Métodos de generalización para sistemas clasificadores de Pittsburgh

Jaume Bacardit Peñarroya y Josep Maria Garrell Guiu

Resumen— **Los sistemas clasificadores basados en algoritmos genéticos usando el enfoque de Pittsburgh tradicionalmente han tenido problemas para controlar el tamaño de los individuos de su población (conjuntos de reglas de clasificación) y, como consecuencia, para generar conjuntos minimales máximamente generales. En este artículo se presentan diversos métodos para favorecer la generalización, actuando en diversas áreas como la evaluación de los individuos, el proceso de selección y la eliminación de reglas.**

Palabras clave— **algoritmos genéticos, sistemas clasificadores, enfoque de Pittsburgh, generalizaci'on**

I. Introducción

Los ALGORITMOS GENÉTICOS (GA) [1],[2] son métodos genéricos para solucionar problemas basados en ciertos aspectos del funcionamiento de la evolución de las especies. Tradicionalmente se han aplicado a problemas de optimización, aunque también son efectivos en otras áreas, como los problemas de clasificación. La aplicación de los GA a problemas de clasificación normalmente se ha realizado desde dos puntos de vista diferentes, el enfoque de Pittsburgh y el de Michigan, siendo ejemplos iniciales de los dos enfoques los sistemas LS-1 [3] y CS-1 [4] respectivamente.

El enfoque de Pittsburgh, plantea como solución al problema de clasificación un conjunto de reglas de clasificación. Cada individuo de la población es uno de estos conjuntos, es decir, una solución completa al problema. En cambio, el enfoque de Michigan define cada individuo como una única regla, siendo toda la población la solución al problema. Este enfoque se aleja del esquema clásico de los GA, siendo este, solo una parte del sistema clasificador.

En los últimos años los sistemas de Michigan han sido mucho más populares que los de Pittsburgh, parte de dicha popularidad es debida al desarrollo de sistemas eficientes, como el XCS de Wilson [5], que presenta un coste computacional menor que los sistemas de Pittsburgh.

Uno de los problemas de los sistemas de Pittsburgh es el control del tamaño (número de reglas) de los individuos. Éste tamaño puede crecer de forma desmesurada, elevando de forma alarmante el coste computacional del sistema. Una consecuencia de este hecho es la obtención de soluciones no suficientemente generalizadas. Entendiendo por generalización la capacidad de aprender a partir de los ejemplos de entrenamiento y poder clasificar correctamente ejemplos nuevos.

Grup de Recerca en Sistemes Intel·ligents, Departament d'Informàtica, Enginyeria i Arquitectura La Salle, Universitat Ramon Llull, Passeig Bonanova 8, 08022 Barcelona, España. {jbacardit,josepmg}@salleURL.edu

En este artículo se presentan diversas técnicas que buscan satisfacer dos objetivos: el primero de ellos es controlar el crecimiento desmesurado de los individuos; el segundo, obtener conjuntos de reglas suficientemente generalizados.

La estructura del artículo es la siguiente: En primer lugar se describe el sistema clasificador básico sobre el que se aplican las técnicas presentadas. Posteriormente se detalla la problemática expuesta en el párrafo anterior y las técnicas que intentan solucionarla, así como las pruebas realizadas para avaluar dichas técnicas. Seguidamente se exponen y analizan los resultados pasando finalmente, a presentar las conclusiones y el trabajo futuro.

II. Entorno de trabajo

En esta sección se describe el sistema GAssist (*GA based claSSIfier sySTem*) [6],[7], sistema clasificador basado en la propuesta GABL de Kenneth A. de Jong [8]. A continuación se detallan sus partes más importantes.

A. Representación de individuo

La representación usada es aproximadamente la misma que la del GABL. Cada individuo, como se ha dicho, es un conjunto variable de reglas de clasificación con la siguiente forma: $I = (R_1 \vee R_2 \ldots \vee R_n)$. Cada regla de clasificación tiene la forma *condición* $\rightarrow$ *clase*, entendiendo que predecimos *clase* a los ejemplos que cumplan con *condición*. La condición es un predicado en forma normal conjuntiva (FNC) el cual presenta la siguiente forma:

$$((A_1 = V_1^1 \vee \ldots \vee A_1 = V_m^1) \bigwedge \ldots \bigwedge (A_n = V_2^n \vee \ldots A_n = V_m^b))$$

donde A_i es el atributo i del problema y V_i^j es el valor j que puede tomar el atributo i. Este último punto implica que los atributos tienen que ser discretos. Para trabajar con problemas que tengan atributos reales, hay que realizar un proceso de discretización previo [9].

Esta tipología de reglas se mapea en el cromosoma usando el alfabeto definido por dos símbolos {0,1} utilizado el método del GABL, al que se le ha añadido el gen que codifica la clase que clasifica la regla. Por ejemplo, si tenemos un problema con dos atributos, cada uno con tres valores diferentes {1,2,3}, y tenemos una regla de la forma "Si el atributo 1 es 1 o 2 y el atributo 2 es 3 luego la clase es 1", esta regla se representa como:

A1	*A2*	*Clase*
110	001	1

Esta representación nos permite usar operadores genéticos clásicos de codificación binaria. Solo hay un caso especial, cuando el problema a solucionar tenga más de dos clases. En este caso, el gen que contiene la clase asociada a cada regla pasará a ser un entero en vez de un bit.

Existen otras representaciones de individuo posibles. Por ejemplo, la representación de predicados totalmente conjuntivos con el alfabeto $\{0, 1, \#\}$, donde el símbolo # significa cualquier valor. Esta representación se ha usado satisfactoriamente en sistemas tanto del enfoque de Michigan ([5],[10]) como del de Pittsburgh ([11]), con un espacio de búsqueda menor que el de la representación de este artículo pero con algo menos de expresividad.

B. Operadores genéticos

Para el cruce se usan los operadores binarios con uno (1px) o dos (2px) puntos de cruce [12], adaptados a trabajar con individuos de tamaño variable. Los puntos de cruce pueden aparecer en cualquier regla pero tienen que ser semánticamente equivalentes para cada individuo. Es decir, los puntos deben cortar por la misma posición de las reglas escogidas de cada individuo.

La mutación es también la clásica mutación binaria con una excepción: los problemas con más de dos clases. En estos casos la mutación cambia la clase por otra diferente elegida al azar.

El operador de selección usado es el basado en torneo [13] (Tournament Selection). Para el reemplazo de poblaciones, se aplica el esquema generacional [14] en su variante elitista.

C. Evaluación del individuo

La evaluación de los individuos se realiza a partir del resultado de la clasificación de todos los ejemplos de entrenamiento disponibles por parte de cada individuo. Dicha evaluación se computa como:

$$\text{evaluación}(I_i) = (\text{porcentaje de acierto})^2 \tag{1}$$

donde el porcentaje de acierto se refiere al porcentaje de ejemplos de entrenamiento clasificados correctamente.

D. Activación de las reglas

El proceso de activación de las reglas define en que orden se evaluan las reglas para determinar qué regla clasifica cada muestra. El método usado es el llamado activación por *lista* [6] donde el orden de evaluación de las reglas es directamente el orden definido por el cromosoma del individuo. Las reglas se evaluan de una en una, empezando por la primera regla del cromosoma hasta que se encuentra una cuya condición concuerde con la muestra, esta es la escogida para clasificar. Si no se activa ninguna regla, la muestra no se clasifica.

E. Inicialización de los individuos

La inicialización se realiza de forma aleatoria restringida a los siguientes criterios:

- El número de reglas de cada individuo es un natural entre $r_{initMin}$ y $r_{initMax}$, límites previamente definidos
- La distribución de zeros y unos de la regla se controla con P_1, la probabilidad que un gen sea uno.

Habitualmente P_{uno} es 0.5, dando igual probabilidad al 0 que al 1, pero en ciertos problemas es necesario elevarla, ya que se puede dar el caso de que ningún individuo de la inicialización clasifique ningún caso, especialmente en problemas con atributos reales de los que desconozcamos suficiente información para realizar una discretización específica. Si se da esta situación, el GA tiene muchas dificultades para converger hacia la solución óptima.

F. criterio de finalización

La población evoluciona a lo largo de las iteraciones hasta que algún individuo clasifique el 100% de las muestras de entrenamiento o se haya realizado el número máximo de iteraciones predefinido.

III. Mejoras de la generalización

En esta sección detallaremos las propuestas anteriores para satisfacer los objetivos descritos en la introducción: el control del tamaño de los individuos, y la obtención de conjuntos de reglas correctamente generalizados. Antes de cada propuesta se detallará los problemas que motivan su realización.

El problema del tamaño de los individuos

Los individuos de los sistemas clasificadores de Pittsburgh, como se ha comentado anteriormente, tienden a crecer de forma desmesurada, colapsando muchas veces los recursos computacionales. Este fenómeno en el campo de la programación genética se suele llamar *bloat* [15] y, en general, está relacionado a representaciones con tamaño de individuo variable [16].

Una de las posibles causas de este problema se debe buscar en el operador de cruce utilizado. Este echo se puede ver claramente en el siguiente ejemplo:

- Supongamos que el algoritmo de cruce decide cruzar dos individuos, uno con 10 reglas y el otro con 8. Representamos las reglas del individuo 1 como R_i^1 y las del individuo 2 como R_j^2.
  - $R_1^1 R_2^1 R_3^1 R_4^1 R_5^1 R_6^1 R_7^1 R_8^1 R_9^1 R_{10}^1$
  - $R_1^2 R_2^2 R_3^2 R_4^2 R_5^2 R_6^2 R_7^2 R_8^2$
- El operador de cruce decide cortar el primer individuo por la regla 2 y el segundo por la regla 8. En este ejemplo para simplificar suponemos que los puntos de corte están en extremos de las reglas pero en realidad no tiene porque ser así.
- Los hijos resultantes son:
  - $R_1^1 R_2^1 R_8^2$
  - $R_1^2 R_2^2 R_3^2 R_4^2 R_5^2 R_6^2 R_7^2 R_3^1 R_4^1 R_5^1 R_6^1 R_7^1 R_8^1 R_9^1 R_{10}^1$
- El hijo con muchas reglas tiene más capacidad para memorizar y a consecuencia de ello, tendrá más posibilidades de sobrevivir al proceso de selección.

Limitar el operador de cruce para que no genere individuos demasiado grandes no es conveniente, ya que limitar la fuente de la recombinación genética no es aconsejable. Pero analizando los individuos se puede observar que muchas de sus reglas no se activan nunca. Enfoquemos pues el análisis del problema hacia estas reglas no utilizadas.

La presencia de reglas inútiles puede deberse a tres motivos: el primero es que alguno de los terminos conjuntivos de la condición sea falso sea cual sea la muestra de entrada. En la representación de reglas utilizada este hecho corresponde a tener todos genes asociados a un atributo dentro de una regla con valor 0. El segundo motivo es que, aunque la regla sea activable (su condición puede ser cierta el algún caso), no lo haga nunca ya que existe otra regla similar en una posición anterior dentro del cromosoma. El tercero son reglas para las que, simplemente, no existen ejemplos afectados.

A. La purga de reglas

Para controlar la aparición de reglas inútiles, se propone un operador que elimine las reglas que no se activen nunca. Llamándolo *operador de purga*. Este operador se aplica después de la evaluación de los individuos. Una vez se conoce cuales son las reglas que no se han activado nunca, estas son eliminadas.

Este operador puede eliminar la diversidad de la población, ya que elimina material genético de la población de forma determinista (en vez de hacerse de forma "natural" a través del proceso de selección). Es por eso que su acción se limita con varios factores:

- La purga se activa cuando la población ya ha tenido tiempo de adaptarse mínimamente al problema a solucionar.
- Nunca se reduce el número de reglas de un individuo por debajo de un cierto mínimo. Este mínimo se puede determinar a partir de información previa del problema a solucionar (si se dispone de ella) o, como cota inferior se pueden tomar dos opciones:
  - A: 2 o 3 reglas ya que este es el número mínimo de reglas a partir del cual nuestro operador de cruce de uno o dos puntos de corte respectivamente pueda generar de nuevo indivíduos más grandes.
  - B: tantas reglas como clases tenga el problema.

Reglas útiles pero específicas

Con el operador de purga se controla el crecimiento sin control del tamaño de los individuos, y los recursos computacionales necesarios para el proceso de aprendizaje se hacen factibles, pero si se analizan los individuos de la población se observa otro problema: las reglas específicas.

El algoritmo genético se adapta al operador de purga de la forma más fácil que encuentra, en lugar de aprender reglas generalizadas, el entrenamiento genera individuos con muchas reglas específicas, adaptadas a clasificar solo los ejemplos de entrenamiento y que escapan al efecto del operador de purga. Consecuentemente, al llegar a la fase de test (cuando se evalua el grado de aprendizaje adquirido), el rendimiento de estos individuos no es satisfactorio.

B. Presión de generalización

Para conseguir conjuntos de reglas con el grado de generalización correcto a continuación se proponen diversos operadores, que afectan diversas fases del GA como la evaluación y la selección. Hay que aclarar que lo que realmente buscamos es obtener buenas soluciones con el menor número de reglas posible, ya que hay mas posibilidades que estos estén bien generalizadas.

B.1 Presión por penalización

La primera medida para favorecer la generalización de los individuos es penalizar los individuos con demasiadas reglas. Esta penalización se puede obtener modificando la evaluación de los individuos de la siguiente forma:

$$I_i' = \begin{cases} I_i & \text{Si } r \leq r_{max} \\ I_i \cdot factorPen & \text{Si } r > r_{max} \end{cases} \tag{2}$$

$$factorPen = max \begin{cases} 1 - 0.0005 \cdot (r - r_{max})^2 \\ 0.01 \end{cases} \tag{3}$$

r es el número de reglas del individuo a penalizar y r_{max} es el número máximo de reglas permitido sin penalizar. Podemos observar este sistema de penalización en la figura 1, donde vemos que la penalización se realiza de manera suave y gradual para no limitar la evolución, ya que de usar un método de penalización más rígido, el factor predominante en la evaluación de los individuos puede pasar a ser el número de reglas y no el grado de aprendizaje adquirido por el individuo. Este problema también puede aparecer en propuestas de penalización más simples de la forma $eval' = \frac{r_{max}}{r} eval$ ya que la penalización se aplica siempre, y no solo cuando se supera el límite r_{max}

Por otro lado, una penalización de la forma propuesta no tiene los problemas de las propuestas basadas en sumas ponderadas [17] donde los factores se tienen que mantener en escalas similares, y es fácilmente adaptable a formas de evaluaciones basadas en minimización de errores.

B.2 Comparación entre individuos por tamaño

El objetivo de generar conjuntos de reglas reducidos siempre tiene que estar supeditado a que los conjuntos mantengan su capacidad de clasificación. Esta relación entre los objetivos (principal y secundario) puede no ser interpretada correctamente por el GA si la función de evaluación no se define correctamente. Por otro lado, no siempre sabemos como dimensionar r_{max} para el operador de penalización del apartado anterior. Puede ser interesante buscar algún método para romper esta dependencia, tratando los objetivos por separado, y poder expresar correctamente la importancia de cada uno de ellos.

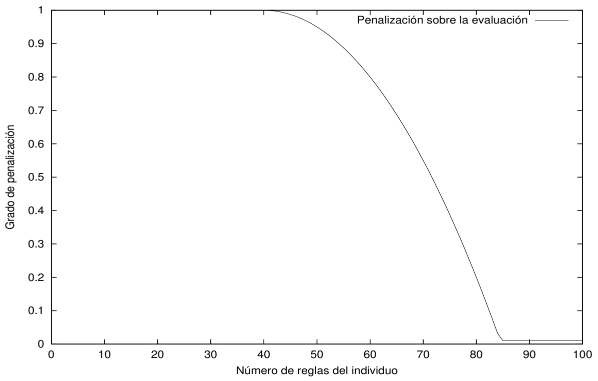

Fig. 1. Penalización de la evaluación de los individuos según el número de reglas. Límite fijado en 40 reglas

Con esta idea pero sin entrar de lleno en el terreno de la optimización multi-objetivo, donde también se han realizado trabajos con objetivos similares [10], [18] se propone un método para comparar individuos que tenga en cuenta en su medida adecuada los dos objetivos.

Esta idea solo es aplicable si el algoritmo de selección actúa comparando directamente individuos o si se basa en el rango de los individuos dentro de la población. La selección por torneo se ajusta perfectamente al primer caso.

Para aplicar esta idea se define un mecanismo de comparación de individuos con los siguientes criterios:

- Supongamos dos individuos, a y b, con evaluaciones a_{eval} y b_{eval} y tamaño en reglas a_{numR} y b_{numR}, definamos e_{comp} como la diferencia máxima entre las evaluaciones de los individuos dentro de la cual se comparará dos individuos por su número de reglas. Por lo tanto, no se trata de una comparación por el material genético de los individuos, solo de su evaluación.
- Si $|a_{eval} - b_{eval}| < e_{comp}$ entonces:
  - Si $a_{numR} < b_{numR}$ entonces a es mejor que b.
  - Si $a_{numR} > b_{numR}$ entonces b es mejor que a.
  - Si $a_{numR} = b_{numR}$ entonces se va al caso general
- Sino, pasa a usarse el caso general: es mejor el individuo con mayor evaluación.

e_{comp} definirá la importancia que tiene el tamaño del conjunto de reglas respecto a la evaluación del individuo, pudiéndose ajustar según el grado de generalización que se considere correcto aunque a costa de la evaluación.

Como el proceso de purga descrito anteriormente, este operador es aconsejable activarlo después de haber realizado algunas iteraciones para evitar que en determinados problemas se puede quedar atascado en un mínimo local con todos los individuos de la población con solo una regla.

IV. Planificación experimental

En esta sección se expone qué configuraciones del GA se probarán para ver la eficiencia y la eficacia de los métodos de generalización expuestos en este artículo. Para realizar estas pruebas se han usado diversos problemas, tanto sintéticos como reales, que detallemos primero.

A. Datos de prueba

Para tener una visión amplia de la actuación de las técnicas probadas, se han usado nueve problemas con diferente número de atributos, número de clases, número de ejemplos,... Cuatro de ellos (*mx11, lrnsyn, tao y led*) son problemas sintéticos y los otros cinco (*mamm, lrn, iris, pima y breast*) son basados en datos reales.

De los sintéticos, *mx11* [5] es el multiplexor de 11 entradas binarias, *lrnsyn* es un problema multiclase construido a partir de reglas con una distribución desigual entre las clases, inspirado en el problema real *lrn* que describiremos en el siguiente parágrafo. *Tao* es un problema con atributos reales y fronteras entre clases no ortogonales [19]. Finalmente el *led*, que proviene del repositorio de aprendizaje artificial de la Universidad de California en Irvine (UCI) [20], es un problema de 10 clases con atributos binarios donde las muestras son representaciones de un visualizador numérico de 7 segmentos, pero con un 10% de muestras con ruido.

De los problemas reales, tres son también del repositorio UCI: Pima-indians-diabetes (*pima*), *iris* y breast-cancer-winsconsin (*Breast*). Los otros dos son de nuestro repositorio privado. *Lrn* contiene casos de notas de alumnos a partir de las cuales predecimos su nota final [7], [21]. En *mamm* tratamos de diagnosticar cáncer de mama a partir de los datos extraídos a imágenes de mamografías [22].

Las características de los 9 problemas se pueden resumir en la tabla I.

Las particiones de los datos de cada problema en entrenamiento y test usarán la metodología *stratified ten-fold cross-validation* [9], usando tres versiones de los *folds* para cada problema excepto en el problema del *led* donde debido al volumen de ejemplos se ha realizado una única partición de tipo *hold-out* [9] con 2000 ejemplos de entrenamiento y 4000 de test.

B. Configuraciones del GA de las pruebas

Para evaluar las técnicas expuestas, se realizará pruebas con las configuraciones de la tabla II. Todas las configuraciones usaran el operador de purga, ya que es absolutamente necesario. Podremos demostrar esta necesidad observando las gráficas de evolución del tamaño medio de los individuos de la población en diversos problemas. Probaremos el operador de penalización (prueba Pen) y el de comparación por tamaño con dos valors diferentes de e_{comp} (0.001 para Com1 y 0.01 para Comp2) para comprobar si cambia su comportamiento de forma notable. Finalmente mezclaremos "Comp1" y "Pen" en la configuración "Todo". Como base para comparar los métodos, también tendremos la configuración "Nada" donde solo se aplica purga.

Respecto al resto de parámetros del GA, algunos

TABLA I

CARACTERÍSTICAS DE LOS PROBLEMAS DE LAS PRUEBAS

Nombre	ejemplos	Atr. reales	Atr. discretos	Clases
breast	699	-	9	2
iris	150	4	-	3
led	6000	-	7	10
mamm	216	21	-	2
mx11	2048	-	11	2
pima	768	8	-	2
tao	1888	2	-	2
lrn	648	4	2	5
lrnsyn	1000	-	4	5

TABLA II

CONFIGURACIONES DE OPERADORES DE GENERALIZACIÓN A EVALUAR

Nombre	Penalización	Comp. por tamaño	e_{comp}
Comp1	-	x	0.001
Comp2	-	x	0.01
Nada	-	-	-
Pen	x	-	-
Todo	x	x	0.001

se han adaptado a medida para cada problema a partir de tests cortos (tabla IV) mientras que otros tienen valores comunes para todas las pruebas (tabla III). Hay que hacer mención especial para la mutación: debido a que la probabilidad de mutación es a nivel de bit, el operador es muy sensible al crecimiento del tamaño de los individuos. Por ejemplo, una población donde los individuos tengan el doble de reglas que otra población sufrirá también el doble de mutaciones por iteración si usan la misma p_{mut}. Es por eso que se ha adaptado la probabilidad de mutación a medida también para cada configuración de operadores a evaluar teniendo en cuenta también el tamaño de cada regla, como se puede ver en la tabla V.

V. RESULTADOS

En esta sección presentamos los resultados de las pruebas realizadas y procedemos a su análisis. En primer lugar, y tal como se ha dicho en la sección anterior, todas las pruebas realizadas usaban el operador de purga presentado en este artículo. Para demostrar su necesidad a continuación mostramos dos gráficas donde se puede observar la evolución de la media de reglas por individuo a lo largo de las iteraciones para dos problemas, el *mamm* (figura 3) y el *mx11* (figura 2) usando la configuración "Nada". En estas figuras, superpuestas a la evolución del número

TABLA III

PARÁMETROS COMUNES DEL GA

Parámetro	Valor
Prob. de cruce	0.6
Itera. activ. purga	30
Itera. activ. compar. tamaño	30
Núm. mín. reglas en inicializ.	10
Núm. máx. reglas en inicializ.	15
Alg. selección	Torneo de tamaño 3
Reemplazo	Generacional con elitismo
Núm. mín. reglas en la purga	3
Tamaño de la población	200

TABLA IV

PARÁMETROS DEL GA SEGÚN EL PROBLEMA A SOLUCIONAR

Código	Parámetro
#iter	Número máximo de iteraciones
P_{uno}	Probabilidad de uno en la inicialización
cruce	Operador de cruce
r_{max}	Número màximo de reglas de la penalización
r_{size}	Tamaño en genes de cada regla

Probl.	Parámetro				
	#iter	P_{uno}	cruce	r_{max}	r_{size}
breast	2500	50%	2px	15	91
iris	1500	75%	1px	15	41
led	500	50%	1px	35	15
mamm	2500	75%	2px	25	105
mx11	1500	75%	1px	20	23
pima	2000	75%	2px	25	81
tao	3000	50%	1px	40	49
lrn	2000	50%	1px	25	34
lrnsyn	1500	50%	1px	20	21

TABLA V

PROBABILIDADES DE MUTACIÓN PARA CADA PROBLEMA Y TIPO DE PRUEVA

Probl.	Comp1	Comp2	Pen	Nada	Todo
breast	0.00015	0.00016	0.00017	0.00010	0.00018
iris	0.00087	0.00084	0.00049	0.00042	0.00081
led	0.00240	0.00400	0.00140	0.00110	0.00250
mamm	0.00020	0.00032	0.00020	0.00020	0.00015
mx11	0.00120	0.00190	0.00140	0.00077	0.00170
pima	0.00030	0.00080	0.00035	0.00014	0.00040
tao	0.00060	0.00130	0.00038	0.00030	0.00070
lrn	0.00068	0.00210	0.00065	0.00050	0.00080
lrnsyn	0.00160	0.00240	0.00170	0.00070	0.00240

de reglas, también se muestra la evolución de la media de evaluación del mejor individuo de la población. Esta gráfica demuestra que la purga no afecta de forma negativa a la evolución de la evaluación.

Dado que el objetivo de las técnicas presentadas en este artículo era, por un lado, reducir el coste computacional del sistema y, por el otro, generar conjuntos de reglas correctamente generalizados, para cada tipo de prueba y problema se mostrará las medias de tiempo de ejecución (en segundos) y de reglas del mejor individuo de la población así como las de porcentaje de acierto en entrenamiento y de *cross-validation* para la fase de test. De todos estos resul-

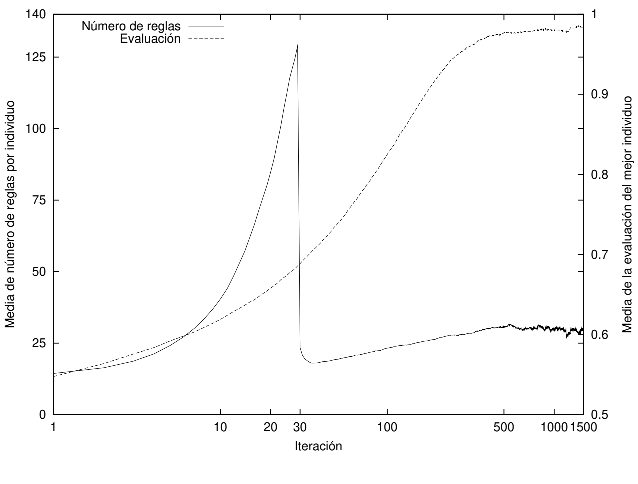

Fig. 2. Evolución del número de reglas de la población:Problema *mx11*

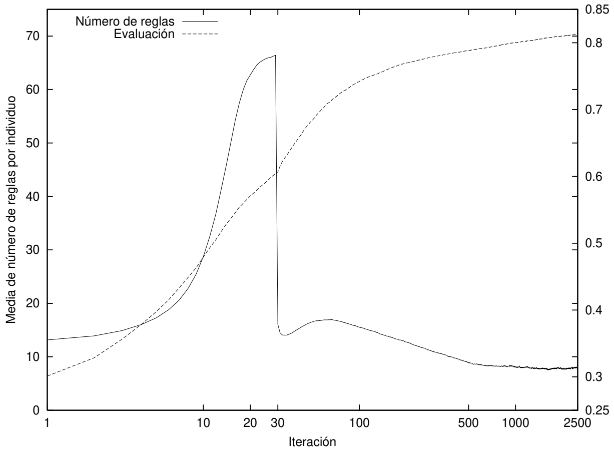

Fig. 3. Evolución del número de reglas de la población:Problema *mamm*

tados también se mostrará su desviación estándar. Las pruebas se han realizado en diferentes procesadores y los tiempos de ejecución se han adaptado para tener todos la misma referencia, que es un PC con procesador Intel Celeron a 600MHz y sistema operativo Linux. El lenguaje de programación es C++.

Para cada tipo de prueba y juego de datos se han realizado 5 ejecuciones con semillas del generador de números aleatorios distintas, es decir, 150 ejecuciones para las pruebas de tipo "Cross-validation" (10 folds x 3 versiones de los folds x 5 semillas) y 5 para las pruebas de tipo "Hold-out". Pasemos a mostrar los resultados de las pruebas realizadas que se pueden observar en la tabla VI y donde para cada problema se ha marcado en negrita qué tipo de prueba ha sido superior para cada uno de los resultados que mostramos.

En primer lugar se puede observar claramente que en todos los problemas de prueba, los métodos de generalización propuestos superan la configuración base (la llamada "nada"), y que también en todos los problemas excepto en uno (el problema *mamm*) el tiempo de ejecución es menor y el conjunto de reglas resultante es más reducido (en algunos casos, como en el del problema "Breast" las diferencias son notables).

Si intentamos ver qué método de generalización es el mejor, observamos que hay mucha variabilidad en los distintos problemas. Para aclararla crearemos un ranking que tenga en cuenta los tres mejores métodos para cada problema de prueba asignando 3 puntos al mejor, 2 puntos al segundo y un punto al tercero para cada problema. Este ranking aparece en la tabla VII y analizándolo tampoco podemos determinar de forma clara qué método es el mejor aunque la configuración "todo" tiene ligera ventaja, lo único que podemos decir claramente es que el método de penalización por si solo no funciona bien, confirmando el problema que intuíamos para dimensionar correctamente r_{max}.

Pero si separamos el ranking por problemas sintéticos (tabla VIII) y reales (tabla IX), las posiciones del ranking sufren cambios. En los problemas reales destaca la configuración "Comp2", la configuración de tipo comparación por tamaño donde e_{comp} es mayor, es decir, donde le damos más importancia al tamaño del individuo. En las pruebas iniciales realizadas con este operador teníamos la impresión que el e_{comp} de "Comp2" era demasiado grande y frenaría el proceso evolutivo y, de hecho, observado los porcentajes de acierto en la fase de entrenamiento parece cierta esta afirmación, ya que la prueba "Comp1" siempre es superior, cosa que también pasa en la fase de test para los problemas sintéticos. Así pues, ha sido una sorpresa el rendimiento de "Comp2" en los problemas reales.

TABLA VI

Resultados de las pruebas realizadas

Probl.	Prueba	Train	Test	Reglas	Tiempo
mx11	Comp1	100.0±0.3	100.0±0.3	19.7±6.5	214.5±106.1
	Comp2	99.9±0.7	99.9±0.9	15.8±3.6	200.5±95.5
	Pen	100.0±0.1	100.0±0.2	17.9±2.1	225.5±110.9
	Nada	99.9±0.4	99.9±0.5	32.9±9.0	283.2±168.7
	Todo	**100.0±0.0**	**100.0±0.0**	**16.0±2.7**	**179.7±69.6**
tao	Comp1	**95.2±1.2**	**93.6±2.0**	18.8±3.6	620.7±71.9
	Comp2	94.1±1.4	92.5±2.4	**7.7±1.7**	**488±41.4**
	Pen	93.6±1.6	91.8±2.7	31.9±3.9	652±63
	Nada	94.1±1.7	92.4±2.4	41.8±8.1	712.3±95.6
	Todo	95.1±1.0	93.4±2.0	18.9±3.5	689.6±94.9
led	Comp1	**76.1±0.2**	**74.6±0.2**	18.6±2.9	474.3±23.2
	Comp2	72.9±2.1	72.1±2.4	**10.8±1.0**	**365.1±39.6**
	Pen	75.8±0.4	74.3±0.5	26.2±2.2	387.8±62.4
	Nada	74.8±2.0	73.4±2.1	30.4±6.3	431.4±18.2
	Todo	75.2±0.8	74.0±0.7	16.8±1.3	386.2±25.6
lrnsyn	Comp1	**99.8±1.0**	97.9±2.5	20.4±8.0	95.9±50.6
	Comp2	99.6±1.3	98.2±2.5	**13.4±4.6**	103.7±62.1
	Pen	99.5±1.4	97.6±3.0	19.0±2.1	122.4±66.1
	Nada	99.8±0.7	97.2±2.6	37.4±8.6	172±117.7
	Todo	99.7±1.3	**98.4±2.7**	15.9±3.4	**89.5±48.6**
lrn	Comp1	75.1±2.8	65.1±5.6	21.6±4.6	227±42
	Comp2	71.2±3.2	**65.5±5.7**	**6.9±1.7**	161.2±21.2
	Pen	73.8±2.5	64.3±5.7	22.2±3.3	206.2±29.2
	Nada	**75.3±2.4**	64.8±5.8	30.1±5.8	**120±27**
	Todo	74.5±2.7	64.5±5.4	17.9±3.5	193.9±29.8
iris	Comp1	98.3±0.8	**91.6±7.1**	**5.5±1.4**	22.9±5.1
	Comp2	98.2±0.7	91.6±7.1	5.6±1.2	22.4±3
	Pen	**98.5±0.6**	90.7±7.5	10.2±2	25±4
	Nada	98.5±0.8	90.7±7.1	12.9±3.4	28±5.7
	Todo	98.3±0.7	90.8±7.5	5.8±1.4	**21.6±3.4**
mamm	Comp1	81.1±1.9	63.9±9.9	13.6±4.1	307±57
	Comp2	79.5±1.8	**64.8±9.4**	**6.0±2.1**	359.4±42.2
	Pen	80.4±1.8	63.2±10.1	14.9±4.1	**295±61.6**
	Nada	**81.3±2.2**	63.0±9.7	18.2±5.1	329.4±82
	Todo	79.7±1.9	64.4±9.8	9.9±3.7	251.6±39.4
pima	Comp1	**88.8±1.4**	68.8±5.1	37.0±7.2	373.56±94.3
	Comp2	85.1±1.9	**70.6±5.4**	**11.2±3.6**	**216.7±58.1**
	Pen	86.9±1.4	68.7±5.4	25.3±0.8	272.2±40.8
	Nada	88.5±1.5	68.6±5.6	60.1±8.3	429.5±90.4
	Todo	87.3±1.3	69.3±5.3	23.5±2.0	271.9±44.7
breast	Comp1	**99.7±0.2**	85.7±4.4	48.2±7.3	951.8±154.6
	Comp2	98.2±2.7	89.7±4.4	15.8±6.3	567±126.9
	Pen	98.7±0.5	90.6±3.9	15.8±0.9	490.8±43.6
	Nada	99.3±0.4	84.8±3.9	89.0±4.3	1705±189.2
	Todo	98.7±0.5	**90.6±3.5**	**14.7±1.6**	**344.9±32.9**

TABLA VII

Ranking general de los métodos de generalización probados

Prueba	1º	2º	3º	Puntos	Ranking
Comp1	3	1	4	15	2
Comp2	3	2	2	15	2
Pen	-	3	-	6	4
Nada	-	-	1	1	5
Todo	3	3	2	17	1

TABLA VIII
Ranking de los métodos de generalización probados para problemas sintéticos

Prueba	1°	2°	3°	Puntos	Ranking
Comp1	2	-	1	8	2
Comp2	-	1	1	3	4
Pen	-	2	-	4	3
Nada	-	-	-	0	5
Todo	2	1	1	9	1

TABLA IX
Ranking de los métodos de generalización probados para problemas reales

Prueba	1°	2°	3°	Puntos	Ranking
Comp1	1	1	3	8	2
Comp2	3	1	-	11	1
Pen	-	1	-	2	4
Nada	-	-	1	1	5
Todo	1	2	1	8	2

VI. Conclusiones y lineas de futuro

En este artículo se han presentado diversos métodos con el objetivo doble de controlar el efecto de *bloat* de los individuos y generar soluciones correctamente generalizadas para sistemas clasificadores genéticos bajo el enfoque de Pittsburgh. Los objetivos se han intentado conseguir mediante acciones para reducir en la fase de entrenamiento el número de reglas de los individuos de la población pero manteniendo su capacidad de clasificación, ya que este tipo de individuos tiene más garantías de estar correctamente generalizados.

El operador de purga ha resultado eficiente en controlar el crecimiento desmesurado de los individuos de la población sin afectar de forma negativa el proceso de evolución pero sin favorecer tampoco una buena generalización. Para favorecer la generalización, se ha propuesto dos métodos, uno basado en penalización de la evaluación y otro basado en comparaciones entre individuos teniendo en cuenta el tamaño de sus conjuntos de reglas que se aplican en el proceso de selección.

Los resultados de las pruebas nos han demostrado que la clasificación mejora con los dos métodos de generalización planteados, y también se reduce el tiempo de ejecución de las pruebas, a parte, nos han dejado también conclusiones concretas para cada método:

- El método de penalización tiene el problema de dimensionar correctamente r_{max}, si no conocemos de antemano ningún dato del problema a solucionar. Este problema ha quedado demostrado en las pruebas realizadas, donde su rendimiento ha sido dispar.
- Respecto al operador de comparación por tamaño, ha resultado funcionar mejor que la penalización, ya que el tamaño de los individuos de la población se va adaptando a las necesidades del problema de forma automática. Respecto al valor de e_{comp}, en los problemas reales ha resultado ser mejor un valor grande del parámetro, ya que ha creado individuos más compactos y generalizados. No obstante, creemos que aumentar mucho este parámetro puede frenar la evolución, y es necesario estudiar el tema más a fondo.

Otro elemento que consideramos importante es el detalle con que se ha expuesto las probabilidades de mutación para las pruebas realizadas, ya que dimensionar correctamente este parámetro es muy crucial en representaciones de individuo de tamaño variable. Seria muy interesante pues, encontrar un método donde la mutación se adaptase a las necesidades de cada momento de la evolución.

Finalmente, analizando los resultados comparando con otros sistemas [11], [23] se puede observar que en los problemas con atributos reales el sistema aquí expuesto tiene resultados inferiores. Este es un punto muy importante a tratar en el futuro.

Agradecimientos

Este artículo se ha realizado con el soporte del *Departament d'Universitats, Recerca i Societat de la Informació de la Generalitat de Catalunya*, de la Comisión Interministerial de Ciencia y Tecnología y el Fondo de Investigación Sanitaria con las becas 2001FI 00514, CICYT/Tel98-0408-02 y FIS 00/0033-02 respectivamente. Los resultados de este trabajo han sido parcialmente obtenidos usando equipamiento cofinanciado por la *Direcció General de Recerca de la Generalitat de Catalunya (D.O.G.C 30/12/1997)*. Finalmente agradecemos también a *Enginyeria i Arquitectura La Salle* el soporte a nuestro grupo de investigación en sistemas inteligentes.

Referencias

[1] John H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

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

[3] Stephen F. Smith, "Flexible learning of problem solving heuristics through adaptive search," in *Proceedings of the 8th International Joint Conference on Artificial Intelligence (IJCAI-83)*, Los Altos, CA, 1983, pp. 421–425, Morgan Kaufmann.

[4] John H. Holland, "Escaping Brittleness: The possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems," in *Machine learning, an artificial intelligence approach. Volume II*, Mitchell, Michalski, and Carbonell, Eds., pp. 593–623. Morgan Kaufmann, 1986.

[5] Steward W. Wilson, "Classifier fitness based on accuracy.," *Evolutionary Computation*, vol. 3, no. 2, pp. 149–175, 1995.

[6] Jaume Bacardit i Peñarroya, *Creació d'un assistent digital per a la predicció del fracàs acadèmic basat en computació evolutiva*, Proyecto final de carrera s0491, Enginyeria La Salle, Universitat Ramon Llull, 2000.

[7] Elisabet Golobardes, Xavier Llorà, Josep Maria Garrell, David Vernet, and Jaume Bacardit, "Genetic classifier system as a heuristic weighting method for a case-based classifier system," *Butlletí de l'Associació Catalana l'Intel.ligència Artificial*, vol. 22, pp. 132–141, 2000.

[8] Kenneth A. De Jong et al, "Learning concept classification rules using genetic algorithms," *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 651–656, 1991.

[9] Ian H. Witten and Eibe Frank, *Data Mining: practical machine learning tools and techniques with java implementations*, Morgan Kaufmann, 2000.

[10] Ester Bernadó i Mansilla and Josep Maria Garrell i Guiu, "Multiobjective learning in a genetic classifier

system (MOLeCS)," *Butlletí de l'Associació Catalana l'Intel.ligència Artificial*, vol. 22, pp. 102–111, 2000.

[11] Xavier Llorà and Josep Maria Garrell i Guiu, "Knowledge-independent data mining with fine-grained parallel evolutionary algorithms," in *Proceeding of the Genetic and Evolutionary Computation Conference (GECCO2001)*. 2001, Morgan Kaufmann.

[12] David Beasley, David R. Bull, and Ralph R. Martin, "An overview of genetic algorithms: Part 2, research topics," *University Computing*, vol. 15, no. 4, pp. 170–181, 1993.

[13] David E. Goldberg and Kalyanmoy Deb, "A comparative analysis of selection schemes used in genetic algorithms," in *Foundations of Genetic Algorithms*. pp. 69–93, Morgan Kaufmann.

[14] Lawrence Davis, *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991.

[15] Terence Soule and James A. Foster, "Effects of code growth and parsimony pressure on populations in genetic programming," *Evolutionary Computation*, vol. 6, no. 4, pp. 293–309, Winter 1998.

[16] Peter Nordin and Wolfgang Banzhaf, "Complexity compression and evolution," in *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, L. Eshelman, Ed., Pittsburgh, PA, USA, 15-19 1995, pp. 310–317, Morgan Kaufmann.

[17] Dipankar Dasgupta and Fabio A. Gonzalez, "Evolving complex fuzzy classifier rules using a linear tree genetic representation," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*. 2001, pp. 299–305, Morgan Kaufmann.

[18] Fernando Jiménez, Antonio F. Gómez-Skarmeta, Hans Roubos, and Robert Babuška, "Accurate, transparent and compact fuzzy models for function approximation and dynamic modeling through multi-objective evolutionary optimization," in *Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization (EMO 2001)*. 2001, pp. 653–667, Springer-Verlag.

[19] Xavier Llorà and Josep Maria Garrell, "Inducing partially-defined instances with evolutionary algorithms," in *Proceedings of the 18th International Conference on Machine Learning*, 2001.

[20] C.L. Blake and C.J. Merz, "UCI repository of machine learning databases," 1998.

[21] David Vernet, Maria Salamó, Carles Vallespí, Joan Camps, Elisabet Golobardes, and Jaume Bacardit, "¿cómo predecir la evolución del alumno?," in *Proceedings de las VII Jornadas sobre Enseñazas Universitarias de Informática (JENUI2001)*, 2001.

[22] J. Martí, X. Cufí, J. Regincós, and et al., "Shape-based feature selection for microcalcification evaluation," in *Imaging Conference on Image Processing, 3338:1215-1224*, 1998.

[23] Ester Bernadó, Xavier Llorà, and Josep M. Garrell, "XCS and GALE: a comparative study of two learning classifier systems with six other learning algorithms on classification tasks," in *Fourth International Workshop on Learning Classifier Systems - IWLCS-2001*, 2001, pp. 337–341.