Incorporación de conocimiento en forma de restricciones sobre algoritmos evolutivos para la búsqueda de funciones de similitud

A. Fornells Herrera, J. Camps Dausà, E. Golobardes i Ribé
J.M. Garrell i Guiu

Grup de Recerca en Sistemes Intel·ligents
Enginyeria i Arquitectura La Salle, Universitat Ramon Llull
Quatre Camins 2, 08022 Barcelona
{afornells, joanc, elisabet, josepmg}@salleURL.edu

Resumen

Uno de los puntos claves en el razonamiento basado en casos es la recuperación de los casos más parecidos mediante una función de similitud. Debido a la complejidad de los dominios en los problemas reales, es difícil definir funciones precisas y se han de utilizar funciones de propósito general, que dan resultados aceptables dentro de unos márgenes de error. El objetivo de este artículo es proponer una estrategia de exploración de funciones de similitud basada en la Computación Evolutiva y el Razonamiento Basado en Casos. La gran diferencia respecto otras propuestas es la facilidad de incorporar restricciones mediante gramáticas, las cuales orientan la búsqueda de la solución. De esta manera, la función de similitud encontrada permite una recuperación más eficiente de los casos y por tanto, mejorar los resultados de clasificación.

Palabras Claves: Computación evolutiva, razonamiento basado en casos, función de similitud

1. Introducción

El Razonamiento Basado en Casos (CBR) [1] es un paradigma basado en el aprendizaje analógico: dado un problema nuevo, recupera los más similares a partir de su experiencia utilizando una función de similitud, la cual permite obtener los más similares para adaptarlos y proponer una solución. El CBR justifica la clasificación mediante la similitud entre los problemas previamente resueltos y el nuevo. Por tanto, el porcentaje de acierto está altamente relacionado con esta función, la cual debe ser propuesta por expertos en el dominio.

No obstante, esto no es una tarea trivial porque normalmente los dominios reales son complejos y/o parcialmente desconocidos, con lo que los expertos no pueden modelar su comportamiento. Sin embargo, esto no significa que estos no puedan solventarse, ya que una solución es utilizar funciones de similitud de propósito general (i.e. la métrica Euclidiana). El problema está en que no se obtienen normalmente los resultados esperados, y es muy difícil intentar ajustar dichas funciones con restricciones del dominio. Sería deseable poder definir mecanismos para buscar funciones de similitud para un dominio concreto.

Los Algoritmos Genéticos (GA) [6] son algoritmos de búsqueda de propósito general basados en los principios de la evolución. A partir de una población de individuos (conjunto de posibles soluciones) se recombina su información mediante operaciones genéticas, para obtener una solución aplicando presión selectiva. La gran pregunta es, ¿puede esto usarse para buscar funciones de similitud?

La Programación Genética (GP) [11] es una variante de la Computación Evolutiva (CE) que se caracteriza porque los individuos son programas que pueden ejecutarse directamente. No es descabellado pensar que es-

ta virtud puede aprovecharse para buscar funciones de similitud. En este caso se tiene el gran inconveniente que el espacio de búsqueda es demasiado grande, y podemos encontrarnos ante un problema *NP-Hard*. Con el fin de reducir este efecto se pueden añadir restricciones, aunque el GP no permite añadir restricciones de una manera 'natural' ya que su integración es muy compleja.

Sin embargo, existe una variante de la CE que permite introducir de una manera sencilla restricciones. *Grammar Evolution* (GE) [19] persigue la misma idea que GP pero con una representación y ciclo de vida iguales a los de los GA. La única diferencia está en la evaluación de los individuos, etapa en la que se aplica un proceso de mapeo con una gramática en *Backus Naur Form* (BNF) para transformarlo en una función. De esta manera, se separa el espacio de búsqueda y el de soluciones, con la ventaja que ahora es sencillo añadir restricciones para reducir el espacio de búsqueda, ya que sólo se modifica la gramática.

Este trabajo presenta la integración de un núcleo GE y otro CBR, para disponer de un generador de funciones ad hoc a un dominio que permita añadir restricciones de una manera sencilla y personalizada. Esto permite definir un modelo general de comportamiento a partir de la gramática, permitiendo así encontrar la función de similitud buscada.

El artículo se estructura de la siguiente manera. La sección 2 presenta antecedentes y trabajos relacionados. La sección 3 se explica la integración y funcionamiento del CBR y el GE. La sección 4 muestra el análisis de resultados sobre diferentes problemas. Finalmente, la sección 5 expone las conclusiones y líneas futuras.

2. Antecedentes

Cuando se habla de mejorar los resultados de clasificación del CBR, tradicionalmente todos los focos se centran en la función de similitud. Las funciones más populares son Minkowski [3], Mahalanobis [15], Camberra, Chebychev, Cuadrática, Correlación, y Chicuadrado [13], funciones basadas en hiperrectángulos [20] o las funciones Heterogéneas [22].

Los resultados obtenidos con estas funciones son mejorables si se pueden conseguir funciones a medida para el problema. Existen muchos sistemas que utilizan estrategias basadas en la CE con el fin de mejorar la 'adaptación' de las funciones, como por ejemplo ajustando la importancia de los atributos mediante esquemas de ponderación [12] [10], o usando procesos de selección de características [9].

Otros enfoques se centran en utilizarlos como algoritmos de extracción de características como en [18] [2] mediante GP. No obstante, ninguna de estas estrategias permite modelar la función con restricciones o conocimiento de específico del experto.

Nuestro grupo de investigación trabaja en la detección de cáncer de mama [7]. Actualmente en el proyecto TIC2002-04160-C02-02 se desarrolla una herramienta de recuperación de imágenes Mamográficas por análisis de contenido (HRIMAC) para el asesoramiento en el diagnóstico del cáncer de mama. Uno de los objetivos de este sistema es definir la manera como recuperar historiales de pacientes. En [8] y [5] se propuso un sistema basado en CBR y GP para encontrar funciones de similitud ad hoc un dominio. Los resultados fueron positivos, pero era complejo añadir conocimiento específico por parte de los expertos que agilizara la búsqueda de la función.

No obstante, existen variantes dentro de la computación evolutiva que permiten la incorporación de restricciones mediante gramáticas BNF [19], libres de contexto [21] [16], o simplemente definen un tipo especial de representación para datos complejos como los vectores o las matrices [14].

3. BRAIN: La plataforma

BRAIN (*hyBRid system to discover And Improve similarity fuNctions*) es una plataforma desarrollada con el fin de integrar diferentes paradigmas basados en la computación evolutiva (GP y GE), para definir/optimizar funciones de similitud aplicables al CBR. A continuación exponemos brevemente el CBR y GE, para posteriormente centrarnos en su integración e interacción.

3.1. Razonamiento basado en casos

El CBR es una técnica enmarcada dentro del aprendizaje analógico inspirada en la filosofía de las personas para resolver problemas: Dado un problema nuevo, intenta resolverlo a partir de la experiencia de otros similares anteriormente resueltos. El método se caracteriza por cuatro fases [1]: la recuperación (*Retrieval phase*) de casos anteriores ya resueltos que sean similares al nuevo caso que se desea resolver; la adaptación (*Reuse phase*) de estos al caso a resolver; la revisión (*Revise phase*) de la solución propuesta; y, finalmente, la fase de almacenamiento (*Retain phase*) de la información relevante obtenida.

Uno de los puntos claves es la fase de recuperación, en la que mediante una función de similitud se comparan los casos resueltos respecto el nuevo para obtener los más parecidos. Aunque existen muchas funciones de propósito general, los resultados del CBR pueden mejorarse si se define una función ad hoc al problema. Muchas veces esto no es posible debido a la complejidad o falta de conocimiento del dominio, factores que dificultan su definición.

3.2. *Grammar Evolution*

GE [19] es una variante de la CE que busca encontrar programas como en GP. Su ciclo de vida es casi idéntico al de los GA: A partir de una población inicial de individuos que tienen asignado un *fitness*, durante un conjunto de generaciones la población va evolucionando (aplicando operadores genéticos: cruce, mutación y reproducción) hasta encontrar la solución deseada. Cualquier mejora aplicable a los GA, lo es también a GE.

La diferencia de GE con GA está en la manera como se evalúan los individuos, en este caso, el *fitness* representa el grado de funcionamiento del programa. A partir de una gramática BNF se transforman los individuos (representados por cadena de bits) en programas (representados por cadenas de caracteres) que siguen una sintaxis. Una gramática BNF se define con una túpla {N, T, S, P}, donde 'N' y 'T' representan el conjunto de no terminales y terminales respectivamente, 'S' la producción inicial, y 'P' define las reglas de cada una de las producciones de los no terminales.

La principal ventaja respecto GP es la separación del espacio de búsqueda del de soluciones, aspecto que permite añadir restricciones complejas sin necesidad de modificar el algoritmo. En cambio, introducir restricciones en GP puede ser una tarea muy compleja que implica una recodificación total de los operadores, que puede no ser factible.

Para poder aplicar el proceso de mapeo genotipo-fenotipo de un individuo, es necesario agrupar los bits en *codons*. Un *codon* es un conjunto de 'X' bits que representa un entero, donde 'X' dependerá del número más grande de reglas que tenga una producción.

Inicialmente, el programa asociado al individuo está representado por los no terminales de la producción inicial 'S'. Mediante un proceso iterativo, se substituyen los no terminales del programa por una de sus posibles reglas según la ecuación 1. En cada una de las substituciones se utiliza un *codon* del individuo. El proceso finaliza cuando el programa asociado sólo tiene terminales, y por tanto, es ejecutable. Si se acaban todos los *codons* y aún no se ha finalizado la traducción se aplica el operador de *wrapping*, que consiste en reutilizar *codons* desde el principio. Con el fin de evitar un bucle infinito se limita su aplicación.

$$regla = MOD \frac{codon}{\#\ reglas\ no\ terminal} \quad (1)$$

3.3. Integración de CBR y GE en BRAIN

El objetivo es disponer de un sistema capaz de encontrar una función de similitud ad hoc a un problema. Para ello se propone integrar dentro de un mismo entorno un sistema con capacidad para explorar soluciones (GE), y otro que pueda evaluar las propuestas (CBR).

Como se aprecia en la figura 1, el punto de interacción del GE y el CBR es cuando se evalúan los individuos. En la figura 2 se muestra este proceso más detalladamente. A partir de la cadena de bits y la gramática BNF que modela la función que queremos buscar, se generan un conjunto de códigos intermedios, los

cuales asocian un entero a cada terminal posible (constantes, atributos de ambos casos, y operadores unarios y binarios). Con esta información el CBR en la fase de recuperación substituye, para cada ejecución de la función de similitud, los terminales que representan los atributos por sus valores reales (caso a resolver y resuelto). Una vez esto, se aplica un parser que procesa la expresión que tiene como resultado la similitud entre ambos casos.

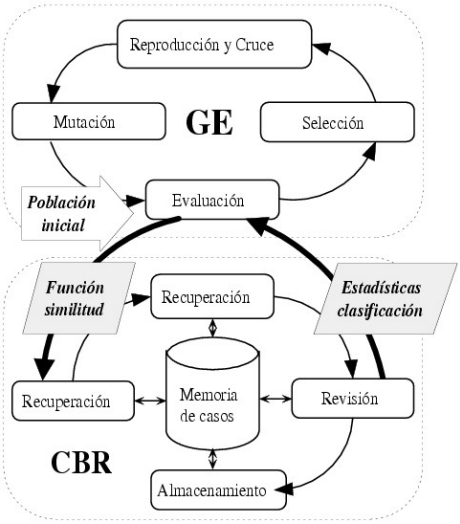

Figura 1: Interacción entre GE y CBR

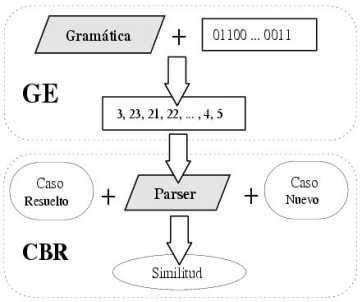

Figura 2: Proceso de evaluación de un individuo

Inicialmente se parte de una población de individuos generada aleatoriamente, aunque puede forzarse que el 100 % (*Full*), 50 % (*Ramped Half and Half*) y 0 % (*Grow*) de la población tengan todos los atributos. Cuando el CBR ha ejecutado todos los ejemplos de *test*, genera estadísticas que contienen información sobre la simulación: % clasificaciones correctas, incorrectas y no clasificados, sensitividad y especificidad de cada ejecución, así como las medias y desviación típica si se ejecuta en modo *CrossValidation*. A partir de esta información, el GE aplica la ecuación 2 para asignar un *fitness* al individuo. Además de estos, podrian usarse otros criterios.

$$fitness = w_1\,\%sen. + w_2\,\%esp. - w_3\,\%no \quad (2)$$

Donde:

- w_i representa la importancia.
- %sen. representa el % de sensitividad.
- %esp. representa el % de especificidad.
- %no representa el % de no clasificados.

Durante todo el proceso el CBR no memoriza casos.

4. Experimentación y comparación

Para evaluar el impacto de las funciones generadas respecto las de propósito general, se ha realizado una experimentación sobre diferentes problemas del repositorio UCI (HS y SO) [4] y otros propios (MF y TA), que se detallan en la tabla 1. Se han elegido problemas con diferente número de atributos para analizar su influencia sobre los resultados. Todos los problemas tienen atributos reales para poder aplicar las funciones de propósito general, y dos clasificaciones posibles para aplicar la ecuación 2 de evaluación.

Problema	Atrib.	Train	Test
Tao (TA)	2+1	1700	800
Heart-Statlog(HS)	14+1	243	27
Mamografias (MF)	21+1	196	20
Sonar (SO)	60+1	187	21

Tabla 1: Problemas estudiados

Todos los problemas se han estudiado con diferentes configuraciones del CBR y GE+CBR. En la tabla 2 se detallan las funciones de similitud, los métodos de ponderación de los atributos, y finalmente los diferentes casos utilizados en la votación de la clase mayoritaria.

Parámetros	Valores
Función	Clark Ec.3 Cosinus Ec.4 Manhattan Ec.5 Minkowski (r=1, 2, 3) Ec.6
Ponderación	Sin ponderación *Principal Component Analysis* Correlación Muestral
K-NN	1, 3, 5

Tabla 2: Configuraciones del CBR

$$distancia(x,y) = \sqrt{\frac{\sum_{i=1}^{p} |x_i - y_i|}{\sum_{i=1}^{p} |x_i - y_i|}} \quad (3)$$

$$distancia(x,y) = \frac{\sum_{i=1}^{p} w_i(x_i \cdot y_i)}{\sqrt{\sum_{i=1}^{p} x_i^2 \cdot \sum_{i=1}^{p} y_i^2}} \quad (4)$$

$$distancia(x,y) = \sum_{i=1}^{p} |x_i - y_i| \quad (5)$$

$$distancia(x,y) = \sqrt[r]{\sum_{i=1}^{p} w_i |x_i - y_i|^r} \quad (6)$$

Donde:

- x, y son los casos a comparar.
- x_i , y_i es el valor del atributo 'i' de 'x' y de 'y' respectivamente.
- w_i es la ponderación del atributo 'i'.
- p es el número de atributos del caso.

Parámetros	Valores
Generaciones	200
Población	500
Finalización	0.95% del fitness ideal
Operadores	Prob. Cruce (0.8) Prob. Repro. (0.2) Prob. Mutar (0.2) Max. Wrappring (2)
Selección	Torneo de 2 individuos
# Codons	10 codons por atributo
Evaluación	Ec. 2 con diferentes w_i
Inicialización	*Grow* *Full* *Ramped*
Reemplazamiento	*Steady-State* (SS) Generacional (GN)
Semillas	10

Tabla 3: Configuraciones del GE+CBR

Las configuraciones del GE (tabla 3) se centran en definir unos parámetros fijos y cambiar la manera de inicializar, evaluar y reemplazar los individuos de la población. La figura 3 muestra la gramática empleada, en la que fijamos restricciones que permiten generar funciones de similitud con sentido, como que sólo se permitan operaciones directas entre atributos del mismo tipo, siempre tengan un peso específico dentro de la función, y además, se tenga en cuenta el número de atributos que se utilizan. Para considerar válidas las evaluaciones del CBR siempre se realiza un *10-fold stratified cross-validation*.

Gramática G={N, T, S, P}
N = {<expr>,<op_binario>,<op_unario>,<var>, <op_bis>,<constantes>}
T = x_0,..,x_{P-1},y_0,..,y_{P-1},+,-,*,/,%,abs,sqr,sqrt
S = <expr> / (#Atributos usados)
P =
<expr> ← (<expr> <op_binario> <expr>)
← <op_unario> (<expr>)
← <constantes> * (<var>)
<op_binario> ← + | - | * | / |%
<op_unario> ← abs | sqr | sqrt
<var> ← x_0<op_bis>y_0
← ...
← x_{P-1}<op_bis>y_{P-1}
<op_bis> ← + | - | * | / |%
<constantes> ← 0 | 0.1 | ... | 1

Figura 3: Gramática definida en el GE

Las tablas 4, 6, 8 y 10 agrupan los mejores resultados, y los 5, 7, 9 y 11 sus configuraciones. Los parámetros a analizar son la sensitividad, la especificidad y los aciertos, con sus desviaciones típicas. La exploración de funciones se ha hecho con el mismo número de individuos y generaciones independientemente del problema. En este sentido, podemos entender el número de atributos como un indicador de complejidad.

Para problemas donde el número de atributos es reducido (TA, HS o MF) obtenemos una mejora de los resultados respecto las funciones de propósito general. En cambio, si hay un volumen más importante de atributos (SO), el GE necesitaría más generaciones y individuos para poder encontrar una solución mejor.

Función	%Sens.	%Spe.	%Acierto
Clark	95.1(2.1)	96.1(2.7)	95.3(1.4)
Cosinus	00.0(0.0)	50.0(0.3)	50.0(0.3)
Manhattan	96.4(2.2)	96.5(2.1)	96.4(1.5)
Mink.(r=1)	96.4(2.2)	96.5(2.1)	96.4(1.5)
Mink.(r=2)	96.7(2.4)	96.5(1.9)	96.6(1.4)
Mink.(r=3)	96.9(2.8)	96.4(2.1)	96.6(1.6)
GE+CBR	**97.6(0.9)**	**95.3(1.5)**	**96.8(1.4)**

Tabla 4: Resultados para el TA

Función	Configuración
Clark	K-NN=3, PCA
Cosinus	K-NN=1, Sin ponderación
Manhattan	K-NN=3
Mink.(r=1)	K-NN=5, Sin ponderación
Mink.(r=2)	K-NN=5, Sin ponderación
Mink.(r=3)	K-NN=5, PCA
GE+CBR	*Full*, SS, w_i={0.4,0.4,0.2}

Tabla 5: Mejores configuraciones para el TA

Función	%Sens.	%Spe.	%Acierto
Clark	44.8(1.1)	0 (0)	44.44(0)
Cosinus	28.3(32.5)	53.5(3.4)	51.1(6.15)
Manhattan	82.22(8.2)	81.9(8.3)	81.4(6.4)
Mink.(r=1)	81.1(9.9)	82.8(6.5)	81.4(6.4)
Mink.(r=2)	81.2(8.2)	78.9(9.8)	79.2(6.8)
Mink.(r=3)	81.6(9.5)	79.2(8.6)	80.0 (9.6)
GE+CBR	**85.1(8.5)**	**85.8(5.4)**	**85.2(6.4)**

Tabla 6: Resultados para el HS

Función	Configuración
Clark	K-NN=3, Correlación Muestral
Cosinus	K-NN=3, PCA
Manhattan	K-NN=3
Mink.(r=1)	K-NN=3, Correlación Muestral
Mink.(r=2)	K-NN=3, PCA
Mink.(r=3)	K-NN=5, PCA
GE+CBR	*Ramped*, GN, w_i={0.4,0.4,-0.2}

Tabla 7: Mejores configuraciones para el HS

Función	%Sens.	%Spe.	%Acierto
Clark	61.2(9.7)	70.5(7.1)	65.7(8.6)
Cosinus	10.0(30.0)	56.2(2.5)	56.4(4.9)
Manhattan	61.3(6.4)	71.2(6.4)	66.6(8.4)
Mink.(r=1)	62.3(6.9)	73.7(11.3)	68.1(10.0)
Mink.(r=2)	62.0(8.6)	72.3(8.7)	67.1(12.4)
Mink.(r=3)	61.5(7.3)	71.1(7.2)	66.6(8.12)
GE+CBR	**61.6(7.5)**	**81.3(10.9)**	**69.0(9.6)**

Tabla 8: Resultados para el MF

Función	Configuración
Clark	K-NN=5, Correlación Muestral
Cosinus	K-NN=3, PCA
Manhattan	K-NN=5
Mink.(r=1)	K-NN=3, PCA
Mink.(r=2)	K-NN=5, PCA
Mink.(r=3)	K-NN=3, Sin ponderación
GE+CBR	*Ramped*, SS, w_i={0.4,0.4,-0.2}

Tabla 9: Mejores configuraciones para el MF

Otro dato importante a estudiar es el tiempo necesario para encontrar una solución. Todas las configuraciones se han simulado sobre un clúster formado por 6 máquinas (P-IV a 2.6 Ghz con 1 GB de RAM) controladas por *OpenMosix* [17]. La operación más costosa es

Función	%Sens.	%Spe.	%Acierto
Clark	77.4(6.6)	94.6(6.5)	82.6(8.9)
Cosinus	49.7(13.1)	42.2(8.2)	45.1(8.3)
Manhattan	86.1(9.2)	88.1(10.6)	87.9(11.4)
Mink.(r=1)	88.0(7.4)	91.2(8.1)	88.9(7.7)
Mink.(r=2)	**88.2(6.3)**	**88.5(11.7)**	**87.9(11.4)**
Mink.(r=3)	86.0(9.2)	88.1(10.6)	86.5(12.1)
GE+CBR	85.7(9.1)	89.2(8.5)	86.7(8.5)

Tabla 10: Resultados para el SO

Función	Configuración
Clark	K-NN=3, Correlación Muestral
Cosinus	K-NN=3, PCA
Manhattan	K-NN=3
Mink.(r=1)	K-NN=1, Correlación Muestral
Mink.(r=2)	K-NN=1, Correlación Muestral
Mink.(r=3)	K-NN=1, Sin ponderación
GE+CBR	*Ramped*, GN, w_i={0.2,0.2,-0.6}

Tabla 11: Mejores configuraciones para el SO

Problema	T. evaluación	T. total
TA	0.09	16
HS	0.01	23
MF	0.01	29
SO	0.02	38

Tabla 12: $\overline{T_{eje}}$ parcial y total en minutos

la evaluación de un individuo por parte del CBR, la cual a su vez está condicionada por el número de atributos y casos de los conjuntos de *train* y *test* (Tabla 1). La tabla 12 resume los tiempos medios en evaluar un individuo en el CBR, y el total en ejecutar una configuración para cada problema.

Aunque el problema del TA es el que tiene el mayor tiempo de evaluación por individuo (debido a que tiene muchos ejemplos de *train* y *test*), es el que antes finaliza. Como se ha comentado antes, el número de atributos representa la complejidad del problema, al tener sólo 2 atributos el sistema converge mucho antes que finalicen todas las generaciones. En cambio, el resto de problemas (que tienen una proporción similar de ejemplos *train* y *test*) la relación del tiempo de ejecución es proporcional al número de atributos.

Teniendo en cuenta todas las configuraciones estudiadas sobre todos los problemas, se han realizado aproximadamente unos 80 millones de ejecuciones del CBR en modo *10-fold stratified cross-validation*.

Para acabar, las ecuaciones 7 y 8 muestran a detalle de ejemplo las funciones de similitud encontradas para los problemas de TA y HS.

$$f(x,y) = 0{,}5(x_0 - y_0) + \sqrt{\|0{,}9(x_0 - y_0) + \\ + \sqrt{\|(x_1 - y_1)\|}\,\|} + 0{,}5(x_1 - y_1)\| \quad (7)$$

$$f(x,y) = \|0{,}3(x_2 - y_2) + 0{,}1(x_{11} - y_{11}) + \\ + 0{,}5(x_{12} - y_{12})\| \quad (8)$$

5. Conclusiones y líneas futuras

En las secciones anteriores se ha presentado la integración de un CBR y un GE en la plataforma BRAIN, para conseguir funciones de similitud ad hoc a un dominio.

La gran baza del sistema es la facilidad con la que los expertos del dominio pueden añadir restricciones o conocimientos complejos (expresiones condicionales, etc.) con una gramática. Gracias a estas reducciones en el espacio de búsqueda las posibilidades de encontrar una solución buena son más elevadas. No obstante, hay que ser cauteloso al definirlas ya que unas restricciones demasiado fuertes pueden impedir encontrarla.

Los resultados obtenidos son positivos, ya que usando complejidades diferentes (2, 13, 21 y 60 atributos) sobre una configuración fija de individuos y generaciones, el sistema ha encontrado soluciones en tiempos razonables. Además, al aplicar el *t-Student* respecto la mejor configuración del CBR obtenemos una mejora significativa en HS, en TA y MF es poco significativa, y en SO los resultados no se mejoran. Por tanto, el sistema necesitaria realizar más exploraciones ante problemas de mayor complejidad para alcanzar mejores resultados, es decir, el número de atributos influye en el número de generaciones y tamaño de la población. Este último aspecto también influye en el tiempo de evaluación del individuo en el CBR, así como el número de ejemplos de *train* y *test*.

Se abren diferentes líneas futuras para mejorar los resultados, que podrían dividirse en la evaluación y evolución de individuos. Por parte de la evaluación, sería interesante definir nuevas maneras de evaluar las estadísticas del CBR, por ejemplo, que se puntúe más a los individuos que representen funciones más robustas. También se podría jugar con la posibilidad que los individuos se mapearan sobre diferentes gramáticas en la evaluación, y disponer así de una población más diversa en cuanto a soluciones. Estas gramáticas podrían definirse por ejemplo a partir de esquemas básicos de funciones de propósito general, que tuvieran en cuenta condiciones, etcétera.

En la evolución de individuos, se podrían añadir nuevos operadores de cruce que realizaran intercambio de bloques, o dotar a la parte del GE de mecanismos basados en los GA para mejorar la diversidad y calidad de los individuos de la población.

Agradecimientos

Quisiéramos agradecer al Ministerio de Ciencia y Tecnología, y al Fondo Europeo de Desarrollo Regional (FEDER) por su soporte a los proyectos TIC2002-04036-C05-03 y TIC2002-04160-C02-02, y al *Departament d'Universitats, Recerca i Societat de la Informació* (DURSI) el apoyo proporcionado mediante las becas 2005FIR 00237 y 2002 SGR-00/55. Finalmente, agradecemos a *Enginyeria i Arquitectura La Salle*, de la Universitat Ramon Llull, por el soporte a nuestro Grupo de Investigación en Sistemas Inteligentes.

Referencias

[1] A. Aamodt and E. Plaza. Case-based reasoning: Foundations issues, methodological variations, and system approaches. *IA Communications*, 7:39–59, 1994.

[2] M. Ahluwalia and L. Bull. Coevolving functions in genetic programming: Classification using k-nearest-neighbour. *In Proceedings of the genetic and evolutionary computation conference*, pages 947–952, 1999.

[3] B. Bachelor. Pattern recognition: Ideas in practice. *New York: Plenum Press*, pages 71–72, 1978.

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

[5] J. Camps, J.M. Garrell, E. Golobardes, and D. Vernet. Diseño de funciones de similitud para el razonamiento basado en casos usando programación genética: estudio con problemas sintéticos. *MAEB*, pages 409–416, 2003.

[6] D. E. Goldberg. *Genetic Algorithm in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[7] E. Golobardes, X. Llorà, M. Salamó, and J. Martí. Computer aided diagnosis with case-based reasoning and genetic algorithms. *Journal of Knowlegde Based Systems*, pages 45–52, 2002.

[8] E. Golobardes, M. Nieto, M. Salamó, J.Camps, G. Calzada, J. Martí, and D. Vernet. Generació de funcions de similitud mitjançant la programació genètica pel raonament basat en casos. *CCIA*, 25:100–107, 2001.

[9] J. Jarmulak, S. Craw, and R. Crowe. Genetic algorithms to optimise cbr retrieval. In *EWCBR '00: Proceedings of the 5th European Workshop on Advances in Case-Based Reasoning*, pages 136–147, London, UK, 2000. Springer-Verlag.

[10] J. Kelly and L. Davis. Hybridizing the genetic algorithms and the k nearest neighbors. *Proceedings of the 4th international conference on genetic algorithms*, pages 337–383, 1991.

[11] J. R. Koza. *Genetic Programming. Programing of computers by means of natural selection*. MIT Press, 1992.

[12] L. Kuncheva. Editing for the k-nearest neighbors rule by a genetic algorithm. *Pattern Recognition Letters, Special Issue on Genetic Algorithms*, 16:809–814, 1995.

[13] R. Michalski, R. Stepp, and E. Diday. A recent advance in data analysis: Clustering objects into classes characterized by conjunctive concepts. *Progress in Pattern Recognition, Vol. 1, Laveen N. Kanal and Azriel Rosenfeld (Eds.). New York: North-Holland*, pages 33–56, 1981.

[14] D. J. Montana. Strongly typed genetic programming. Technical Report 7866, Cambridge, MA 02138, USA, 7 1993.

[15] Morton Nadler and Eric P. Smith. Pattern recognition engineering. *New York: Wiley*, pages 293–294, 1993.

[16] M. Oltean. Evolving evolutionary algorithms for function optimization. In Ken Chen, editor, *Proceedings of the 5th International Workshop on Frontiers in Evolutionary Algorithms*, pages 295–298, Research Triangle Park, Carolina, 2003.

[17] Open source linux cluster project. http://openmosix.sourceforge.net/.

[18] M.L. Raymer, W.F. Punch, E. Goodman, and L.A. Kung. Genetic programing for improved data mining: An application to the biochemistry of proteins interactions. *Proceedings of the firt annual conference*, pages 375–380, 1996.

[19] C. Ryan, J. J. Collins, and M. O'Neill. Grammatical evolution: Evolving programs for an arbitrary language. In Wolfgang Banzhaf and Riccardo Poli, editors, *Proceedings of the First European Workshop on Genetic Programming*, volume 1391, pages 83–95, Paris, 14-15 1998. Springer-Verlag.

[20] S. Salzberg. A nearest hyperrectangle learning method. *Machine Learning*, 6:277–309, 1991.

[21] P. A. Whigham. Grammatically-based genetic programming. In Justinian P. Rosca, editor, *Proceedings of the Workshop on GP: From Theory to Real-World Applications*, pages 33–41, Tahoe City, California, USA, 9 1995.

[22] D.R. Wilson and T.R. Martinez. Improved heterogeneus distance functions. *Journal of Artificial Intelligence Research (JAIR)*, 6:1–34, 1997.