Introducción

La capacidad de aprender es una de las características centrales de la inteligencia, y esto hace que el aprendizaje haya formado parte de la investigación en Inteligencia Artificial desde prácticamente los orígenes de la misma.

Las aplicaciones del aprendizaje son hoy en día muy numerosas. En concreto, es importante destacar la gran aportación que los algoritmos de aprendizaje tienen en el desarrollo práctico de algunos sistemas basados en el conocimiento, ya que como es conocido, las técnicas de elicitación son de difícil aplicación en muchos casos. El aprendizaje permite, en estos casos, una extracción del conocimiento mucho más eficaz. Este conocimiento podrá ser posteriomente validado por un experto.

La utilización de la lógica difusa, por otra parte, ha demostrado ser una herramienta extraordinariamente útil en la descripción y modelación de muchos sistemas. La utilización de dicha lógica ha permitido la modelización cualitativa de sistemas. Esta modelización es en muchos más comprensible y simple que otras alternativas. Sin embargo, el éxito comercial de muchos sistemas basados en el conocimiento descrito mediante la lógica difusa hace necesario el desarrollo de nuevos modelos de aprendizaje que permitan adquirir tal conocimiento. Por supuesto, ya se ha iniciado el trabajo, y desde distintos puntos de vista se han aportado modelos de aprendizaje que permiten identificar sistemas difusos. En concreto, en este trabajo, nos proponemos como objetivo el desarrollar un modelo de aprendizaje de reglas difusas basado en la utilización de algoritmos genéticos.

Objetivos

El objetivo de esta memoria es el desarrollo de un algoritmo de aprendizaje inductivo con las siguientes características:

- sea capaz de obtener un conocimiento que, además de proporcionar una buena descripción del problema, sea fácilmente interpretable desde un punto de vista humano. En este sentido, las reglas difusas son una forma de representación que permite dotar a dicho conocimiento de un alto grado de comprensibilidad,

- obtenga un buen comportamiento ante sistemas complejos, es decir, ante sistemas con un número elevado de variables o ante sistemas fuertemente afectados por la imprecisión de los datos o por el ruido,
- sea aplicable a problemas cuya variable consecuente toma valores tanto sobre un referencial nominal o discreto como sobre un referencial continuo. En este sentido, el algoritmo que se desea desarrollar pasa a ocupar un vacío existente en el campo de los algoritmos de clasificación, ya que los algoritmos que se pueden encontrar en la bibliografía se han desarrollodo para abordar uno de estos dos problemas, presentando, en general, un mal comportamiento sobre el otro. Por consiguiente, se desea desarrollar una nueva metodología de aprendizaje que sea válida para ambos tipos de problemas de clasificación.

Además, en el proceso de búsqueda asociado al algoritmo, queremos explorar el uso de algoritmos genético en el desarrollo de sistemas de aprendizaje. En este sentido, exploraremos un nueva vía de aplicación de los algoritmos genéticos al problema de la clasificación, que sea una alternativa a los modelos Michigan y Pittsburgh, que permita reducir el espacio de búsqueda, y por consiguiente el tiempo de respuesta del sistema.

Descripción por capítulos

En el primer capítulo, situamos el entorno en el que se ha desarrollado este trabajo, definiendo los conceptos básicos que se utilizarán en el desarrollo del algoritmo. Además, se describen las metodologías más utilizadas en el desarrollo de algoritmos de clasificación, incluyendo aquellas que tienen como componente fundamental los algoritmos genéticos.

En la figura 1 se muestra la evolución que a lo largo de esta memoria se realizará del algoritmo de aprendizaje. Así, en el capítulo 2 se propone la metodología básica del algoritmo de clasificación, que llamaremos SLAVE, describiendo los elementos básicos que componen el sistema de aprendizaje.

La metodología propuesta está basada en la inclusión de un algoritmo genético como componente principal de un modelo de aprendizaje iterativo. Este modelo aprende de forma aislada cada una de las reglas necesarias para describir el sistema. Este hecho conlleva que las reglas obtenidas sean buenas reglas individuales, pero no contempla cómo es su funcionamiento con respecto al resto de las reglas que describen el conocimiento. Por esta razón, en el capítulo 3 sugerimos dos posibles vías para mejorar el comportamiento del algoritmo mediante la inclusión de mecanismos que permitan la colaboración en unos casos, y la competición en otros, de las reglas aprendidas. La primera de las vías propuestas consiste en la inclusión de un algoritmo de refinamiento de teorías como fase final del algoritmo de aprendizaje. Este algoritmo de refinamiento será el encargado de establecer

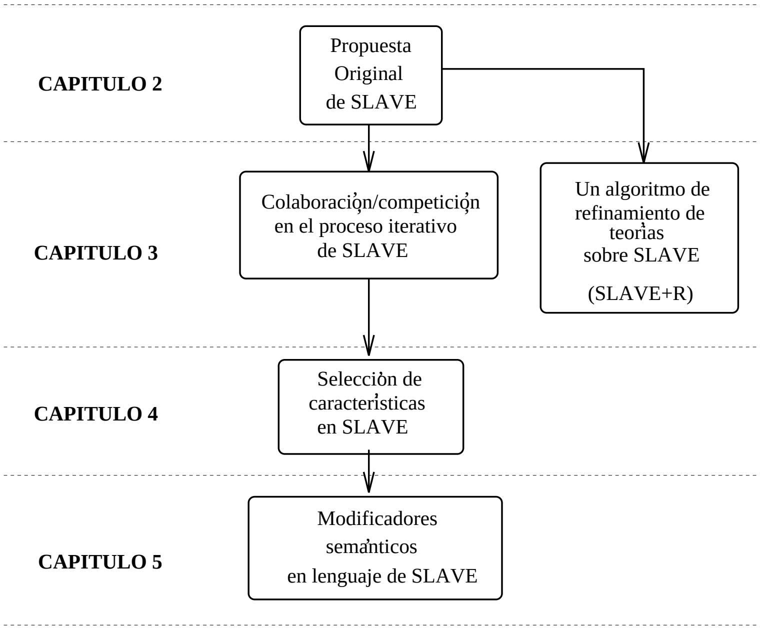

Figura 1: Evolución de SLAVE por capítulos

de forma conveniente la colaboración o competición entre las reglas, teniendo como objetivo no sólo mejorar la capacidad de predicción de las mismas, sino también de mejorar en la medida de lo posible la comprensibilidad del conocimiento extraído. Denominamos SLAVE+R a la inclusión del algoritmo de refinamiento de teorías propuesto como fase final del proceso de aprendizaje de SLAVE.

Como segunda vía, se contempla la posibilidad de que sea el propio algoritmo el que incorpore la colaboración/competición entre las reglas durante el proceso de aprendizaje. Con tal fin, se propone un nuevo mecanismo de evaluación de reglas, que contempla el conocimiento previamente extraído para los conceptos ya aprendidos, para determinar en qué medida las nuevas reglas que se proponen afectarán a la clasificación de los conceptos. La idea es en definitiva añadir más información al modelo iterativo en el que se basa SLAVE.

En el capítulo 4, se describe una nueva modificación de SLAVE con el fin de aumentar su capacidad de detección de variables irrelevantes. Así, en el módulo de selección de reglas se establecen dos procesos de búsqueda diferentes, uno que trata de encontrar el subconjunto de variables relevantes que formarán parte de la descripción de la regla, y otro que intenta encontrar la mejor asignación de valores a estas variables. Estas dos búsquedas se llevan a cabo de forma simultánea mediante la ejecución de forma paralela de dos algoritmos genéticos que mediante coevolución tratan de encontrar la mejor descripción para un concepto. Así, se consigue no sólo agilizar la selección, sino que además, se aumenta el nivel de exploración del espacio de los subconjuntos de variables relevantes. En este capítulo se proponen para su estudio, tres formas posibles de llevar a cabo esta modificación.

En el capítulo 5 nos planteamos finalmente la posibilidad de enriquecer el lenguaje de representación de SLAVE incluyendo modificadores semánticos. Con esta modificación intentaremos dotar al algoritmo de la capacidad de alterar la semántica de sus etiquetas iniciales, de manera que estas respondan lo más posible a la información que contienen los ejemplos.

La parte experimental de esta memoria se ha hecho seleccionando un conjunto de bases de datos y una serie de algoritmos de aprendizaje para comprobar el comportamiento de SLAVE. Las bases de datos estudiadas y los algoritmos de aprendizaje utilizados se describen en los apéndices A, B y C. En cada capítulo se recoge una parte experimental con el fin de probar cada nuevo sistema diseñado y de justificar la siguiente modificación del algoritmo.

Capítulo 1

Preliminares

1.1 El aprendizaje inductivo de conceptos

La adquisición de conocimiento es uno de los mayores problemas que se ha encontrado en el desarrollo de los sistemas basados en el conocimiento. Los elementos básicos que aparecen en los métodos tradicionales de elicitación son: **el experto**, que normalmente es una persona o conjunto de personas de las cuales se desea elicitar el conocimiento, y **el elicitador**, que es la persona o conjunto de personas que mediante técnicas de entrevista tratan de adquirir dicho conocimiento. Otros elementos importantes son **el modelo de comunicación**, que especifica la estructura en la cual el conocimiento se transmite entre experto y elicitador, y **la representación del conocimiento**, que indica la forma en la que el conocimiento adquirido se almacena. La figura 1.1 ([39]) muestra el proceso de transmisión del conocimiento:

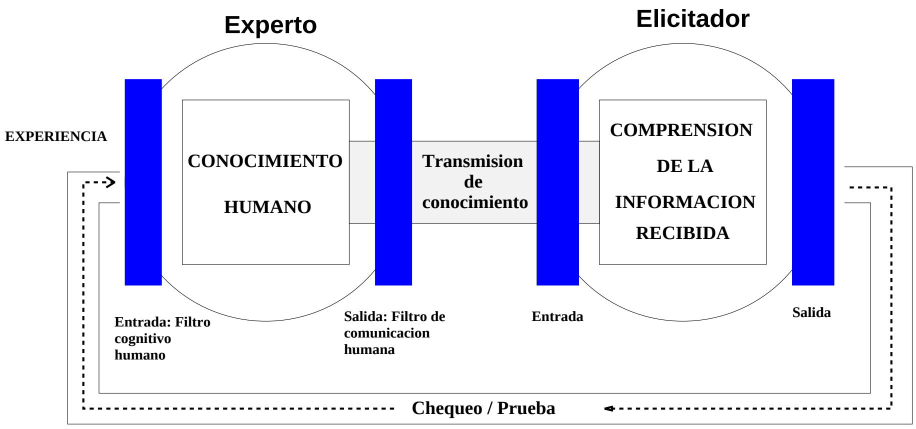

Figura 1.1: Proceso de transmisión de conocimiento

La elicitación de este conocimiento utilizando técnicas de entrevista [39] es una tarea compleja que consume mucho tiempo. Una alternativa a estos métodos tradicionales, que ha tenido recientemente un amplio desarrollo, consiste en la utilización de técnicas de aprendizaje automático [114, 115, 118, 125].

Uno de los modelos más estudiados es el aprendizaje inductivo, que consiste en un proceso de adquisición de conocimiento mediante la aplicación de inferencias inductivas sobre un conjunto de hechos. El conjunto de hechos representa el modelo de comunicación del conocimiento que, o bien es propuesto basándose en la experiencia de un experto, o bien se extrae directamente del sistema que se desea aprender. El algoritmo de aprendizaje toma el papel del elicitador usando para ello un modelo de representación del conocimiento. Un diagrama general de estos modelos se describe en la figura 1.2.

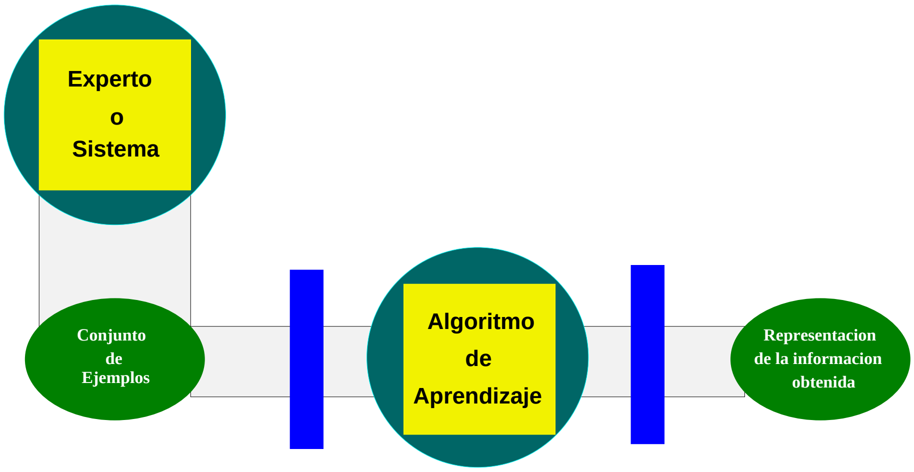

Figura 1.2: Diagrama de un modelo de aprendizaje inductivo

La inducción, que es una de las formas más comunes de aprendizaje, se basa en un fundamento bastante débil, ya que en principio, la adquisición de conocimiento no se puede validar completamente. Las aserciones obtenidas mediante inducción producen un número potencialmente infinito de hipótesis, sobre un número finito de comprobaciones que se pueden realizar.

La inducción tradicional trata de establecer cual es el mejor criterio para guiar la solución de las aserciones inductivas y cómo éstas se pueden confirmar. La búsqueda de respuestas a tales preguntas se ha convertido en un campo de discusión entre filósofos y lógicos. En este sentido, muchos filósofos tales como Karl Popper, Beltrand Russell y George Polya afirmaron que es imposible que una máquina aprenda correctamente haciendo uso de la inducción. Debido al gran desarrollo producido en los ordenadores modernos y a las últimas investigaciones en el campo de la inteligencia artificial, las visiones pesimitas mencionadas anteriormente se estan revisando. En la actualidad se ha comprobado que los ordenadores no sólo son capaces de almacenar una gran cantidad de conocimiento sino que también son capaces de trabajar con él.

Actualmente, el problema del aprendizaje inductivo se ha simplificado en el siguiente sentido: hay que dotar a las máquinas de la capacidad de poder proporcionar hipótesis plausibles, mientras que son los seres humanos los encargados de verificar su validez.

Frente a la deducción, la inferencia inductiva parte de un conjunto de hechos o ejemplos más que de un conjunto de axiomas. Su objetivo consiste en formular hipótesis que expliquen los hechos dados y sean capaces de predecir nuevos hechos. En otras palabras, la inferencia inductiva trata de obtener una descripción completa y correcta de un fenómeno dado, a partir de un conjunto de observaciones específicas de ese fenómeno o parte de él.

De los dos aspectos de la inferencia inductiva, la generación de hipótesis plausibles y su validación, sólo el primero de ellos es el cuerpo de estudio de los investigadores en el campo del aprendizaje inductivo.

En base a lo descrito anteriormente, podemos realizar la siguiente formulación del paradigma de la inducción propuesta por Michalski R.S. en [114]:

Dados

- *Un conjunto de hechos* que representan el conocimiento específico sobre objetos, situaciones o procesos,
- *una hipótesis inductiva inicial* que puede ser nula,
- *y un conocimiento sobre el problema o conocimiento subyacente (background knowledge)* que define las restricciones y afinidades impuestas sobre las observaciones dadas, y permite generar hipótesis inductivas candidatas, y que además, puede incluir alguna información sobre la teoría del dominio que permite a su vez definir criterios de preferencia para caracterizar las propiedades deseables de la hipótesis inducida.

Encontrar

- *Una hipótesis* que tautológicamente o débilmente implique las observaciones dadas y satisfaga las restricciones y preferencias impuestas por el conocimiento subyacente.

Definición 1.1 *Una hipótesis H implica tautológicamente a un conjunto de hechos F si y sólo si F es una consecuencia lógica de H, es decir, H=>F.*

La consecuencia lógica, H=>F se puede expresar de la siguiente manera:

- H especifica a F
- F generaliza a H

Derivando el conjunto de hechos F de la hipótesis H (inferencia deductiva), la fórmula H=>F preserva la veracidad, es decir, si H es cierta, F es una consecuencia lógica de H. Si por el contrario, H se ha derivado de F (inferencia inductiva), entonces la fórmula H=>F no preserva la veracidad, aunque sí preserva la falsedad, es decir, si algunos hechos hacen falso a F, entonces estos hechos hacen falso también a H.

Para algunos problemas puede no existir ninguna hipótesis que implique tautológicamente al conjunto de hechos. Para solventar estas situaciones, aparece la noción de *implicación débil.*

Definición 1.2 *Diremos que H implica débilmente a F, si no todos los hechos de F son una consecuencia lógica de H, pero sí al menos un subconjunto de F no vacío verifica la implicación tautológica.*

Admitiendo la implicación débil, este paradigma incluye métodos para generar hipótesis "blandas", que permiten explicar con un cierto grado de incertidumbre o de forma parcial un determinado fenómeno.

Dado un conjunto de hechos, existe un número potencialmente infinito de hipótesis que implican estos hechos. Por tanto, se hace imprescindible un conocimiento subyacente que permita restringir el espacio de búsqueda, así como criterios de preferencia para reducir el número infinito de posibles elecciones.

Una forma típica para definir tales criterios, consiste en especificar las propiedades deseables de la hipótesis, tales como:

a) Requerir que la hipótesis sea la descripción consistente más corta o más económica que se deduce de los hechos. Este tipo de restricción se conoce por *sesgo en la elección* y es utilizado en lenguajes de descripción completos tales como el propuesto en [108].

b) Otra opción es usar, para expresar las hipótesis, lenguajes de descripción restringidos [117]. Esta segunda opción se conoce como *sesgo en el lenguaje.*

1.1.1 Tipos de aprendizaje inductivo

1.1.1.1 La adquisición de conceptos frente a la generalización descriptiva

Los máximos exponentes del aprendizaje inductivo son: el aprendizaje por ejemplos, también llamado *adquisición de conocimiento* y el aprendizaje por observación, también conocido por *generalización descriptiva.* Para el primero, las observaciones son caracterizaciones de algunos objetos clasificados previamente por un profesor en una o varias clases (conceptos). La hipótesis de inducción se puede ver como una regla de reconocimiento de un concepto. El segundo de ellos consiste en determinar una descripción general, ley o teoría que caracterize a un conjunto de objetos.

Como ejemplo, presentamos diferentes problemas pertenecientes a cada una de estas dos categorias:

1. **Adquisición de conocimiento**:

   – Aprender una *descripción característica* de una clase de objetos que especifique todas las propiedades comunes de los objetos conocidos de la clase. Algunos trabajos sobre este problema son [9, 20, 67, 110, 117, 158].

   – Aprender una *descripción discriminante* de una clase de objetos que permita distinguir la clase dada de un número finito de otras clases. Este problema se aborda en [108, 110, 124].

   – Inferir secuencias de reglas de extrapolación que permiten predecir el siguiente elemento en una secuencia dada por ejemplo [32, 142].

2. **Generalización descriptiva**:

   – Formular una teoría caracterizando una colección de entidades. Por ejemplo, una teoría de números, como en [98].

   – Descubrir patrones en datos obtenidos mediante observación, como por ejemplo en los trabajos [65, 146].

   – Determinar una descripción taxonómica de una colección de objetos, como por ejemplo en [112, 113].

En esta memoria nos centraremos en el problema de la adquisición de conocimiento, donde el conjunto de hechos (o ejemplos) que describen un fenómeno, se pueden ver como un conjunto de implicaciones del tipo:

$$F : \{e_{ik} \rightarrow C_i\}, \ i \in I$$

donde e_{ik} es la descripción del k-ésimo ejemplo de la clase C_i, y I es un conjunto de índices para las clase C_i.

Una hipótesis inductiva H se puede caracterizar como un conjunto de reglas de reconocimiento:

$$H : \{D_i \rightarrow C_i\}, \ i \in I$$

donde D_i es una descripción de la clase C_i, es decir, una expresión de condiciones de tal forma que cuando se satisfaga esta expresión para un objeto, podemos considerar a dicho objeto como una instancia de la clase C_i.

De acuerdo con la definición de hipótesis inductiva, H especifica a F, y sustituyendo H y F por las anteriores expresiones y haciendo algunas transformaciones apropiadas, obtenemos las dos siguientes condiciones que se deben satisfacer para que se verifique la definición de la hipótesis inductiva:

$$(a) \qquad \forall i \in I,\ (E_i \ \ implica \ \ D_i)$$

$$(b) \qquad \forall i, j \in I,\ (D_i \ \ implica \ \ \overline{E_j}),\ si\ j \neq i$$

donde E_i es una descripción que verifican todos los ejemplos del conjunto de entrenamiento de la clase K_i y sólo estos ejemplos.

La expresión (a) se conoce como **condición de completitud** y la expresión (b) como **condición de consistencia**. Para que una regla de reconocimiento de conceptos sea aceptable, se requiere que la hipótesis inductiva verifique estas dos condiciones. Las condiciones de completitud y consistencia proporcionan el fundamento lógico de los algoritmos para el aprendizaje de conceptos mediante ejemplos.

1.1.1.2 Descripción característica frente a la descripción discriminante

La condición de completitud y la condición de consistencia permiten explicar claramente la diferencia entre estos dos tipos de descripciones. Una descripción característica para una clase de objetos es una expresión que verifica la condición de completitud. Normalmente, esta descripción es una conjunción de algunas propiedades simples, comunes a todos los objetos de la clase. Desde un punto de vista práctico, las más interesantes son *las descripciones características maximales* (las más largas) que son las más especificas. Tales descripciones intentan discriminar la clase dada del resto de las clases.

Una descripción discriminante satisface tanto la condición de completitud como la condición de consistencia y especifica uno o más caminos para distinguir la clase dada de un número fijo de otras clases. Las más interesantes son *las descripciones discriminantes minimales* (las más cortas) que son las más generales. Estas descripciones intentan minimizar la información necesaria para identificar la clase dada entre un número fijo de otras clases.

1.1.1.3 Aprendizaje de un único concepto frente al aprendizaje de múltiples conceptos

Respecto a los ejemplos contenidos en el conjunto de entrenamiento podemos distinguir dos tipos básicos de aprendizaje, el aprendizaje de un único concepto y el aprendizaje de una colección de conceptos. En el primer caso, podemos distinguir las dos siguientes situaciones:

a) El conjunto de entrenamiento sólo contiene instancias del concepto que se debe aprender, es decir, sólo hay ejemplos positivos. En este caso no es aplicable la condición de consistencia por existir una única clase. Por consiguiente, tampoco existe un límite que permita detener la generalización de una descripción. Los algoritmos que abordan este tipo de problemas, imponen ciertas restricciones sobre las descripciones. Las más comunes son:

   – imponer que sean descripciones características maximales, como en [67, 158].

   – imponer que la descripción no exceda un cierto grado de generalidad, como en [147].

b) En el conjunto de entrenamiento hay ejemplos y contraejemplos de un determinado fenómeno. En este caso los contraejemplos se pueden tratar como ejemplos pertenecientes a una clase diferente, por lo que es aplicable la condición de consistencia y por tanto, existe un límite para generalizar las descripciones [167, 169].

En el problema del aprendizaje de múltiples conceptos, también podemos distinguir dos subtipos en función de la naturaleza de las descripciones deseables:

a) En un primer caso, se impone como restricción que todas las descripciones D_i sean disjuntas entre sí, es decir,

$$\bigcap_{i=1}^{n} D_i = \emptyset.$$

b) En un segundo caso, se permite el solapamiento entre las descripciones. En este caso, puede ocurrir que un ejemplo verifique más de una descripción. Este hecho, puede ser deseable para algunos problemas, por ejemplo, supongamos que tenemos un paciente que tiene dos enfermedades distintas, por tanto, debe ser miembro de dos clases diferentes. Cuando esto ocurre, la condición de consistencia no es aplicable. Esta situación especial requiere una reformulación de la regla de inducción, para la cual se han propuesto diferentes soluciones [26, 108, 109].

1.1.2 El conocimiento subyacente

Como mencionamos anteriormente, dado un conjunto de observaciones se puede construir un número potencialmente infinito de hipótesis inductivas que implican a dichas observaciones. Por consiguiente, se hace necesaria la incorporación de información adicional, *conocimiento sobre el problema*, para restringir el espacio de posibles descripciones y localizar las más deseables.

Ahora, describiremos los componentes más usuales introducidos como conocimiento subyacente y detallaremos algunos de ellos. Estos componentes son:

- Información sobre los descriptores.
- Criterios de preferencia que especifican las propiedades deseables de las descripciones.
- Incorporación de heurísticas y procedimientos especiales que permitan al sistema de aprendizaje generar nuevos conceptos.

Un problema subyacente, fundamental para el desarrollo de sistemas automáticos de aprendizaje inductivo, es qué información recibe la máquina y qué información se espera que la máquina aprenda. El conjunto de datos de entrada al sistema es la componente más importante del mismo, como propone el propio paradigma de inducción. Los descriptores utilizados en los datos son características observables y medidas disponibles de los objetos. Los descriptores que formarán parte de las descripciones finales pueden ser seleccionados por un profesor siguiendo algún criterio de relevancia sobre el problema específico, o bien, dados todos los descriptores disponibles, dejar que el algoritmo de aprendizaje seleccione aquellos que son relevantes para el problema concreto. En este segundo caso, el papel fundamental del aprendizaje inductivo consiste en obtener las propiedades esenciales de los objetos, también conocido como **selección de características**.

Con respecto a la capacidad de selección de características, se propone en [115] la siguiente clasificación de los algoritmos de aprendizaje:

- **Relevancia completa:** En este caso todos los descriptores proporcionados al sistema se consideran como relevantes para la tarea de aprender.
- **Relevancia parcial:** Las observaciones pueden contener un número elevado de descriptores irrelevantes o redundantes, siendo algunos de ellos relevantes. La tarea de aprender consiste en seleccionar los más relevantes y construir a partir de ellos descripciones apropiadas. Este tipo de sistemas de aprendizaje son llamados *sistemas de aprendizaje inductivo selectivo*. Algunos sistemas de esta clase son AQ11 [109] e ID3 [125].
- **Relevancia indirecta:** Las observaciones pueden contener descriptores que no son relevantes de forma directa, pero que a partir de algunos de los descriptores iniciales pueden construirse nuevos descriptores que sean relevantes de forma directa. En este caso, la tarea de aprender consiste en construir nuevos descriptores a partir de los iniciales y formular en base a ellos descripciones apropiadas. Estos sistemas son también llamados *sistemas de aprendizaje inductivo constructivo*. Algunos sistemas de esta clase son BACON [94], INDUCE [97] y AM [98].

Estos tres casos representan situaciones que progresivamente van restando importancia a la relevancia de los descriptores iniciales, (es decir, menos trabajo para la persona que define el problema) y más demanda al sistema de aprendizaje.

Además, los descriptores almacenan información del problema que se debe considerar durante el proceso de aprendizaje. Esta información puede ser:

(1) Una especificación del dominio y del tipo del descriptor.

(2) Una especificación de los operadores que se le pueden aplicar.

(3) Una especificación de las restricciones y las relaciones entre este descriptor y otros descriptores.

(4) Para descriptores numéricos, el significado, la varianza o una distribución completa de probabilidad sobre los valores que puede tomar.

(5) Una caracterización de los objetos a los cuales se les puede aplicar este descriptor.

(6) Sinónimos que pueden usarse para denotar este descriptor.

(7) Una definición de este descriptor en base a otros descriptores.

De entre todas estas informaciones que pueden incluirse como parte de un descriptor, a continuación detallaremos brevemente la (1) y la (3).

Para un problema concreto, normalmente es posible determinar el conjunto de valores que puede tomar cada descriptor. Al conjunto de estos valores se le conoce con el nombre de *dominio*. El dominio se usa para restringir el nivel de generalización que admite un descriptor.

Otra información importante para restringir el proceso de generalización está relacionado con la estructura de su dominio, es decir, con las relaciones existentes entre los elementos que lo constituyen. Atendiendo a la estructura interna de un dominio, podemos distinguir tres tipos de descriptores:

- **Nominales:** El conjunto de valores que toman los descriptores está formado por símbolos o nombres independientes.
- **Lineales:** El conjunto de valores de los descriptores lineales está completamente ordenado. Variables medidas sobre ordinales, intervalos o escalas son casos particulares de este tipo de descriptores.
- **Estructurados:** El dominio de los descriptores posee una estructuración jerarquizada que refleja las relaciones existentes entre sus valores.

Algunas otras informaciones importantes vienen impuestas en forma de restricciones. Para un problema concreto, algunas descripciones no se pueden aceptar como admisibles debido a propiedades específicas de los descriptores y a las relaciones que se establecen entre ellos. Algunos ejemplos de estas restricciones son:

- *Interdependencia entre valores.* En problemas prácticos, algunas variables especifican un estado de un objeto, mientras otras variables especifican características sobre ese estado. Dependiendo del valor de las variables de estado, el valor de una variable que caracteriza el estado, puede ser o no necesario.
- *Propiedades de los descriptores.* Descriptores que especifican relaciones entre objetos pueden poseer ciertas propiedades generales: simetría, transitividad, reflexividad, etc. Este tipo de información se puede incorporar en forma de regla para la obtención de nuevos hechos.
- *Interrelación entre descriptores.* En algunos problemas pueden existir relaciones entre descriptores que restringen sus valores.

En [114] se ofrecen normas generales para la incorporación de otra información como conocimiento subyacente de un problema.

1.2 Adquisición de conocimiento difuso

Uno de los objetivos principales que se plantean en este trabajo, es el desarrollo de un algoritmo que sea capaz de expresar el comportamiento de un sistema en términos difusos. Fundamentalmente, dos razones motivan este objetivo: la primera de ellas es que los modelos difusos han tenido un exito evidente en la modelización de sistemas. Tanto es así, que hoy día podemos ver la aparición de productos que incluyen en su diseño este tipo de modelos. La segunda razón, es que a menudo, la forma en que los humanos transmiten su conocimiento adopta una forma muy parecida a la forma en la cual representa su conocimiento los sistemas difusos. Por consiguiente, la representación mediante difusos de la información de un sistema aumenta la compresibilidad del mismo desde el punto de vista humano.

Por ejemplo, términos como *grande, alto, pequeño* y *mediano* son corrientes en una conversación para describir el tamaño de un objeto. Estos términos imprecisos son conocidos como *etiquetas lingüísticas* si están asociados a una semántica que los dota de significado. Las variables que toman estos términos son conocidas como *variables lingüísticas*. Por tanto, es interesante para muchos problemas la utilización de descripciones expresadas en términos de variables lingüísticas, sobre todo si el fin último del aprendizaje es que la información extraída sea fácilmente interpretable por un ser humano.

Formalmente, la definición de variable lingüística fué propuesta por Zadeh en [172], apoyándose en las propiedades de la teoría de subconjuntos difusos [171].

La teoría de subconjuntos difusos se basa en la relajación del concepto de pertenencia de un elemento a un conjunto. En la teoría tradicional, un elemento pertenece o no a un conjunto, mientras que en la teoría de subconjuntos difusos, un elemento pertenece a un conjunto en un cierto grado.

Una forma de representar las descripciones en términos lingüísticos lo constituyen las reglas difusas. Una regla difusa consta de un parte antecedente, descripción o condición y una parte consecuente o de clasificación. La estructura básica de un regla difusa es la siguiente:

$$\text{IF } X_1 \text{ is } A_1 \text{ and } \ldots \text{ and } X_n \text{ is } A_n \text{ THEN } Y \text{ is } B$$

donde cada variable X_i tiene un conjunto referencial U_i y toma valores en un dominio difuso D_i para $i \in \{1, \ldots, n\}$. El conjunto referencial para Y es V y su dominio es F. El valor de la variable Y es B, donde $B \in F$ y el valor de la variable X_i es A_i tal que $A_i \in D_i$.

Las reglas difusas, por tanto, se presentan desde el punto de vista humano, como una herramiento útil y confortable para la incorporación de conocimiento en modelos computacionales. Por consiguiente, como modelos válidos para la construcción de sistemas expertos.

Frente a la estructura básica de regla propuesta anteriormente, en la literatura podemos encontrar otras propuestas alternativas tal como el modelo de regla TSK [23] que se aplicado fundamentalmente al control de procesos.

En este trabajo, se propondrá un modelo de regla que generaliza el modelo básico propuesto anteriormente, que resulta ser más útil que otros modelos por dos razones: el número de reglas necesarias para expresar el comportamiento de un sistema es menor, y aumenta significativamente la comprensibilidad del conocimiento extraído.

La estructura de regla que aquí se propone, es la extensión a la utilización de etiquetas lingüísticas de un modelo de regla ampliamente utilizado en diferentes problemas de clasificación conocido como modelo DNF modificado. Además, en el capítulo 5 de esta memoria, generalizamos aún más este modelo de regla para incluir modificadores semánticos. Esta extensión esta basada en la definición de variable lingü'istica booleana propuesto en [172]. La inclusión de los modificadores semánticos dota de un mayor grado de flexibilidad al proceso de aprendizaje, y por tanto de una mejor aproximación entre el comportamiento del sistema y el conocimiento obtenido.

En el desarrollo de esta memoria, se propondrán dos modelos de razonamiento [34, 55] asociados a este modelo de regla que se adaptan a los dos diferentes tipos problemas de clasificación, que se pueden presentar en función del tipo de dominio asociado a la variable de clasificación. En este sentido, distinguimos problemas cuya variable de clasificación tiene asociado un referencial nominal o discreto, que denominaremos caso crisp, y problemas cuya variable de clasificación tiene un referencial continuo que discretizaremos utilizando conjuntos difusos. En el primer caso, a diferencia del segundo, el dominio asociado a la variable de clasificación toma valores no difusos. Este hecho, obligará a modificar la conducta de los algoritmos de aprendizaje en función del tipo de dominio asociado a la variable de clasificación. La justificación de este cambio de conducta se expondrá detenidamente en el siguiente capítulo de esta memoria.

1.3 Descripción de algunos algoritmos de aprendizaje

De entre los distintos tipos de aprendizaje inductivo existentes en la bibliografía, destacan por su importancia dos grupos de algoritmos: aquellos que utilizan árboles de decisión para representar el conocimiento adquirido y que normalmente utilizan una estrategia de divide y vencerás descendente y aquellos que utilizan una estrategia ascendente basada en operaciones de generalización y especificación. A continuación describimos las características más relevantes de estos grupos.

1.3.1 Basados en árboles de decisión

Dentro de los algoritmos que usan árboles de decisión como forma de representación del conocimiento adquirido, podemos considerar las tres siguientes familias:

- La familia ID3, cuyos máximos exponentes son el propio programa ID3 desarrollado por Quinlan [125] y el programa CLS de Hunt [81] que se puede considerar como el preculsor de esta filosofía de aprendizaje.
- La familia de árboles de regresión, cuyo máximo exponente es CART, desarrollado por Breiman et al. [13].
- La familia compuesta por otros métodos que utilizan criterios estadísticos ([140]).

Todos estos sistemas se caracterizan por emplear una estrategia de divide y vencerás en forma descendente que particiona un conjunto de ejemplos dado, en subconjuntos cada vez más pequeños a medida que el árbol crece. La diferencia fundamental entre las tres familias es el criterio de selección utilizado para subdividir el conjunto de ejemplos en cada paso del algoritmo.

Estos algoritmos de aprendizaje utilizan criterios de selección de características para elegir el atributo por el cual dividir el conjunto de ejemplos. El conjunto de ejemplos se divide progresivamente usando los atributos seleccionados hasta que los ejemplos de cada subconjunto generado pertenecen a una única clase. A los subconjuntos que verifican esta condición se les conoce como *hojas* del árbol, mientras aquellos que no lo verifican se les conoce como *nodos*. Cada hoja del árbol está asociada a un concepto, que corresponde a la clase de los ejemplos que pertenecen a dicho subconjunto.

De las tres familias anteriores, detallaremos a continuación la primera que es la más referenciada en el campo de la inteligencia artificial y sobre la que actualmente hay una gran cantidad de investigadores trabajando sobre esta filosofía de aprendizaje. Para más información sobre la segunda y tercera familia ver [140].

El algoritmo ID3 (Induction of Decision Trees) de Quinlan [125, 126] es el descendiente más fiel del originario algoritmo CLS (Concept Learning System) de Hunt [81] y el más conocido de los algoritmos de aprendizaje que usan árboles de decisión.

ID3 construye un árbol de decisión de forma descendente (es decir, partiendo de los descriptores hacia los ejemplos) siendo guiado por una medida de información que se establece sobre los ejemplos. La estructura básica del algoritmo es iterativa. El objetivo consiste en construir un árbol que sea capaz de clasificar correctamente todos los ejemplos del conjunto de entrenamiento. Para ello, selecciona un atributo y en base a él divide el conjunto de ejemplos. Si este atributo es suficiente para clasificar todos los ejemplos (es decir, todos los subconjuntos generados por este atributo son hojas) el proceso termina. En otro caso, se selecciona otro atributo y se repite el proceso.

La selección del siguiente atributo por el cual se debe seguir dividiendo el conjunto de ejemplos es la componente fundamental de este algoritmo. Con este fin, se define un criterio de selección basado en una medida de información. Este criterio nos dice la ganancia de información obtenida si se selecciona un determinado atributo. El atributo más relevante será aquel que mayor información proporcione.

Las ventajas principales de este algoritmo son su simplicidad y su bajo costo computacional. Por contra, este algoritmo presenta también algunos inconvenientes importantes como:

- Tiene grandes problemas para aprender coherentemente ante la presencia de ruido, es decir, es muy vulnerable al ruido.
- Tampoco trabaja correctamente cuando aparecen valores desconocidos en los ejemplos.
- Tiene poca capacidad de generalización, lo que ocasiona la aparición de las llamadas hojas vacías (*null leaf*). Las hojas vacías son hojas que representan a subconjuntos vacíos de ejemplos, y por tanto, no tienen ninguna clase asociada. Esto provoca que no se puedan clasificar ejemplos que pertenecen a estos subconjuntos de ejemplos, por no aparecer en el conjunto de entrenamiento.
- El hecho de no soportar bien el ruido y la inconsistencia de algunos datos en el conjunto de ejemplos provoca la aparición de las llamadas hojas muertas (*dead-end leaf*). Este tipo de hojas aparece cuando se han explorado todos los atributos y alguno de los subconjuntos finales contiene ejemplos de más de una clase.
- La necesidad de precisión total del algoritmo obliga a que todos los ejemplos tienen

que estar correctamente clasificados, lo que puede provocar árboles de decisión demasiado grandes que complican la comprensión de los mismos.

- El lenguaje utilizado para el aprendizaje es demasiado pobre, y no contempla las posibles relaciones existentes entre distintos atributos.

- Es incapaz de trabajar con atributos continuo-valuados. Sólo trabaja con atributos o variables cuyo dominio es nominal.

El algoritmo ID3 supuso un punto de partida válido e interesante para el aprendizaje automático de conceptos y en base a él han aparecido en la literatura nuevas propuestas que intentan corregir sus deficiencias. Entre estas nuevas propuestas, podemos destacar el sistema de aprendizaje ACLS (Analogue Concept Learning System) que es una generalización de ID3 que admite como valores para los atributos un número ilimitado de valores enteros y en una filosofía semejante de generalización [18, 154]. El sistema ASSISTANT [17] es el primer descendiente de ID3 que trabaja con atributos continuo-valuados. Otra cualidad que introduce este sistema es la posibilidad de determinar y construir clases jerárquicas, es decir, que una clase puede ser una división más fina de una clase superior.

La teoría de subconjuntos difusos se ha utilizado como método para resolver el problema que supone trabajar con atributos continuo-valuados para la construcción de árboles de decisión. Los sistemas Fuzzy-ID3 [160] y FIDMV [86] constituyen dos exponentes de esta forma de resolver el problema, donde los atributos continuos son discretizados usando variables lingüísticas. Esta vía para dar respuesta al problema de los atributos continuo-valuados ha motivado un creciente interés de los investigadores de este campo que ha ocasionado la aparición de multitud de mecanismos para mejorar el comportamiento de estos sistemas. Entre estos trabajos, destacamos los métodos de poda definidos para este tipo de árboles que se proponen en [6].

Para resolver algunos problemas de ID3, el propio Quinlan propone el sistema C4 [127] y más tarde el C4.5 [128], donde además de incorporar métodos para trabajar con variables continuas, introduce criterios de poda para aumentar tanto la comprensibilidad del árbol como su precisión. Este sistema lo usaremos en la parte experimental para comparar su comportamiento con el algoritmo que se desarrollará a lo largo de esta memoria.

Otras propuestas destacables son los sistemas ID4 e ID5 [148] que transforman a ID3 en un sistema de aprendizaje incremental, así como el trabajo de Uthurusamy [154] que introduce mecanismos para eliminar la sensibilidad al ruido de los árboles de decisión.

1.3.2 Basados en la metodología STAR

Frente a los sistemas que utilizan una estrategia divide y vencerás descendente donde el conocimiento se representa mediante árboles de decisión, están los sistemas de aprendizaje que utilizan la estrategia de generalización y especificación de forma ascendente. La mayoría de estos sistemas representan el conocimiento mediante reglas de producción. Uno de los máximos exponentes de este tipo de sistemas es AQ11 [109] diseñado por Michalski y que siguen la filosofía STAR. Los antecedentes de esta metodología se encuentran en los trabajos de Winston [167] con su programa ARCH y el algoritmo de eliminación de candidatos propuesto por Mitchell [116].

El programa ARCH describe un mecanismo para aprender conceptos basado en la búsqueda de relaciones entre las representaciones mediante redes semánticas en el mundo de bloques. En este programa son importantes dos procesos:

(i) encontrar y explorar las semejanzas existentes entre las descripciones estructurales para un mismo tipo de configuración,

(ii) encontrar diferencias significativas entre los ejemplos positivos y los ejemplos negativos.

Estos dos procesos corresponden en la terminología anteriormente descrita en este capítulo a la generalización y a la especificación. El programa ARCH generaliza las descripciones para cubrir todos los ejemplos positivos y las especifica para eliminar todos los ejemplos negativos. Si al sistema se le presenta un nuevo ejemplo, si este es positivo se generalizan las descripciones para cubrirlo, si por contra es negativo, se especifican las descripciones existentes.

El algoritmo de eliminación de candidatos de Mitchell tiene una estrategia semejante a la propuesta por Winston en el sentido que ambos usan procesos de generalización y especificación, pero se diferencian en la forma de explorar el espacio de soluciones. Mientras en el programa ARCH las hipótesis son generadas y verificadas una a una, en el método de Mitchell las hipótesis son sistemáticamente eliminadas del espacio de todas las posibles hipótesis cuando estas no son satisfactorias. Lo novedoso de este método es el uso de un sistema muy eficiente para representar el conjunto de todas las hipótesis como un espacio de versiones.

Frente a estos dos sistemas de aprendizaje iniciales, AQ11 propone una variante más sofisticada del uso de la generalización y la especificación, donde la principal ventaja de AQ11 y de todos los algoritmos que utilizan la metodología STAR es el hecho de trabajar con un espacio de versiones que aseguran una correcta representación de las descripciones y un mecanismo muy apropiado para el uso de estos procesos de generalización. Además, con esta representación se construyen descripciones tan generales como permitan los ejemplos negativos.

El algoritmo AQ11 presenta también algunos problemas:

- El hecho de utilizar el algoritmo de eliminación de candidatos conlleva arrastrar todos los problemas de este algoritmo. Un problema típico de este algoritmo es que las entradas para un concepto no estén en zonas adyacentes del espacio de soluciones.

- El conjunto de reglas que AQ11 obtiene depende del orden en el cual se le presentan los ejemplos del conjunto de entrenamiento al sistema. En general, la primera regla inducida será más general que las que se obtengan posteriormente.

- El costo computacional de este algoritmo es mucho mayor que el necesario en sistemas que utilizan árboles de decisión tales como ID3, aunque por contra, es mucho mejor la comprensibilidad de los resultados.

- Es un algoritmo muy sensible al ruido.

Así como ID3 se puede considerar como el punto de partida para el desarrollo de nuevos sistemas de aprendizaje que usan árboles de decisión, a la familia de algoritmos AQ los podemos considerar en el mismo sentido para estrategias de aprendizaje ascendentes.

De los anteriores problemas de AQ11, el más importante es su alta sensibilidad al ruido, para el que muchos investigadores han propuesto soluciones. El propio Michalski en [115] propone una primera solución, introduciendo el uso de procedimiento de adaptación parcial a la hora de ejecutar las reglas. Una segunda aproximación es propuesta por Clark con su algoritmo CN2 [19] que introduce un algoritmo descendente que produce las mismas reglas (del tipo If-then-else) pero controlando el nivel de ruido.

Otras mejoras sustanciales de este sistema pueden verse en las siguientes generaciones de la familia AQ, como son AQ15, AQ15-GA y AQ17. En AQ15 [115] Michalski introduce una nueva herramienta muy poderosa y flexible que permite la construcción de nuevos descriptores en base a los descriptores iniciales [114]. Esto supone que AQ entra en el mundo de la relevancia indirecta de los descriptores iniciales. Este nuevo sistema demuestra ser mejor que AQ11 en cuanto a la comprensibilidad de las descripciones obtenidas y en cuanto al porcentaje de acierto. AQ15-GA es una implementación basada en el uso de algoritmos genéticos sobre AQ15 propuesta por Thrun [148].

En los últimos años han aparecido nuevas versiones más poderosas de la familia AQ tales como:

- AQ17-DCI [148], donde además de la generación de nuevos descriptores se define una *función de calidad* para los atributos que sobrepasan una determinada cota prefijada para determinar la relevancia de cada uno de ellos en las descripciones.

- AQ17-FCLS (Flexible Concept Learning), [148] que es capaz de trabajar con representaciones numéricas y simbólicas para describir los conceptos y además las reglas se aprenden en dos fases: una primera fase semejante a la metodología STAR que permite una preclasificación de los ejemplos, y una segunda fase de especificación de las reglas para mejorar su porcentaje de aciertos.

- AQ17-HCI (Hypothesis-driven Constructive Induction) [148], donde se generan nuevos atributos en base a las descripciones obtenidas en iteraciones anteriores.

1.4 La aplicación de la computación evolutiva al aprendizaje automático

Los organismos vivos poseen una consumada destreza en la resolución de problemas, esta habilidad la obtienen a través de la *evolución*. En casi todos los organismos, la evolución se produce mediante dos procesos elementales: 1) la *selección natural*, que determina qué miembros de la población sobrevivirán hasta reproducirse y 2) la *reproducción*, que garantiza la mezcla y recombinación de sus genes entre la descendencia. El efecto conjunto de estos dos procesos marca el éxito de 3 billones de años de vida sobre el planeta: cualquier organismo, en sucesivas generaciones, es capaz de *adaptarse* a cambios en su forma de vida, e incluso en su estructura, para combatir cualquier agente que hiciera peligrar su supervivencia.

A través de la historia, la naturaleza ha sido una fuente inagotable de inspiración para el desarrollo técnico y científico, el cual, a cambio, ha contribuido a una mejor comprensión de la propia naturaleza. J. Holland estudia el fenómeno de la adaptación tal y como ocurre en la naturaleza y desarrolla mecanismos para incorporarla en un sistema computerizado. En 1975, en su libro *Adaptación en sistemas naturales y artificiales* ([72]) propone una clase de algoritmos adaptativos, ideados como una abstracción de la evolución para resolver problemas prácticos y para servir de modelos computacionales de sistemas naturales evolutivos, algoritmos que posteriormente recibieron el nombre de *algoritmos genéticos*.

En la naturaleza, *los procesos evolutivos* ocurren cuando se satisfacen las siguientes condiciones:

- Una entidad o individuo tiene la habilidad de reproducirse.
- Hay una población de tales individuos que son capaces de reproducirse.
- Existe alguna variedad, diferencia, entre los individuos que se reproducen.
- Algunas diferencias en la habilidad para sobrevivir en el entorno están asociadas con esa variedad.

Esta diversidad se manifiesta en cambios en los cromosomas de los individuos de la población, y se traslada a la variación de la estructura y el comportamiento de los individuos en su entorno. Estos cambios en la estructura y comportamiento se reflejan en el grado de supervivencia, adaptación, y en el nivel de reproducción. Los individuos que mejor se adaptan a su entorno son los que más sobreviven y más se reproducen. En un periodo de tiempo y con muchas generaciones de individuos, la población llega a contener más individuos cuyos cromosomas se trasladan a estructuras y a comportamientos adecuados a su entorno, sobreviviendo y reproduciéndose en un alto índice, de forma que con el tiempo la estructura de individuos en la población cambia debido a la selección natural ([24]).

J. Holland fue uno de los pioneros en la aplicación de los principios en los que se basan los procesos de la evolución natural como reglas en el diseño de sistemas artificiales para la resolución de problemas. En [72], Holland proporciona un marco general en el que se muestra cómo aplicar los procesos evolutivos a los sistemas artificiales diseñados a partir de los sistemas naturales. Cualquier problema de adaptación puede formularse generalmente en términos genéticos. Una vez formulados en estos términos, tales problemas pueden resolverse mediante lo que hoy se conoce como *algoritmos genéticos* (AGs).

Paralelamente a los estudios realizados por Holland, la idea de aplicar los principios en los que se basan los procesos de la evolución natural como reglas en los procedimientos de búsqueda fue investigada con otros enfoques ([41]). Actualmente, todas estas aportaciones iniciales han cristalizado en lo que se denomina *computación evolutiva*, y recoge las distintas vertientes en el diseño de los *algoritmos de evolución* como herramientas para el análisis de sistemas no lineales complejos por medio de simulación computacional y basados en los principios en los que se basan los procesos de evolución natural. La idea general de estos algoritmos es manejar, durante cierto tiempo, una población de soluciones candidatas para un problema particular, a través de operadores inspirados en los mecanismos naturales que producen variación genética y en la selección natural. Las diferentes variantes de algoritmos de evolución existentes actualmente son:

- *Algoritmos genéticos* ([48, 72]).
- *Estrategias de evolución* ([138]).
- *Programación evolutiva* ([40, 41]).
- *Programación genética* ([91, 92]).

En los años sesenta y setenta, se introducen las estrategias de evolución como métodos para optimizar parámetros de dispositivos aerodinámicos. El modelo era simple; a partir de un vector de parámetros reales, solución factible para un problema dado, se generaba otro vector mediante cambios aleatorios sobre el primero, y se seleccionaba el mejor de ellos para reiterar de nuevo el proceso. Posteriormente, Schwefel desarrolló esta idea con más profundidad ([137, 138]). El estudio de las estrategias de evolución ha sido un área de investigación muy activa, independiente del área de los AGs (aunque recientemente se han comenzado los intercambios entre ambos campos ([4, 5])).

Fogel, Owens y Walsh desarrollaron en los años sesenta la programación evolutiva, una técnica en la que las soluciones candidatas para un problema dado se representan como máquinas de estados finitos, las cuales evolucionan mediante las mutaciones aleatorias producidas sobre sus diagramas estado-transición y la selección de los mejores ([40, 41]).

Koza propone en 1991 ([91]) un modelo de algoritmo de evolución específico para obtener programas de ordenador que resuelven problemas dados. Para un problema concreto, un algoritmo de este tipo maneja un conjunto de árboles que representan expresiones en lenguaje lisp que resuelven el problema a cierto nivel. La aplicación de operadores especializados y de un mecanismo de selección produce árboles de expresiones que resuelven el problema concreto cada vez con mayor precisión. Tras el trabajo de Koza, se han presentado bastantes modelos basados en la misma idea pero que usan otros lenguajes de programación, tales como el C y C++. Esto ha dado lugar a la creación de un área dentro de la computación evolutiva con nombre propio: la programación genética ([92]).

En los últimos años, se ha dado una amplia interacción entre los investigadores de las distintas vertientes de la computación evolutiva. En muchos casos, esto ha permitido romper las fronteras entre los AGs, las estrategias de evolución, la programación evolutiva y la programación genética.

1.4.1 Algoritmos genéticos

Los AGs ([48, 72]) son procedimientos adaptativos para la búsqueda de soluciones en espacios complejos, inspirados en los procesos genéticos de los organismos naturales y en los principios de la evolución natural de poblaciones. Su idea básica es mantener una *población* de *cromosomas*, los cuales representan soluciones candidatas a un problema concreto, que evolucionan con el tiempo a través de un proceso de competición y variación controlada. Cada cromosoma tiene una bondad o adaptación asociada, que describe la adecuación de la solución a la que representa. El proceso de competición, denominado *mecanismo de selección*, utiliza estas adaptaciones para determinar los cromosomas que se usan para crear otros nuevos. Los nuevos cromosomas se generan a través de los operadores genéticos denominados *cruce* y *mutación*.

Los AGs se han aplicado con éxito en problemas de búsqueda y optimización. Gran parte de este éxito se debe a su capacidad para explotar la información acumulada sobre un espacio de búsqueda, y así dirigir las siguientes búsquedas hacia los mejores subespacios. Esta es su principal ventaja, sobre todo en espacios grandes, complejos y parcialmente definidos donde las técnicas clásicas de búsqueda no son apropiadas. Los AGs ofrecen una aproximación válida a problemas que requieren técnicas de búsqueda efectivas y eficientes.

Bajo la formulación inicial de los AGs, las soluciones se han representado mediante cadenas binarias de longitud fija, debido principalmente a que existen razones teóricas que recomiendan el uso de esta representación ([50]) y además, porque ésta permite un diseño simple de los operadores genéticos. Sin embargo, las buenas propiedades de los AGs no radican en el uso de cadenas de ceros y unos ([3, 130]). Por esta razón, frecuentemente se han utilizado representaciones no binarias más adecuadas para cada uno de los problemas particulares de aplicación. Una de las representaciones alternativas más importantes es la *representación real*. Esta representación parece particularmente natural cuando se tratan problemas con variables continuas. El uso de nuevas representaciones y la construcción de nuevos operadores para manipularlas ha propiciado que la concepción actual de AG sea bastante diferente y más general que la idea original de Holland ([72]).

A continuación ofrecemos una descripción más detallada de los AGs, considerando los siguientes aspectos: la estructura de un AG, la representación, el mecanismo de selección y los operadores genéticos de cruce y mutación.

1.4.1.1 Estructura de un AG

Un AG comienza con una población de cromosomas generados aleatoriamente y va obteniendo mejores cromosomas gracias a la aplicación de los operadores genéticos, los cuales están basados en los procesos genéticos que ocurren en la naturaleza. La evolución se produce sobre la población en forma de selección natural. Durante sucesivas iteraciones, llamadas *generaciones*, se evalúa la adecuación (adaptación) de los cromosomas como soluciones, y en base a esta evaluación, se forma una nueva población de cromosomas usando un mecanismo de selección y operadores genéticos específicos, tales como el cruce y la mutación. Para cada problema a resolver, se ha de proporcionar una *función de evaluación o adaptación*, f. Dado un cromosoma, la función de evaluación devuelve un valor numérico que se supone proporcional a la utilidad o adaptación de la solución que representa.

Aunque existen muchas posibles variantes de AGs, los mecanismos fundamentales bajo los cuales funcionan consisten en operar sobre una población de individuos, que inicialmente se genera de forma aleatoria, y cambiar en cada generación la población atendiendo a los tres pasos siguientes:

1. Evaluación de los individuos de la población.
2. Formación de una población intermedia a través del mecanismo de selección.
3. Formación de una nueva población a partir de los operadores genéticos de cruce y mutación.

Estos tres pasos se repiten hasta que el sistema deja de mejorar, o se alcanza un número máximo, T, de generaciones especificado por el usuario. El esquema 1.1 muestra la estructura de un AG.

Debido a la naturaleza aleatoria de los AGs, normalmente todo el proceso se repite más de una vez. Los investigadores de los AGs suelen obtener estadísticas tales como el mejor elemento encontrado durante una ejecución, la media de la función de evaluación de todos los cromosomas muestreados, generación en la que se encontró el mejor elemento, etc, todas ellas, promediadas sobre el número total de ejecuciones.

Esquema 1.1 *Estructura de un algoritmo genético*

Procedimiento Algoritmo Genético
Inicio
$t = 0$;
Inicializar $P(t)$;
Evaluar $P(t)$;
Mientras *(No se cumpla la condición de parada)* **Hacer**
Inicio
$t = t + 1$;
Seleccionar $P(t)$ *desde* $P(t-1)$;
Aplicar Cruce y Mutación sobre $P(t)$;
Evaluar $P(t)$;
Final
Final

1.4.1.2 Representación

La representación es uno de los elementos clave en la operatividad de un AG, debido a dos razones: 1) los AGs manipulan una representación codificada del problema y 2) la representación puede limitar en gran medida la visión que tiene un sistema sobre su entorno ([92]).

Con frecuencia, las soluciones se representan usando cadenas binarias. Para representar un espacio de búsqueda $S = S_1 \times .... \times S_n$ por medio del alfabeto binario se usa un conjunto de funciones $cod_i : S_i \rightarrow \{0,1\}^{L_i}$ con $L_i \in N$ y $i = 1, ..., n$, tal que cod_i codifica cada elemento de S_i usando una cadena de ceros y unos. Un elemento $x = (x_1, ..., x_n) \in S$ ($x_i \in S_i$) queda representado a través de la concatenación de la codificación de cada una de sus componentes: $cod(x) = cod_1(x_1)...cod_n(x_n)$. Los AGs basados en este tipo de representación serán denominados AGs con codificación binaria (AGCBs)

En muchos casos, se han aplicado representaciones no binarias más naturales para los problemas particulares de aplicación, tales como en los trabajos [28, 69, 82, 101].

1.4.1.3 Mecanismo de Selección

Consideremos un población P con los cromosomas $C_1, ..., C_N$. El mecanismo de selección produce una población intermedia, P', con copias de cromosomas en P. El número de copias recibidas por cada cromosoma depende de su función de evaluación; aquellos cromosomas con función de evaluación alta tienen usualmente mayor probabilidad de contribuir con copias para P'. El mecanismo de selección se realiza siguiendo dos pasos:

1. Para cada cromosoma C_i en P se calcula la probabilidad, $p_s(C_i)$, de incluir una copia suya en P'. El modelo más utilizado para calcular esta probabilidad es el modelo *proporcional* ([48, 72]), donde $p_s(C_i)$, $i = 1, ..., N$ se calcula como

$$p_s(C_i) = \frac{f(C_i)}{\sum_{j=1}^{N} f(C_j)}.$$

   De esta forma, los cromosomas con función de evaluación por encima de la media reciben más copias que aquéllos con función de evaluación por debajo de la media.

2. Basándose en las probabilidades de selección calculadas en el paso anterior, se asigna, aleatoriamente, a cada elemento de P el número de copias suyas que aparecerán en P'. Este paso se realiza a través de un método de *muestreo*. El más simple, llamado *muestreo aleatorio simple* ([48, 72]), consiste en simular el comportamiento de una ruleta en la que existe para cada elemento de la población una región proporcional al valor de su probabilidad de selección. Cada vez que se juega, la ruleta determinará un elemento para la población intermedia. Jugando N veces, la población intermedia queda totalmente determinada. Los elementos con mayor probabilidad de selección tendrán, en la ruleta, una mayor proporción de espacio asignado, de esta forma la probabilidad de que la ruleta se detenga en las zonas asignadas a estos elementos es grande.

La figura 1.3 muestra un ejemplo de la aplicación del mecanismo de selección, mostrándose a la izquierda la ruleta asociada.

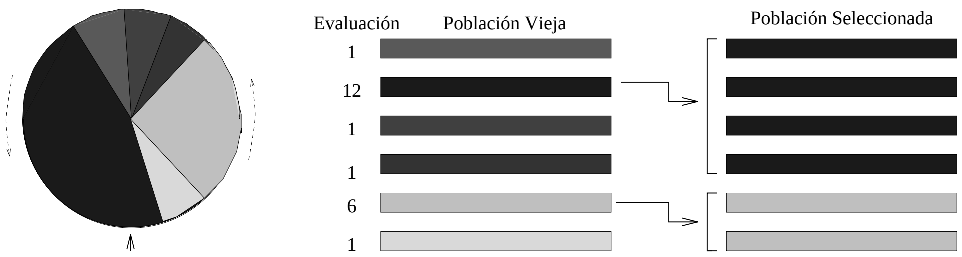

Figura 1.3: Mecanismo de selección

Una estrategia adicional, que se utiliza frecuentemente como complemento del mecanismo de selección, es el *elitismo* ([29]), el cual asegura que el mejor elemento de una generación t estará presente en la generación $t + 1$. Esto es conveniente ya que es posible que el mejor elemento de la generación t desaparezca debido a la aplicación del propio mecanismo de selección o por los operadores de cruce y mutación.

1.4.1.4 Operadores Genéticos

Tras llevar a cabo el mecanismo de selección, se aplican los operadores de cruce y mutación sobre los elementos de la población intermedia.

Operador de Cruce

El operador de cruce es un método para compartir información entre los cromosomas. Generalmente se combinan las características (genes) de dos cromosomas padres para generar dos cromosomas hijos. El operador de cruce juega un papel central en los AGs. De hecho puede considerarse como una de las características que definen a estos algoritmos y es uno de los componentes a tener en cuenta para mejorar su eficacia ([99]). A veces, al operador de cruce se le denomina operador de *recombinación*.

Las definiciones para los operadores de cruce (y mutación) dependen de la representación particular escogida. Para el caso particular de los AGCBs, el operador clásico de cruce es el *cruce simple* ([48, 72]), en el que dados dos cromosomas padres, $C_1 = (c_1^1 \ldots c_L^1)$ y $C_2 = (c_1^2 \ldots c_L^2)$, se generan los dos cromosomas hijos, $H_1 = (c_1^1, .., c_i^1, c_{i+1}^2, ..., c_L^2)$ y $H_2 = (c_1^2, .., c_i^2, c_{i+1}^1, ..., c_L^1)$, donde i es un número aleatorio perteneciente a $\{1, ..., L-1\}$. La figura 1.4 muestra un ejemplo de la aplicación de este operador.

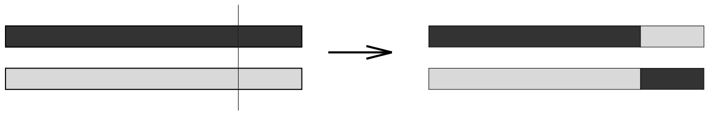

Figura 1.4: Cruce simple

Un parámetro de control del AG, denominado *probabilidad de cruce*, p_c, determina la probabilidad de aplicar el operador de cruce sobre parejas de cromosomas de la población intermedia.

Operador de Mutación

El operador de mutación altera arbitrariamente uno o más componentes (genes) de un cromosoma para aumentar la variabilidad estructural de la población. El papel de la mutación en los AGs es restaurar material genético perdido o no explorado en la población. De esta forma, se asegura que la probabilidad de alcanzar cualquier punto del espacio de búsqueda nunca es cero. Cada gen en cada cromosoma sufre una mutación de acuerdo con un parámetro de control, denominado probabilidad de mutación, p_m.

En los AGCBs, la mutación de un gen es fácil: se cambia su valor, "1" por "0" y viceversa. La figura 1.5 muestra un ejemplo de la aplicación del operador de mutación.

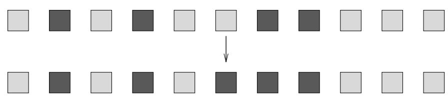

Figura 1.5: Mutación

1.4.2 Algoritmos genéticos con codificación real

Los AGCBs tienen ciertas dificultades cuando se aplican sobre espacios de búsqueda continuos con grandes dimensiones y donde se requiere una gran precisión numérica. También se plantea un problema cuando las variables que se manejan toman un número finito de valores discretos válidos, el cual no es potencia de 2. En este caso, algunos códigos binarios serán redundantes.

Una de las alternativas más importantes a la codificación binaria es la codificación real. Este tipo de codificación parece particularmente natural para resolver problemas con variables en espacios de búsqueda continuos. Un cromosoma es un vector de números reales cuyo tamaño es igual a la longitud del vector solución del problema; de esta forma, *cada gen representa una variable del problema.* Los valores de los genes están forzados a permanecer en el intervalo establecido para las variables a las cuales representan, de modo que los operadores genéticos tendrán que preservar este requerimiento.

La utilización de la codificación real aparece inicialmente en aplicaciones particulares, tales como en [101] para problemas de quimiometría. Posteriormente, los AGCRs se han usado principalmente en problemas de optimización numérica sobre dominios continuos ([28, 69]). Hasta 1991 no se realizó ningún estudio teórico sobre estos algoritmos; su utilización provocaba cierta controversia: los investigadores familiarizados con la teoría fundamental de los AGs no comprendían los buenos resultados empíricos mostrados por los AGCRs, ya que esta teoría sugería que los alfabetos de cardinal bajo debían ser más efectivos que aquellos con cardinales altos. Más tarde, se presentaron distintas herramientas para el tratamiento teórico de los AGCRs ([50, 129, 170]), lo que permitió corroborar su buen comportamiento.

El uso de parámetros reales posibilita manejar grandes dominios (incluso dominios desconocidos) para las variables, tarea difícil en las implementaciones binarias donde extender el dominio significaría sacrificar precisión, supuesta una longitud fija para los cromosomas. Otra ventaja es la capacidad para explotar la *gradualidad* de las funciones con variables continuas, donde el concepto gradualidad se refiere a que pequeños cambios en las variables se corresponden con pequeños cambios en la función. En esta línea, se destaca la habilidad de los AGCRs para *el ajuste local* de las soluciones, existiendo operadores de mutación (*mutación no uniforme* ([106])) que permiten que ésta se realice de forma más rápida y eficaz que en los AGCB.

Usando codificación real, la representación de las soluciones está muy cercana a la formulación natural de muchos problemas. Así se evita el proceso de codificación y decodificación binaria que se debe producir en los AGCBs, de manera que se aumenta la rapidez del AG. En [130], Radcliffe aduce que no es necesario que existan diferencias entre el genotipo y el fenotipo para que se produzca evolución. Por ello, no se justifica que los operadores genéticos deban definirse en razón a la representación escogida. Muy al contrario, el autor aboga por definir tanto los operadores genéticos como los conceptos de esquema directamente sobre el espacio de los fenotipos siempre y cuando esto sea posible. Los operadores genéticos y los conceptos de esquema presentados para los AGCRs en la literatura concuerdan con la idea de Radcliffe.

1.4.2.1 Operadores de Cruce

Supongamos que los cromosomas $C_1 = (c_1^1, ..., c_n^1)$ y $C_2 = (c_1^2, ..., c_n^2)$ se han seleccionado para aplicarles el operador de cruce. A continuación, se muestran los efectos de diferentes operadores de cruce para AGCRs sobre estos cromosomas.

Hay que notar que cada operador produce un número distinto de hijos por lo que a veces se hace necesaria una selección para obtener los cromosomas que formarán parte de la población. Esta selección se denominará *selección de descendientes*.

- *Cruce plano* ([129])

  Se genera el hijo, $H = (h_1, ..., h_i, ..., h_n)$, donde h_i es un valor escogido de forma aleatoria (uniformemente) del intervalo $[c_i^1, c_i^2]$.

- *Cruce simple* ([106, 170])

  Aleatoriamente se elige una posición $i \in \{1, 2, ..., n-1\}$ y se generan

  $$H_1 = (c_1^1, c_2^1, ..., c_i^1, c_{i+1}^2, ..., c_n^2), \text{ y } H_2 = (c_1^2, c_2^2, ..., c_i^2, c_{i+1}^1, ..., c_n^1).$$

- *Cruce aritmético* ([106])

  Se generan dos hijos, $H_k = (h_1^k, ..., h_i^k, ..., h_n^k)$ $k = 1, 2$, donde $h_i^1 = \lambda c_i^1 + (1-\lambda)c_i^2$ y $h_i^2 = \lambda c_i^2 + (1-\lambda)c_i^1$. λ es una constante (cruce aritmético uniforme) o varía en función del número de generaciones realizadas (cruce aritmético no uniforme).

- *Cruce BLX-α* ([35])

  Se genera un hijo, $H = (h_1, ..., h_i, ..., h_n)$, donde h_i es un número escogido de forma aleatoria (uniformemente) del intervalo $[c_{min} - I\alpha, c_{max} + I\alpha]$, donde $c_{max} = \max(c_i^1, c_i^2)$, $c_{min} = \min(c_i^1, c_i^2)$ y $I = c_{max} - c_{min}$. El cruce BLX-0.0 es igual al cruce plano.

- *Cruce lineal* ([170])

  Se generan tres hijos, $H_k = (h_1^k, ..., h_i^k, ..., h_n^k)$ $k = 1, 2, 3$, donde $h_i^1 = \frac{1}{2}c_i^1 + \frac{1}{2}c_i^2$, $h_i^2 = \frac{3}{2}c_i^1 - \frac{1}{2}c_i^2$ y $h_i^3 = -\frac{1}{2}c_i^1 + \frac{3}{2}c_i^2$.

  Con este tipo de cruce se realiza una *selección de descendientes* consistente en que los dos cromosomas más prometedores son los seleccionados para formar parte de la nueva población.

1.4.2.2 Operadores de Mutación

Supongamos que $C = (c_1, ..., c_i, ..., c_n)$ es un cromosoma y c_i es un elemento a mutar, a continuación se muestra el gen c_i' resultante de aplicar los dos operadores de mutación más habitualmente utilizados para AGCRs son:

- *Mutación aleatoria* ([129])

  c_i' se toma de forma aleatoria (uniformemente) del dominio $[a_i, b_i]$.

- *Mutación no uniforme* ([106])

  Si se supone que este operador se aplica durante una generación t y que T es el máximo número de generaciones del AGCR, entonces

  $$c_i' = \begin{cases} c_i + \triangle(t, b_i - c_i), & \text{Si } \beta = 0 \\ c_i - \triangle(t, c_i - a_i), & \text{Si } \beta = 1 \end{cases}$$

  donde β es un número aleatorio que puede valer cero o uno, y

  $$\triangle(t, y) = y(1 - r^{(1-\frac{t}{T})^b}),$$

  donde r es un número aleatorio del intervalo $[0, 1]$ y b es un parámetro elegido por el usuario, que determina el grado de dependencia con el número de iteraciones. La función $\triangle(\cdot, \cdot)$ devuelve un valor en el rango $[0, y]$ de forma que la probabilidad de devolver un número cercano a cero crece conforme t es mayor. De esta forma, el intervalo de generación de genes es más pequeño conforme pasan las generaciones. Esta propiedad permite que este operador produzca una búsqueda uniforme del espacio de búsqueda cuando t es pequeño, y una búsqueda local en las etapas finales, favoreciendo el ajuste local.

1.4.3 Modelos de aprendizaje basados en algoritmos genéticos

A partir de los años 80 surgió un creciente interés en aplicar los métodos basados en los AGs a problemas de aprendizaje automático, en particular al aprendizaje de reglas de producción en base a conjuntos de ejemplos atributo-valorados. El principal problema en tales aplicaciones consiste en encontrar una representación "confortable" en el sentido de que sea capaz tanto de recoger las características del problema como de representar las soluciones potenciales. Cuando usamos una representación para aprender conceptos mediante reglas tenemos un problema añadido, el número de tales reglas es desconocido a priori, por tanto, la representación tradicional con longitud fija no es recomendable. Tradicionalmente se han propuesto dos modelos de aprendizaje :

- **Modelo de Michigan**, en donde la población consta de cromosomas de longitud fija y la solución del problema se representa por todos los cromosomas de la población. Este modelo es usualmente presentado en la literatura con el nombre de *Sistemas Clasificadores* (SCs). Su base es un mecanismo especial de asignación de pesos mediante algoritmos como el *Bucket-Brigade* y fué desarrollado originariamente por Holland [77].
- **Modelo de Pittsburgh**, que constituye una extensión del modelo tradicional de cromosomas de longitud fija. Los cromosomas ahora tienen longitud variable y se utilizan para representar soluciones individuales, es decir, bases de reglas completas. Este modelo fué inicialmente propuesto por Smith [145].

La diferencia fundamental entre estos dos modelos, es que mientras el primero busca como solución a una población completa, el segundo sólo busca un único individuo de la población.

Las dos modelos presentan algunos inconvenientes debido principalmente a que implican una readaptación de los tradicionales AGs.

El modelo de Pittsburgh tiene dos inconvenientes fundamentales, el primero de ellos se debe a que los individuos en la población son de longitud variable, lo que ocasiona un tratamiento especial de dichos elementos y una redefinición de todos los operadores genéticos para adaptarse a estas circunstancias. La segunda está relacionada con el espacio de posibles soluciones. Supongamos que N es el número de reglas necesario para describir el problema, K es el número de clases, n es el número de atributos y V es el número de valores por atributo. El número de reglas posibles que constituyen el espacio de búsqueda viene definido por la siguiente fórmula [157]:

$$N^{K(V+1)^n}$$

es decir, el espacio de búsqueda es extremadamente grande, con lo que el tiempo requerido para encontrar una buena solución lo es también.

Frente al modelo de Pittsburgh, el modelo Michigan no necesita readaptar los operadores tradicionales de los AGs y además el número de posibles soluciones viene determinado por la siguientes expresión:

$$K(V+1)^n$$

que es sensiblemente menor que el del otro modelo, aunque aún sigue siendo un espacio de búsqueda bastante grande. Por otro lado, el hecho de que la solución al problema se represente por una subpoblación de individuos (que difiere de los tradicionales AGs que buscan el mejor individuo) provoca la introducción de una función de evaluación que debe tener en cuenta a todos los miembros de la población. Esto implica una función de evaluación algo más compleja, lo que ocasiona generalmente un costo computacional mayor frente a los AGs tradicionales.

Para la construcción de nuestro algoritmo exploraremos una vía alternativa de la aplicación de los algoritmos genéticos al problema del aprendizaje automático, que reduzca el espacio de las soluciones posibles.

En las siguientes secciones pasaremos a estudiar algunos algoritmos que usan estos modelos.

1.4.4 Sistemas clasificadores (SCs)

Las primeras ideas desarrolladas sobre los SCs fueron presentadas por Holland en [72, 73], y fué este mismo autor junto con Reitman en [74] quien desarrolló el primer SC. A partir de este momento se han propuesto en la literatura muchos tipos diferentes de SCs [12, 16, 48, 121, 155, 165, 166].

Los SCs se definen como sistemas paralelos basados en reglas, de paso de mensajes, que son capaces de interactuar con el entorno y de aprender mediante refuerzo (asignando recompensas) y descubrimiento de reglas.

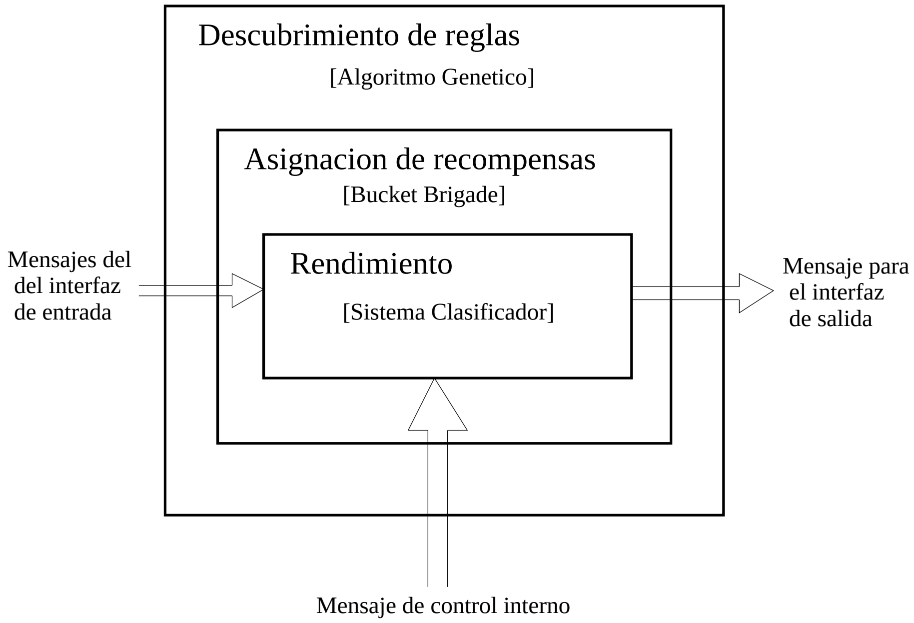

Figura 1.6: Organización general de un sistema clasificador.

En un SC podemos distinguir tres niveles de actividad (como se muestra en la fig. 1.6):

- **Sistema de rendimiento:** Es la actividad que se produce a más bajo nivel y es la encargada de interactuar con el entorno del SC. El sistema de rendimiento suele ser un sistema basado en reglas como la mayoría de los sistemas expertos. Este tipo de reglas son llamadas *clasificadores*. La comunicación con el medio que rodea al SC se realiza a través del paso de mensajes.

- **Sistema de asignación de recompensas:** El sistema de rendimiento debe determinar cuando las reglas son correctas para clasificar. Por consiguiente, es necesario tener un segundo nivel de actividad que

  a) determine la efectividad (bondad para clasificar) de cada uno de los clasificadores (reglas), y

  b) determine las reglas que son incorrectas y deben ser reemplazadas del conjunto de los clasificadores.

  La forma en la cual se determina la bondad para la clasificación de las reglas usadas en el sistema de rendimiento se realiza a través de un algoritmo de aprendizaje por refuerzo, en el cual, una vez que las reglas son lanzadas, el sistema premia a aquellas reglas que se han comportado de acuerdo a la situación existente y castiga a aquellas que han tenido un comportamiento erróneo. Tradicionalmente se vienen usando dos esquemas diferentes para la asignación de recompensas:

  1. **El Algoritmo del Bucket Brigade**. Este algoritmo trabaja con un esquema de aprendizaje local que precisa pocos requerimientos computacionales, tanto de memoria como de tiempo de CPU. Es un algoritmo de asignación de recompensas que se usa en muchos SCs como por ejemplo en [11, 46, 76].

     En un SC este algoritmo se enlaza con el sistema de asignación de recompensas formando un mecanismo que conduce a la competición entre los distintos clasificadores del sistema. Algunas variantes sobre este algoritmo se pueden encontrar en [33, 79].

  2. **El plan de Profit-Sharing**. Este es un esquema de aprendizaje global que presenta un comportamiento mejor en líneas generales que el algoritmo de Bucket Brigade. Básicamente el proceso de elección del clasificador a disparar es semejante al del algoritmo anterior. La diferencia entre ambos esquemas aparece en el método utilizado para la asignación de recompensas. En la literatura se puede encontrar más información sobre este esquema, como por ejemplo en los trabajos [63, 74] y algunas variantes en [63, 162]

- **Sistema de descubrimiento de reglas.** Es el encargado de determinar entre todas la reglas posibles, cuales de ellas pueden ser las más útiles, evaluando una pequeña porción de ellas. Por el hecho de explorar sólo una porción de reglas es posible que entre las examinadas no se encuentren reglas buenas para clasificar, por este motivo es necesario que el sistema permita la incorporación de nuevas reglas reemplazándolas por las no que son útiles.

  El conjunto inicial de reglas se puede obtener aleatoriamente y mediante pequeñas alteraciones y recombinando partes de distintas reglas intentar mejorar su bondad.

Esta forma progresiva de tratar de mejorar las reglas paso a paso combinando información de otras reglas y con alteraciones aleatorias encaja perfectamente con la filosofía de los algoritmos genéticos.

1.4.5 El modelo Pittsburgh en el aprendizaje automático de conceptos

Los algoritmos de adquisición automática de conceptos que utilizan los AGs con el modelo Pittsburgh son llamados Sistemas de Aprendizaje (SAs). Conceptualmente, estos algoritmos son simples de describir porque el método de aprendizaje es directamente la descripción del algoritmo genético. Por tanto, un SA queda determinado en función de las siguientes componentes del algoritmo genético:

- **El espacio de búsqueda y la representación de las reglas:** Básicamente, los SAs usan dos lenguajes para la representación de las reglas.
  - El primero de ellos, conocido como *forma normal disyuntiva modificada*, fué propuesto por Michalski [114] y supone un lenguaje útil para representar conceptos complejos en base a un conjunto disyuntivo de reglas de clasificación. La parte antecedente de la regla se compone de la conjunción de una o más variables, mientras la parte consecuente indica el concepto que debe ser asignado a un ejemplo cuando este se adapte a la parte condición. Además en la parte antecedente se permiten disyunciones internas, es decir, que una misma variable pueda contener una disyunción de valores de su dominio.

    Una regla usando este lenguaje permite construciones del tipo:

    Si (F1 = rojo) y (F2 = esférico o cúbico) entones es de la clase C1.

    suponiendo que el dominio de la variable F1 es {rojo, verde, blanco, azul} y el de la variable F2 es {esférico, cilíndrico, cúbico}.

    Usando este tipo de lenguaje es posible determinar si una variable es útil para describir una clase. Si una variable toma como valor todo su dominio de definición, esto indica que esta variable no es necesaria para describir una clase, por ejemplo, en la siguiente regla

    Si (F1 = rojo o verde o blanco o azul) y
    (F2 es cúbico) entonces es de la clase C1.

    la única variable relevante de la regla es F2, y la regla se puede reformular como:

    Si (F2 = cúbico) entonces es de la clase C1

Normalmente, en este lenguaje se suele utilizar una codificación binaria para representar las reglas, de manera que si una variable puede tomar k valores diferentes, su representación requiere k bits, donde un 0 en el bit i-ésimo indica que la variable no toma el valor i-ésimo de su dominio y un 1 indica que la variable toma el valor i-ésimo de su dominio. Así, el ejemplo anterior tendría la siguiente representación genética, suponiendo que los ejemplos se pueden clasificar en tres clases:

F1	F2	Clase
1111	001	100

– El segundo lenguaje es conocido por (VL_1) y fué propuesto por Michalski y otros en [115]. Este lenguaje es más poderoso que el descrito anteriormente en el sentido que permite descripciones más complejas para representar un concepto. Los elementos más destacables de este lenguaje son los siguientes:

(-) Los valores que pueden tomar los atributos (variables antecedente) pueden ser de tres tipos: nominales, intervalares o estructurados (los mismos tres tipos que se describieron anteriormente).

(-) Existen un conjunto de relaciones que pueden ser aplicadas para la definición de una regla. Las relaciones permitidas son $(=, <>, >, <, \leq, \geq)$.

(-) La parte antecedente de la regla está compuesta por un conjunto de selectores. Cada selector es de la siguiente forma

$$[\textit{variable relación valor}\,]$$

(-) Cuando la relación utilizada es el =, entonces se permite dentro de cada selector que una variable tome diferentes valores de su dominio que son interpretados como disyunciones internas, al igual que en el anterior lenguaje.

Una regla expresada en este lenguaje es de la forma:

Si [F1 = verde] [F2 <> cilíndrico] entonces es de la clase C1

asumiendo que F1 y F2 tiene los mismos dominios que en el caso anterior.

El problema fundamental cuando trabajamos con el lenguaje VL_1 es encontrar un buena representación genética que permita contemplar todas las posibilidades que ofrece el lenguaje, frente a la forma normal disyuntiva modificada que ofrece una representación binaria bastante confortable.

- **El mecanismo de evaluación:** La función de evaluación para estos sistemas, debido a que cada cromosoma en la población representa un conjunto de reglas, intenta recoger conceptos de la teoría clásica de aprendizaje tales como la consistencia y la completitud [114]. Normalmente, los SA usan los porcentajes de acierto de cada elemento de la población sobre el conjunto de ejemplos para determinar el grado en el que los anteriores conceptos se satisfacen. Además, se suelen incluir en la función de evaluación otros parámetros que son importantes para algunos problemas concretos. Parámetros tales como el costo asociado al uso de los valores de cada variable para construir la descripción, la longitud de las reglas o exigir un nivel de generalidad a las reglas.

- **El conjunto de operadores genéticos usados:** Los SAs necesitan redefinir los operadores genéticos tradicionales para adaptarlos al problema del aprendizaje estableciendo restricciones sobre su uso. Además, se introducen nuevos operadores que permitan evolucionar al sistema más rápidamente hacia soluciones deseadas. Existen diferentes propuestas de nuevos operadores para este modelo, por ejemplo, desde los operadores simples propuestos por De Jong [31] que consisten en la inclusión de dos operadores especiales de mutación, uno que permite la generalización de reglas y el otro permita la especificación de las mismas, hasta algunos más complejos propuestos por Janikow [84] o por Grefenstette [64] que proponen diferentes operadores para diferentes niveles en la evolución del sistema.

GABIL y GIL junto con SAMUEL son los tres SAs más conocidos y referenciados en la literatura. Mientras podemos considerar al sistema GABIL (propuesto por De Jong en [31]) como la versión más simple de un SA, tanto SAMUEL (propuesto por Grefenstette en [64]) y como GIL (propuesto por Janikow en [85]) representan el otro extremo de los SAs, donde incorporan muchos conceptos de los algoritmos de aprendizaje tradicionales.

En los dos siguientes apartados describimos el sistema GABIL como ejemplo de un SA simple y el sistema GIL como máximo exponente de los SAs más sofisticados.

1.4.5.1 El sistema de aprendizaje GABIL

GABIL (GA-based concept learner) es un SA desarrollado por De Jong para el problema del aprendizaje de conceptos simples, donde el conjunto de entrenamiento sólo contiene ejemplos del concepto que debe ser aprendido. GABIL utiliza la forma normal disyuntiva modificada para expresar las reglas y una representación binaria de las mismas, basada en la idea que se mostraba en el apartado anterior. Cada cromosoma tiene una longitud variable (formado por la concatenación de reglas simples) y representa un conjunto de reglas para clasificar el concepto descrito por el conjunto de ejemplos.

Las características fundamentales del sistema vienen dadas por la función de evaluación utilizada y por los nuevos operadores genéticos que son introducidos. Con respecto a la función de evaluación, GABIL mide la bondad de un conjunto de reglas (un individuo de la población) en función del porcentaje de aciertos de este conjunto de reglas sobre el conjunto de ejemplos de entrenamiento de la siguiente forma:

$$fitness(individuo_i) = (porcentaje\ de\ acierto)^2$$

Esta forma de evaluar los individuos es equivalente a medir la consistencia y la completitud de las reglas sobre el conjunto de entrenamiento, en el sentido de que cuanto mayor sea el porcentaje de aciertos, mayor será la satisfacción tanto de la consistencia como de la completitud de las reglas.

Este sistema se caracteriza también por la inclusión de dos nuevos operadores genéticos de mutación llamados *operador de adición de nuevas alternativas* y *operador de eliminación de alternativas*. Estos dos operadores son adaptaciones de los propuestos por Michalski [114] que permiten generalizar y especificar el conjunto de reglas. El primero de los operadores añade una disyunción interna en un atributo de una regla, mientras que el segundo elimina una disyunción interna de un atributo.

Estos dos operadores se introducen con la siguiente intención: si una regla es buena, puede que una regla más general o una regla más especifica que ella sea mejor. Con la introducción de estos dos operadores se permiten movimientos aleatorios de generalización y especialización de reglas cuya permanencia en las sucesivas iteraciones dependerá de la bondad de dichas modificaciones.

Por último destacaremos una modificación que propone el propio De Jong sobre GABIL [31] para transformarlo en un SA adaptativo. La idea consiste en que los dos operadores anteriores no se usen indiscriminadamente sobre todos los individuos de la población, sino que sean los propios individuos, durante el proceso de evolución del sistema, los que determinen cual de los dos operadores se debe aplicar sobre ellos. Para llevar a cabo esta modificación, en la representación de los cromosomas se añaden dos nuevos bits que indicarán si el elemento deber ser generalizado o especificado. El sistema con esta modificación es bautizado con el nombre de GABIL adaptativo.

1.4.5.2 El sistema de aprendizaje GIL

GIL (Genetic-based Inductive Learning) es un sistema bastante más complejo que GABIL que fué desarrollado por Janikow [85]. Utiliza como lenguaje de representación de las reglas una versión restringida del lenguaje VL_1. Esta restricción consiste en que el único operador de relación permitido es el "=". GIL es un SA para el aprendizaje de múltiples conceptos, es decir, que el conjunto de entrenamiento contiene ejemplos de clases diferentes. Además, utiliza una representación genética binaria semejante a la descrita en [31] para GABIL.

La función de evaluación que utiliza GIL es algo más compleja, y tiene en cuenta tanto la consistencia como la completitud del conjunto de reglas sobre el conjunto de entrenamiento, así como el costo asociado a la obtención de los valores para los atributos. La combinación de estos tres factores se realiza de la siguiente forma: define una medida llamada corrección que es una suma ponderada de las medidas de consistencia y completitud

$$\text{corrección}(individuo_i) = \frac{w_1\ completitud + w_2\ consistencia}{w_1 + w_2}.$$

esta medida de corrección se combina con el costo asociado del conjunto de reglas dando lugar a la siguiente función de evaluación:

$$fitness(individuo_i) = correccion(individuo_i)(1 + w_3 * (1 - costo(individuo_i)))^f$$

donde w_1, w_2 y w_3 representan la importancia de cada uno de los factores que entran en juego y f es un valor que crece lentamente en el intervalo [0,1] a medida que el genético evoluciona.

Además, la parte más sofisticada del sistema GIL son los operadores genéticos que introduce, así como la forma en que estos se activan en el proceso evolutivo. Los operadores genéticos de GIL se dividen en tres niveles de actuación:

- **A nivel de conjuntos de reglas:** donde actuán sobre todos los individuos de la población, operan con uno o dos individuos al mismo tiempo.
- **A nivel de regla:** donde actuan únicamente sobre una única regla, es decir, sobre un segmento de un individuo de la población.
- **A nivel de la parte condición de una regla:** donde actuán únicamente sobre una condición de una regla, es decir, solo con un segmento de un individuo de la población que corresponda a la parte precondición de una regla.

Sobre cada uno de estos niveles se define un conjunto de operadores, donde a su vez estos operadores pueden clasificarse en tres tipos: de generalización, de especificación y de independencia. Los dos primeros siguen la misma filosofía que los operadores definidos en GABIL para permitir movimientos aleatorios de generalización y especificación, mientras que el último tipo introduce nuevos individuos en la población para aumentar la diversidad del proceso evolutivo. Para una descripción más detallada de todos los operadores genéticos utilizados en GIL ver [85].