentadas da unos resultados muy superiores a los obtenidos por los algoritmos clásicos.

En relación al conjunto de Ruspini, como ya dijimos anteriormente únicamente hemos aplicado la aproximación basada en el orden. Los resultados con este segundo conjunto se pueden ver en la tabla 6.4.

	Basada en el Orden	
	Media	Mínimo
PMX	36.422.418	26.570.410
CX	34.793.051	28.710.376
OX1	34.378.658	27.794.166
AP	54.254.448	41.654.588
SM	41.204.050	29.015.210
SIM	39.485.054	27.256.516
ISM	43.268.791	29.295.194
IVM	38.795.052	28.710.376
EM	37.807.301	27.437.560
DM	38.826.927	26.570.410

Tabla 6.4: Resultados obtenidos en el conjunto Ruspini

Los resultados obtenidos para los métodos implementados en SAS en el conjunto de datos Ruspini se encuentran en la tabla 6.5.

Métodos	Mínimo	UPGMA	Máximo
Resultados	486.150.880	351.939.904	413.378.016
Métodos	Flexible	Media	Ward
Resultados	518.739.814	351.883.584	465.034.027

Tabla 6.5: Resultados obtenidos por los métodos de SAS en el conjunto Ruspini

Nuevamente puede verse que la aproximación basada en el orden obtiene mucho mejores resultados que los métodos clásicos.

Como conclusión se pueden apuntar tres cosas. Por un lado ambas aproximaciones obtienen buenos resultados. Por otro, la aproximación basada en el orden es la mejor de ambas, no solamente utiliza menos recursos computacionales sino que además obtiene mejores resultados. Finalmente, es importante destacar el hecho de que la aproximación basada en el orden soporta su aplicación a grandes conjuntos de datos sin perder en ningún momento los buenos resultados que obtiene en los conjuntos pequeños. Incluso estos resultados podrían haberse mejorado si una vez obtenido el mejor individuo para el AAGG se hubiese resuelto el problema cuadrático para este único individuo.

Al igual que con los experimentos del capítulo anterior hemos realizado unos test estadísticos para medir la validez de cada uno de los operadores a la hora de llevar a cabo la tarea de clasificación jerárquica. Varios test de Kruskall-Wallis se realizaron entre los operadores de cruce y mutación en relación con el valor obtenido por la función objetivo y el número de evaluaciones necesario para alcanzar la convergencia.

En la aproximación basada en el orden y con el primer conjunto de datos, se obtuvo, que había una diferencia estadísticamente significativa entre los operadores de cruce en relación con el número de evaluaciones, siendo el operador de cruce CX el mejor. En el conjunto Ruspini obtuvimos resultados positivos en relación con ambos aspectos, siendo en ambos casos el operador de mutación EM el mejor.

En la aproximación basada en la penalización no se obtuvo ningún resultado estadísticamente significativo.

Capítulo 7

Algoritmos Genéticos Aplicados a la CNS Piramidal

7.1 Introducción

En este capítulo se expondrá la forma de aplicar los AAGG a la clasificación piramidal de cara a obtener la piramide óptima que representa un conjunto de datos. Recoge y completa el trabajo Lozano y Larrañaga [101].

Como se vió en el tercer capítulo, los algoritmos que construyen pirámides indexadas, actúan, al igual que los que construyen jerarquías, de forma local. Esto justifica la utilización de los AAGG como técnica de optimización, capaz de buscar el óptimo global dentro del conjunto de pirámides.

Del mismo modo que en la clasificación jerárquica nos basábamos en la relación que existía entre las jerarquías indexadas y un tipo especial de disimilaridad, en algo similar nos vamos a basar en este capítulo. Comenzaremos definiendo un tipo particular de disimilaridad asociado a una pirámide indexada en sentido amplio. Como se verá, existe una biyección entre estas disimilaridades y las pirámides indexadas en sentido amplio. La búsqueda con los AAGG la realizaremos por tanto dentro del conjunto de las disimilaridades.

El capítulo comienza introduciendo las disimilaridades piramidales, para continuar planteando el problema de la búsqueda de la disimilaridad piramidal óptima. En la tercera sección se resuelve el problema cuadrático que aparece , planteando en la siguiente la búsqueda del mejor orden. Se termina con una sección dedicada a la exposición de los experimentos y las conclusiones.

7.2 Disimilaridades Robinsonianas

Como se vió en el tercer capítulo de esta memoria, una pirámide indexada esta compuesta por un par (P, f) siendo P una pirámide y f una función definida sobre los nodos de P que cumplía ciertas propiedades. Es posible, a partir de (P, f) definir una nueva disimilaridad sobre el conjunto de datos, de la siguiente forma:

$$d(\omega, \omega^{'}) = \min_{\{R|\omega,\omega^{'}\in R\}} f(R).$$

donde R son los elementos de la pirámide. Esta disimilaridad va a poseer unas propiedades especiales, y la llamaremos disimilaridad piramidal o Robinsoniana.

Si se tiene en cuenta la píramide indexada de la figura 7.2, entonces la matriz de disimilaridad Robinsoniana asociada a dicha pirámide indexada sería:

$$\mathcal{P} = \begin{pmatrix} 0 & 5 & 5 & 10 & 10 & 12 \\ & 0 & 2 & 10 & 10 & 12 \\ & & 0 & 4 & 7 & 12 \\ & & & 0 & 7 & 12 \\ & & & & 0 & 9 \\ & & & & & 0 \end{pmatrix}$$

Definición 15 *Dado un conjunto de datos Ω, una disimilaridad:*

$$d : \Omega \times \Omega \longrightarrow I\!R$$

definida sobre los datos es una disimilaridad piramidal o Robinsoniana si cumple:

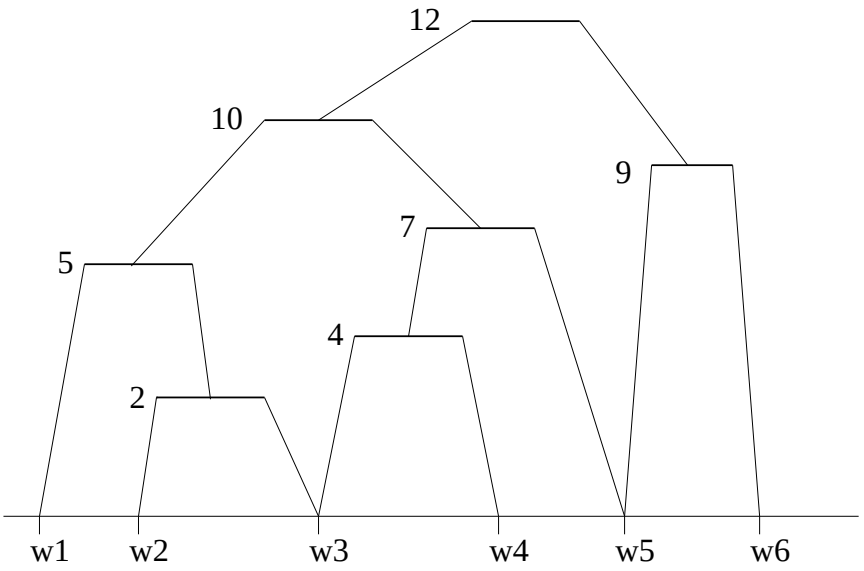

Figura 7.2: Ejemplo de una clasificación piramidal indexada

i) $d(\omega,\omega^{'}) = 0 \iff \omega = \omega^{'}$

ii) $d(\omega,\omega^{'}) = d(\omega^{'},\omega) \ \ \forall\ \omega,\omega^{'} \in \Omega$

iii) *Existe un orden* θ *en* Ω *tal que para todos los elementos* $\omega,\omega^{'}$ *y* $\omega^{''}$ *en* Ω *con* $\omega\ \theta\ \omega^{'}\ \theta\ \omega^{''}$ *se tiene:*

$$d(\omega,\omega^{''}) \geq \max\{d(\omega,\omega^{'}), d(\omega^{'},\omega^{''})\}.$$

Las disimilaridades piramidales tienen unas propiedades estructurales especiales. Si las filas y las columnas de la matriz de disimilaridad se ordenan según el orden θ entonces dicha matriz $\mathcal{R}_\theta$ es una matriz Robinsoniana (Robinson [137]).

Definición 16 *Una matriz cuadrada n-dimensional* $\mathcal{R} = [r_{ij}]_{i,j=1,2,\ldots,n}$ *se dice Robinsoniana si es no negativa, simétrica,* $r_{ii} = 0$ *para todo* $i = 1,2,\ldots,n$*, y las sucesiones siguientes:*

a) $\{r_{ii}, r_{ii+1}, r_{ii+2}, \ldots, r_{in}\}$ *con* $i = 1,2,\ldots,n-1$

b) $\{r_{ii}, r_{i-1i}, r_{i-2i}, \ldots, r_{1i}\}$ *con* $i = 2, 3, \ldots, n$

son no decrecientes.

Dicho en otras palabras, la últimas condiciones significan que al salir de la diagonal en cualquier dirección siempre voy a encontrar sucesiones no decrecientes.

El orden θ de la definición anterior, sin embargo, puede no ser único. Es decir, dada una disimilaridad piramidal, puede darse el caso de que exista más de un orden para el cual, la ordenación de las filas y columnas de la matriz en dichos órdenes nos conduzca a matrices Robinsonianas.

Es importante no confundir la noción de disimilaridad piramidal o Robinsoniana con la de matriz Robinsoniana. Dada una disimilaridad piramidal es posible ordenar las filas y columnas de la matriz que representa esta disimilaridad de manera que tras la ordenación sea Robinsoniana, sin embargo, dicha matriz con cualquier otro orden podría tener un aspecto muy diferente.

El siguiente teorema (Diday [49]) es fundamental para el uso de los AAGG en la búsqueda de la mejor estructura piramidal de un conjunto de datos:

Teorema 8 *Existe una biyección entre las pirámides indexadas en sentido amplio y las disimilaridades piramidales.*

Al igual que ocurría con la biyección entre los dendogramas y las disimilaridades ultramétricas, esta biyección nos permite realizar la búsqueda de la mejor pirámide en el conjunto de las disimilaridades piramidales en lugar de hacerlo en el de las pirámides indexadas en sentido amplio directamente.

7.3 Planteamiento del problema

Visto lo anterior, la búsqueda de la mejor estructura piramidal para un conjunto de datos, se puede escribir de manera formal como: dado un conjunto de datos Ω y una matriz de disimilaridad para los datos $D = [d_{ij}]_{i,j=1,2,\ldots,n}$, se trata de buscar la matriz de disimilaridad piramidal $\mathcal{P}^* = [p^*_{ij}]_{i,j=1,2,\ldots,n}$ tal que:

$$d^*(D, \mathcal{P}^*) = \min_{\mathcal{P} \in PD} d^*(D, \mathcal{P})$$

donde PD es el conjunto de las matrices que representan disimilaridades piramidales. La distancia d^* (el asterisco es para distinguirla de la distancia entre los elementos de Ω), puede ser cualquier distancia entre matrices n-dimensionales. Como ya vimos en el tema anterior, se pueden encontrar en la literatura gran cantidad de distancias diferentes. En nuestro caso utilizaremos nuevamente el cuadrado de la distancia Euclidea, L_2. Esta distancia solamente se calculará entre los elementos de la parte diagonal superior debido a que ambas matrices son simétricas.

Se sabe que si $\mathcal{P}$ es una matriz representando una disimilaridad piramidal entonces existe un orden θ tal que si se ordenan las filas y columnas de acuerdo a este orden, entonces la matriz resultante $\mathcal{P}_\theta$ es una matriz Robinsoniana. Por lo tanto si conociésemos el orden óptimo θ^*, entonces el problema se reduciría a hallar la matriz Robinsoniana $\mathcal{R}^*$ más cercana a la matriz de disimilaridad de los datos en este orden, esto es:

$$d^*(D_{\theta^*}, \mathcal{R}^*) = \min_{\mathcal{R} \in RM} d^*(D_{\theta^*}, \mathcal{R})$$

donde RM es el conjunto de matrices Robinsonianas.

De esta manera el problema de la búsqueda de la disimilaridad piramidal más cercana a la disimilaridad de los datos lo hemos dividido en dos problemas:

1. Búsqueda del mejor orden θ^* dentro del conjunto de órdenes de los datos. (Búsqueda del orden)

2. Dado el orden óptimo θ^* hallar la matriz Robinsoniana más cercana a la matriz de disimilaridad de los datos D_{θ^*} en el orden θ^*. (Búsqueda de la matriz Robinsoniana).

Pasaremos inicialmente a resolver el problema de la búsqueda de la mejor matriz Robinsoniana, para más tarde, y apoyándonos en la resolución de este problema, resolver el problema de la búsqueda del mejor orden.

7.4 Hallar la matriz Robinsoniana más cercana a una disimilaridad

El problema que tratamos en esta sección es el de hallar, dado un conjunto de datos Ω y una disimilaridad para los mismos D, la matriz Robinsoniana $\mathcal{R}^*$ más cercana a dicha disimilaridad. Este problema, dado que la distancia que estamos considerando es el cuadrado de la distancia Euclidea puede ser planteado como un problema de programación cuadrática con restricciones. Claramente la función objetivo es cuadrática, y si los elementos de la matriz que estamos buscando son reenumerados por filas comenzando por la primera (solamente utilizamos los elementos de la parte triangular superior), en este caso las restricciones son del tipo $r_i^* - r_j^* \leq 0$.

Veamos como se enunciaría el problema cuadrático para un ejemplo concreto. Supongamos que la matriz de disimilaridad de los datos D es:

$$D = \begin{pmatrix} 0 & 3 & 1 & 9 \\ & 0 & 4 & 3 \\ & & 0 & 5 \\ & & & 0 \end{pmatrix}.$$

Entonces la matriz que estamos buscando tras la reenumeración es:

$$\mathcal{R} = \begin{pmatrix} 0 & r_1^* & r_2^* & r_3^* \\ & 0 & r_4^* & r_5^* \\ & & 0 & r_6^* \\ & & & 0 \end{pmatrix}$$

y el problema cuadrático puede plantearse de la siguiente manera:

$$\min (r_1^* - 3)^2 + (r_2^* - 1)^2 + (r_3^* - 9)^2 + (r_4^* - 4)^2 + (r_5^* - 3)^2 + (r_6^* - 5)^2$$

sujeto a las siguientes restricciones:

$$\begin{array}{c} r_1^* - r_2^* \leq 0 \\ r_2^* - r_3^* \leq 0 \\ r_4^* - r_2^* \leq 0 \\ r_5^* - r_3^* \leq 0 \\ r_6^* - r_5^* \leq 0 \\ r_i^* \geq 0 \quad i = 1, 2, \ldots, 6 \end{array}.$$

Es posible encontrar, tanto en la literatura de Investigación Operativa (Prawda [127], Winston [157]), como en la de Análisis Numérico Fletcher [55], Lawson y Hanson [98], gran cantidad de métodos dedicados a resolver este tipo de problemas. En esta memoria hemos utilizado un método de tipo "conjunto activo" que puede encontrarse en Fletcher [55]. Este método ha sido mejorado utilizando algunos resultados teóricos que se presentan a continuación.

Dado que resolver un problema cuadrático es relativamente caro desde un punto de vista computacional, se decidió realizar algunas investigaciones acerca de la estructura de la matriz óptima, esperando que esto nos condujese a la simplificación del problema.

Los resultados que se exponen a continuación son válidos para cualquier tipo de norma, no sólo, la L_2. Por contra el algoritmo final si que depende del tipo de norma que se utilice.

Teorema 9 *Sea $D = [d_{ij}]_{i,j=1,2...,n}$ una matriz de disimilaridad en un conjunto de datos Ω y $\mathcal{R}^* = [r^*_{ij}]_{i,j=1,2,...,n}$ la matriz Robinsoniana más cercana a D, entonces se cumple:*

a) $r^*_{ii+1} \leq d_{ii+1}\ \forall i \in \{1, 2 \ldots, n-1\}$

b) $r^*_{1n} \geq d_{1n}$

c) *Si existe (i,j) tal que $r^*_{ij} < d_{ij}$ entonces existe un camino que partiendo de (i,j) recorre elementos de la matriz hacia la derecha o hacia arriba de tal forma que todos los elementos de r^*_{ij} son iguales y el camino termina en $(i^{'}, j^{'})$ tal que $d_{ij} > d_{i^{'}j^{'}}$.*

d) *Si existe (i,j) tal que $r^*_{ij} > d_{ij}$ entonces existe un camino que partiendo de (i,j) recorre elementos de la matriz hacia la izquierda o hacia abajo de tal forma que todos los elementos de r^*_{ij} son iguales y el camino termina en $(i^{'}, j')$ tal que $d_{ij} < d_{i^{'}j^{'}}$.*

e) *En la primera fila si existe i tal que $r^*_{1i} < d_{1i}$ entonces una de dos:*

- $r^*_{1j} = r^*_{1i} \qquad \forall\ j > i$
- $\exists\ j > i$ *tal que* $d_{1i} > d_{1j}$.

f) *En la última columna si existe j tal que $r^*_{jn} < d_{jn}$ entonces una de dos:*

* $r^*_{jn} = r^*_{in} \qquad \forall i < j$
* $\exists\ i < j$ *tal que* $d_{nj} > d_{ni}$.

Desmostración

a) Supongamos que existe i tal que $r^*_{ii+11} > d_{ii+11}$, entonces, es suficiente tomar la matriz Robinsoniana $\mathcal{R}^{'}$ tal que:

$$r^{'}_{jk} = \begin{cases} d_{ii+1} & \text{si } j = i \text{ y } k = i+1 \\ r^*_{jk} & \text{en otro caso} \end{cases}$$

Claramente $d^*(D, \mathcal{R}^{'})$ es menor que $d^*(D, \mathcal{R}^*)$ por lo que llegaríamos a que $\mathcal{R}^*$ no es óptima y por lo tanto un absurdo, de lo que se deduce el resultado **a)**.

b) Un razonamiento similar se puede utilizar para deducir el segundo resultado.

c) Supongamos que se tiene que $r^*_{ij} < d_{ij}$, claramente se tiene que cumplir que $r_{ij} = r_{ij+1}$ o $r_{ij} = r_{i-1j}$ ya que en caso contrario argumentando de forma similar a como lo hicimos en el apartado **a)** sería posible encontrar una matriz Robinsoniana $\mathcal{R}^{'}$ más cercana a D que $\mathcal{R}^*$. Nuevamente en r_{ij+1} o en r_{i-1j} se puede argumentar de la misma manera a menos que $r_{ij+1} \geq d_{ij+1}$ con lo que llegaríamos al resultado deseado ($d_{ij} > d_{ij+1}$) o $r_{i-1j} \geq d_{i-1j}$ con lo que nuevamente llegaríamos al resultado esperado. En caso contrario la argumentación continuaría, hasta llegar al par $(i^{'}, j^{'})$ buscado.

d) Similar a la demostración anterior.

e) Es un corolario de **c)**.

f) Es un corolario de **d)**.

El teorema siguiente nos da ciertas condiciones que la matriz Robinsoniana óptima $\mathcal{R}^*$ debe cumplir en relación con D:

Teorema 10 *Dada D matriz de disimilaridad sobre Ω , si existen (i,j) y $(i^{'},j^{'})$ con $i \geq i^{'}$ y $j \leq j^{'}$ tal que $d_{i,j} > d_{i^{'}j^{'}}$, entonces existen en R^{*} al menos dos elementos iguales.*

Demostración
Sea $\mathcal{R}^{*}$ la matriz Robinsoniana óptima. Claramente en (i,j) se pueden dar tres casos diferentes:

- (a) $r_{ij} < d_{ij}$. Entonces por lo visto en el teorema anterior, $r_{ij} = r_{i-1j}$ o $r_{ij} = r_{ij+1}$.
- (b) $r_{ij} = d_{ij}$, claramente $r_{i^{'}j^{'}} \leq r_{ij}$, por lo tanto $r_{i^{'}j^{'}} > d_{i^{'}j^{'}}$ y ahora es suficiente tener en cuenta el caso d) del teorema anterior.
- (c) $r_{ij} > d_{ij}$. Evidentemente deben existir dos elementos iguales luego es suficiente razonar de forma similar a como lo hicimos en el anterior teorema.

El resultado anterior nos permite sacar una conclusión muy interesante. A menos que la matriz de disimilaridad de los datos en el orden óptimo sea una matriz Robinsoniana, la matriz Robinsoniana más cercana a la dada no va a crear una pirámide saturada (posee todos los grupos posibles), ya que van a existir varios elementos en la matriz iguales. Esto es muy interesante ya que la mayoría de los algoritmos de construcción piramidal construyen una pirámide saturada para más tarde dar algoritmos que la simplifican de cara a que su interpretación sea más directa Bertrand y col. [21], Carles y col. [32].

Como resultado de los anteriores teoremas se puede enunciar el siguiente corolario que nos permite fijar ciertos elementos en $\mathcal{R}^{*}$ a partir de D y al mismo tiempo nos acota la búsqueda de la mejor matriz Robinsoniana $\mathcal{R}^{*}$ de manera importante.

Corolario 1 *Dada una matriz de disimilaridad D sobre un conjunto de datos Ω, sea $\mathcal{R}^{*}$ la matriz Robinsoniana mas cercana a D, si d_{ij} es tal que:*

i) $d_{ij} \geq d_{i^{'}j^{'}} \qquad \forall\ i^{'} \leq i\ y\ j^{'} \geq j$

ii) $d_{ij} \leq d_{i^{'}j^{'}} \qquad \forall\ i^{'} \geq i\ y\ j^{'} \leq j$

entonces se tiene que $r^*_{ij} = d_{ij}$.

Demostración
Supongamos por reducción al absurdo que $r_{ij} > d_{ij}$ entonces por el apartado **d)** del teorema 1 se contradeciría la hipótesis **ii)**, por lo tanto imposible. Si por el contrario suponemos que $r^*_{ij} < d_{ij}$ en este caso se contradeciría el apartado **c)** del teorema 1, y por lo tanto se tiene el resultado deseado.

Este resultado no sólo nos fija algunos elementos de la matriz $\mathcal{R}^*$ sino que además limita de forma importante el valor de los demás elementos de la propia matriz.

7.5 Hallando el mejor orden θ^*

Resolveremos en esta sección el problema de la búsqueda del orden óptimo. Se trata por lo tanto de dada una disimilaridad D en un conjunto de datos Ω, hallar el orden θ^* tal que la distancia entre la matriz Robinsoniana óptima y la matriz D en dicho orden sea mínima. Es decir, se busca θ^* tal que:

$$d^*(D_{\theta^*}, \mathcal{R}^*_{\theta^*}) = \min_{\theta \in \Theta} d^*(D_\theta, \mathcal{R}^*_\theta)$$

donde Θ es el conjunto de órdenes para el conjunto de datos.

Nuevamente este problema es un problema de optimización y la clave está en definir una función que alcance su óptimo en el orden θ^* óptimo. Lo ideal desde un punto de vista, tanto metodológico, como práctico sería, encontrar una función que no utilizase para nada la búsqueda de la mejor matriz Robinsoniana hecha en la sección anterior, sin embargo, esto es bastante complicado. Se propondrán a continuación tres funciones, una de las cuales se apoya en la resolución del problema de la sección anterior y las otras son independientes de éste.

Una aproximación directa aunque computacionalmente cara, consiste en hallar el valor de la función de la siguiente manera. Dado un orden θ se resuelve el problema cuadrático enunciado anteriormente para dicho orden, hallando la matriz $\mathcal{R}^*_\theta$. Se toma entonces como valor la diferencia cuadrática entre la matriz $\mathcal{R}^*_\theta$ matriz de disimilaridad de

los datos D_θ en el orden θ. El problema de esta función es el consumo de recursos informáticos que tiene en sí la resolución del problema **1.**, incluso con las simplificaciones dadas en la sección anterior. Formalmente esta función puede ser escrita como:

$$g_1(\theta) = d^*(D_\theta, \mathcal{R}^*_\theta) = \sum_{1 \le i < j \le n} (d_{ij} - r^*_{ij})^2.$$

Dado que la función g_1 consume demasiados recursos, se pensaron otras dos funciones heurísticas para resolver el problema sin el coste computacional en el que incurre la función g_1. Sin embargo, dado su carácter de heurísticas, su validez solamente podrá ser demostrada en los resultados experimentales. Es más, incluso no está demostrado que dichas funciones alcancen el óptimo cuando el orden θ^* sea el óptimo.

La primera de estas dos funciones utiliza el corolario 1, y trata de buscar el orden que maximiza el número de elementos fijos en la matriz D. Así dado un orden θ se tiene:

$$g_2(\theta) = \text{número de elementos fijados en la matriz } D_\theta.$$

Finalmente la segunda función heurística está basada en medir en que cantidad los elementos de la función D_θ violan la condición de Robinsoniana:

$$g_3(\theta) = \sum_{1 \le i < j \le n} \Big(\sum_{i' \ge i} \sum_{j \ge j'} (d_{i'j'} - d_{ij}) \chi_{(-\infty, 0]}(d_{i'j'} - d_{ij}) \Big)$$

donde χ es la función indicador.

De aquí en adelante llamaremos a la función g_1 la *función cuadrática*, la g_2 la *función de fijos* y por último a la g_3 la *función de diferencias*.

Para optimizar estas funciones hemos utilizado los AAGG. En las tres funciones se utilizó la misma estructura de AG, en el que sólo variaba la función objetivo y algunos de los parámetros.

Un individuo del AAGG va a ser un vector de tamaño n representando una permutación de los números del 1 al n. Los operadores utilizados son los mismos que en los experimentos anteriores. Como operadores de cruce se utilizaron: CX, AP, OX1 y PMX, y como operadores de mutación: SM, ISM, SIM, IVM, EM, DM.

7.6 Resultados Experimentales

Se llevaron a cabo varios experimentos para comprobar la validez de cada una de las tres funciones. Para realizar estos experimentos se utilizó el conjunto que recogía disimilaridades entre 18 flores de jardín (Kaufman y Rousseeuw [87]), y que ya se utilizó para comprobar la validez de la aproximación con AAGG dada para la clasificación jerárquica.

Los parámetros del algoritmo fueron diferentes dependiendo del tipo de función, así tenemos que para la función cuadrática estos fueron:

$$\lambda = 30, \quad p_m = 0.01, \quad p_c = 1$$

y el criterio de terminación consistía en limitar el número de evaluaciones de la función objetivo a 5000.

Para las otras dos funciones los parámetros fueron:

$$\lambda = 100, \quad p_m = 0.01, \quad p_c = 1.$$

y el número máximo de evaluaciones permitido fué elevado a 50000.

Las razones de las diferencias en los parámetros radica en el diferente consumo de recursos computacionales de cada una de las funciones.

Nuevamente se hicieron 240 experimentos con cada función. Por cada combinación de operador de cruce y mutación se realizaron 10 experimentos, dando en total 60 por cada operador de cruce y 40 por cada operador de mutación.

En la función de fijos y en la de diferencias, una vez conseguido, utilizando el AG, el orden óptimo, se resolvió el problema cuadrático para de esta manera hallar la matriz Robinsoniana $\mathcal{R}^*$, y calcular su diferencia cuadrática con D. Por lo tanto todos los experimentos son comparados utilizando la diferencia cuadrática entre $\mathcal{R}^*$ y D.

En la tabla 7.6 se pueden ver los resultados:

	Cuadrática		Fijos		Diferencias	
	Media	Mínimo	Media	Mínimo	Media	Mínimo
CX	1.678.880	1.109.720	3.642.697	3.147.214	3.049.698	3.049.697
AP	2.214.235	1.653.200	3.436.948	2.690.929	3.086.802	2.716.686
OX1	1.891.147	1.191.550	3.508.471	2.781.258	3.060.997	2.935.708
PMX	1.989.509	1.088.250	3.511.422	3.046.747	3.041.980	2.911.613
SM	1.902.028	1.088.250	3.456.989	2.690.929	3.088.329	2.938.914
SIM	1.906.257	1.409.360	3.499.278	3.004.857	3.046.146	2.911.613
ISM	1.997.465	1.258.950	3.501.517	2.858.822	3.068.682	3.046.947
IVM	1.990.582	1.274.430	3.465.102	2.781.258	3.051.722	2.994.870
EM	1.961.707	1.399.260	3.623.973	2.979.606	3.052.952	2.716.686
DM	1.902.617	1.355.880	3.602.450	3.159.027	3.051.386	2.911.613

Tabla 7.6: Resultados de los experimentos

Claramente los resultados destacan el hecho de que la función cuadrática es la que mejores valores obtiene. La función cuadrática devuelve diferencias cuadráticas mucho más pequeñas que las funciones heurísticas lo que hace dudar de su utilización en este tipo de problemas.

Para comparar los resultados obtenidos con nuestra aproximación hemos utilizado el algoritmo CAP propuesto por Capdevila [30]. Este algoritmo tiene implementadas tres medidas diferentes entre clases: Mínimo, Máximo y UPGMA.

Utilizando el algoritmo hemos conseguido tres clasificaciones piramidales para el conjunto de datos, una con cada uno de los índices de agregación. Para cada una de las clasificaciones hemos hallado su disimilaridad piramidal asociada y finalmente su diferencia cuadrática con la disimilaridad de los datos. El mejor resultado obtenido fué 5.987.237.

Claramente cualquiera de las aproximaciones supera los resultados obtenidos por el algoritmo CAP. Sin embargo, una vez observados los resultados finales, se vió que en el caso de la función cuadrática la población final del AG no había convergido. Esto motivó la realización de más experimentos aumentando el valor de los parámetros, tanto del tamaño población, como del número de evaluaciones máximo. EL mejor resultado obtenido por nuestro algoritmo fué menor que 1.000.000, exactamente 935.280.

Hay que decir en favor del CAP que nuestro algoritmo consume

mucho más recursos computacionales de lo que lo hace el CAP.

Al igual que en las utilizaciones previas de los AAGG, se llevaron a cabo varios test estadísticos de Kruskall-Wallis para comprobar si existía alguna diferencia en la eficacia de los diferentes operadores. Dado que en este caso el criterio de parada era el número máximo de evaluaciones, únicamente re realizaron los test en relación con el valor obtenido en la función objetivo. También debido al hecho de que la función cuadrática es la que obtiene resultados considerablemente mejores, únicamente realizamos los experimentos con los resultados obtenidos en dicha función.

No se obtuvieron diferencias estadísticamente significativas entre los operadores de mutación. En cuanto a los operadores de cruce se pudo comprobar que si existían diferencias estadísticas significativas, siendo el operador CX el mejor.

7.7 Conclusiones

En este capítulo se ha planteado el problema de la clasificación piramidal como un problema de optimización. Se han dado algunos resultados teóricos que ayudan en la resolución del problema cuadrático que surge del planteamiento anterior, a la vez que clarifican la estructura de la disimilaridad piramidal óptima.

Se utilizaron tres funciones diferentes para buscar el orden óptimo. De estas tres la cuadrática es la que mejor funciona, sin embargo, todas obtienen mejores resultados que el algoritmo CAP.

El algoritmo propuesto en el capítulo no puede competir en tiempo computacional con los algoritmos clásicos, sin embargo, obtiene resultados mejores que estos. El algoritmo podría ser utilizado por alguien interesado en conseguir la mejor clasificación piramidal de un conjunto de datos, y que no le importase emplear para ello cierta cantidad de tiempo.

Como conclusión podríamos decir, que se ha hecho una primera propuesta de solución en la búsqueda de la mejor pirámide representando a un conjunto de datos.

Parte V

Conclusiones

Capítulo 8

Conclusiones

8.1 Introducción

En este capítulo final expondremos, para empezar, las aportaciones de esta memoria, para a continuación enumerar las conclusiones extraídas del trabajo desarrollado. Por último nos centraremos en los trabajos futuros y las líneas abiertas de investigación.

8.2 Aportaciones

Las principales aportaciones de la presente memoria son:

i) Algoritmos Genéticos:

- Revisión bibliográfica acerca de los híbridos AAGG-EE y utilización de la distribución de Boltzmann en AAGG.
- Enunciado de un nuevo algoritmo de optimización, basado en AAGG y en EE.
- Demostración de la convergencia hacia el óptimo del algoritmo bajo condiciones débiles.
- Relajación de la condición suficiente de convergencia (elitismo) conocida hasta el momento para AAGG.
- Posibilidad de controlar el grado de elitismo y la diversidad en la población del AAGG mediante un parámetro.

ii) Clasificación Particional:

- Revisión bibliográfica de las aplicaciones anteriores de los AAGG al problema de la clasificación particional.
- Desarrollo de un algoritmo flexible basado en AAGG que puede utilizarse para hallar el número de clases y las propias clases o únicamente para hallar éstas, conocido su número.
- Utilización de los AAGG en la selección de casos cuando el objetivo posterior es realizar una clasificación particional.
- Análisis estadístico de los diferentes operadores de cruce y mutación a la hora de realizar la tarea de clasificación particional.

iii) Clasificación Jerárquica:

- Construcción de un nuevo algoritmo capaz de buscar la clasificación jerárquica óptima de un conjunto de datos.
- Posibilidad de aplicar el algoritmo a conjuntos grandes de datos (100 puntos).
- Planteamiento de un heurístico que permite la rápida evaluación de una clasificación jerárquica, dando la evaluación exacta en el caso de que el dendograma sea el óptima.
- Análisis estadístico de la validez de los operadores.

iv) Clasificación Piramidal:

- Construcción de un algoritmo capaz de buscar la clasificación piramidal óptima.
- Aportaciones sobre la matriz de disimilaridad piramidal óptima de un conjunto de datos.
- Simplificación del problema de programación cuadrático que se plantea.
- Análisis estadístico de la validez de los operadores.

8.3 Conclusiones

Como conclusión general de esta memoria se puede decir que los AAGG son una muy buena alternativa a los métodos clásicos de clasificación no supervisada. Aunque, en general, son más caros, computacionalmente hablando, ofrecen al usuario la posibilidad de obtener mejores clasificaciones. Analizaremos a continuación cada parte novedosa por separado.

Algoritmos Genéticos

Dos son las principales conclusiones que se pueden obtener a nivel teórico.

- La condición suficiente de convergencia conocida hasta el momento, el elitismo, puede ser relajada mediante un modelo probabilístico.

- Las buenas propiedades de convergencia del algoritmo de EE pueden ser exportadas, e incluso mejoradas, al campo de los AAGG.

AAGG en clasificación particional

El algoritmo desarrollado obtiene resultados muy buenos en aquellos conjuntos de datos en los que las clases están distribuidas con distribuciones cercanas a la normal o a la uniforme. No sólo halla el número de clases óptimas sino que recupera las clases casi de forma exacta.

La aplicación directa del algoritmo a conjuntos con gran cantidad de datos (más de 1.000) es computacionalmente muy costosa. Sin embargo, la selección de casos realizada utilizando los AAGG, hace que este problema pueda subsanarse permitiendo la ejecución del algoritmo en este tipo de conjuntos.

AAGG en clasificación jerárquica

El método planteado tiene un coste computacional no muy alto, sin perder en ningún momento su efectividad. Esto permite, y es su gran

cualidad, la aplicación del mismo a conjuntos de datos de tamaño grandes.

Como se puede ver en los resultados experimentales, se obtienen resultados mucho mejores que los algoritmos clásicos.

AAGG en clasificación piramidal

Es en la clasificación piramidal donde nuestra aproximación es más costosa computacionalmente hablando. Sin embargo, es en este tipo de clasificación donde no se había nunca planteado la idea de buscar la piramide óptima que representa los datos. La principal conclusión por lo tanto es la apertura de esta posibilidad.

Hasta el momento, la única alternativa para hallar una clasificación óptima consistía en utilizar algoritmos que seguían el esquema de los algoritmos clásicos de clasificación jerárquica. Nuestro algoritmo, aunque a un coste computacional alto, permite buscar por la mejor pirámide.

8.4 Trabajo futuro

Nuestras primeras ideas acerca del trabajo futuro más inmediato se centran en aspectos complementarios de esta memoria:

- Comprobar la validez práctica del algoritmo diseñado en el capítulo 3.
- Intentar mejorar el rendimiento del algoritmo introducido para clasificación piramidal. Para ello tenemos en mente varias ideas: tomar como población inicial del AAGG los resultados de algún algoritmo de tipo clásico, intentar buscar el mejor orden con la función utilizada en la clasificación jerárquica u otro tipo de función diferente.
- Aplicar los AAGG a otro tipo de clasificación no supervisada, cuyo resultado es un árbol (no necesariamente con raíz) y cuya disimilaridad asociada se denominan *distancias de tipo árbol*.

Mirando hacia un futuro más lejanos hay varios campos donde se piensa que es posible obtener resultados interesantes:

- Continuar el estudio teórico de los AAGG.
- Intentar utilizar los AAGG en la clasificación no supervisada basada en modelos mixtos, por ejemplo, como alternativa al algoritmo EM.
- Investigar el uso de las *redes bayesianas* para realizar clasificación no supervisada.

Bibliografía

[1] D.H. Ackley. *A Connectionist Machine for Genetic Hillclimbing*. Kluwer Academic Press, 1987.

[2] R. Agarwala, V. Bafina, M. Farach, B. Narayanan, M. Paterson y Mikkel Thorup. On the approximability of numerical taxonomy. Informe Interno 95-46, DIMACS, Rutgers University, Piscataway, NJ 08855, USA, 1995.

[3] K.S. Al-Sultan. A tabu search approach to the clustering problem. *Pattern Recognition*, **28**(9):1443–1451, 1995.

[4] J.T. Alander. On optimal populations size of genetic algorithms. En: *Proc. CompEuro 92*, págs. 65–70. IEEE Computer Society Press, 1992.

[5] C. Alippi y R. Cucchiara. Cluster partitioning in image analysis classification: a genetic algorithm approach. En: *Proc. CompEuro 92*, págs. 139–144. IEEE Computer Society Press, 1992.

[6] M.R. Anderberg. *Cluster Analysis for Applications*. Academic Press, New York, 1973.

[7] S. Anily y A. Federgruen. Ergodicity in parametric nonstationary markov chains: An application to simulated annealing methods. *Operational Research*, **35**(6):867–874, 1987.

[8] C.A. Ankenbrandt. An extension to the theory of convergence and a proof of the time complexity of genetic algorithms. En: G.J.E. Rawlins (ed.), *Foundations of Genetic Algorithms*, págs. 53–68. Morgan Kaufmann, 1991.

[9] J. Antonisse. A new interpretation of schema notation that overturns the binary coding constraint. En: J.D. Schaffer (ed.), *Proceedings of Third International Conference on Genetic Algorithms*, págs. 86–91. Morgan Kaufmann, 1989.

[10] J. Arabas, Z. Michalewicz y J. Mulawka. GAVaPS - a genetic algorithm with varying populations size. En: D.B. Fogel (ed.), *Proceedings of The First IEEE Conference on Evolutionary Computation*, volume I, págs. 73–79. IEEE Computer Society Press, 1994.

[11] G.P. Babu y M.N. Murty. Clustering with evolution strategies. *Pattern Recognition*, **27**(2):321–329, 1994.

[12] T. Bäck. Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. En: D.B. Fogel (ed.), *Proceedings of The First IEEE Conference on Evolutionary Computation*, volume I, págs. 57–62. IEEE Computer Society Press, 1994.

[13] T. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, 1996.

[14] J.E. Baker. Reducing bias and inefficiency in the selection algorithm. En: J.J. Grenfenstette (ed.), *Proceedings of the Second International Conference on Genetic Algorithms and Their Applications*, págs. 14–21. Lawrence Erlbaum Associates, 1987.

[15] G.H. Ball y D.J. Hall. A clustering technique for summarizing multivariate data. *Behavioral Science*, **12**:153–155, 1967.

[16] J.D. Bandfield, G. Celeux, A.E. Raftery y C.P. Robert. Inference in model-based cluster analysis. Informe Interno 285, Dpto. of Statistics, University of Washington, 1995.

[17] J.D. Bandfield y A.E. Raftery. Model-based gaussian and non-gaussian clustering. *Biometrics*, **49**:803–821, 1993.

[18] W. Banzhaf. The "molecular" traveling salesman problem. *Biologica Cybernetics*, **64**:7–14, 1990.

[19] C.B. Barber, D.P. Dobkin y H.T. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, **20**(4):469–483, 1996.

[20] P. Bertrand. *Etude de la Répresentation Pyramidale.* Thèse de Doctorat. Université de Paris-Duphine, 1986.

[21] P. Bertrand, Y. Lechevallier, M. Csernel y A. Morineau. A·pyros: a pyramidal representation of data. En: *IFCS-96, Proceedings of the fifth Conference of International Federation of Classification Societies*, volume I, págs. 126–128. Kobe, Japón, 1996.

[22] J.C. Bezdeck. Numerical taxonomy with fuzzy sets. *Journal of Mathematical Biology*, **1**:57–71, 1974.

[23] J.C. Bezdeck y S.K. Pal. *Fuzzy Models for Pattern Recognition.* IEEE Press, 1992.

[24] J.C. Bezdek, S. Boggavaparu, L.O. Hall y A. Bensaid. Genetic algorithm guided clustering. En: D.B. Fogel (ed.), *Proceedings of The First IEEE Conference on Evolutionary Computation*, volume I, págs. 34–40. IEEE Computer Society Press, 1994.

[25] J.N. Bhuyan, V.V. Raghavan y V.K. Elayavalli. Genetic algorithms with an ordered representation. En: R. Belew y L.B. Booker (eds.), *Proc. of the Fourth International Conference on Genetic Algorithms*, págs. 408–415. Morgan Kaufmann, 1991.

[26] G.L. Bilbro, L.C. Hall y L.A. Ray. A probably convergent inhomogeneous genetic annealing algorithm. En: *Neural and Stochastic Methods in Image and Signal Processing*, volume 1766, págs. 50–60. SPIE, 1992.

[27] L.B. Booker. Improving search in genetic algorithms. En: L. Davis (ed.), *Genetic Algorithms and Simulated Annealing*, págs. 61–73. Morgan Kaufmann, 1987.

[28] P. Bretrand y E. Diday. A visual representation of the compatibility between an order and a dissimilarity index: the pyramids. *Computational Statistics Quarterly*, **2**(1):31–41, 1985.

[29] A. Brindle. *Genetic algorithm for function optimization.* Tesis Doctoral, Universidad de Alberta, 1991.

[30] C. Capdevila. *Contribuciones al Estudio del Problema de la Clasificación Mediante Grafos Piramidales.* Tesis Doctoral. Universidad de Barcelona, 1993.

[31] C. Capdevila y E. Arcas. Pirámides indexadas y disimilaridades piramidales. *Qüestiió*, **19**:131–151, 1995.

[32] C. Capdevila, A.G. Gil y A. Arcas. Algunas propiedades de un algoritmo de clasificación piramidal. *Estadística española*, **37**(138):101–126, 1995.

[33] J.D. Caroll y S. Pruzansky. Fitting of hierarchical tree structures. En: *US-Japan Seminar on Multidimensional Scaling.* University of California at San Diego, 1975.

[34] J.D. Caroll y S. Pruzansky. Discrete and hybrid scaling models. En: E.D. Lenterman y H. Feger (eds.), *Similarity and Choice*, págs. 108–139. Huber, Bern, 1980.

[35] G. Celeux y G. Govaert. Gaussian parsimonious clustering models. *Pattern Recognition*, **28**(5):781–793, 1995.

[36] G. Celeux y G. Soromenho. An entropy criterion for assessing the number of clusters in a mixture model. *Journal of Classification*, **13**(2):195–212, 1996.

[37] U.K. Chakraborty y D.G. Dastidar. Using reliability analysis to estimate the number of generations to convergence in genetic algorithms. *Information Processing Letters*, **46**:199–209, 1993.

[38] J.L. Chandon, J. Lemaire y J. Pouget. Construction de l'ultrametrique la plus proche d'une dissimilarité au sens des moindres carrés. *R.A.I.R.O. Recherche Operationalle*, **14**:157–170, 1980.

[39] J.L. Chandon y G. De Soete. Fitting a least squares ultrametric to dissimilarity data: approximation versus optimization. En:

E. Diday y col. (eds.), *Data Analysis and Informatics, III*, págs. 213–221. North-Holland, Amsterdam, 1984.

[40] T.F. Cox y M.A.A. Cox. *Multidimensional Scaling*. Chapman and Hall, London, 1994.

[41] R. Cucchiara. Analysis and comparison of different genetic models for the clustering problem in image analysis. En: R.F. Albretch, C.R. Reeves y N.C. Steele (eds.), *Artificial Neural Networks and Genetic Algorithms*, págs. 423–427. Springer-Verlag, 1993.

[42] L. Davis. Applying adaptative algorithms to epistatic domains. En: *Proceedings of the International Joint Conference on Artificial Intelligence*, págs. 162–164, 1985.

[43] L. Davis. Adapting operator probabilities in genetic algorithms. En: J.D. Schaffer (ed.), *Proceedings of Third International Conference on Genetic Algorithms*, págs. 61–69. Morgan Kaufmann, 1989.

[44] L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991.

[45] T.E. Davis y J.C. Príncipe. A markov chain framework for the simple genetic algorithm. *Evolutionary Computation*, **1**(3):269–288, 1993.

[46] A.P. Dempster, N.M. Laird y D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, **39**(1):1–38, 1977.

[47] A. Díaz. *Optimización Hurística y Redes Neuronales*. Editorial Paraninfo, 1996.

[48] E. Diday. Crossings, orders and ultrametrics: Application to visualization of consensus for comparing classification. En: *Proceedings in Computer Statistics*, págs. 186–191. Physica-Verlag, 1982.

[49] E. Diday. Orders and overlapping clusters by pyramids. En: *Multidimensional Data Analysis*, págs. 201–234. DSWO Press, 1986.

[50] R.O. Duda y P.E. Hart. *Pattern Classification and Scene Analysis*. Wiley, 1973.

[51] A.W.F. Edwards y L.L. Cavalli-Sforza. A method for cluster analysis. *Biometrics*, **21**:363–375, 1965.

[52] A.E. Eiben, E.H.L. Aarts y K.M. Van Hee. Global convergence of genetic algorithms: A markov chain analysis. En: *Parallel Problem Solving from Nature. Lectures Notes in Computer Science*, volume 496, págs. 4–12. Springer-Verlag, 1991.

[53] W. Feller. *An introduction to probability theory and its applications*. Wiley, New York, 3rd. ed., 1971.

[54] D.H. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, **2**:139–172, 1987.

[55] R. Fletcher. *Practical Methods of Optimization, Vol. 2*. John Wiley & Sons, 1981.

[56] T.C. Fogarty. Varying the probability of mutation in the genetic algorithm. En: J.D. Schaffer (ed.), *Proceedings of Third International Conference on Genetic Algorithms*, págs. 104–109. Morgan Kaufmann, 1989.

[57] D.B. Fogel. An evolutionary approach to the traveling salesman problems. *Biological Cybernetics*, **60**:139–144, 1988.

[58] D.B. Fogel. Applying evolutionary programming to selected traveling salesman problems. *Cybernetics and Systems*, **24**:27–36, 1993.

[59] L.J. Fogel. *On the Organization of Intellect*. Tesis Doctoral. University of California, Los Angeles, CA, 1964.

[60] L.J. Fogel, A.J. Owens y M.J. Walsh. *Artificial Intelligence through Simulated Evolution*. Wiley, New York, 1966.

[61] E.W. Forgy. Cluster analysis of multivariate data: efficency versus interpretability of classifications (abstract). *Biometrics*, **21**:768–769, 1965.

[62] F. Glover. Tabu search-part I. *ORSA Journal on Computing*, **1**(3):190–206, 1989.

[63] F. Glover. Tabu search-part II. *ORSA Journal on Computing*, **2**(1):4–32, 1990.

[64] D.E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, 1989.

[65] D.E. Goldberg. Sizing populations for serial and parallel genetic algorithms. En: J.D. Schaffer (ed.), *Proceedings of Third International Conference on Genetic Algorithms*, págs. 70–79. Morgan Kaufmann, 1989.

[66] D.E. Goldberg. A note on boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing. *Complex Systems*, **4**:445–460, 1990.

[67] D.E. Goldberg y K. Deb. A comparative analysis of selection schemes used in genetic algorithms. En: G.J.E. Rawlins (ed.), *Foundations of Genetic Algorithms*, págs. 69–93. Morgan Kaufmann, 1991.

[68] D.E. Goldberg y Jr.R. Lingle. Alleles, loci and the travelling salesman problem. En: J.J. Grenfenstette (ed.), *Proceedings of the First International Conference on Genetic Algorithms*, págs. 154–159. Lawrence Erlbaum Associates, 1985.

[69] A.D. Gordon. *Classification. Methods for the exploratory analysis of multivariate data*. Chapman & Hall, 1981.

[70] A.D. Gordon. A review of hierarchical classification. *Journal of the Royal Statistical Society, Series A*, **150**(2):119–137, 1987.

[71] A.D. Gordon. Cluster validation. En: C. Hayashi N. Ohsumi K. Yajima Y Tanaka H.H. Bock y Y. Baba (ed.), *Data Science, Classification and Related Methods*, págs. 22–39. Springer, 1998.

[72] J.J. Grenfensttete, R. Gopal, B. Rosmaita y D. Van Gucht. Genetic algorithms for traveling salesman problem. En: J.J. Grenfenstette (ed.), *Proceedings of the First International Conference on Genetic Algorithms*, págs. 160–165. Lawrence Erlbaum Associates, 1985.

[73] R. Hanson, J. Stutz y P. Cheesman. Bayesian classification theory. Informe Interno FIA-90-12-7-01, NASA, Ames Research Center, 1990.

[74] A. Hardy y J.P. Rasson. Une nouvelle approche des problèmes de classification automatique. *Statistiques et Analyse des Données*, **7**(2):41–56, 1982.

[75] J.A. Hartigan. Representation of dissimilarity matrices by trees. *J. Amer. Statist. Ass.*, **62**:1140–1158, 1967.

[76] J. Hesser y R. Manner. Towards an optimal mutation probability for genetic algorithms. En: *Parallel Problem Solving from Nature. Lectures Notes in Computer Science*, volume 496, págs. 23–32. Springer-Verlag, 1990.

[77] J.H. Holland. *Adaptation in natural and artificial systems.* The University of Michigan Press, Ann Arbor, MI, 1975.

[78] L.J. Hubert. Monotone invariant clustering procedures. *Psychometrika*, **38**:47–62, 1973.

[79] D.L. Isaacson y R.W. Madsen. *Markov Chains Theory and Applications.* Wiley, New York, 1976.

[80] A.K. Jain y R.C. Dubes. *Algorithms for Clustering Data.* Prentice Hall, 1988.

[81] C.J. Jardine, N. Jardine y R. Sibson. The structure and construction of taxonomic hierarchies. *Math. Biosci.*, **1**:171–179, 1967.

[82] S.C. Johnson. Hierarchical clustering schemes. *Psychometrika*, **32**:241–245, 1967.

[83] D. R. Jones y M. A. Beltramo. Solving partitioning problems with genetic algorithms. En: R. Belew y L. B. Booker (eds.), *Proc. of the Fourth International Conference on Genetic Algorithms*, págs. 442–449. Morgan Kaufmann, 1991.

[84] D.R. Jones y M.A. Beltramo. Clustering with genetic algorithms. Informe Interno GMR-7156, Operating Sciences Department, General Motors Research Laboratories, 1990.

[85] K.A. De Jong. *An Analysis of the behavior of a class of genetic adaptive systems*. Tesis Doctoral. University of Michigan, 1975.

[86] K.A. De Jong, W.M. Spears y D.F. Gordon. Using markov chains to analyze GAFOs. En: D. Whitley y M.D. Vose (eds.), *Foundations of Genetic Algorithms 3*, págs. 115–138. Morgan Kaufmann, 1995.

[87] L. Kaufman y P.J. Rousseeuw. *Finding Groups in Data*. John Wiley & Sons, Inc., 1990.

[88] R.W. Kelin y R.C. Dubes. Experiments in projection and clustering by simulated annealing. *Pattern Recognition*, **22**:213–220, 1989.

[89] M.G. Kendall. Discirmination and classification. En: P.R. Krishnaiah (ed.), *Multivariate Analysis*, págs. 165–185. Academic Press, New York, 1966.

[90] F. Kettaf y J. Asselin de Beauville. Genetic and fuzzy based clustering. En: *IFCS-96, Proceedings of the fifth Conference of International Federation of Classification Societies*, volume I, págs. 100–103. Kobe, Japón, 1996.

[91] S. Kirpatrick, C.D. Jr. Gelatt y M.P. Vecchi. Optimization by simulated annealing. Informe Interno RC 9355, IBM Research Report, 1982.

[92] P.J.M. Van Laarhoven y E.H.L. Aarts. *Simulated Annealing: Theory and Applications*. D. Reidel Publishing Company, Dordretch, Holland, 1987.

[93] G.N. Lance y W.T. Williams. A general theory of classificatory sorting strategies I. Hierarchical systems. *Computer J.*, **9**:373–380, 1967.

[94] P. Larrañaga, M. Graña, A. D'Anjou y J.F. Torrealdea. Genetic algorithms elitist probabilistic of degree 1, a generalisation of simulated annealing. En: P. Torasso (ed.), *Advance in Artificial Intelligence*, volume 728, págs. 208–217. Springer-Verlag, 1993.

[95] P. Larrañaga, C.M.H. Kuijpers, R.H. Murga, I. Inza y S. Dizdarevic. Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review*, A Publicar.

[96] P. Larrañaga, C.M.H. Kuijpers, R.H. Murga y Y. Yurramendi. Learning bayesian networks structures by searching for the best ordering with genetic algorithms. *IEEE Transactions on Systems Man and Cybernetics*, **26**(7):487.493, 1996.

[97] P. Larrañaga y M. Poza. Structure learning of bayesian networks by genetic algorithms. En: E. Diday (ed.), *New Approches in Classification and Data Analysis*, págs. 300–307. Springer-Verlag, 1994.

[98] C.L. Lawson y R.J. Hanson. *Solving Least Squares Problems*. Prentice-Hall, 1974.

[99] I.C. Lerman. *Classification et analyse ordinale des données*. Dunod, Paris, 1981.

[100] F.T. Lin, C.Y. Kao y C.C. Hsu. Applying the genetic approach to simulated annealing in solving some NP-hard problems. *IEEE Transactions on Systems, Man, and Cybernetics*, **23**(6):1752–1767, 1993.

[101] J. A. Lozano y P. Larrañaga. Optimizing pyramidal clustering via genetic algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Enviado.

[102] J. A. Lozano, P. Larrañga, F.X. Albizuri y M. Graña. Genetic algorithms: Bridging the convergence gap. *Journal of Theoretical Computer Science*, A publicar.

[103] J. A. Lozano, P. Larrañga y M. Graña. Partitional cluster analysis with genetic algorithms: searching for the number of clusters. En: C. Hayashi N. Ohsumi K. Yajima Y Tanaka H.H. Bock y Y. Baba (ed.), *Data Science, Classification and Related Methods*, págs. 117–125. Springer, 1998.

[104] J.A. Lozano y P. Larrañaga. Aplicación de los algoritmos genéticos al problema del clustering jerárquico. *Revista Iberoamericana de Inteligencia Artificial*, A publicar.

[105] J.A. Lozano y P. Larrañaga. Applying genetic algorithms to search for the best hierarchical clustering of a dataset. *Pattern Recognition Letters*, Enviado.

[106] J.A. Lozano y P. Larrañaga. Using genetic algorithms to get the classes and their number in a partitional cluster analysis of large data sets. *Pattern Recognition*, Enviado.

[107] C.B. Lucasius, A.D. Dane y G. Kateman. On k-medoid clustering of large data sets with the aid of a genetic algorithm: background, feasibility and comparison. *Analytica Chimica Acta*, **282**:647–669, 1993.

[108] C.B. Lucasius y G. Kateman. Towards solving subset selection problems with the aid of genetic algorithms. En: R. Manner y B. Manderick (eds.), *Parallel Problem Solving from Nature II*, págs. 239–247. North-Holland, 1992.

[109] S. Lunchian, H. Lunchian y M. Petriuc. Evolutionary automated classification. En: D.B. Fogel (ed.), *Proceedings of The First IEEE Conference on Evolutionary Computation*, volume I, págs. 585–589. IEEE Computer Society Press, 1994.

[110] P. Macnaughton-Smith, W.T. Williams, M.B. Dale y L.G. Mockett. Dissimilarity analysis: a new technique of hierarchical subdivision. *Nature*, **202**:1034–1035, 1964.

[111] J.B. MacQueen. Some methods for classification and analysis of multivariate observations. En: *Proceedings of Fifth Berkeley Symposium*, volume 2, págs. 281–297, 1967.

[112] S.W. Mahfoud. Finite markov chain models of an alternative selection strategy for the genetic algorithm. *Complex Systems*, **7**:155–170, 1993.

[113] S.W. Mahfoud y D.E. Goldberg. Parallel recombinative simulated annealing: A genetic algorithm. *Parallel Computing*, **21**:1–28, 1995.

[114] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller y E. Teller. Equation of state calculations by fast computing machines. *Journal of Chem. Physics*, **21**:1087–1092, 1953.

[115] Z. Michalewicz. *Genetic Algorithms + Data Structures= Evolution Programs.* Springer-Verlag, Berlin Heidelberg, 1992.

[116] Z. Michalewicz y C.Z. Janikov. Handling constraints in genetic algorithms. En: R. Belew y L.B. Booker (eds.), *Proc. of the Fourth International Conference on Genetic Algorithms*, págs. 151–157. Morgan Kaufmann, 1991.

[117] R.S. Michalski y R.E. Stepp. Automated construccion of classifications: Conceptual clustering versus numerical taxonomy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **5**:219–243, 1983.

[118] R.S. Michalski y R.E. Stepp. Learning from observation: Conceptual clustering. En: R.S. Michalski, J.G. Carbonell y T.M. Mitchell (eds.), *Machine learning: An artificial intelligence approach.* Morgan Kaufmann, Los Altos, CA, 1983.

[119] G.W. Milligan y M.C. Cooper. An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, **50**(2):159–179, 1985.

[120] B. Mirkin. *Mathematical Classification and Clustering.* Kluwer Academic Publishers, 1996.

[121] M. Mitchell. *An Introduction to Genetic Algorithms*. The MIT Press, Cambridge, Massachusetts, 1996.

[122] I.R. Moraczewski, W. Borkowski y A. Kierzek. Clustering geobotanical data with the use of a genetic algorithm. *COENOSES*, **10**(1):17–28, 1995.

[123] R. Ngouenet. *Analyse géométrique des donnés de dissimilarité par le multidimensional scaling : une approche parallèle basée sur les algorithmes génétiques. Application aux séquences biologiques.* Tesis Doctoral. Université Rennes I, 1996.

[124] A.E. Nix y M.D. Vose. Modeling genetic algorithms with markov chains. *Annals of Mathematics and Artificial Intelligence*, **5**:79–88, 1992.

[125] L.M. Oliver, D.J. Smith y J.R.C. Holland. A study of permutation crossover operators on the tsp. En: J.J. Grenfenstette (ed.), *Proceedings of the Second International Conference on Genetic Algorithms and Their Applications*, págs. 224–230. Lawrence Erlbaum Associates, 1987.

[126] C.C. Peck y A.P. Dhawan. Genetic algorithms as global random search methods: An alternative perspective. *Evolutionary Computation*, **3**(1):39–80, 1995.

[127] J. Prawda. *Métodos y Modelos de Investigación de Operaciones.* Limusa, 1995.

[128] N.J. Radcliffe. Forma analysis and random respectful recombination. En: R. Belew y L.B. Booker (eds.), *Proc. of the Fourth International Conference on Genetic Algorithms*, págs. 222–229. Morgan Kaufmann, 1991.

[129] N.J. Radcliffe. Genetic set recombination. En: D. Whitley (ed.), *Foundations of Genetic Algorithms*, págs. 203–219. Morgan Kauffman, 1992.

[130] N.J. Radcliffe y F.A.W. George. A study in set recombination. En: S. Forrest (ed.), *Proc. of the Fifth International Conference on Genetic Algorithms*, págs. 23–30. Morgan Kaufmann, 1993.

[131] J.P. Rasson y T. Kubushishi. The gap test: an optimal method for determining the number of natural classes in cluster analysis. En: E. Diday, Y. Lechevallier, M. Schader, P. Bertrand y B. Burtschy (eds.), *New Approaches in Classification and Data Analysis*, págs. 186–193. Springer-Verlag, 1993.

[132] I. Rechenberg. *Evolutionstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Fromman-Holzboog, Stuttgart, 1973.

[133] C.R. Reeves. *Modern Heuristic Techniques for Combinatorial Optimization*. Blackwell Scientific Publications, 1993.

[134] C.R. Reeves. Using genetic algorithms with small populations. En: S. Forrest (ed.), *Proceedings of the Fifth International Conference on Genetic Algortihms*. Morgan Kaufman, 1993.

[135] C.R. Reeves. A genetic algorithm for flowshop sequencing. *Computer & Opns. Res.*, **22**:5–13, 1995.

[136] C.R. Reeves. *Comunicación personal*. 1997.

[137] W.S. Robinson. A method for chronologically ordering archeological deposites. *Amer. Aniquity*, **19**:293–301, 1951.

[138] G. Rudolph. Convergence analysis of canonical genetic algorithms. *IEEE Transactions on Neural Networks*, **5**(1):96–101, 1994.

[139] G. Rudolph. Massively parallel simulated annealing and its relation to evolutionary algorithms. *Evolutionary Computation*, **1**(4):361–382, 1994.

[140] G. Rudolph. *Convergence Properties of Evolutionary Algorithms*. Kovač, Hamburg, 1997.

[141] M. Sarkar, B. Yegnanarayana y D. Khemani. A clustering algorithm using evolutionary programming-based approach. *Pattern Recognition Letters*, **18**:975–986, 1997.

[142] J.D. Schaffer, R.A. Caruna, L.J. Eshelman y R. Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. En: J.D. Schaffer (ed.), *Proceedings of Third International Conference on Genetic Algorithms*, págs. 51–60. Morgan Kaufmann, 1989.

[143] H.-P. Shwefel. *Evolutionstrategie und numerische Optimierung.* Tesis Doctoral, Technische Universität, Berlin,1975.

[144] P. Sneath y R. Sokal. *Numerical Taxonomy.* Freeman, 1973.

[145] G. De Soete. A least square algorithm for fitting an ultrametric tree to a dissimilarity matrix. *Pattern Recognition Letters*, **2**:133–137, 1984.

[146] J.A. Storer, A.J. Nicas y J. Becker. Uniform circuit placement. En: P. Bertolazziy F. Luccio (ed.), *VLSI: Algorithms and Architectures*, págs. 255–273. Elsevier's Science Pub., Amsterdam, 1985.

[147] J. Suzuki. A markov chain analysis on simple genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, **25**(4):655–659, 1995.

[148] J. Suzuki. A further result on the markov chain model of genetic algorithms and its application to a simulated annealing-like strategy. *IEEE Transactions on Systems Man and Cybernetics*, A publicar.

[149] G. Syswerda. Schedule optimization using genetic algorithms. En: L. Davis (ed.), *Handbook of Genetic Algorithms*, págs. 332–349. Van Nostrand Reinhold, 1991.

[150] M.D. Vose. Modeling simple genetic algorithms. Informe Interno TN 37996–1301, Computer Science Dept., 107 Ayres Hall, The University of Tennesse, Knoxville, 1993.

[151] M.D. Vose y G.E. Liepins. Punctuated equilibria in genetic search. *Complex Systems*, **5**:31–44, 1991.

[152] D. Whitley. The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. En: J.D. Schaffer (ed.), *Proceedings of Third International Conference on Genetic Algorithms*, págs. 116–121. Morgan Kaufmann, 1989.

[153] D. Whitley y J. Kauth. GENITOR: A different genetic algorithm. En: *Proceedings of the Rocky Mountain Conference on Artificial Intelligence*, volume II, págs. 118–130, 1988.

[154] D. Whitley y T. Starkweather. GENITOR II: A distributed genetic algorithm. *Journal of Experimental and Theoretical Aritificial Intelligence*, **2**:189–214, 1990.

[155] L.D. Whitley. An executable model of a simple genetic algorithms. En: D. Whitley (ed.), *Foundations of Genetic Algorithms 2*, págs. 45–62. Morgan Kauffmann, 1992.

[156] W.T. Williams y J.M. Lambert. Multivariate methods in plant ecology I. Association analysis in plant communities. *Journal of Ecology*, **47**:83–101, 1959.

[157] W.L. Winston. *Operations Research*. Duxbury Press, 1991.

[158] D.H. Wolpert y W.G. Macready. No free lunch theorems for search. Informe Interno 95-02-10, Santa Fe Institute, 1995.