UNIVERSIDAD DE GRANADA

Programa Oficial de Doctorado en Tecnologías de la Información y la Comunicación

Departamento de Ciencias de la Computación e Inteligencia Artificial

Nuevos Modelos de Clasificación mediante Algoritmos Evolutivos

Memoria de Tesis presentada por

Alberto Cano Rojas

como requisito para optar al grado de Doctor en Informática

Directores

Sebastián Ventura Soto

Amelia Zafra Gómez

Granada Enero 2014

Editor: Editorial de la Universidad de Granada
Autor: Alberto Cano Rojas
D.L.: GR 1049-2014
ISBN: 978-84-9028-963-1

UNIVERSITY OF GRANADA

Official PhD Program on Information and Communication Technologies

Department of Computer Science and Artificial Intelligence

New Classification Models through Evolutionary Algorithms

A Thesis presented by

Alberto Cano Rojas

as a requirement to aim for the degree of

Ph.D. in Computer Science

Advisors

Sebastián Ventura Soto

Amelia Zafra Gómez

Granada January 2014

La memoria de Tesis Doctoral titulada "*Nuevos Modelos de Clasificación mediante Algoritmos Evolutivos*", que presenta Alberto Cano Rojas para optar al grado de Doctor, ha sido realizada dentro del Programa Oficial de Doctorado en Tecnologías de la Información y la Comunicación del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada, bajo la dirección de los doctores Sebastián Ventura Soto y Amelia Zafra Gómez, cumpliendo los requisitos exigidos a este tipo de trabajos, y respetando los derechos de otros autores a ser citados, cuando se han utilizado sus resultados o publicaciones.

Granada, Enero de 2014

El Doctorando

Fdo: Alberto Cano Rojas

El Director

Fdo: Dr. Sebastián Ventura Soto

La Directora

Fdo: Dra. Amelia Zafra Gómez

Mención de Doctorado Internacional

Esta tesis cumple los criterios establecidos por la Universidad de Granada para la obtención del Título de Doctor con Mención Internacional:

1. Estancia predoctoral mínima de 3 meses fuera de España en una institución de enseñanza superior o centro de investigación de prestigio, cursando estudios o realizando trabajos de investigación relacionados con la tesis doctoral:

   Department of Computer Science, School of Engineering, Virginia Commonwealth University, Richmond, Virginia, United States.
   Responsable de la estancia: **Ph.D. Krzysztof Cios**, Professor and Chair.

2. La tesis cuenta con el informe previo de dos doctores o doctoras expertos y con experiencia investigadora acreditada pertenecientes a alguna institución de educación superior o instituto de investigación distinto de España:

   - **Ph.D. Virgilijus Sakalauskas**, Associate Professor, Department of Informatics, Kaunas Faculty of Humanities, Vilnius University, Lithuania.
   - **Ph.D. Vojislav Kecman**, Associate Professor, Department of Computer Science, Virginia Commonwealth University, Richmond, Virginia, United States.

3. Entre los miembros del tribunal evaluador de la tesis se encuentra un doctor procedente de una institución de educación superior distinto de España y diferente del responsable de la estancia predoctoral:

   Ph.D. Mario Gongora, Principal Lecturer, Department of Informatics, Faculty of Technology at De Montfort University, Leicester, United Kindgom.

4. Parte de la tesis doctoral se ha redactado y presentado en dos idiomas: castellano e inglés.

Granada, Enero de 2014

El Doctorando

Fdo: Alberto Cano Rojas

Tesis Doctoral parcialmente subvencionada por la Comisión Interministerial de Ciencia y Tecnología (CICYT) con el proyecto **TIN2011-22408**.

Asímismo ha sido subvencionada por el programa predoctoral de Formación del Profesorado Universitario (FPU) del Ministerio de Educación (referencia **AP-2010-0042**), convocatoria publicada en el B.O.E. N^0 20 de 24 de enero de 2011 y resuelta en el B.O.E. N^0 305 de 20 de diciembre de 2011.

Agradecimientos

Quisiera agradecer a todos los que han hecho posible conseguir que este día haya llegado, por su apoyo, influencia, educación, interés y cariño que me han formado personal y profesionalmente.

A mis padres, Antonio y Ángela, a mi hermano Carlos, a Fran y al resto de mi familia por haber estado siempre ahí.

A mis directores Sebastián y Amelia por su dedicación y apoyo a lo largo de los últimos 5 años.

A mis compañeros de laboratorio, con los que he compartido grandes momentos.

Resumen

Los algoritmos evolutivos inspiran su funcionamiento en base a los procesos evolutivos naturales con el objetivo de resolver problemas de búsqueda y optimización. En esta Tesis Doctoral se presentan nuevos modelos y algoritmos que abordan problemas abiertos y nuevos retos en la tarea de clasificación mediante el uso de algoritmos evolutivos. Concretamente, nos marcamos como objetivo la mejora del rendimiento, escalabilidad, interpretabilidad y exactitud de los modelos de clasificación en conjuntos de datos complejos. En cada uno de los trabajos presentados, se ha realizado una búsqueda bibliográfica exhaustiva de los trabajos relacionados en el estado del arte, con el objetivo de estudiar propuestas similares de otros autores y su empleo como comparativa con nuestras propuestas.

En primer lugar, hemos analizado el rendimiento y la escalabilidad de los modelos evolutivos de reglas de clasificación, que han sido mejorados mediante el uso de la programación paralela en tarjetas gráficas de usuario (GPUs). El empleo de GPUs ha demostrado alcanzar una gran eficiencia y rendimiento en la aceleración de los algoritmos de clasificación. La programación de propósito general en GPU para tareas de aprendizaje automático y minería de datos ha resultado ser un nicho de investigación con un amplio abanico de posibilidades. El gran número de publicaciones en este campo muestra el creciente interés de los investigadores en la aceleración de algoritmos mediante arquitecturas masivamente paralelas.

Los modelos paralelos desarrollados en esta Tesis Doctoral han acelerado algoritmos evolutivos poblacionales, paralelizando la evaluación de cada unos de los individuos, además de su evaluación sobre cada uno de los casos de prueba de la función de evaluación. Los resultados experimentales derivados de los modelos propuestos han demostrado la gran eficacia de las GPUs en la aceleración de los algoritmos, especialmente sobre grandes conjuntos de datos, donde anteriormente era inviable la ejecución de los algoritmos en un tiempo razonable. Esto abre la

puerta a la aplicación de esta tecnología a los nuevos retos de la década en aprendizaje automático, tales como Big Data y el procesamiento de streams de datos en tiempo real.

En segundo lugar, hemos analizado la dualidad intpretabilidad-precisión de los modelos de clasificación. Los modelos de clasificación han buscado tradicionalmente maximizar únicamente la exactitud de los modelos de predicción. Sin embargo, recientemente la interpretabilidad de los modelos ha demostrado ser de gran interés en múltiples campos de aplicación tales como medicina, evaluación de riesgos crediticios, etc. En este tipo de dominios es necesario motivar las razones de las predicciones, justificando las características por las que los modelos ofrecen tales predicciones. No obstante, habitualmente la búsqueda de mejorar la interpretabilidad y la exactitud de los modelos es un problema conflictivo donde ambos objetivos no se pueden alcanzar simultáneamente.

El problema conflictivo de la interpretabilidad y exactitud de los modelos de clasificación ha sido tratado mediante la propuesta de un modelo de clasificación basado en reglas interpretables, llamado ICRM, que proporciona reglas que producen resultados exactos y a la vez, son altamente comprensibles por su simplicidad. Este modelo busca buenas combinaciones de comparaciones atributo-valor que compongan el antecedente de una regla de clasificación. Es responsabilidad del algoritmo evolutivo encontrar las mejores combinaciones y las mejores condiciones que compongan las reglas del clasificador.

En tercer lugar, hemos analizado conjuntos datos no balanceados. Este tipo de conjuntos de datos se caracterizan por el alto desbalanceo entre las clases, es decir, el número de instancias que pertenecen a cada una de las clases de datos no se encuentra equilibrado. Bajo estas circunstancias, los modelos tradicionales de clasificación suelen estar sesgados a predecir las clases con un mayor número de ejemplos, olvidando habitualmente las clases minoritarias. Precisamente, en ciertos dominios el interés radica verdaderamente en las clases minoritarias, y el verdadero problema es clasificar correctamente estos ejemplos minoritarios. En esta Tesis Doctoral hemos realizado una propuesta de un modelo evolutivo de clasificación basado en gravitación. La idea de este algoritmo se basa en el concepto físico de gravedad y la interacción entre las partículas. El objetivo era desarrollar un modelo de predicción que lograse buenos resultados tanto en conjuntos de datos balanceados como no balanceados. La adaptación de la función de ajuste teniendo en cuenta las características del dominio y propiedades del conjunto de datos ha ayudado a lograr su buen funcionamiento en ambos tipos de datos. Los resultados obtenidos han demostrado alcanzar una gran exactitud, y a su vez una suave y buena generalización de la predicción a lo largo del dominio del conjunto de datos.

Todos los modelos propuestos en esta Tesis Doctoral han sido evaluados bajo un entorno experimental apropiado, mediante el uso de un gran número de conjuntos de datos de diversa dimensionalidad, número de instancias, atributos y clases, y mediante la comparación de los resultados frente a otros algoritmos del estado del arte y recientemente publicados de probada calidad. La metodología experimental empleada busca una comparativa justa de la eficacia, robustez, rendimiento y resultados de los algoritmos. Concretamente, los conjuntos de datos han sido particionados en un esquema de 10 particiones cruzadas, y los experimentos han sido repetidos al menos 10 veces con diferentes semillas, para reflejar la naturaleza estocástica de los algoritmos evolutivos. Los resultados experimentales obtenidos han sido verificados mediante la aplicación de tests estadísticos no paramétricos de comparaciones múltiples y por pares, tales como el de Friedman, Bonferroni-Dunn, o Wilcoxon, que apoyan estadísticamente los mejores resultados obtenidos por los modelos propuestos.

Abstract

This Doctoral Thesis presents new computational models on data classification which address new open problems and challenges in data classification by means of evolutionary algorithms. Specifically, we pursue to improve the performance, scalability, interpretability and accuracy of classification models on challenging data.

The performance and scalability of evolutionary-based classification models were improved through parallel computation on GPUs, which demonstrated to achieve high efficiency on speeding up classification algorithms.

The conflicting problem of the interpretability and accuracy of the classification models was addressed through a highly interpretable classification algorithm which produced very comprehensible classifiers by means of classification rules.

Performance on challenging data such as the imbalanced classification was improved by means of a data gravitation classification algorithm which demonstrated to achieve better classification performance both on balanced and imbalanced data.

All the methods proposed in this Thesis were evaluated in a proper experimental framework, by using a large number of data sets with diverse dimensionality and by comparing their performance against other state-of-the-art and recently published methods of proved quality. The experimental results obtained have been verified by applying non-parametric statistical tests which support the better performance of the methods proposed.

Table of Contents

List of Acronyms

AI Artificial Intelligence

ANN Artificial Neural Network

ARM Association Rule Mining

AUC Area Under the Curve

BNF Backus-Naur Form

CFG Context-Free Grammar

CUDA Compute Unified Device Architecture

DGC Data Gravitation Classification

DM Data Mining

EA Evolutionary Algorithm

EP Evolutionary Programming

FPR False Positive Rate

FNR False Negative Rate

G3P Grammar Guided Genetic Programming

GA Genetic Algorithm

GCCL Genetic Cooperative-Competitive Learning

GP Genetic Programming

GPops/s Genetic Programming operations per second

GPU Graphic Processing Unit

GPGPU General Purpose Graphic Processing Unit

IR Imbalance Ratio

IRL Iterative Rule Learning

KDD Knowledge Discovery in Databases

NN Nearest Neighbour

MIL Multiple Instance Learning

ML Machine Learning

MOO Multi-Objective Optimization

NSGA-II Non-dominated Sorting Genetic Algorithm II

RBS Rule-Based System

ROC Receiver Operating Characteristic

SVM Support Vector Machine

Part I: Ph.D. Dissertation

1 Introduction

Discovering knowledge in large amounts of data collected over the last decades has become significantly challenging and difficult, especially in high-dimensional and large-scale databases. Knowledge discovery in databases (KDD) is the process of discovering useful, nontrivial, implicit, and previously unknown knowledge from a collection of data [1]. Data mining (DM) is the step of the KDD process that involves the use of data analysis tools to discover valid patterns and relationships in large data sets. The data analysis tools used for DM include statistical models, mathematical methods, and machine learning algorithms. Machine learning (ML) is a branch of artificial intelligence that aims for the construction of computer algorithms that can learn from data to accomplish a task.

Classification is a data mining and machine learning task which consists in predicting the class membership of uncategorized examples, whose label is not known, using the properties of examples in a model learned previously from training examples, whose label was known. Classification tasks include a broad range of domains and real world application: disciplines such as bioinformatics, medical diagnosis, image recognition, and financial engineering, among others, where domain experts can use the model learned to support their decisions [2, 3].

Evolutionary computation and its application to machine learning and data mining, and specifically, to classification problems, has attracted the attention of researchers over the last decade [4–8]. Evolutionary algorithms (EAs) are search methods inspired by natural evolution to find a reasonable solution for data mining and knowledge discovery [9, 10]. Genetic programming (GP) is a specialization of EAs, where each individual represents a computer program. It is a machine learning technique used to optimize a population of computer programs according to a fitness function that determines the program's ability to perform a task. Recently, GP has been applied to different common data mining tasks such as classification [11], feature selection [12], and clustering [13].

In spite of the numerous studies and applications of EAs to DM, there are still many open problems and new arising issues to the scientific community. These open tasks call for new computational methods capable of solving the new challenges of the present decade in DM, focusing on performance, scalability, accuracy and interpretability of the classification models on heterogeneous types of data. This chapter introduces the different issues that will be faced in the dissertation and provides their the motivation and justification.

1.1 Performance and scalability of classification algorithms

The increasing amount of data is a big challenge for machine learning algorithms to perform efficiently. This issue, known as *Big data*, comprises a collection of data sets so large and complex that it becomes very difficult to process using traditional methods. Therefore, performance and scalability of algorithms to large-scale and high-dimensional datasets become crucial. Many parallelization strategies have been adopted to overcome this problem of machine learning algorithms. The use of multi-core CPUs, many-core graphic processing units (GPUs), and distributed computation systems is nowadays vital to handle vast amounts of data under the constraints of computation time.

Parallel computation designs and implementations have been employed to speed up evolutionary algorithms [14, 15], including multi-core and distributed computing [16, 17], master–slave models [18], and grid computing environments [19, 20]. Over the last few years, GPUs have focused increasing attention in academia.

GPUs are devices with many-core architectures and massive parallel processor units, which provide fast parallel hardware at a fraction of the cost of a traditional parallel system. Since the introduction of the computer unified device architecture (CUDA) [21] in 2007, researchers all over the world have harnessed the power of the GPU for general purpose GPU computing (GPGPU) [22–25]. The use of GPGPU models has already been studied for speeding up algorithms within the framework of evolutionary computation and data mining [26–29], achieving high performance and promising results.

1.2 Interpretability vs accuracy of classification models

The interpretability of the classifiers is also a key issue in decision support systems. There are hundreds of classification algorithms in the literature which provide accurate classification models but many of them must be regarded as black boxes, i.e., they are opaque to the user. Artificial neural networks (ANN) [30], support vector machines (SVM) [31], and instance-based learning methods [32] belong to this type of algorithms. Opaque predictive models prevent the user from tracing the logic behind a prediction and obtaining interesting knowledge previously unknown from the model. These classifiers do not permit human understanding and inspection, they are not directly interpretable by an expert and it is not possible to discover which are the relevant attributes to predict the class of an example. This opacity prevents them from being used in many real-life knowledge discovery applications where both accuracy and comprehensibility are required, such as medical diagnosis [33], credit risk evaluation [34], and decision support systems [35], since the prediction model must explain the reasons for classification. On the other hand, there are machine learning approaches which overcome this limitation and provide transparent and comprehensible classifiers such as decision trees [36] and rule-based systems [37].

Evolutionary Algorithms, and specifically Genetic Programming, have been successfully applied to build decision trees and rule-based systems easily. Rule-based systems are especially user-friendly and offer compact, understandable, intuitive and accurate classification models. To obtain comprehensibility, accuracy is often sacrificed by using simpler but transparent models, achieving a trade-off between

accuracy and comprehensibility. Even though there are many rule based classification models, it has not been until recently that the comprehensibility of the models is becoming a more relevant objective. Proof of this trend is found in recent studies of this issue [38–41], i.e, improving the comprehensibility of the models is a new challenge as important as obtaining high accuracy.

1.3 Improving performance on challenging data

The problem of learning from imbalanced data is also a novel challenging task that attracted attention of both academical and industrial researchers. It concerns the performance of learning algorithms in the presence of severe class distribution skews (some classes have many times more instances than other classes). Traditional algorithms fail minority class predictions in the presence of imbalanced data because of the bias of classical algorithms to majority class instances. Thereby, this issue calls for new algorithm capable of handling appropriately both balanced and imbalanced data. Proof of this trend is found in many recent studies of this issue [42–45]. Therefore, it is essential to adapt algorithms to consider the presence of imbalanced and challenging data.

For example, the nearest neighbor (NN) algorithm [46] is an instance-based method which might be the simplest classification algorithm. Its classification principle is to classify a new sample with the class of the closest training sample. The extended version of NN to k neighbors (KNN) and their derivatives are indeed one of the most influential data mining techniques, and they have been shown to perform well in many domains [47]. However, the main problem with these methods is that they severely deteriorate with imbalanced, noisy data or high dimensionality: their performance becomes very slow, and their accuracy tends to deteriorate as the dimensionality increases, especially when classes are nonseparable, imbalanced, or they overlap [48].

In recent years, new instance-based methods based on data gravitation classification (DGC) have been proposed to solve the aforementioned problems of the nearest neighbor classifiers [49–51]. DGC models are inspired by Newton's law of universal gravitation and simulate the accumulative attractive force between data samples to perform the classification. These gravitation-based classification methods extend

the NN concept to the law of gravitation among the objects in the physical world. The basic principle of DGC is to classify data samples by comparing the data gravitation among the training samples for the different data classes, whereas KNNs vote for the k training samples that are the closest in the feature space.

DGC models can be combined with evolutionary-based learning to improve the accuracy of the classifiers [49]. The learning process may be useful for detecting properties of data in order to help to overcome the problems of the presence of imbalanced or noisy data.

2

Objectives

The main objective of this thesis is to develop new classification models using the evolutionary computation paradigm which solve open problems and new challenges in the classification task domain. Specifically, the following objectives were pursued to successfully accomplish this aim:

- Analysis of the state of the art in the classification task domain to identify open problems and new challenges. Review of recently proposed and best performance algorithms to analyze their approach to new challenges and open issues in classification. Listing of the unsolved problems of algorithms and classification databases, focusing on performance, scalability, accuracy and interpretability of the classification models.
- Review of the application of evolutionary-based computation to the classification task, giving special attention to the Genetic Programming paradigm. Analysis of the open issues of algorithms when addressing large and complex databases. Design of evolutionary-based model to solve the classification problem.

- Design and implementation of parallel computation algorithms which speed up the classification task, especially seeking for their scalability to large-scale and high-dimensional datasets. Analysis of the efficiency and the performance of current algorithms after their parallelization on multi-core CPUs and many-core GPUs. Development of new, efficient, scalable and high performance classification algorithms on large datasets, taking advantage of the massively parallel computation capabilities of GPUs.

- Exploration of the interpretability of classification models, focusing on the comprehensibility, complexity, accuracy and human understanding of rule-based classifiers. Development of classification models capable of handling the conflicting objectives of accuracy and interpretability of the classifiers.

- Development of novel high performance classification models to strive for their application to complex data, open problems and new challenges in the classification task, such as the imbalanced data classification and the presence of noise in the data input.

3

Methodology

This chapter summarizes the methods and tools used for the development of the algorithms proposed in this dissertation. Detailed information about the methodology employed in each of the experimental studies is provided in their respective article's documentation.

Datasets

The datasets used in the experimentation of the algorithms were collected from the UCI machine learning repository [52] and the KEEL data sets repository [53]. The datasets represented a wide variety of data problem complexities, with significantly different number of instances, attributes, and classes. This variety allows us for evaluating the performance and soundness of the algorithms under types of data problems.

Concretely, it is possible to find datasets already formatted in ARFF and KEEL formats for classification on particular domains such as multi-label, multi-instance and imbalanced data classification.

Software

The parallel computation was performed with the NVIDIA CUDA Toolkit [21], which allows for programming GPUs for general purpose computation, using a C-style encoding scheme. Java was employed when developing algorithms and genetic operators under the JCLEC [54] software environment, an open-source software for evolutionary computation. Java was also employed for encoding algorithms within the well-known WEKA software tool [55].

Hardware

The experiments were run on a machine equipped with an Intel Core i7 quad-core processor running at 3.0 GHz and 12 GB of DDR3-1600 host memory. The GPU video cards used were two dual-GPU NVIDIA GTX 690 equipped with 4 GB of GDDR5 video RAM. Each GTX 690 video card had two GPUs with 1,536 CUDA cores. In total there were 4 GPUs and 6,144 CUDA cores at default clock speeds. Older hardware was also employed with two NVIDIA GeForce 480 GTX video cards equipped with 1.5GB of GDDR5 video RAM, 15 multiprocessors and 480 CUDA cores clocked at 1.4 GHz. The host operating system was GNU/Linux Ubuntu 64 bits along with NVIDIA CUDA runtime.

Performance evaluation

The evaluation framework we employed in the experimentation of the algorithms followed the 10-fold cross-validation procedure [56, 57] (5-fold cross-validation for imbalanced data). Stochastic algorithms such as seed-based evolutionary methods were also run at least 10 times with different seeds. The statistical analysis of the results was carried out by means of the Bonferroni–Dunn [58] and Wilcoxon rank-sum [59] non-parametric statistical tests, in order to validate multiple and pairwise comparisons among the algorithms [60, 61].

4

Results

This chapter summarizes the different proposals developed in this dissertation and presents a joint discussion of the results achieved in regard to the objectives aimed in the thesis.

4.1 Performance and scalability of classification algorithms

The performance of evolutionary-based classification algorithms decreases as the size of the data and the population size increase. Therefore, it is essential to propose parallelization strategies which allow to scale algorithms to larger data sets and more complex problems.

In [62, 63] we proposed a parallel evaluation model for evolutionary rule learning algorithms based on the parallelization of the fitness computation on GPUs. The proposed model parallelized the evaluation of the individuals of the algorithm's population, which is the phase that requires the most computational time in the evolutionary process of EAs. Specifically, the efficient and scalable evaluator model designed uses GPUs to speed up the performance, receiving rule-based classifiers and returning the confusion matrix of the classifiers on a database. To show the generality of the model proposed, it was applied to some of the most popular

GP classification algorithms [64–66] and several datasets with distinct complexity. Thus, among the datasets used in experiments, there were some widely used as benchmark datasets on the classification task characterized by its simplicity and with others that had not been commonly addressed to date because of their extremely high complexity when applied to previous models. The use of these datasets of varied complexity allowed us to demonstrate the high performance of the GPU proposal on any problem complexity and domain. Experimental results showed the efficiency and generality of our model, which can deal with a variety of algorithms and application domains, providing a great speedup of the algorithm's performance of up to 820 times as compared with the non-parallel version executed sequentially, and up to 196 as compared with the CPU parallel implementation. Moreover, the speedup obtained was higher when the data problem complexity increased. The proposal was compared with other different GPU computing evolutionary learning system called BioHEL [67]. The comparison results showed the efficiency far better obtained by our proposal.

In [29, 68] we presented an efficient Pittsburgh individuals evaluation model on GPUs which parallelized the fitness computation for both rules and rules sets, applicable to any individual = set of rules evolutionary algorithm. The GPU model was scalable to multiple GPU devices, which allowed to address larger data sets and population sizes. The rules interpreter, which checks the coverage of the rules over the instances, was carefully designed to maximize its efficiency compared to traditional rules stack-based interpreters. Experimental results demonstrated the great performance and high efficiency of the proposed model, achieving a rules interpreter performance of up to 64 billion operations per second. On the other hand, the individual evaluation performance achieved a speedup of up to 3.461× when compared to the single-threaded CPU implementation, and a speedup of 1.311× versus the parallel CPU version using 12 threads.

In [69] we presented a GPU parallel implementation of the G3P-MI [70] algorithm, which allowed the acceleration of the evaluation of multiple instance rules learning [71, 72]. G3P-MI is an evolutionary algorithm based on classification rules that has proven to be a suitable model because of its high flexibility, rapid adaptation, excellent quality of knowledge representation, and competitive data classification results. However, its performance becomes slow when learning form large-scale and

high-dimensional data sets. The proposal aimed to be a general purpose model for evaluating multi-instance classification rules on GPUs, which was independent to algorithm behavior and applicable to any of the multi-instance hypotheses [72]. The proposal addressed the computational time problem of evolutionary rule-based algorithms for the evaluation of the rules on multi-instance data sets, especially when the number of rules was high, or when the dimensionality and complexity of the data increased. The design of the model comprised three different GPU kernels which implement the functionality to evaluate the classification rules over the examples in the data set. The interpreter of the rules was carefully designed to maximize efficiency and performance. The GPU model was distributable to multiple GPU devices, providing transparent scalability to multiple GPUs. Moreover, it was designed to achieve good scalability across large-scale and high-dimensional data sets. The proposal was evaluated over a series of real-world and artificial multi-instance data sets and its execution times were compared with the multi-threaded CPU ones, in order to analyze its efficiency and scalability to larger data sets. Experimental results showed the great performance and efficiency of the model, achieving an speedup of up to 450× when compared to the multi-threaded CPU implementation. The efficient rules interpreter demonstrated the ability to run up to 108 billion Genetic Programming operations per second (GPops/s) whereas the multi-threaded CPU interpreter run up to 98 million GPops/s. Moreover, it showed great scalability to two and four GPUs. This means that more complex multi-instance problems with larger number of examples can be addressed.

The publications associated to this part of the dissertation are:

A. Cano, A. Zafra, and S. Ventura. ***Speeding up the evaluation phase of GP classification algorithms on GPUs***. Soft Computing, 16 (2), pages 187-202, 2012.

A. Cano, A. Zafra, and S. Ventura. ***Parallel evaluation of Pittsburgh rule-based classifiers on GPUs***. Neurocomputing, vol. 126, pages 45-57, 2014.

A. Cano, A. Zafra, and S. Ventura. ***Speeding up Multiple Instance Learning Classification Rules on GPUs***. Knowledge and Information Systems, submitted, 2013.

A. Cano, A. Zafra, and S. Ventura. ***A parallel genetic programming algorithm for classification***. In Proceedings of the 6th International Conference on Hybrid Artificial Intelligent Systems (HAIS). Lecture Notes in Computer Science, 6678 LNAI(PART 1):172-181, 2011.

A. Cano, A. Zafra, and S. Ventura. ***Solving classification problems using genetic programming algorithms on GPUs***. In Proceedings of the 5th International Conference on Hybrid Artificial Intelligent Systems (HAIS). Lecture Notes in Computer Science, 6077 LNAI(PART 2):17-26, 2010.

4.2 Interpretability vs accuracy of classification models

Accuracy and interpretability are conflicting objectives in building classification models, and usually it is necessary to achieve a trade-off between the accuracy and comprehensibility of the classifiers. In [73, 74] we proposed a classification algorithm focusing on the interpretability, trying to reach more comprehensible models than most of the current proposals and thus covering the needs of many application domains that require greater comprehensibility than the provided by other available classifiers. This model was based on Evolutionary Programming, and called ICRM (Interpretable Classification Rule Mining). It was designed to obtain a base of rules with the minimum number of rules and conditions, in order to maximize its interpretability, while obtaining competitive accuracy results. The algorithm used an individual = rule representation, following the Iterative Rule Learning (IRL) model. Individuals were constructed by means of a context-free grammar [75, 76], which established a formal definition of the syntactical restrictions of the problem to be solved and its possible solutions, so that only grammatically correct individuals were generated.

The main characteristics of the proposal were the following. Firstly, the algorithm guaranteed obtaining the minimum number of rules. This was possible because it generated one rule per class, together with a default class prediction, which was assigned when none of the available rules were triggered. Moreover, it was guaranteed that there were no contradictory or redundant rules, i.e., there was no pair of rules with the same antecedents and different consequents. Finally, it also guaranteed the minimum number of conditions forming the antecedents of these rules, which was achieved by selecting only the most relevant and discriminating attributes that separate the classes in the attribute domains.

The experiments carried out on 35 different data sets and 9 other high-performance rule-based classification algorithms showed the competitive performance of the ICRM algorithm in terms of predictive accuracy and execution time, obtaining significantly better results than the other methods in terms of the interpretability measures considered in the experimental study: the minimum number of rules, minimum number of conditions per rule, and minimum number of conditions of the classifier. The experimental study included a statistical analysis based on the Bonferroni–Dunn [58] and Wilcoxon [59] non-parametric tests [60, 61] in order to evaluate whether there were statistically differences in the results of the algorithms. The ICRM algorithm demonstrated to obtain more comprehensible rule-based classifiers than other known algorithms such as C4.5 [77] and MPLCS [78].

The ICRM algorithm was also applied on the educational data mining domain using real data from high school students from Zacatecas, Mexico [79]. Predicting student failure at school has become a difficult challenge due to both the high number of factors that can affect the low performance of students and the imbalanced nature of these types of datasets. Firstly, we selected the best attributes in order to resolve the problem of high dimensionality. Then, rebalancing of data and cost sensitive classification were applied in order to resolve the problem of classifying imbalanced data. We compared the performance of the ICRM algorithm versus different white box techniques in order to obtain both more comprehensible and accurate classification rules.

The publications associated to this part of the dissertation are:

A. Cano, A. Zafra, and S. Ventura. ***An Interpretable Classification Rule Mining Algorithm***. Information Sciences, vol. 240, pages 1-20, 2013.

C. Márquez-Vera, A. Cano, C. Romero, and S. Ventura. ***Predicting student failure at school using genetic programming and different data mining approaches with high dimensional and imbalanced data***. Applied Intelligence, 38 (3), pages 315-330, 2013.

A. Cano, A. Zafra, and S. Ventura. ***An EP algorithm for learning highly interpretable classifiers***. In Proceedings of the 11th International Conference on Intelligent Systems Design and Applications, ISDA'11, pages 325-330, 2011.

4.3 Improving performance on challenging data

Challenging and complex data such as imbalanced and noisy data classification call for new algorithms capable of handling complex data appropriately. In [80] we presented a data gravitation classification algorithm, named DGC+, that uses the gravitation concept to induct data classification. The algorithm compared the gravitational field for the different data classes to predict the data class with the highest magnitude. The proposal improved previous data gravitation algorithms by learning the importance of attributes in classification by means of optimizing weights of the attributes for each class. The proposal solved some of the known issues of previous methods such as nominal attributes handling, imbalanced data classification performance, and noisy data filtering. The weights of the attributes in the classification of each class were learned by means of the covariance matrix adaptation evolution strategy (CMA-ES) [81] algorithm, which is a well-known, robust, and scalable global stochastic optimizer for difficult nonlinear and nonconvex continuous domain objective functions [82]. The proposal improved accuracy results by considering both global and local data information, especially in decision boundaries.

The experiments were carried out on 35 standard and 44 imbalanced data sets collected from the KEEL [53] and UCI [52] repositories. A total of 14 algorithms were compared for evaluating standard and imbalanced classification performance. These algorithms belong to the KEEL [83] and WEKA [55] software tools, and they comprise neural networks, support vector machines, rule-based, and instance-based classifiers. The experiments considered different problem domains, with a wide variety on the number of instances, attributes, and classes. The results showed the competitive performance of the proposal, obtaining significantly better results in terms of predictive accuracy, Cohen's kappa rate [84, 85], and area under the curve (AUC) [86, 87]. DGC+ performance differences with other algorithms were more significant as compared with classical instance-based nearest neighbor classifiers, especially they are noteworthy on imbalanced datasets where bias of algorithms to majority class instances is notable. The experimental study was also completed with a statistical analysis based on the Bonferroni–Dunn [58] and Wilcoxon [59] nonparametric tests [60, 88] in order to evaluate whether there were significant differences in the results of the algorithms.

The publication associated to this part of the dissertation is:

A. Cano, A. Zafra, and S. Ventura. ***Weighted Data Gravitation Classification for Standard and Imbalanced Data***.
IEEE Transactions on Cybernetics, 43 (6), pages 1672-1687, 2013.

5

Conclusions and future work

This chapter summarizes the concluding remarks obtained from the research of the dissertation and provides research lines for future work.

5.1 Conclusions

In this Ph.D. thesis we have explored the domain of the classification task on data mining and machine learning, and analyzed the open problems and challenges of data and algorithms. We have reviewed the application of evolutionary algorithms to solve this task and identified the open issues for the new data of the present decade. Specifically, we pursued for several objectives, namely the performance, scalability, interpretability and accuracy of classification algorithms.

Performance and scalability of classification algorithms

First, we analyzed performance and scalability of evolutionary-based classification algorithms. The classification of large datasets using EAs is a high time consuming computation as the problem complexity increases. In [63] we proposed a GPU-based evaluation model to speed up the evaluation phase of GP classification algorithms.

The parallel execution model proposed along with the computational requirements of the evaluation of individuals of the algorithm's population, created an ideal execution environment where GPUs demonstrated to be powerful. Experimental results showed that our GPU-based contribution greatly reduced the execution time using a massively parallel model that takes advantage of fine-grained and coarse-grained parallelization to achieve a good scalability as the number of GPUs increases. Specifically, its performance was better in high dimensional problems and databases with a large number of patterns where our proposal achieved a speedup of up to 820× as compared with the non-parallel version.

In [68] we presented a high-performance and efficient evaluation model on GPUs for Pittsburgh genetic rule-based algorithms. The rule interpreter and the GPU kernels were designed to maximize the GPU occupancy and throughput, reducing the evaluation time of the rules and rule sets. The experimental study analyzed the performance and scalability of the model over a series of varied data sets. It was concluded that the GPU-based implementation was highly efficient and scalable to multiple GPU devices. The best performance was achieved when the number of instances or the population size was large enough to fill the GPU multiprocessors. The speedup of the model was up to 3.461× when addressing large classification problems with two GPUs, significantly higher than the speedup achieved by the CPU parallel 12-threaded solution. The rule interpreter obtained a performance above 64 billion GPops/s and even the efficiency per Watt was up to 129 million GPops/s/W.

Interpretability vs accuracy of classification models

Second, we addressed the conflicting problem of interpretability vs accuracy of data classification models. In [69] we proposed an efficient algorithm for interpretable classification rule mining (ICRM), which is a rule-based evolutionary programming classification algorithm. The algorithm solved the cooperation–competition problem by dealing with the interaction among the rules during the evolutionary process. The proposal minimized the number of rules, the number of conditions per rule, and the number of conditions of the classifier, increasing the interpretability of the solutions. The algorithm did not require to configure or optimize its parameters; it was self-adapted to the data problem complexity. The experiments

performed compared the performance of the algorithm with several other machine learning classification methods, including crisp and fuzzy rules, decision trees, and an ant colony algorithm. Experimental results showed the competitive performance of our proposal in terms of predictive accuracy, obtaining significantly better results. ICRM obtained the best results in terms of interpretability, i.e, it minimized the number of rules, the number of conditions per rule, and the number of conditions of the classifier. Experimental results showed the good performance of the algorithm, but it would be honest to note its limitations. The algorithm was capable of finding comprehensible classifiers with low number of rules and conditions, while achieving competitive accuracy. However, the comprehensibility was prioritized between these conflicting objectives. The one rule per class design allows to obtain very interpretable solutions and it is useful to extract fast "big pictures" of the data. Nevertheless, the accuracy on very complex data might be lower than the obtained by other algorithms but with much more complex classifiers. There was no classifier which achieved best accuracy and comprehensibility for all data. Thus, this algorithm focused on the interpretability of the classifier, which is very useful for knowledge discovery and decision support systems.

Improving performance on challenging data

Third, we challenged to improve classification performance on complex data such as imbalanced data sets. In [80] we presented a data gravitation classification algorithm called DGC+. The proposal included attribute–class weight learning for distance weighting to improve classification results, especially on imbalanced data and to overcome the presence of noisy data. The weights were optimized by means of the CMA-ES algorithm, which showed to perform an effective learning rate of optimal weights for the different attributes and classes, ignoring noisy attributes and enhancing relevant ones. The effects of gravitation around the instances allowed an accurate classification considering both local and global data information, providing smooth classification and good generalization. Gravitation was successfully adapted to deal with imbalanced data problems. The gravitation model achieved better classification accuracy, Cohen's kappa rate and AUC results than other well-known instance-based methods, especially on imbalanced data.

5.2 Future work

In this section we provide some remarks for futures lines of research that arise from the developed research in this dissertation.

First, there are many other applications of the GPU methodology to classification and data mining problems. The latest developed method about feature extraction and data visualization opens a new field of research in which the use of GPUs for parallel computation is crucial. Moreover, we intend to extend the application of the GPU parallelization to other artificial intelligence tasks such as video tracking through evolutionary algorithms. This problem seems to be very appropriate for GPU computing since it involves high computational resources and implies very fast response, currently not available for real-time systems.

Second, we seek the scalability of algorithms to big data. Data processing in a fast and efficient way is an important functionality in machine learning, especially with the growing interest in data storage that has made the data size to be exponentially increased. Thereby, it is necessary to design new efficient distributed data structures to support the manage of massive amount of data. We will develop a Map-Reduce model on big data, which can bring to machine learning the ability to handle efficiently and compute faster enormous datasets.

Third, we will extend the developed models to the multi-instance and multi-label classification problems, which are now hot topics in classification. In multi-instance multi-label classification, examples are described by multiple instances and associated with multiple class labels. Therefore, it is not much complex to adapt the developed methods in this dissertation to the multi-instance and multi-label classification problems.

Bibliography

[1] U. Fayyad, G. Piatetsky-shapiro, and P. Smyth, "From data mining to knowledge discovery in databases," *AI Magazine*, vol. 17, pp. 37–54, 1996.

[2] A. K. Jain, R. P. Duin, and J. Mao, "Statistical pattern recognition: A review," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 1, pp. 4–37, 2000.

[3] D. T. Larose, *Discovering Knowledge in Data: An Introduction to Data Mining*. Wiley, 2005.

[4] A. Ghosh and L. Jain, Eds., *Evolutionary Computation in Data Mining*, ser. Studies in Fuzziness and Soft Computing. Springer, 2005, vol. 163.

[5] A. Abraham, E. Corchado, and J. M. Corchado, "Hybrid learning machines," *Neurocomputing*, vol. 72, no. 13-15, pp. 2729–2730, 2009.

[6] E. Corchado, M. Graña, and M. Wozniak, "New trends and applications on hybrid artificial intelligence systems," *Neurocomputing*, vol. 75, no. 1, pp. 61–63, 2012.

[7] E. Corchado, A. Abraham, and A. de Carvalho, "Hybrid intelligent algorithms and applications," *Information Sciences*, vol. 180, no. 14, pp. 2633–2634, 2010.

[8] W. Pedrycz and R. A. Aliev, "Logic-oriented neural networks for fuzzy neurocomputing," *Neurocomputing*, vol. 73, no. 1-3, pp. 10–23, 2009.

[9] A. A. Freitas, *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2002.

[10] X. Yu and M. Gen, *Introduction to Evolutionary Algorithms*. Springer, 2010.

[11] P. Espejo, S. Ventura, and F. Herrera, "A Survey on the Application of Genetic Programming to Classification," *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, vol. 40, no. 2, pp. 121–144, 2010.

[12] J. Landry, L. D. Kosta, and T. Bernier, "Discriminant feature selection by genetic programming: Towards a domain independent multi-class object detection system," *Journal of Systemics, Cybernetics and Informatics*, vol. 3, no. 1, 2006.

[13] I. De Falco, A. Della Cioppa, F. Fontanella, and E. Tarantino, "An Innovative Approach to Genetic Programming-based Clustering," in *Proceedings of the 9th Online World Conference on Soft Computing in Industrial Applications*, 2004.

[14] E. Alba and M. Tomassini, "Parallelism and evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 5, pp. 443–462, 2002.

[15] G. Luque and E. Alba, *Parallel Genetic Algorithms: Theory and Real World Applications*, ser. Studies in Computational Intelligence. Springer, 2011.

[16] P. E. Srokosz and C. Tran, "A distributed implementation of parallel genetic algorithm for slope stability evaluation," *Computer Assisted Mechanics and Engineering Sciences*, vol. 17, no. 1, pp. 13–26, 2010.

[17] M. Rodríguez, D. M. Escalante, and A. Peregrín, "Efficient Distributed Genetic Algorithm for Rule extraction," *Applied Soft Computing*, vol. 11, no. 1, pp. 733–743, 2011.

[18] S. Dehuri, A. Ghosh, and R. Mall, "Parallel multi-objective genetic algorithm for classification rule mining," *IETE Journal of Research*, vol. 53, no. 5, pp. 475–483, 2007.

[19] A. Folling, C. Grimme, J. Lepping, and A. Papaspyrou, "Connecting Community-Grids by supporting job negotiation with coevolutionary Fuzzy-Systems," *Soft Computing*, vol. 15, no. 12, pp. 2375–2387, 2011.

[20] P. Switalski and F. Seredynski, "An efficient evolutionary scheduling algorithm for parallel job model in grid environment," *Lecture Notes in Computer Science*, vol. 6873, pp. 347–357, 2011.

[21] NVIDIA Corporation, "NVIDIA CUDA Programming and Best Practices Guide, http://www.nvidia.com/cuda," 2013.

[22] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. E. Lefohn, and T. J. Purcell, "A survey of general-purpose computation on graphics hardware," *Computer Graphics Forum*, vol. 26, no. 1, pp. 80–113, 2007.

[23] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU Computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, 2008.

[24] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, and K. Skadron, "A performance study of general-purpose applications on graphics processors using CUDA," *Journal of Parallel and Distributed Computing*, vol. 68, no. 10, pp. 1370–1380, 2008.

[25] D. M. Chitty, "Fast parallel genetic programming: Multi-core CPU versus many-core GPU," *Soft Computing*, vol. 16, no. 10, pp. 1795–1814, 2012.

[26] S. N. Omkar and R. Karanth, "Rule extraction for classification of acoustic emission signals using Ant Colony Optimisation," *Engineering Applications of Artificial Intelligence*, vol. 21, no. 8, pp. 1381–1388, 2008.

[27] K. L. Fok, T. T. Wong, and M. L. Wong, "Evolutionary computing on consumer graphics hardware," *IEEE Intelligent Systems*, vol. 22, no. 2, pp. 69–78, 2007.

[28] L. Jian, C. Wang, Y. Liu, S. Liang, W. Yi, and Y. Shi, "Parallel data mining techniques on Graphics Processing Unit with Compute Unified Device Architecture (CUDA)," *The Journal of Supercomputing*, pp. 1–26, 2011.

[29] A. Cano, A. Zafra, and S. Ventura, "A parallel genetic programming algorithm for classification," in *Proceedings of the 6th International Conference on Hybrid Artificial Intelligent Systems (HAIS). Lecture Notes in Computer Science*, vol. 6678 LNAI, no. PART 1, 2011, pp. 172–181.

[30] M. Paliwal and U. Kumar, "Neural Networks and Statistical Techniques: A Review of Applications," *Expert Systems with Applications*, vol. 36, no. 1, pp. 2–17, 2009.

[31] C. Campbell, "Kernel methods: A survey of current techniques," *Neurocomputing*, vol. 48, pp. 63–84, 2002.

[32] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-based learning algorithms," *Machine Learning*, vol. 6, no. 1, pp. 37–66, 1991.

[33] S. Tsumoto, "Mining diagnostic rules from clinical databases using rough sets and medical diagnostic model," *Information Sciences*, vol. 162, no. 2, pp. 65–80, 2004.

[34] D. Martens, B. Baesens, T. V. Gestel, and J. Vanthienen, "Comprehensible Credit Scoring Models using Rule Extraction from Support Vector Machines," *European Journal of Operational Research*, vol. 183, no. 3, pp. 1466–1476, 2007.

[35] S. Alonso, E. Herrera-Viedma, F. Chiclana, and F. Herrera, "A web based consensus support system for group decision making problems and incomplete preferences," *Information Sciences*, vol. 180, no. 23, pp. 4477–4495, 2010.

[36] N. Xie and Y. Liu, "Review of Decision Trees," in *Proceedings of the 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT)*, vol. 5, 2010, pp. 105–109.

[37] D. Richards, "Two Decades of Ripple Down Rules Research," *Knowledge Engineering Review*, vol. 24, no. 2, pp. 159–184, 2009.

[38] J. Cano, F. Herrera, and M. Lozano, "Evolutionary Stratified Training Set Selection for Extracting Classification Rules with trade off Precision-Interpretability," *Data and Knowledge Engineering*, vol. 60, no. 1, pp. 90–108, 2007.

[39] S. García, A. Fernández, J. Luengo, and F. Herrera, "A Study of Statistical Techniques and Performance Measures for Genetics-based Machine Learning: Accuracy and Interpretability," *Soft Computing*, vol. 13, no. 10, pp. 959–977, 2009.

[40] J. Huysmans, K. Dejaeger, C. Mues, J. Vanthienen, and B. Baesens, "An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models," *Decision Support Systems*, vol. 51, pp. 141–154, 2011.

[41] W. Verbeke, D. Martens, C. Mues, and B. Baesens, "Building comprehensible customer churn prediction models with advanced rule induction techniques," *Expert Systems with Applications*, vol. 38, no. 3, pp. 2354–2364, 2011.

[42] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[43] H. Kaizhu, Y. Haiqin, K. Irwinng, and M. R. Lyu, "Imbalanced learning with a biased minimax probability machine," *IEEE Transactions on Systems and Man and and Cybernetics and Part B: Cybernetics*, vol. 36, no. 4, pp. 913–923, 2006.

[44] R. Akbani, S. Kwek, and N. Japkowicz, "Applying support vector machines to imbalanced datasets," in *Proceedings of 15th European Conference on Machine Learning*, 2004, pp. 39–50.

[45] S.-H. Wu, K.-P. Lin, H.-H. Chien, C.-M. Chen, and M.-S. Chen, "On generalizable low false-positive learning using asymmetric support vector machines," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 5, pp. 1083–1096, 2013.

[46] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[47] I. Kononenko and M. Kukar, *Machine Learning and Data Mining: Introduction to Principles and Algorithms*. Combridge, U.K.: Horwood Publ., 2007.

[48] B. Li, Y. W. Chen, and Y. Q. Chen, "The nearest neighbor algorithm of local probability centers," *IEEE Transactions on Systems and Man and and Cybernetics and Part B: Cybernetics*, vol. 38, no. 1, pp. 141–154, 2008.

[49] L. Peng, B. Peng, Y. Chen, and A. Abraham, "Data gravitation based classification," *Information Sciences*, vol. 179, no. 6, pp. 809–819, 2009.

[50] C. Wang and Y. Q. Chen, "Improving nearest neighbor classification with simulated gravitational collapse," in *Proceedings the International Conference on Computing, Networking and Communications*, vol. 3612, 2005, pp. 845–854.

[51] Y. Zong-Chang, "A vector gravitational force model for classification," *Pattern Analysis and Applications*, vol. 11, no. 2, pp. 169–177, 2008.

[52] D. J. Newman and A. Asuncion, "UCI Machine Learning Repository, University of California, Irvine, School of Information and Computer Sciences," 2007. [Online]. Available: http://archive.ics.uci.edu/ml/

[53] J. Alcalá-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. García, L. Sánchez, and F. Herrera, "KEEL Data-Mining Software Tool: Data Set Repository, Integration of Algorithms and Experimental Analysis Framework," *Journal of Multiple-Valued Logic and Soft Computing*, vol. 17, pp. 255–287, 2011.

[54] S. Ventura, C. Romero, A. Zafra, J. A. Delgado, and C. Hervás, "JCLEC: a Java framework for evolutionary computation," *Soft Computing*, vol. 12, pp. 381–392, 2007.

[55] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemannr, and I. H. Witten, "The WEKA data mining software: An update," *SIGKDD Explorations*, vol. 11, no. 1, pp. 10–18, 2009.

[56] R. Kohavi, "A Study of Cross-validation and Bootstrap for Accuracy Estimation and Model Selection," in *Proceedings of the 14th international joint conference on Artificial intelligence (1995)*, vol. 2, 1995, pp. 1137–1143.

[57] T. Wiens, B. Dale, M. Boyce, and G. Kershaw, "Three way k-fold cross-validation of resource selection functions," *Ecological Modelling*, vol. 212, no. 3-4, pp. 244–255, 2008.

[58] O. J. Dunn, "Multiple comparisons among means," *American Statistical Association*, vol. 56, no. 293, pp. 52–64, 1961.

[59] F. Wilcoxon, "Individual Comparisons by Ranking Methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[60] S. García, A. Fernández, J. Luengo, and F. Herrera, "Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power," *Information Sciences*, vol. 180, no. 10, pp. 2044–2064, 2010.

[61] S. García, D. Molina, M. Lozano, and F. Herrera, "A Study on the use of Non-parametric Tests for Analyzing the Evolutionary Algorithms'Behaviour: A Case Study," *Journal of Heuristics*, vol. 15, pp. 617–644, 2009.

[62] A. Cano, A. Zafra, and S. Ventura, "Solving classification problems using genetic programming algorithms on GPUs," in *Proceedings of the 5th International Conference on Hybrid Artificial Intelligent Systems (HAIS). Lecture Notes in Computer Science*, vol. 6077 LNAI, no. PART 2, 2010, pp. 17–26.

[63] A. Cano, A. Zafra, and S. Ventura, "Speeding up the evaluation phase of GP classification algorithms on GPUs," *Soft Computing*, vol. 16, no. 2, pp. 187–202, 2012.

[64] I. De Falco, A. Della Cioppa, and E. Tarantino, "Discovering interesting classification rules with genetic programming," *Applied Soft Computing*, vol. 1, no. 4, pp. 257–269, 2001.

[65] C. Bojarczuk, H. Lopes, A. Freitas, and E. Michalkiewicz, "A Constrained-syntax Genetic Programming System for Discovering Classification Rules: Application to Medical Datasets," *Artificial Intelligence in Medicine*, vol. 30, no. 1, pp. 27–48, 2004.

[66] K. C. Tan, A. Tay, T. H. Lee, and C. M. Heng, "Mining multiple comprehensible classification rules using genetic programming," in *Proceedings of the IEEE Congress on Evolutionary Computation*, vol. 2, 2002, pp. 1302–1307.

[67] M. A. Franco, N. Krasnogor, and J. Bacardit, "Speeding up the evaluation of evolutionary learning systems using GPGPUs," in *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, ser. GECCO '10, 2010, pp. 1039–1046.

[68] A. Cano, A. Zafra, and S. Ventura, "Parallel evaluation of pittsburgh rule-based classifiers on gpus," *Neurocomputing*, vol. 126, pp. 45–57, 2014.

[69] A. Cano, A. Zafra, and S. Ventura, "Speeding up Multiple Instance Learning Classification Rules on GPUs," *Knowledge and Information Systems, submitted*, 2013.

[70] A. Zafra and S. Ventura, "G3P-MI: A Genetic Programming Algorithm for Multiple Instance Learning," *Information Sciences*, vol. 180, pp. 4496–4513, 2010.

[71] T. G. Dietterich, R. H. Lathrop, and T. Lozano-Pérez, "Solving the multiple instance problem with axis-parallel rectangles," *Artificial Intelligence*, vol. 89, pp. 31–71, 1997.

[72] J. Foulds and E. Frank, "A review of multi-instance learning assumptions," *Knowledge Engineering Review*, vol. 25, no. 1, pp. 1–25, 2010.

[73] A. Cano, A. Zafra, and S. Ventura, "An EP algorithm for learning highly interpretable classifiers," in *Proceedings of the 11th International Conference on Intelligent Systems Design and Applications, ISDA'11*, 2011, pp. 325–330.

[74] A. Cano, A. Zafra, and S. Ventura, "An interpretable classification rule mining algorithm," *Information Sciences*, vol. 240, pp. 1–20, 2013.

[75] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation, Chapter 4: Context-Free Grammars*. Addison-Wesley, 2006.

[76] M. L. Wong and K. S. Leung, *Data Mining Using Grammar Based Genetic Programming and Applications*. Kluwer Academic Publisher, 2000.

[77] J. Quinlan, *C4.5: Programs for Machine Learning*, 1993.

[78] J. Bacardit and N. Krasnogor, "Performance and Efficiency of Memetic Pittsburgh Learning Classifier Systems," *Evolutionary Computation*, vol. 17, no. 3, pp. 307–342, 2009.

[79] C. Márquez-Vera, A. Cano, C. Romero, and S. Ventura, "Predicting student failure at school using genetic programming and different data mining approaches with high dimensional and imbalanced data," *Applied Intelligence*, vol. 38, no. 3, pp. 315–330, 2013.

[80] A. Cano, A. Zafra, and S. Ventura, "Weighted data gravitation classification for standard and imbalanced data," *IEEE Transactions on Cybernetics*, vol. 43, no. 6, pp. 1672–1687, 2013.

[81] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.

[82] N. Hansen, "The CMA evolution strategy: A comparing review," in *Towards a New Evolutionary Computation. Advances on Estimation of Distribution Algorithms*, J. A. Lozano, P. Larranaga, I. Inza, and E. Bengoetxea, Eds. New York: Springer-Verlag, 2006, pp. 75–102.

[83] J. Alcalá-Fdez, L. Sánchez, S. García, M. del Jesus, S. Ventura, J. Garrell, J. Otero, C. Romero, J. Bacardit, V. Rivas, J. Fernández, and F. Herrera, "KEEL: A Software Tool to Assess Evolutionary Algorithms for Data Mining Problems," *Soft Computing*, vol. 13, pp. 307–318, 2009.

[84] A. Ben-David, "Comparison of classification accuracy using Cohen's weighted kappa," *Expert Systems with Applications*, vol. 34, no. 2, pp. 825–832, 2008.

[85] A. Ben-David, "About the relationship between ROC curves and Cohen's kappa," *Engineering Applications of Artificial Intelligence*, vol. 21, no. 6, pp. 874–882, Sep. 2008.

[86] A. P. Bradley, "The use of the area under the ROC curve in the evaluation of machine learning algorithms," *Pattern Recognition*, vol. 30, no. 7, pp. 1145–1159, Jul. 1997.

[87] J. Huang and C. X. Ling, "Using AUC and accuracy in evaluating learning algorithms," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 3, pp. 299–310, 2005.

[88] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*. London, U.K.: Chapman & Hall/CRC, 2007.

[89] A. Cano, J. M. Luna, A. Zafra, and S. Ventura, "A Classification Module for Genetic Programming Algorithms in JCLEC," *Journal of Machine Learning Research, submitted*, 2013.

[90] A. Cano, J. Olmo, and S. Ventura, "Parallel multi-objective ant programming for classification using gpus," *Journal of Parallel and Distributed Computing*, vol. 73, no. 6, pp. 713–728, 2013.

[91] A. Cano, J. M. Luna, and S. Ventura, "High performance evaluation of evolutionary-mined association rules on gpus," *Journal of Supercomputing*, vol. 66, no. 3, pp. 1438–1461, 2013.

[92] A. Cano, S. Ventura, and K. J. Cios, "Scalable CAIM Discretization on Multiple GPUs Using Concurrent Kernels," *Journal of Supercomputing, submitted*, 2013.

[93] A. Cano, D. T. Nguyen, S. Ventura, and K. J. Cios, "ur-CAIM: Improved CAIM Discretization for Unbalanced and Balanced Data," *IEEE Transactions on Knowledge and Data Engineering, submitted*, 2013.

Part II: Journal Publications

TITLE:

Speeding up the evaluation phase of GP classification algorithms on GPUs

AUTHORS:

A. Cano, A. Zafra, and S. Ventura

Soft Computing - A Fusion of Foundations, Methodologies and Applications, *Volume 16, Issue 2, pp. 187-202, 2012*

RANKING:

Impact factor (JCR 2012): 1.124

Knowledge area:

Computer Science, Artificial Intelligence: 63/114

Computer Science, Interdisciplinary Applications: 63/99

DOI: 10.1007/s00500-011-0713-4

Soft Comput (2012) 16:187–202
DOI 10.1007/s00500-011-0713-4

FOCUS

Speeding up the evaluation phase of GP classification algorithms on GPUs

Alberto Cano · Amelia Zafra · Sebastián Ventura

Published online: 3 May 2011

Abstract The efficiency of evolutionary algorithms has become a studied problem since it is one of the major weaknesses in these algorithms. Specifically, when these algorithms are employed for the classification task, the computational time required by them grows excessively as the problem complexity increases. This paper proposes an efficient scalable and massively parallel evaluation model using the NVIDIA CUDA GPU programming model to speed up the fitness calculation phase and greatly reduce the computational time. Experimental results show that our model significantly reduces the computational time compared to the sequential approach, reaching a speedup of up to 820×. Moreover, the model is able to scale to multiple GPU devices and can be easily extended to any evolutionary algorithm.

Keywords Evolutionary algorithms · Genetic programming · Classification · Parallel computing · GPU

1 Introduction

Evolutionary algorithms (EAs) are search methods inspired by natural evolution to find a reasonable solution for data mining and knowledge discovery (Freitas 2002). Genetic programming (GP) is a specialization of EAs, where each individual represents a computer program. It is a machine learning technique used to optimize a population of computer programs according to a fitness function that determines the program's ability to perform a task. Recently, GP has been applied to different common data mining tasks such as classification (Espejo et al. 2010), feature selection (Landry et al. 2006), and clustering (De Falco et al. 2004). However, they perform slowly with complex and high-dimensional problems. Specifically, in the case of classification, this slowness is due to the fact that the model must be evaluated according to a fitness function and training data. Many studies, using different approaches (Harding 2010; Schmitz et al. 2003), have focused on solving this problem by improving the execution time of these algorithms. Recently, the use of GPUs has increased for solving high-dimensional and parallelizable problems and in fact, there are already EA models that take advantage of this technology (Franco et al. 2010). The main shortcoming of these models is that they do not provide a general purpose model: they are too specific to the problem domain or their efficiency could be significantly improved.

In this paper, we present an efficient and scalable GPU-based parallel evaluation model to speed up the evaluation phase of GP classification algorithms that overcomes the shortcomings of the previous models. In this way, our proposal is presented as a general model applicable to any domain within the classification task regardless of its complexity and whose philosophy is easily adaptable to any other paradigm.

The proposed model parallelizes the evaluation of the individuals, which is the phase that requires the most computational time in the evolutionary process of EAs. Specifically, the efficient and scalable evaluator model designed uses GPUs to speed up the performance, receiving a classifier and returning the confusion matrix of that

A. Cano · A. Zafra · S. Ventura (✉)
Department of Computing and Numerical Analysis,
University of Córdoba, 14071 Córdoba, Spain
e-mail: sventura@uco.es

A. Cano
e-mail: i52caroa@uco.es

A. Zafra
e-mail: azafra@uco.es

classifier on a database. To show the generality of the model proposed, it is applied to some of the most popular GP classification algorithms and several datasets with distinct complexity. Thus, among the datasets used in experiments, there are some widely used as benchmark datasets on the classification task characterized by its simplicity and with others that have not been commonly addressed to date because of their extremely high complexity when applied to previous models. The use of these datasets of varied complexity allows us to demonstrate the performance of our proposal on any problem complexity and domain. Experimental results show the efficiency and generality of our model, which can deal with a variety of algorithms and application domains, providing a great speedup of the algorithm's performance, of up to 820 times, compared to the non-parallel version executed sequentially. Moreover, the speedup obtained is higher when the problem complexity increases. The proposal is compared with other different GPU computing evolutionary learning system called BioHEL (Franco et al. 2010). The comparison results show that the efficiency obtained is far better using our proposal.

The remainder of this paper is organized as follows. Section 2 provides an overview of previous works related to evolutionary classification algorithms and GPU implementations. Section 3 discusses the GP model and analyzes the computational cost associated with its different phases. Section 4 describes the GPU architecture and the CUDA programming model. Section 5 explains our proposal and its advantages as a scalable and efficient evaluation model. Section 6 describes the experimental study. In Sect. 7, the results will be announced and finally the last section presents the final remarks of our investigation and outlines future research work.

2 Related works

GP has been parallelized in multiple ways to take advantage both of different types of parallel hardware and of different features of particular problem domains. Most of the parallel approaches during the last decades deal with the implementation over CPU machine clusters. More recently, works about parallelization have been focusing on using graphics processing units (GPUs) which provide fast parallel hardware for a fraction of the cost of a traditional parallel system. GPUs are devices with multicore architectures and parallel processor units. The GPU consists of a large number of processors and recent devices operate as multiple instruction multiple data (MIMD) architectures. Today, GPUs can be programmed by any user to perform general purpose computation (GPGPU) (General-Purpose Computation on Graphics Hardware 2010). The use of GPUs has been already studied for speeding up algorithms within the framework of evolutionary computation. Concretely, we can cite some studies about the evaluation process in genetic algorithms and GP on GPUs (Harding 2010).

Previous investigations have focused on two evaluation approaches (Banzhaf et al. 1998): population parallel or fitness parallel and both methods can exploit the parallel architecture of the GPU. In the fitness parallel method, all the fitness cases are executed in parallel with only one individual being evaluated at a time. This can be considered an SIMD approach. In the population parallel method, multiple individuals are evaluated simultaneously. These investigations have proved that for smaller datasets or population sizes, the overhead introduced by uploading individuals to evaluate is larger than the increase in computational speed (Chitty et al. 2007). In these cases there is no benefit in executing the evaluation on a GPU. Therefore, the larger the population size or the number of instances are, the better the GPU implementation will perform. Specifically, the performance of the population parallel approaches is influenced by the size of the population and the fitness case parallel approaches are influenced by the number of fitness cases, i.e., the number of training patterns from the dataset.

Next, the more relevant proposals presented to date are discussed. Chitty et al. (2007) describes the technique of general purpose computing using graphics cards and how to extend this technique to GP. The improvement in the performance of GP on single processor architectures is also demonstrated. Harding and Banzhaf (2007) goes on to report on how exactly the evaluation of individuals on GP could be accelerated; both proposals are focused on population parallel.

Robilliard et al. (2009) proposes a parallelization scheme to exploit the performance of the GPU on small training sets. To optimize with a modest-sized training set, instead of sequentially evaluating the GP solutions parallelizing the training cases, the parallel capacity of the GPU is shared by the GP programs and data. Thus, different GP programs are evaluated in parallel, and a cluster of elementary processors are assigned to each of them to treat the training cases in parallel. A similar technique, but using an implementation based on the single program multiple data (SPMD) model, is proposed by Langdon and Harrison (2008). They implement the evaluation process of GP trees for bioinformatics purposes using GPGPUs, achieving a speedup of around 8×. The use of SPMD instead of SIMD affords the opportunity to achieve increased speedups since, for example, one cluster can interpret the *if* branch of a test while another cluster treats the *else* branch

independently. On the other hand, performing the same computation inside a cluster is also possible, but the two branches are processed sequentially in order to respect the SIMD constraint: this is called divergence and is, of course, less efficient. Moreover, Maitre et al. (2009) presented an implementation of a genetic algorithm which performs the evaluation function using a GPU. However, they have a training function instead of a training set, which they run in parallel over different individuals. Classification fitness computation is based on learning from a training set within the GPU device which implies memory occupancy, while other proposals use a mathematical representation function as the fitness function.

Franco et al. (2010) introduce a fitness parallel method for computing fitness in evolutionary learning systems using the GPU. Their proposal achieves speed-ups of up to 52× in certain datasets, performing a reduction function (Hwu 2009) over the results to reduce the memory occupancy. However, this proposal does not scale to multiple devices and its efficiency and its spread to other algorithms or to more complex problems could be improved.

These works are focused on parallellizing the evaluation of multiple individuals or training cases and many of these proposals are limited to small datasets due to memory constraints where exactly GPU are not optimal. By contrast, our proposal is an efficient hybrid population and fitness parallel model, that can be easily adapted to other algorithms, designed to achieve maximum performance solving classification problems using datasets with different dimensions and population sizes.

3 Genetic programming algorithms

This section introduces the benefits and the structure of GP algorithms and describes a GP evolutionary system for discovering classification rules in order to understand the execution process and the time required.

GP, the paradigm on which this paper focuses, is a learning methodology belonging to the family of evolutionary algorithms (Bäck et al. 1997) introduced by Koza (1992). GP is defined as an automated method for creating a working computer program from a high-level formulation of a problem. GP performs automatic program synthesis using Darwinian natural selection and biologically inspired operations such as recombination, mutation, inversion, gene duplication, and gene deletion. It is an automated learning methodology used to optimize a population of computer programs according to a fitness function that determines their ability to perform a certain task. Among successful evolutionary algorithm implementations, GP retains a significant position due to such valuable characteristics as: its flexible variable length solution representation, the fact that a priori knowledge is not needed about the statistical distribution of the data (data distribution free), data in their original form can be used to operate directly on them, unknown relationships that exist among data can be detected and expressed as mathematical expressions, and, finally, the most important discriminative features of a class can be discovered. These characteristics suit these algorithms to be a paradigm of growing interest both for obtaining classification rules (De Falco et al. 2001; Freitas 2002; Tan et al. 2002) and for other tasks related to prediction, such as feature selection (Landry et al. 2006) and the generation of discriminant functions (Espejo et al. 2010).

3.1 GP classification algorithms

In this section we will detail the general structure of GP algorithms before proceeding to the analysis of its computational cost.

Individual representation GP can be employed to construct classifiers using different kinds of representations, e.g., decision trees, classification rules, discriminant functions, and many more. In our case, the individuals in the GP algorithm are classification rules whose expression tree is composed by terminal and non-terminal nodes. A classifier can be expressed as a set of IF-antecedent-THEN-consequent rules, in which the antecedent of the rule consists of a series of conditions to be met by an instance in order to consider that it belongs to the class specified by the consequent.

The rule consequent specifies the class to be predicted for an instance that satisfies all the conditions of the rule antecedent. The terminal set consists of the attribute names and attribute values of the dataset being mined. The function set consists of logical operators (AND, OR, NOT), relational operators ($<, \leq, =, <>, \geq, >$) or interval range operators (IN, OUT). These operators are constrained to certain data type restrictions: categorical, real or boolean. Figure 1 shows an example of the expression tree of an individual.

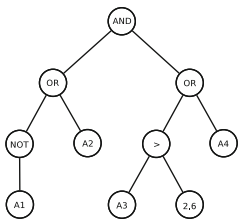


Fig. 1 Example of an individual expression tree

Generational model The evolution process of GP algorithms (Deb 2005), similar to other evolutionary algorithms, consists of the following steps:

1. An initial population of individuals is generated using an initialization criterion.
2. Each individual is evaluated based on the fitness function to obtain a fitness value that represents its ability to solve the problem.
3. In each generation, the algorithm selects a subset of the population to be parents of offspring. The selection criterion usually picks the best individuals to be parents, to ensure the survival of the best genes.
4. This subset of individuals is crossed using different crossover operators, obtaining the offspring.
5. These individuals may be mutated, applying different mutation genetic operators.
6. These new individuals must be evaluated using the fitness function to obtain their fitness values.
7. Different strategies can be employed for the replacement of individuals within the population and the offspring to ensure that the population size in the next generation is constant and the best individuals are kept.
8. The algorithm performs a control stage that determines whether to finish the execution by finding acceptable solutions or by having reached a maximum number of generations, if not the algorithm goes back to step 3 and performs a new iteration (generation).

The pseudo-code of a simple generational algorithm is shown in Algorithm 1.

Algorithm 1 Generational Algorithm

Require: max_generations

```
Initialize (P)
Evaluate (P)
num_generations ← 0
while num_generations < max_generations do
    P ← Parent selection (Population)
    C ← Crossover (P)
    M ← Mutation (C)
    Evaluate (M)
    Population ← Replacement (M ∪ Population)
    num_generations++
end while
```

Evaluation: fitness function The evaluation stage is the evaluation of the fitness function over the individuals. When a rule or individual is used to classify a given training instance from the dataset, one of these four possible values can be obtained: true positive t_p false positive f_p true negative t_n and false negative f_n. The true positive and true negative are correct classifications, while the false positive and false negative are incorrect classifications.

- *True positive* The rule predicts the class and the class of the given instance is indeed that class.
- *False positive* The rule predicts a class but the class of the given instance is not that class.
- *True negative* The rule does not predict the class and the class of the given instance is indeed not that class.
- *False negative* The rule does not predict the class but the class of the given instance is in fact that class.

The results of the individual's evaluations over all the patterns from a dataset are used to build the confusion matrix which allows us to apply different quality indexes to get the individual's fitness value and its calculation is usually the one that requires more computing time. Therefore, our model will also perform this calculation so that each algorithm can apply the most convenient fitness function.

The main problem of the evaluation is the computational time required for the match process because it involves comparing all the rules with all the instances of the dataset. The number of evaluations is huge when the population size or the number of instances increases, thus the algorithm must perform up to millions of evaluations in each generation.

Evaluation: computational study Several previous experiments have been conducted to evaluate the computational time of the different stages of the generational algorithm. These experiments execute the different algorithms described in Sect. 6.1 over the problem domains proposed in Sect. 6.3. The population size was set to 50, 100 and 200 individuals, whereas the number of generations was set to 100 iterations. The results of the average execution time of the different stages of the algorithms among all the configurations are shown in Table 1.

Table 1 GP classification execution time

Phase	Percentage
Initialization	**8.96**
Creation	0.39
Evaluation	8.57
Generation	**91.04**
Selection	0.01
Crossover	0.01
Mutation	0.03
Evaluation	85.32
Replacement	0.03
Control	5.64
Total	**100.00**

The experience using these GP algorithms proves that on average around 94% of the time is taken by the evaluation stage. This percentage is mainly linked to the algorithm, the population size and the number of patterns, increasing up to 99% on large problems. Anyway, the evaluation phase is always more expensive regardless of the algorithm or its parameters. We can conclude that evaluation takes most of the execution time so the most significant improvement would be obtained by accelerating this phase. Therefore, we propose a parallel GPU model detailed in Sect. 5 to speed up the evaluation phase.

4 CUDA programming model

Computer unified device architecture (CUDA) (NVIDIA 2010) is a parallel computing architecture developed by NVIDIA that allows programmers to take advantage of the computing capacity of NVIDIA GPUs in a general purpose manner. The CUDA programming model executes kernels as batches of parallel threads in a SIMD programming style. These kernels comprise thousands to millions of lightweight GPU threads per each kernel invocation.

CUDA's threads are organized into a two-level hierarchy represented in Fig. 2: at the higher one, all the threads in a data-parallel execution phase form a grid. Each call to a kernel execution initiates a grid composed of many thread groupings, called thread blocks. All the blocks in a grid have the same number of threads, with a maximum of 512. The maximum number of thread blocks is 65,535 $\times$ 65,535, so each device can run up to 65,535 $\times$ 65,535 $\times$ 512 $= 2 \times 10^{12}$ threads per kernel call.

To properly identify threads within the grid, each thread in a thread block has a unique ID in the form of a three-dimensional coordinate, and each block in a grid also has a unique two-dimensional coordinate.

Thread blocks are executed in streaming multiprocessors. A stream multiprocessor can perform zero overhead scheduling to interleave warps and hide the overhead of long-latency arithmetic and memory operations.

There are four different main memory spaces: global, constant, shared and local. These GPU memories are specialized and have different access times, lifetimes and output limitations.

- *Global memory* is a large, long-latency memory that exists physically as an off-chip dynamic device memory. Threads can read and write global memory to share data and must write the kernel's output to be readable after the kernel terminates. However, a better way to share data and improve performance is to take advantage of shared memory.
- *Shared memory* is a small, low-latency memory that exists physically as on-chip registers and its contents are only maintained during thread block execution and are discarded when the thread block completes. Kernels that read or write a known range of global memory with spatial or temporal locality can employ shared memory as a software-managed cache. Such caching potentially reduces global memory bandwidth demands and improves overall performance.
- *Local memory* each thread also has its own local memory space as registers, so the number of registers a thread uses determines the number of concurrent threads executed in the multiprocessor, which is called multiprocessor occupancy. To avoid wasting hundreds of cycles while a thread waits for a long-latency global-memory load or store to complete, a common technique is to execute batches of global accesses, one per thread, exploiting the hardware's warp scheduling to overlap the threads' access latencies.
- *Constant memory* is specialized for situations in which many threads will read the same data simultaneously.

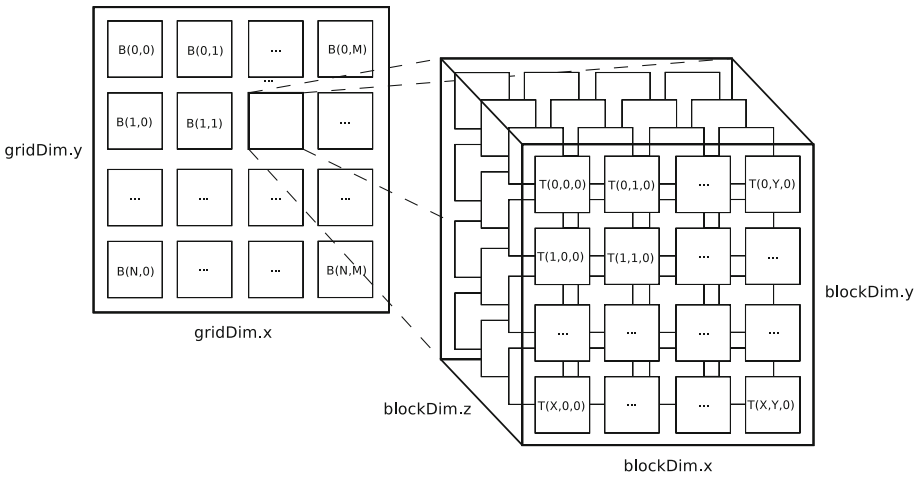


Fig. 2 CUDA threads and blocks model

This type of memory stores data written by the host thread, is accessed constantly and does not change during the execution of the kernel. A value read from the constant cache is broadcast to all threads in a warp, effectively serving 32 loads from memory with a single-cache access. This enables a fast, single-ported cache to feed multiple simultaneous memory accesses. The amount of constant memory is 64 KB.

For maximum performance, these memory accesses must be coalesced as with accesses to global memory. Global memory resides in device memory and is accessed via 32, 64, or 128-byte segment memory transactions. When a warp executes an instruction that accesses global memory, it coalesces the memory accesses of the threads within the warp into one or more of these memory transactions depending on the size of the word accessed by each thread and the distribution of the memory addresses across the threads. In general, the more transactions are necessary, the more unused words are transferred in addition to the words accessed by the threads, reducing the instruction throughput accordingly.

To maximize global memory throughput, it is therefore important to maximize coalescing by following the most optimal access patterns, using data types that meet the size and alignment requirement or padding data in some cases, for example, when accessing a two-dimensional array. For these accesses to be fully coalesced, both the width of the thread block and the width of the array must be multiple of the warp size.

5 Model description

This section details an efficient GPU-based evaluation model for fitness computation. Once it has been proved that the evaluation phase is the one that requires the most of the computational time, this section discusses the procedure of the fitness function to understand its cost in terms of runtime and memory occupancy. We then employ this knowledge to propose an efficient GPU-based evaluation model in order to maximize the performance based on optimization principles (Ryoo et al. 2008) and the recommendations of the NVIDIA CUDA programming model guide (NVIDIA 2010).

5.1 Evaluation complexity

The most computationally expensive phase is evaluation since it involves the match of all the individuals generated over all the patterns. Algorithm 2 shows the pseudo-code of the fitness function. For each individual its genotype must be interpreted or translated into an executable format and then it is evaluated over the training set. The evaluation process of the individuals is usually implemented in two loops, where each individual iterates each pattern and checks if the rule covers that pattern. Considering that the population size is P and the training set size is T the number of iterations is $O(P \times T)$. These two loops make the algorithm really slow when the population size or the pattern count increases because the total number of iterations is the product of these two parameters. This *one by one* iterative model is slow but it only requires $4 \times populationSize \times size\ of(int)$ bytes from memory, i.e., the four integer counters for t_p, t_n, f_p and f_n and values for each individual, this is $O(P)$ complex.

5.2 Efficient GPU-based evaluation

The execution of the fitness function over the individuals is completely independent from one individual to another. Hence, the parallelization of the individuals is feasible. A naive way to do this is to perform the evaluations in parallel using several CPU threads, one per individual. The main problem is that affordable PC systems today only run CPUs with 4 or 8 cores, thus larger populations will need to serialize its execution so the speedup would be limited up to the number of cores that there is. This is where GPGPU systems can exploit its massively parallel model.

Algorithm 2 Evaluation: fitness function

Require: population_size, number_instances

```
1: for each individual within the population do
2:   tp ← 0, fp ← 0, tn ← 0, fn ← 0
3:   for each instance from the dataset do
4:     if individual's rule covers actual instance then
5:       if the consequent matches predicted class then
6:         tp++
7:       else
8:         fp++
9:       end if
10:    else
11:      if the consequent matches predicted class then
12:        fn++
13:      else
14:        tn++
15:      end if
16:    end if
17:  end for
18:  fitnessValue ← computeFitness(tp,tn,fp,fn)
19: end for
```

Using the GPU, the fitness function can be executed over all individuals concurrently. Furthermore, the simple match of a rule over an instance is a self-dependent operation: there is no interference with any other evaluation. Hence, the matches of all the individuals over all the instances can be performed in parallel in the GPU. This means that one thread represents the single match of a pattern over an instance. The total number of GPU threads required would be equal to the number of iterations from the loop of the sequential version. Once each thread has

obtained the result of its match in the GPU device, these results have to be copied back to the host memory and summed up to get the fitness values. This approach would be very slow because in every generation it will be necessary to copy a structure of size $O(P \times T)$, specifically $populationSize \times numberInstances \times sizeof(int)$ bytes from device memory to host memory, i.e, copying the match results obtained from the coverage of every individual over every pattern. This is completely inefficient because the copy transactions time would be larger than the speedup obtained. Therefore, to reduce the copy structure size it is necessary to calculate the final result for each individual inside the GPU and then only copy a structure size $O(P)$ containing the fitness values. Hence, our fitness calculation model involves two steps: matching process and reducing the results for fitness values computation. These two tasks correspond to two different kernels detailed in Sect. 5.2.2.

The source code of our model can be compiled into a shared library to provide the user the functions to perform evaluations in GPU devices for any evolutionary system. The schema of the model is shown in Fig. 3.

At first, the user must call a function to perform the dynamic memory allocation. This function allocates the memory space and loads the dataset instances to the GPU global memory. Moreover, it runs one host thread per GPU device because the thread context is mandatorily associated to only one GPU device. Each host thread runs over one GPU and performs the evaluation of a subset of the individuals from the population. The execution of the host threads stops once the instances are loaded to the GPU awaiting a trigger. The evaluate function call is the trigger that wakes the threads to perform the evaluations over their population's subset. Evaluating the present individuals require the copy of their phenotypes to a GPU memory space. The GPU constant memory is the best location for storing the individual phenotypes because it provides broadcast to all the device threads in a warp. The host threads execute the kernels as batches of parallel threads, first the match kernel obtains the results of the match process and then the reduction kernel calculates the fitness values from these results. The fitness values must be copied back to host memory and associated to the individuals. Once all the individuals from the thread have been evaluated, the host thread sends a signal to the main thread telling it its job has finished and the algorithm process can continue once all the host threads have sent the go-ahead. This stop and go model continues while more generations are performed. At the end, a free memory function must be called to deallocate the dynamic memory previously allocated.

5.2.1 Data structures

The scheme proposed attempts to make full use of global and constant memory. The purpose is to optimize memory usage to achieve maximum memory throughput. Global memory is employed to store the instances from the dataset, the results of the match process, and the fitness values.

The most common dataset structure is a 2D matrix where each row is an instance and each column is an attribute. Loading the dataset to the GPU is simple: allocate a 2D array of width *number Instances* $\times$ *number Attributes* and copy the instances to the GPU. This approach, represented in Fig. 4, is simple-minded and it works but for maximum performance, the memory accesses must be

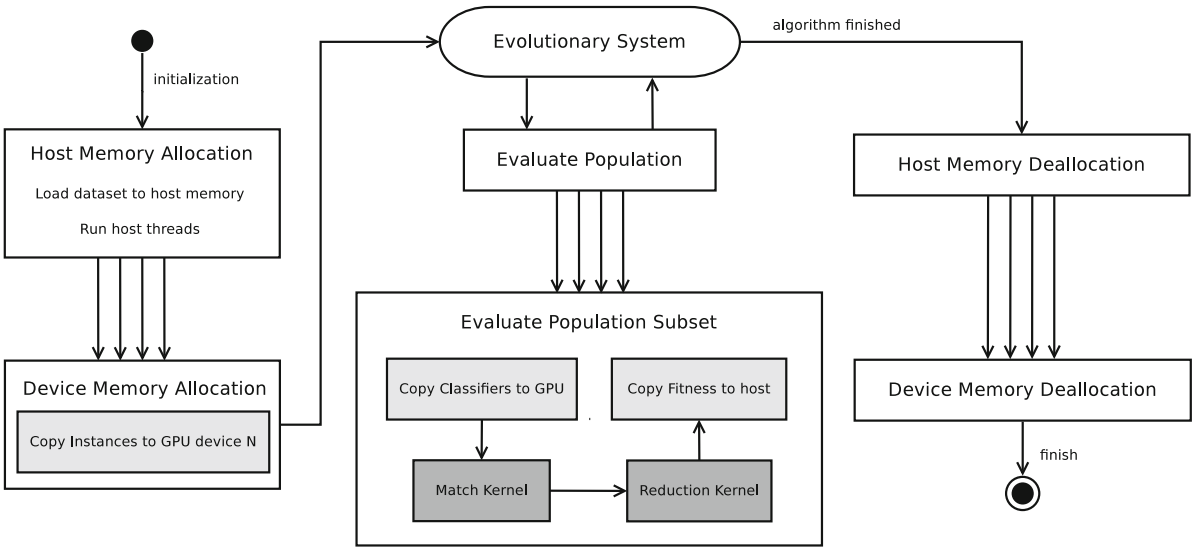


Fig. 3 Model schema

coalesced. When a warp executes an instruction that accesses global memory, it coalesces the memory accesses of the threads within the warp into one or more 32, 64, or 128-byte memory transactions depending on the size of the word accessed by each thread and the distribution of the memory addresses across the threads. In general, the more transactions are necessary, the more unused words are transferred in addition to the words accessed by the threads, reducing the instruction throughput accordingly.

The threads in the match kernel perform the match process of the classifiers over the instances. The *i* thread performs the match process over the *i* instance. Therefore, consecutive threads are executed over consecutive instances. The threads execute the phenotype of the individuals represented in reverse Polish notation using a stack and an interpreter. The phenotype follows the individual representation shown in Sect. 3.1. Thus, when attribute nodes have to obtain the attribute values for each instance covered by the threads within the warp, the attributes' addresses are spaced *numberAttributes* memory addresses (stride is *numberAttributes* × *sizeof* (*datatype*). Depending on the number of attributes, a memory transaction would transfer more or less useful values. Anyway, this memory access pattern shown in Fig. 5 is altogether inefficient because the memory transfer engine must split the memory request into many memory transactions that are issued independently.

The second approach shown in Fig. 6 for storing the instances is derived from the first one. The problem is the stride of the memory requests which is *numberAttributes*. The solution is to lower the stride to one transposing the 2D array that stores the instances. The length of the array remains constant but instead of storing all the attributes of an instance first, it stores the first attributes from all the instances. Now, the memory access pattern shown in Fig. 7 demands attributes which are stored in consecutive memory addresses. Therefore, a single 128-byte memory transaction would transfer the 32 integer or float attributes requested by the threads in the warp.

For these accesses to be fully coalesced, both the width of the thread block and the number of the instances must be a multiple of the warp size. The third approach for achieving fully coalesced accesses is shown in Figs. 8 and 9. Intra-array padding is necessary to align the addresses requested to the memory transfer segment sizes. Thus, the array must be expanded to *multiple*(*numberInstances*,32) × values.

The individuals to be evaluated must be uploaded in each generation to the GPU constant memory. The GPU has a read-only constant cache that is shared by all functional units and speeds up reads from the constant memory space, which resides in device memory. All the threads in a warp perform the match process of the same individual over different instances. Thus, memory requests point to

A(0,0)	A(1,0)	...	A(N,0)	A(0,1)	A(1,1)	...	A(N,1)	...	A(0,I)	...	A(N,I)

Fig. 4 Uncoalesced instances data array structure

Fig. 5 Uncoalesced attributes request

A(0,0)	A(0,1)	...	A(0,I)	A(1,0)	A(1,1)	...	A(1,I)	...	A(N,0)	...	A(N,I)

Fig. 6 Coalesced instances data array structure

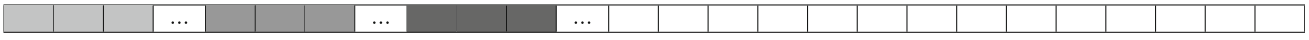


Fig. 7 Coalesced attributes request

A(0,0)	...	A(0,I)	X	A(1,0)	...	A(1,I)	X	...	A(N,0)	...	A(N,I)	X

Fig. 8 Fully coalesced intra-array padding instances data array structure

Fig. 9 Fully coalesced attributes request

R(0,0)	...	R(0,I)	X	R(1,0)	...	R(1,I)	X	...	R(N,0)	...	R(N,I)	X

Fig. 10 Results data array structure

the same node and memory address at a given time. Servicing one memory read request to several threads simultaneously is called broadcast. The resulting requests are serviced at the throughput of the constant cache in case of a cache hit, or at the throughput of device memory otherwise.

The results of the match process for each individual and instance must be stored in global memory for counting. Again, the memory accesses must be coalesced to device global memory. The best data structure is a 2D array *numberInstances* × *populationSize* shown in Fig. 10. Hence, the results write operations and the subsequent read operations for counting are both fully coalesced.

The fitness values calculated by the reduction kernel must be stored in global memory, then copied back to host memory and set to the individuals. A simple structure to store the fitness values of the individuals is a 1D array of length *populationSize*.

5.2.2 Evaluation process on GPU

The evaluation process on the GPU is performed using two kernel functions. The first kernel performs the match operations between the individuals and the instances storing a certain result. Each thread is in charge of a single match. The second kernel counts the results of an individual by a reduction operation. This 2-kernel model allows the user to perform the match processes and the fitness values' calculations completely independently. Once the results of the match process are obtained, any fitness function can be employed to calculate the fitness values. This requires copying back to global memory a large amount of data at the end of the first kernel. Franco et al. (2010) proposes to minimise the volume of data by performing a reduction in the first kernel. However, the experiments carried out indicate to us that the impact on the run-time of reducing data in the first kernel is larger than that of storing the whole data array because our approach allows the kernels to avoid synchronization between threads and unnecessary delays. Furthermore, the threads block dimensions can be ideally configured for each kernel independently.

Match kernel The first kernel performs in parallel the match operations between the classifiers and the instances. Algorithm 3 shows the pseudo-code for this kernel.

The number of matches and hence the total number of threads is *populationSize* × *numberInstances*. The maximum amount of threads per block is 512 or 1,024 depending on the device's computing capability. However, optimal values are multiples of the warp size. A GPU multiprocessor relies on thread-level parallelism to maximize utilization of its functional units. Utilization is therefore directly linked to the number of resident warps. At every instruction issue time, a warp scheduler selects a warp that is ready to execute, if any, and issues the next instruction to the active threads of the warp. The number of clock cycles it takes for a warp to be ready to execute its next instruction is called latency, and full utilization is achieved when the warp scheduler always has some instruction to issue for some warp at every clock cycle during that latency period, i.e., when the latency of each warp is completely hidden by other warps.

Algorithm 3 Match kernel

```
1: instance ← blockDim.y * blockIdx.y + threadIdx.y;
2: if instance < numberInstances then
3:     resultMemPosition ← blockIdx.x * numberInstance-
       sAligned + instance;
4:     if covers(classifier[blockIdx.x],instance) then
5:         if classifiedClass == instancesClass[instance] then
6:             result[resultMemPosition] ← tp
7:         else
8:             result[resultMemPosition] ← fp
9:         end if
10:    else
11:        if classifiedClass != instancesClass[instance] then
12:            result[resultMemPosition] ← tn
13:        else
14:            result[resultMemPosition] ← fn
15:        end if
16:    end if
17: end if
```

The CUDA occupancy calculator spreadsheet allows computing the multiprocessor occupancy of a GPU by a given CUDA kernel. The multiprocessor occupancy is the ratio of active warps to the maximum number of warps supported on a multiprocessor of the GPU. The optimal number of threads per block obtained from the experiments carried out for this kernel is 128 for devices of compute capability 1×, distributed in 4 warps of 32 threads. The active thread blocks per multiprocessor is 8. Thus, the active warps per multiprocessor is 32. This means a 100% occupancy of each multiprocessor for devices of compute capability 1×. Recent devices of compute capability 2× requires 192 threads per block to achieve 48 active warps per multiprocesor and a 100% occupancy. The number of

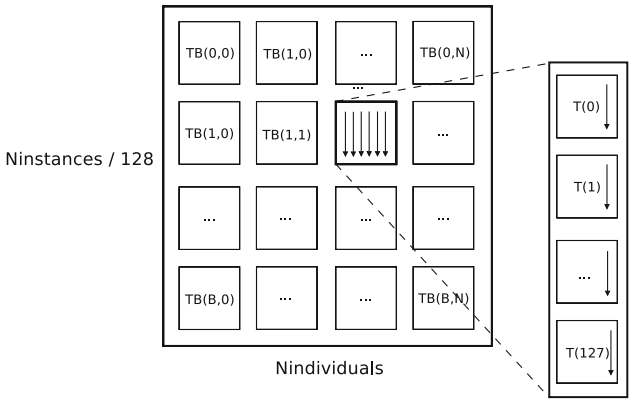


Fig. 11 Match kernel 2D grid of thread blocks

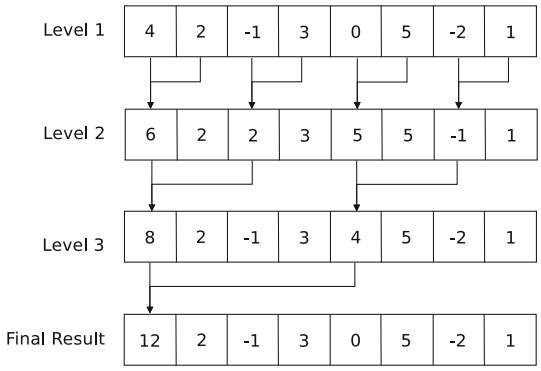


Fig. 12 Parallel reduction algorithm

threads per block does not matter, since the model is adapted to achieve maximum performance in any case.

The kernel is executed using a 2D grid of thread blocks as shown in Fig. 11. The first dimension length is *populationSize*. Using *N* threads per block, the number of thread blocks to cover all the instances is *ceil*(*numberInstances*/*N*) in the second dimension of the grid. Thus, the total number of thread blocks is *populationSize* × *ceil*(*numberInstances*/*N*). This number is important as it concerns the scalability of the model in future devices. NVIDIA recommends that one run at least twice as many thread blocks as the number of multiprocessors.

Reduction kernel The second kernel reduces the results previously calculated in the first kernel and obtains the fitness value for each individual. The naive reduction operation shown in Fig. 12 sums in parallel the values of an array reducing iteratively the information. Our approach does not need to sum the values, but counting the number of t_p, f_p, t_n and f_n resulted for each individual from the match kernel. These four values are employed to build the confusion matrix. The confusion matrix allows us to apply different quality indexes defined by the authors to get the individual's fitness value.

Designing an efficient reduction kernel is not simple because it is the parallelization of a natively sequential task. In fact, NVIDIA propose six different approaches (NVIDIA 2010). Some of the proposals take advantage of the device shared memory. Shared memory provides a small but fast memory shared by all the threads in a block. It is quite desirable when the threads require synchronization and work together to accomplish a task like reduction.

A first approach to count the number of t_p, f_p, t_n and f_n in the results array using *N* threads is immediate. Each thread counts for the *N*th part of the array, specifically *numberInstances/numberThreads* items, and then the values for each thread are summed. This 2-level reduction is not optimal because the best would be the *N*/2-level reduction, but reducing each level requires the synchronization of the threads. Barrier synchronization can impact performance by forcing the multiprocessor to idle. Therefore, a 2 or 3-level reduction has been proved to perform the best.

To achieve a 100% occupancy, the reduction kernel must employ 128 or 192 threads, for devices of compute capability 1× or 2×, respectively. However, it is not trivial to organize the threads to count the items. The first approach involves the thread *i* counting *numberInstances/numberThreads* items from the *thread Idx* × *numberInstances/numberThreads* item. The threads in a thread-warp would request the items spaced *numberInstances* memory addresses. Therefore, once again one has a coalescing and undesirable problem. Solving the memory requests pattern is naive. The threads would count again *numberInstances/number Threads* items but for coalescing purposes the memory access pattern would be *iteration* × *numberThreads* + *thread Idx*. This way, the threads in a warp request consecutive memory addresses that can be serviced in fewer memory transactions. This second approach is shown in Fig. 13. The reduction kernel is executed using a

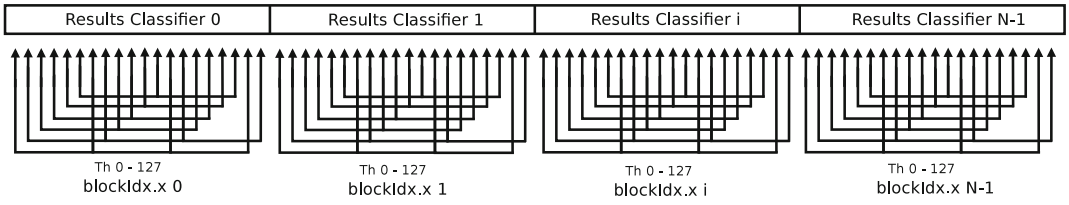


Fig. 13 Coalesced reduction kernel

1D grid of thread blocks whose length is *populationSize*. Using 128 or 192 threads per block, each thread block performs the reduction of the results for an individual. A shared memory array of length 4 × *numberThreads* keeps the temporary counts for all the threads. Once all the items have been counted, a synchronization barrier is called and the threads wait until all the threads in the thread block have reached this point and all global and shared memory accesses made by these threads prior to the synchronization point are visible to all threads in the block. Finally, only one thread per block performs the last sum, calculates the fitness value and writes it to global memory.

6 Experimental study

This section describes the details of the experiments, discusses the application domains, the algorithms used, and the settings of the tests.

6.1 Parallelized methods with our proposal

To show the flexibility and applicability of our model, three different GP classification algorithms proposed in the literature are tested using our proposal in the same way as could be applied to other algorithms or paradigms. Next, the major specifications of each of the proposals that have been considered in the study are detailed.

(1) De Falco et al. (2001) propose a method to get the fitness of the classifier by evaluating the antecedent over all the patterns within the dataset. Falco et al. uses the logical operators AND, OR, NOT, the relational operators $=, <, >, \leq, \geq$ and two numerical interval comparator operators IN and OUT. The evolution process is repeated as many times as classes holds the dataset. In each iteration the algorithm focuses on a class and keeps the best rule obtained to build the classifier.

The crossover operator selects two parent nodes and swaps the two subtrees. The crossover is constrained by two restrictions: the nodes must be compatible and the depth of the tree generated must not exceed a preset depth.

The mutation operator can be applied either to a terminal node or a non-terminal node. This operator selects a node randomly, if it is a terminal node, it is replaced by another randomly selected compatible terminal node, otherwise, the non-terminal node is replaced by another randomly selected compatible terminal node with the same arity and compatibility.

The fitness function calculates the difference between the number of instances where the rule correctly predicts the membership or not of the class and number of examples where the opposite occurs and the prediction is wrong. Finally, the fitness function is defined as:

$$fitness = nI - ((t_p + t_n) - (f_p + f_n)) + \alpha * N$$

where nI is the number of instances, α is a value between 0 and 1 and N is the number of nodes in the rule. The closer is α to 1, the more importance is given to simplicity.

(2) Tan et al. (2002) proposes a modified version of the steady-state algorithm (Banzhaf et al. 1998) which uses an external population and elitism to ensure that some of the best individuals of the current generation survive in the next generation.

The fitness function combines two indicators that are commonplace in the domain, namely the sensitivity (Se) and the specifity (Sp) defined as follows:

$$Se = \frac{t_p}{t_p + w_1 f_n} \quad Sp = \frac{t_n}{t_n + w_2 f_p}$$

The parameters w_1 and w_2 are used to weight the influence of the false negatives and false positives cases in the fitness calculation. This is very important because these values are critical in problems such as diagnosis. Decreasing $w1$ or increasing w_2 generally improves the results but also increases the number of rules. The range [0.2–1] for w_1 y [1–20] for w_2 is usually reasonable for the most cases. Therefore, the fitness function is defined by the product of these two parameters: $fitness = Se \times Sp$. The proposal of Tan et al. is similar to that of Falco et al. but the for OR operator because combinations of AND and NOT operators which can generate all the necessary rules. Therefore, the simplicity of the rules is affected. Tan et al. also introduces the token competition technique proposed by Wong and Leung (2000) and it is employed as an alternative niche approach to promote diversity. Most of the time, only a few rules are useful and cover most of the instances while most others are redundant. The token competition is an effective way to eliminate redundant rules.

(3) Bojarczuk et al. (2004) present a method in which each rule is evaluated for all of the classes simultaneously for a pattern. The classifier is formed by taking the best individual for each class generated during the evolutionary process. The Bojarczuk et al. algorithm does not have a mutation operator.

This proposal uses the logical operators AND, OR and NOT, although AND and NOT would be sufficient; this way the size of the generated rules is reduced. GP does not produce simple solutions. The comprehensibility of a rule is inversely proportional to its size. Therefore Bojarczuk et al. define the simplicity Sy of a rule:

$$Sy = \frac{maxnodes - 0.5 \times numnodes - 0.5}{maxnodes - 1}$$

where *maxnodes* is the maximum depth of the syntaxtree, *numnodes* is the number of nodes of the current rule, and Se and Sp are the sensitivity and the specifity parameters described in the Tan et al. with $w1$ and $w2$ equal to 1. The fitness value is the product of these three parameters.

$$fitness = Se \times Sp \times Sy$$

These three methods implement the match kernel and the reduction kernel. The match kernel obtains the results from the match processes of the prediction of the examples with their actual class. The reduction kernel counts the t_p, t_n, f_p and f_n values and computes the fitness values.

6.2 Comparison with other proposal

One of the most recent works and similar to our proposal is the one by Franco et al. (2010). This work speeds up the evaluation of the BioHEL system using GPGPUs. BioHEL is an evolutionary learning system designed to cope with large-scale datasets. They provide the results, the profiler information, the CUDA and the serial version of the software in the website http://www.cs.nott.ac.uk/~mxf/biohel. The experiments carried out compare our model and its speedup to the speedup obtained from the CUDA version of BioHEL system over several problem domains in order to demonstrate the improvements provided by the parallelization model proposed. The configuration settings of the BioHEL system were the provided by the authors in the configuration files.

6.3 Problem domains used in the experiments

To evaluate the performance of the proposed GP evaluation model, some datasets selected from the UCI machine learning repository (Newman and Asuncion 2007) and the KEEL website (Alcalá-Fdez et al. 2009) are benchmarked using the algorithms previously described. These datasets are very varied considering different degrees of complexity. Thus, the number of instances ranges from the simplest containing 150 instances to the most complex containing one million instances. Also, the number of attributes and classes are different in different datasets. This information is summarized in Table 2. The wide variety of datasets considered allows us to evaluate the model performance in both low and high problem complexity. It is interesting to note that some of these datasets such as KDDcup or Poker have not been commonly addressed to date because they are not memory and CPU manageable by traditional models.

Table 2 Complexity of the datasets tested

Dataset	#Instances	#Attributes	#Classes
Iris	150	4	3
New-thyroid	215	5	3
Ecoli	336	7	8
Contraceptive	1,473	9	3
Thyroid	7,200	21	3
Penbased	10,992	16	10
Shuttle	58,000	9	7
Connect-4	67,557	42	3
KDDcup	494,020	41	23
Poker	1,025,010	10	10

6.4 General experimental settings

The GPU evaluation code is compiled into a shared library and loaded into the JCLEC (Ventura et al. 2007) framework using JNI. JCLEC is a software system for evolutionary computation research developed in the Java programming language. Using the library, our model can be easily employed in any evolutionary learning system.

Experiments were run on two PCs both equipped with an Intel Core i7 quad-core processor running at 2.66GHz and 12 GB of DDR3 host memory. One PC features two NVIDIA GeForce 285 GTX video cards equipped with 2 GB of GDDR3 video RAM and the other one features two NVIDIA GeForce 480 GTX video cards equipped with 1.5 GB of GDDR5 video RAM. No overclock was made to any of the hardware. The operating system was GNU/Linux Ubuntu 10.4 64 bit.

The purpose of the experiments is to analyze the effect of the dataset complexity on the performance of the GPU evaluation model and the scalability of the proposal. Each algorithm is executed over all the datasets using a sequential approach, a threaded CPU approach, and a massively parallel GPU approach.

7 Results

This section discusses the experimental results. The first section compares the performance of our proposal over different algorithms. The second section provides the results of the BioHEL system and compares them with the obtained by our proposal.

7.1 Results obtained using our proposal

In this section we discuss the performance achieved by our proposal using three different GP algorithms. The execution time and the speedups of the three classification algorithms solving the various problems considered are shown in Tables 3, 4 and 5 where each column is labeled with the execution configuration indicated from left to right as follows: the dataset, the execution time of the native sequential version coded in Java expressed in seconds, the speedup of the model proposed using JNI and one CPU thread, two CPU threads, four CPU threads, one GTX 285 GPU, two GTX 285 GPUs, and with one and two GTX 480 GPUs. The results correspond to the executions of the algorithms with a population of 200 individuals and 100 generations.

Table 3 Falco et al. algorithm execution time and speedups

Execution time (s)		Speedup						
Dataset	Java	1 CPU	2 CPU	4 CPU	1 285	2 285	1 480	2 480
Iris	2.0	0.48	0.94	1.49	2.96	4.68	2.91	8.02
New-thyroid	4.0	0.54	1.03	1.99	4.61	9.46	5.18	16.06
Ecoli	13.7	0.49	0.94	1.38	6.36	10.92	9.05	17.56
Contraceptive	26.6	1.29	2.52	3.47	31.43	55.29	50.18	93.64
Thyroid	103.0	0.60	1.15	2.31	37.88	69.66	75.70	155.86
Penbased	1,434.1	1.15	2.26	4.37	111.85	207.99	191.67	391.61
Shuttle	1,889.5	1.03	2.02	3.87	86.01	162.62	182.19	356.17
Connect-4	1,778.5	1.09	2.14	3.87	116.46	223.82	201.57	392.86
KDDcup	154,183.0	0.91	1.77	3.30	136.82	251.71	335.78	653.60
Poker	108,831.6	1.25	2.46	4.69	209.61	401.77	416.30	820.18

Table 4 Bojarczuk et al. algorithm execution time and speedups

Execution time (s)		Speedup						
Dataset	Java	1 CPU	2 CPU	4 CPU	1 285	2 285	1 480	2 480
Iris	0.5	0.50	0.96	1.75	2.03	4.21	2.39	5.84
New-thyroid	0.8	0.51	1.02	1.43	2.99	5.85	3.19	9.45
Ecoli	1.3	0.52	1.02	1.15	3.80	8.05	5.59	11.39
Contraceptive	5.4	1.20	2.40	2.47	14.58	31.81	26.41	53.86
hyroid	27.0	0.56	1.11	2.19	25.93	49.53	56.23	120.50
Penbased	42.6	0.96	1.92	3.81	18.55	36.51	68.01	147.55
Shuttle	222.5	1.18	2.35	4.66	34.08	67.84	117.85	253.13
Connect-4	298.9	0.69	1.35	2.65	42.60	84.92	106.14	214.73
KDDcup	3,325.8	0.79	1.55	2.98	30.89	61.80	135.28	306.22
Poker	6,527.2	1.18	2.32	4.45	39.02	77.87	185.81	399.85

Table 5 Tan et al. algorithm execution time and speedups

Execution time (s)		Speedup						
Dataset	Java	1 CPU	2 CPU	4 CPU	1 285	2 285	1 480	2 480
Iris	2.6	0.44	0.80	1.01	2.94	5.44	4.90	9.73
New-thyroid	6.0	0.77	1.43	1.78	7.13	12.03	9.15	21.74
Ecoli	22.5	0.60	1.16	2.09	9.33	16.26	14.18	25.92
Contraceptive	39.9	1.28	2.44	3.89	40.00	64.52	60.60	126.99
Thyroid	208.5	1.10	2.11	2.66	64.06	103.77	147.74	279.44
Penbased	917.1	1.15	2.23	4.25	86.58	148.24	177.78	343.24
Shuttle	3,558.0	1.09	2.09	3.92	95.18	161.84	222.96	431.80
Connect-4	1,920.6	1.35	2.62	4.91	123.56	213.59	249.81	478.83
KDDcup	185,826.6	0.87	1.69	3.20	83.82	138.53	253.83	493.14
Poker	119,070.4	1.27	2.46	4.66	158.76	268.69	374.66	701.41

The results in the tables provide useful information that in some cases, the external CPU evaluation is inefficient for certain datasets such as Iris, New-thyroid or Ecoli. This is because the time taken to transfer the data from the Java virtual machine memory to the native memory is higher than just doing the evaluation in the Java virtual machine. However, in all the cases, regardless of the size of the dataset, the native GPU evaluation is always considerably faster. If we

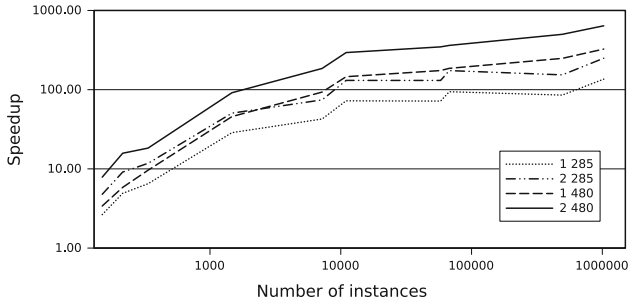


Fig. 14 Average speedups

look at the results of the smallest datasets such as Iris, New-thyroid and Ecoli, it can be seen that its speedup is acceptable and specifically Ecoli performs up to 25× faster. Speeding up these small datasets would not be too useful because of the short run time required, but it is worthwhile for larger data sets. On the other hand, if we focus on complex datasets, the speedup is greater because the model can take full advantage of the GPU multiprocessors' offering them many instances to parallelize. Notice that KDDcup and Poker datasets perform up to 653× and 820× faster, respectively. We can also appreciate that the scalability of the proposal is almost perfect, since doubling the number of threads or the graphics devices almost halves the execution time. Figure 14 summarizes the average speedup, depending on the number of instances.

The fact of obtaining significant enhancements in all problem domains (both small and complex datasets) as has been seen is because our proposal is a hybrid model that takes advantage of both the parallelization of the individuals and the instances. A great speedup is not only achieved by classifying a large number of instances but by a large enough population. The classification of small datasets does not require many individuals but high-dimensional problems usually require a large population to provide diversity in the population genetics. Therefore, a great speedup is achieved by maximizing both parameters. These results allow us to determine that the proposed model achieves a high speedup in the algorithms employed. Specifically, the best speedup is 820× when using the Falco et al. algorithm and the poker dataset; hence, the execution time can be impressively reduced from 30 h to only 2 min.

7.2 Results of the other proposal

This section discusses the results obtained by BioHEL. The results for the BioHEL system are shown in Table 6 where the first column indicates the execution time of the serial version expressed in seconds, the second column shows the speedup of the CUDA version using a NVIDIA GTX 285 GPU, and the third using a NVIDIA GTX 480 GPU. These results show that for a dataset with a low number of instances, the CUDA version of BioHEL performs slower than the serial version. However, the speedup obtained is higher when the number of instances increases, achieving a speedup of up to 34 times compared to the serial version.

Table 6 BioHEL execution time and speedups

Execution time (s)		Speedup	
Dataset	Serial	1 285	1 480
Iris	0.5	0.64	0.66
New-thyroid	0.9	0.93	1.32
Ecoli	3.7	1.14	6.82
Contraceptive	3.3	3.48	3.94
Thyroid	26.4	2.76	8.70
Penbased	147.9	5.22	20.26
Shuttle	418.4	11.54	27.84
Connect-4	340.4	10.18	12.24
KDDcup	503.4	14.95	28.97
Poker	3,290.9	11.93	34.02

The speedup results for the BioHEL system shown in Table 6 compared with the results obtained by our proposal shown in Tables 3, 4 and 5 demonstrate the better performance of our model. One of the best advantages of our proposal is that it scales to multiple GPU devices, whereas BioHEL does not. Both BioHEL and our proposal employ a 2-kernel model. However, we do not to perform a one-level parallel reduction in the match kernel, in order to avoid synchronization between threads and unnecessary delays even if it means storing the whole data array. Thus, the memory requirements are larger but the reduction performs faster as the memory accesses are fully coalesced and synchronized. Moreover, our proposal improves the instruction throughput upto 1.45, i.e., the number of instructions that can be executed in a unit of time. Therefore, our proposal achieves 1 Teraflops performance using two GPUs NVIDIA GTX 480 with 480 cores running at 700 MHz. This information is provided in the CUDA profiler available in the respective websites.

Additional information of the paper such as the details of the kernels, the datasets employed, the experimental results and the CUDA profiler information are published in the website: http://www.uco.es/grupos/kdis/kdiswiki/SOCOGPU.

8 Conclusions and future work

The classification of large datasets using EAs is a time consuming computation as the problem complexity increases. To solve this problem, many studies have aimed at optimizing the computational time of EAs. In recent years, these studies have focused on the use of GPU devices whose main advantage over previous proposals are

their massively parallel MIMD execution model that allows researchers to perform parallel computing where million threads can run concurrently using affordable hardware.

In this paper there has been proposed a GPU evaluation model to speed up the evaluation phase of GP classification algorithms. The parallel execution model proposed along with the computational requirements of the evaluation of individuals, creates an ideal execution environment where GPUs are powerful. Experimental results show that our GPU-based proposal greatly reduces the execution time using a massively parallel model that takes advantage of fine-grained and coarse-grained parallelization to achieve a good scalability as the number of GPUs increases. Specifically, its performance is better in high-dimensional problems and databases with a large number of patterns where our proposal has achieved a speedup of up to 820× compared to the non-parallel version.

The results obtained are very promising. However, more work can be done in this area. Specifically, the development of hybrid models is interesting from the perspective of evolving in parallel a population of individuals. The classical approach of genetic algorithms is not completely parallelizable because of the serialization of the execution path of certain pieces of code. There have been several proposals to overcome these limitations achieving excellent results. The different models used to perform distributed computing and parallelization approaches focus on two approaches (Dorigo and Maniezzo 1993): the islands model, where several isolated subpopulations evolve in parallel and periodically swap their best individuals from neighboring islands, and the neighborhood model that evolves a single population and each individual is placed in a cell of a matrix.

These two models are available for use in MIMD parallel architectures in the case of islands and SIMD models for the neighborhood. Therefore, both perspectives can be combined to develop multiple models of parallel and distributed algorithms (Harding and Banzhaf 2009), which take advantage of the parallel threads in the GPU, the use of multiple GPUs, and the distribution of computation across multiple machines networked with these GPUs.

Acknowledgments This work has been financed in part by the TIN2008-06681-C06-03 project of the Spanish Inter-Ministerial Commission of Science and Technology (CICYT), the P08-TIC-3720 project of the Andalusian Science and Technology Department, and FEDER funds.

References

Alcalá-Fdez J, Sánchez LS, García S, del Jesus M, Ventura S, Garrell J, Otero J, Romero C, Bacardit J, Rivas V, Fernández J, Herrera F (2009) KEEL: KEEL: a software tool to assess evolutionary algorithms for data mining problems. Soft Comput Fusion Found Methodol Appl 13:307–318

Bäck TB, Fogel D, Michalewicz Z (1997) Handbook of evolutionary computation. Oxford University Press

Banzhaf W, Nordin P, Keller RE, Francone FD (1998) Genetic programming—an introduction: on the automatic evolution of computer programs and its applications. Morgan Kaufmann, San Francisco, CA

Bojarczuk CC, Lopes HS, Freitas AA, Michalkiewicz EL (2004) A constrained-syntax genetic programming system for discovering classification rules: application to medical data sets. Artif Intell Med 30(1):27–48

Chitty DM, Malvern Q (2007) A data parallel approach to genetic programming using programmable graphics hardware. In: GECCO G07: Proceedings of the 9th annual conference on genetic and evolutionary computation. ACM Press, pp 1566–1573

De Falco I, Della Cioppa A, Tarantino E (2001) Discovering interesting classification rules with genetic programming. Appl Soft Comput 1(4):257–269

De Falco I, Della Cioppa A, Fontanella F, Tarantino E (2004) An innovative approach to genetic programming-based clustering. In: 9th online world conference on soft computing in industrial applications

Deb K (2005) A population-based algorithm-generator for real-parameter optimization. Soft Comput 9:236–253

Dorigo M, Maniezzo V (1993) Parallel genetic algorithms: introduction and overview of current research. In: Parallel genetic algorithms: theory and applications. IOS Press, Amsterdam, The Netherlands, pp 5–42

Espejo PG, Ventura S, Herrera F (2010) A survey on the application of genetic programming to classification. IEEE Trans Syst Man Cybern Part C (Appl Rev) 40(2):121–144

Franco MA, Krasnogor N, Bacardit J (2010) Speeding up the evaluation of evolutionary learning systems using GPGPUs. In: Proceedings of the 12th annual conference on genetic and evolutionary computation, GECCO '10. ACM, New York, NY, pp 1039–1046

Freitas AA (2002) Data mining and knowledge discovery with evolutionary algorithms. Springer, New York

General-purpose computation on graphics hardware. http://www.gpgpu.org. Accessed November 2010

Harding S (2010) Genetic programming on graphics processing units bibliography. http://www.gpgpgpu.com. Accessed November 2010

Harding S, Banzhaf W (2007) Fast genetic programming and artificial developmental systems on gpus. In: High performance computing systems and applications, 2007. HPCS, pp 2-2

Harding S, Banzhaf W (2009) Distributed genetic programming on GPUs using CUDA. In: Workshop on parallel architectures and bioinspired algorithms. Raleigh, USA

Hwu WW (2009) Illinois ECE 498AL: programming massively parallel processors, lecture 13: reductions and their implementation. http://nanohub.org/resources/7376

Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection (complex adaptive systems). The MIT Press, Cambridge

Langdon WB, Harrison AP (2008) GP on SPMD parallel graphics hardware for mega bioinformatics data mining. Soft Comput 12(12):1169–1183

Landry J, Kosta LD, Bernier T (2006) Discriminant feature selection by genetic programming: towards a domain independent multi-class object detection system. J Syst Cybern Inform 3(1)

Maitre O, Baumes LA, Lachiche N, Corma A, Collet P (2009) Coarse grain parallelization of evolutionary algorithms on gpgpu cards with easea. In: Proceedings of the 11th annual conference on genetic and evolutionary computation, GECCO '09. ACM, New York, NY, USA, pp 1403–1410

Newman DJ, Asuncion A (2007) UCI machine learning repository. University of California, Irvine

NVIDIA (2010) NVIDIA programming and best practices guide. http://www.nvidia.com/cuda. Accessed November 2010

Robilliard D, Marion-Poty V, Fonlupt C (2009) Genetic programming on graphics processing units. Genetic Program Evolvable Mach 10:447–471

Ryoo S, Rodrigues CI, Baghsorkhi SS, Stone SS, Kirk DB, Hwu WW (2008) Optimization principles and application performance evaluation of a multithreaded GPU using CUDA. In: Proceedings of the 13th ACM SIGPLAN symposium on principles and practice of parallel programming, PPoPP '08. ACM, New York, NY, pp 73–82

Schmitz T, Hohmann S, Meier K, Schemmel J, Schürmann F (2003) Speeding up hardware evolution: a coprocessor for evolutionary algorithms. In: Proceedings of the 5th international conference on evolvable systems: from biology to hardware, ICES'03. Springer, pp 274–285

Tan KC, Tay A, Lee TH, Heng CM (2002) Mining multiple comprehensible classification rules using genetic programming. In: Proceedings of the evolutionary computation on 2002. CEC '02. Proceedings of the 2002 Congress, volume 2 of CEC '02. IEEE Computer Society, Washington, DC, pp 1302–1307

Ventura S, Romero C, Zafra A, Delgado JA, Hervás C (2007) JCLEC: a Java framework for evolutionary computation. Soft Comput 12:381–392

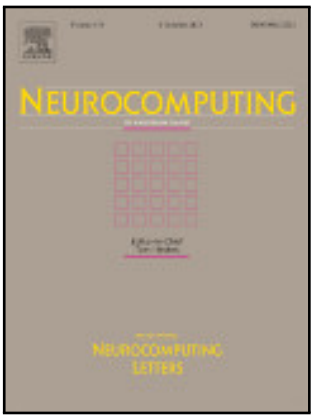
Wong ML, Leung KS (2000) Data mining using grammar-based genetic programming and applications. Kluwer Academic Publishers, Norwell, MA

TITLE:

Parallel evaluation of Pittsburgh rule-based classifiers on GPUs

AUTHORS:

A. Cano, A. Zafra, and S. Ventura



Neurocomputing, *Volume 126, pp. 45-57, 2014*

RANKING:

Impact factor (JCR 2012): 1.634

Knowledge area:

Computer Science, Artificial Intelligence: 37/114

Neurocomputing 126 (2014) 45–57

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Parallel evaluation of Pittsburgh rule-based classifiers on GPUs

Alberto Cano, Amelia Zafra, Sebastián Ventura *

Department of Computer Science and Numerical Analysis, University of Cordoba, Campus Universitario Rabanales, Edificio Einstein, Tercera Planta, 14071 Cordoba, Spain

ARTICLE INFO

Article history:
Received 1 May 2012
Received in revised form
7 November 2012
Accepted 10 January 2013
Available online 7 August 2013

Keywords:
Pittsburgh
Classification
Rule sets
Parallel computing
GPUs

ABSTRACT

Individuals from Pittsburgh rule-based classifiers represent a complete solution to the classification problem and each individual is a variable-length set of rules. Therefore, these systems usually demand a high level of computational resources and run-time, which increases as the complexity and the size of the data sets. It is known that this computational cost is mainly due to the recurring evaluation process of the rules and the individuals as rule sets. In this paper we propose a parallel evaluation model of rules and rule sets on GPUs based on the NVIDIA CUDA programming model which significantly allows reducing the run-time and speeding up the algorithm. The results obtained from the experimental study support the great efficiency and high performance of the GPU model, which is scalable to multiple GPU devices. The GPU model achieves a rule interpreter performance of up to 64 billion operations per second and the evaluation of the individuals is speeded up of up to 3.461 × when compared to the CPU model. This provides a significant advantage of the GPU model, especially addressing large and complex problems within reasonable time, where the CPU run-time is not acceptable.

1. Introduction

Evolutionary computation and its application to machine learning and data mining, and specifically, to classification problems, has attracted the attention of researchers over the last decade [1–5]. Classification is a supervised machine learning task which consists in predicting class membership of uncategorised examples using the properties of a set of train examples from which a classification model has been inducted [6].

Rule-based classification systems are especially useful in applications and domains which require comprehensibility and clarity in the knowledge discovery process, expressing information in the form of IF–THEN classification rules. Evolutionary rule-based algorithms take advantage of fitness-biased generational inheritance evolution to obtain rule sets, classifiers, which cover the train examples and produce class prediction over new examples.

Rules are encoded into the individuals within the population of the algorithm in two different ways: individual = rule, or individual = set of rules. Most evolutionary rule-based algorithms follow the first approach due to its simplicity and efficiency, whereas the latter, also known as Pittsburgh style algorithms, are not so usually employed because they are considered to perform slowly [7]. However, Pittsburgh approaches comprise other advantages such as providing individuals as complete solutions to the problem and allowing considering relations between the rules within the evolutionary process.

The efficiency, computational cost, and run-time of Pittsburgh rule-based systems are a primary concern and a challenge for researchers [8,9], especially when seeking their scalability to large scale databases [10,11], processing vast amounts of data within a reasonable amount of time. Therefore, it becomes crucial to design efficient parallel algorithms capable of handling these large amounts of data [12–15].

Parallel implementations have been employed to speed up evolutionary algorithms, including multi-core and distributed computing [16,17], master–slave models [18], and grid computing environments [19,20]. Over the last few years, increasing attention has focused on graphic processing units (GPUs). GPUs are devices with multi-core architectures and massive parallel processor units, which provide fast parallel hardware for a fraction of the cost of a traditional parallel system. Actually, since the introduction of the computer unified device architecture (CUDA) in 2007, researchers all over the world have harnessed the power of the GPU for general purpose GPU computing (GPGPU) [21–24].

The use of GPGPU has already been studied for speeding up algorithms within the framework of evolutionary computation and data mining [25–28], achieving high performance and promising results. Specifically, there are GPU-accelerated genetic rule-based systems for individual = rule approaches, which have been shown to achieve high performance [29–31]. Franco et al. [29] reported a speedup of up to 58 × using the BioHEL system. Cano et al. [30] reported a speedup of up to 820 ×, considering a scalable model using multiple GPU devices. Augusto [31] reported a speedup of up to 100 × compared to a single-threaded model and delivering

* Corresponding author. Tel.: +34 957212218; fax: +34 957218630.
E-mail addresses: acano@uco.es (A. Cano), azafra@uco.es (A. Zafra), sventura@uco.es (S. Ventura).

0925-2312/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.neucom.2013.01.049

almost 10 × the throughput of a twelve-core CPU. These proposals are all focused on speeding up individual=rule approaches. However, as far as we know, there are no GPU-based approaches to date using an individual=set of rules representation.

In this paper we present an efficient Pittsburgh individuals evaluation model on GPUs which parallelises the fitness computation for both rules and rules sets, applicable to any individual=set of rules evolutionary algorithm. The GPU model is scalable to multiple GPU devices, which allows addressing of larger data sets and population sizes. The rules interpreter, which checks the coverage of the rules over the instances, is carefully designed to maximise its efficiency compared to traditional rules stack-based interpreters. Experimental results demonstrate the great performance and high efficiency of the proposed model, achieving a rules interpreter performance of up to 64 billion operations per second. On the other hand, the individual evaluation performance achieves a speedup of up to 3.461 × when compared to the single-threaded CPU implementation, and a speedup of 1.311 × versus the parallel CPU version using 12 threads.

This paper is organised as follows. In the next section, genetic rule-based systems and their encodings are introduced, together with the definition of the CUDA programming model on the GPU. Section 3 presents the GPU evaluation model and its implementation in CUDA kernels. Section 4 introduces the experimental study setup, whose results are given in Section 5. Finally, Section 6 collects some concluding remarks.

2. Background

This section introduces the genetic rule-based systems and the encoding of the individuals. Finally, the CUDA programming model on the GPU is presented.

2.1. Genetic rule-based systems

Genetic algorithms (GAs) evolve a population of individuals which correspond to candidate solutions to a problem. GAs have been used for learning rules (Genetic rule-based systems), including crisp and fuzzy rules, and they follow two approaches for encoding rules within a population.

The first one represents an individual as a single rule (individual=rule). The rule base is formed by combining several individuals from the population (rule cooperation) or via different evolutionary runs (rule competition). This representation results in three approaches:

- Michigan: they employ reinforcement learning and the GA is used to learn new rules that replace the older ones via competition through the evolutionary process. These systems are usually called learning classifier systems [32], such as XCS [33], UCS [34], Fuzzy-XCS [35], and Fuzzy-UCS [36].
- Iterative Rule Learning (IRL): individuals compete to be chosen in every GA run. The rule base is formed by the best rules obtained when the algorithm is run multiple times. SLAVE [37], SIA [38] and HIDER [39] are examples which follow this model.
- Genetic Cooperative-Competitive Learning (GCCL): the whole population or a subset of individuals encodes the rule base. In this model, the individuals compete and cooperate simultaneously. This approach makes it necessary to introduce a mechanism to maintain the diversity of the population in order to avoid a convergence of all the individuals in the population. GP-COACH [40] or COGIN [41] follow this approach.

The second one represents an individual as a complete set of rules (individual=set of rules), which is also known as the Pittsburgh approach. The main advantage of this approach compared to the first one is that it allows addressing of the cooperation–competition problem, involving the interaction between rules in the evolutionary process [42,43]. Pittsburgh systems (especially naive implementations) are slower, since they evolve more complex structures and they assign credit at a less specific (and hence less informative) level [44]. Moreover, one of their main problems is controlling the number of rules, which increases the complexity of the individuals, adding computational cost to their evaluation and becoming an unmanageable problem. This problem is known as the bloat effect [45], i.e., a growth without control of the size of the individuals.

One method based on this approach is the Memetic Pittsburgh Learning Classifier System (MPLCS) [8]. In order to avoid the bloat effect, they employ a rule deletion operator and a fitness function based on the minimum description length [46], which balances the complexity and accuracy of the rule set. Moreover, this system uses a windowing scheme [47] that reduces the run-time of the system by dividing the training set into many non-overlapping subsets over which the fitness is computed at each GA iteration.

2.2. CUDA programming model

Computer unified device architecture (CUDA) [48] is a parallel computing architecture developed by NVIDIA that allows programmers to take advantage of the parallel computing capacity of NVIDIA GPUs in a general purpose manner. The CUDA programming model executes kernels as batches of parallel threads. These kernels comprise thousands to millions of lightweight GPU threads per each kernel invocation.

CUDA's threads are organised into thread blocks in the form of a grid. Thread blocks are executed in streaming multiprocessors. A stream multiprocessor can perform zero-overhead scheduling to interleave warps (a warp is a group of threads that execute together) and hide the overhead of long-latency arithmetic and memory operations. GPU's architecture was rearranged from SIMD (Single Instruction, Multiple Data) to MIMD (Multiple Instruction, Multiple Data), which runs independent of separate program codes. Thus, up to 16 kernels can be executed concurrently as long as there are multiprocessors available. Moreover, asynchronous data transfers can be performed concurrently with the kernel executions. These two features allow speeding up of the execution compared to a sequential kernel pipeline and synchronous data transfers, as in the previous GPU architectures.

There are four different main memory spaces: global, constant, shared, and local. These GPU memories are specialised and have different access times, lifetimes, and output limitations.

- Global memory: A large long-latency memory that exists physically as an off-chip dynamic device memory. Threads can read and write global memory to share data and must write the kernel's output to be readable after the kernel terminates. However, a better way to share data and improve performance is to take advantage of shared memory.
- Shared memory: A small low-latency memory that exists physically as on-chip registers and its contents are only maintained during thread block execution and are discarded when the thread block completes. Kernels that read or write a known range of global memory with spatial or temporal locality can employ shared memory as a software-managed cache. Such caching potentially reduces global memory bandwidth demands and improves overall performance.
- Local memory: Each thread also has its own local memory space as registers, so the number of registers a thread uses determines the number of concurrent threads executed in the multiprocessor, which is called multiprocessor occupancy. To avoid wasting hundreds of cycles while a thread waits for a

long-latency global-memory load or store to complete, a common technique is to execute batches of global accesses, one per thread, exploiting the hardware's warp scheduling to overlap the threads' access latencies.

- Constant memory: This memory is specialised for situations in which many threads will read the same data simultaneously. This type of memory stores data written by the host thread, is accessed constantly, and does not change during the execution of the kernel. A value read from the constant cache is broadcast to all threads in a warp, effectively serving all loads from memory with a single-cache access. This enables a fast, single-ported cache to feed multiple simultaneous memory accesses.

There are some recommendations for improving the performance on a GPU [49]. Memory accesses must be coalesced as with accesses to global memory. Global memory resides in device memory and is accessed via 32, 64, or 128-byte segment memory transactions. It is recommended to perform a fewer but larger memory transactions. When a warp executes an instruction which accesses global memory, it coalesces the memory accesses of the threads within the warp into one or more of these memory transactions depending on the size of the word accessed by each thread and the distribution of the memory addresses across the threads. In general, the more the transactions are necessary, the more the unused words are transferred in addition to the words accessed by the threads, reducing the instruction throughput accordingly.

To maximise global memory throughput, it is therefore important to maximise the coalescing, by following optimal access patterns, using data types that meet the size and alignment requirements, or padding data. For these accesses to be fully coalesced, both the width of the thread block and the width of the array must be a multiple of the warp size.

3. Parallel Pittsburgh evaluation on GPU

This section first introduces the encoding of the Pittsburgh individuals on the GPU. Then, it will present the evaluation procedure of an individual's rules. Finally, it will describe the evaluation process of an individual's fitness.

3.1. Pittsburgh individual encoding

Pittsburgh individuals are variable-length sets of rules which may include a default rule class prediction, interesting when using decision lists [50] as individual representation. Rules are one of the formalisms most often used to represent classifiers (decision trees can be easily converted into a rule set [51]). The IF part of the rule is called the antecedent and contains a combination of attribute-value conditions on the predicting attributes. The THEN part is called the consequent and contains the predicted value for the class. This way, a rule assigns a data instance to the class pointed out by the consequent if the values of the predicting attributes satisfy the conditions expressed in the antecedent. Rule specification can be formally defined by means of a context-free grammar [52] as shown in Fig. 1.

$$
\begin{aligned}
&\langle S\rangle \rightarrow \langle S\rangle\ OR\ \langle cmp\rangle\\
&\langle S\rangle \rightarrow \langle S\rangle\ AND\ \langle cmp\rangle\\
&\langle S\rangle \rightarrow NOT\ \langle S\rangle\\
&\langle S\rangle \rightarrow \langle cmp\rangle\\
&\langle cmp\rangle \rightarrow \langle op\_num\rangle\ \langle variable\rangle\ \langle value\rangle\\
&\langle cmp\rangle \rightarrow \langle op\_cat\rangle\ \langle variable\rangle\ \langle value\rangle\\
&\langle op\_num\rangle \rightarrow \geq \mid > \mid < \mid \leq\\
&\langle op\_cat\rangle \rightarrow = \mid \neq\\
&\langle variable\rangle \rightarrow Any\ valid\ attribute\ in\ dataset\\
&\langle value\rangle \rightarrow Any\ valid\ value
\end{aligned}
$$

Fig. 1. Grammar specification for the rules.

Fig. 2 shows how the rules are stored in the GPU memory. Rules are usually computed by means of a stack-based interpreter [53,54]. Traditional stack-based interpreters perform push and pop operations on a stack, involving the operator and operands found in the rule. The rule encoding we employ allows the interpreter to achieve maximal efficiency by minimizing the number of push and pop operations on the stack, reading the rules from the left to the right. Attribute-value comparisons are expressed in prefix notation, which places operators to the left of their operands, whereas logical operators are expressed in postfix notation, in which the operator is placed after the operands. This way, the efficiency of the interpreter is increased by minimizing the number of operations on the stack. The interpreter avoids pushing or popping unnecessary operands and behaves as a finite-state machine. For example, the first rule represented in the individual from Fig. 2 reads the first element and finds the $>$ operator. The interpreter knows the cardinality of the $>$ operator, which has two operands. Thus, it directly computes $> At_1\ V_1$ and pushes the result into the stack. Then, the next element is $<$, it computes $< At_2\ V_2$ and pushes the result. Finally, the AND operator is found, the interpreter pops the two operands from the stack and returns the AND Boolean computation.

This interpreter model provides a natural representation which allows dealing with all types of logical operators with different cardinalities and operand types while keeping an efficient performance.

3.2. Evaluation of particular rules

Rules within individuals must be evaluated over the instances of the data set in order to assign a fitness to the rules. The evaluation of the rules is divided into two steps, which are implemented in two GPU kernels. The first one, the coverage kernel, checks the coverage of the rules over the instances of the data set. The second one, the reduction kernel, performs a reduction count of the predictions of the rules, to compute the confusion matrix from which the fitness metrics for a classification rule can be obtained.

3.2.1. Rule coverage kernel

The coverage kernel executes the rule interpreter and checks whether the instances of the data set satisfy the conditions comprised in the rules within the individuals. The interpreter takes advantage of the efficient representation of the individuals described in Section 3.1 to implement an efficient stack-based procedure in which the partial results coming from the child nodes are pushed into a stack and pulled back when necessary.

The interpreter behaves as a single task being executed on the Single Instruction Multiple Data (SIMD) processor, while the rules and instances are treated as data. Therefore, the interpreter parallelises the fitness computation cases for individuals, rules, and instances. Each thread is responsible for the coverage of a single rule over a single instance, storing the result of the matching of the coverage and the actual class of the instance to an array. Threads are grouped into a 3D grid of thread blocks, whose size depends on the number of individuals (width), instances (height), and rules (depth), as represented in Fig. 3.

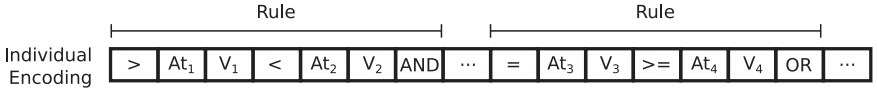


Fig. 2. Pittsburgh individual encoding.

Thus, a thread block represents a collection of threads which interpret a common rule over a subset of different instances, avoiding a divergence of the kernel, which is known to be one of the major efficiency problems of NVIDIA CUDA programming.

The number of threads per block is recommended to be a multiple of the warp size (a warp is a group of threads that execute together in a streaming multiprocessor), usually being 128, 192, 256, …, up to 1024 threads per block. This number is important as it concerns the scalability of the model in future GPU devices with a larger number of processors. NVIDIA recommends running at least twice as many thread blocks as the number of multiprocessors in the GPU, and provides an occupancy calculator which reports the GPU occupancy regarding the register and shared memory pressure, and the number of threads per block. Table 1 shows the GPU occupancy to be maximised for different block sizes. 192 threads per block is the best choice since it achieves 100% occupancy and provides more active thread blocks per multiprocessor to hide latency arising from register dependencies, and therefore, a wider range of possibilities given to the scheduler to issue concurrent block to the multiprocessors. Moreover, while the occupancy is maximal, the smaller number of threads per block there is, the higher the number of blocks, which provides better scalability to future GPU devices capable of handling more active blocks concurrently. Scalability to multiple GPU devices is achieved by splitting the population into as many GPUs as available, and each GPU is responsible for evaluating a subset of the population.

Thread accesses to global memory must be coalesced to achieve maximum performance and memory throughput, using data types that meet the size and alignment requirements, or padding data arrays. For these accesses to be fully coalesced, both the width of the thread block and the width of the array must be a multiple of the warp size. Therefore, the results array employs intra-array padding to align the memory addresses to the memory transfer segment sizes [30,55]. Since the number of threads per block is said to be 192, the results array intra-array padding forces the memory alignment to 192 float values, i.e., 768 bytes. Thus, memory accesses are fully coalesced and the best throughput is achieved. Memory alignment and padding details can be found in Section 5.3.2 from the NVIDIA CUDA programming guide.

Listing 1. Rule coverage kernel and interpreter.

```
__global__ void coverageKernel(unsigned char* result, double* instancesData,
                               int* instancesClass, float** rules, int** consequent)
{
  int instance = blockDim.y * blockIdx.y + threadIdx.y;
  int memIndex = (blockDim.z*blockIdx.x+blockIdx.z)*nInstances+instance;

  if(covers(&rules[blockIdx.x][blockIdx.z], instance, instancesData))
  {
    if(instancesClass[instance] == consequent[blockIdx.x][blockIdx.z])
      result[memIndex] = 0; // TRUE POSITIVE
    else
      result[memIndex] = 2; // FALSE POSITIVE
  }
  else
  {
    if(instancesClass[instance] != consequent[blockIdx.x][blockIdx.z])
      result[memIndex] = 1; // TRUE NEGATIVE
    else
      result[memIndex] = 3; // FALSE NEGATIVE
  }
}

__device__ bool covers(float* rule, int instance, double* instancesData)
{
  int sp, bufp, attribute;
  float stack[MAX_STACK], op1, op2;

  for(sp = 0, bufp = 0; ;)
  {
    switch(rule[bufp])
    {
      case GREATER:
        attribute = rule[bufp+1];
        op1 = instancesData[numberInstances * attribute + instance];
        op2 = rule[bufp+2];
        if (op1 > op2)                  push(1, stack, &sp);
        else                            push(0, stack, &sp);
        bufp += 3;
        break;
      ...
      case AND:
        op1 = pop(stack, &sp);
        op2 = pop(stack, &sp);
        if (op1 * op2 == 1)             push(1, stack, &sp);
        else                            push(0, stack, &sp);
        bufp++;
        break;
      ...
      case END_RULE:
        return pop(stack, &sp) == 1 ? true : false;
    }
  }
}
```

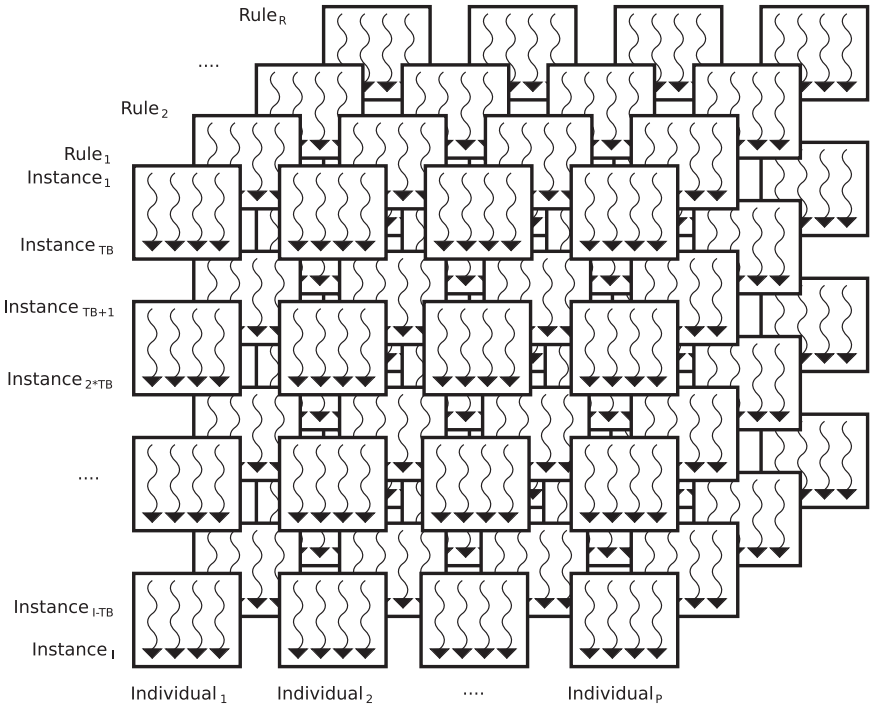


Fig. 3. 3D grid of thread blocks.

Table 1
Threads per block and GPU occupancy.

Threads per block	128	192	256	320
Active threads per multiprocessor	1024	1536	1536	1280
Active warps per multiprocessor	32	48	48	40
Active thread blocks per multiprocessor	8	8	6	4
Occupancy of each multiprocessor (%)	67	100	100	83

Threads within a warp shall request consecutive memory addresses that can be serviced in fewer memory transactions. All the threads in a warp evaluate the same rule but over different instances. Thus, the data set must be stored transpose in memory to provide fully coalescing memory requests to the threads from the warp.

The codes for the coverage kernel and the rule interpreter are shown in Listing 1. The coverage kernel receives as input four arrays: an array of attributes values, an array of class values of the instances of the dataset, an array containing the rules to evaluate, and an array containing the consequents of the rules. It computes the matching of the results and returns them in an array of matching results. The result of the matching of the rule prediction and the actual class of an instance can take four possible values: true positive (T_P), true negative (T_N), false positive (F_P), or false negative (F_N). Threads and blocks within the kernel are identified by the built-in CUDA variables *threadIdx*, *blockIdx* and *blockDim*, which specify the grid and block dimensions and the block and thread indexes, following the 3D representation shown in Fig. 3. Further information about CUDA threads indices can be seen in Section B.4 from CUDA programming guide.

3.2.2. Rule fitness kernel

The rule fitness kernel calculates the fitness of the rules by means of the performance metrics obtained from the confusion matrix. The confusion matrix is a two dimensional table which counts the number of true positives, false positives, true negatives, and false negatives resulting from the matching of a rule over the instances of the data set. There are many well-known performance metrics for classification, such as sensitivity, specificity, precision, recall, F-Measure. The algorithm assigns the fitness values corresponding to the objective or objectives to optimise, e.g., to maximise both sensitivity and specificity at the same time.

The rule fitness kernel is implemented using a 2D grid of thread blocks, whose size depends on the number of individuals (width) and the number of rules (height). The kernel performs a parallel reduction operation over the matching results of the coverage kernel. The naive reduction operation sums in parallel the values of an array reducing iteratively the information.

Our approach does not need to sum the values, but counting the number of T_P, T_N, F_P and F_N. $O(\log_2 N)$ parallel reduction is known to perform most efficiently in multi-core CPU processors with large arrays. However, our best results on GPUs were achieved using a 2-level parallel reduction with sequential addressing using 128 threads per block, which is shown in Fig. 4. Accessing sequential memory address in parallel is more efficient than accessing non-contiguous addresses since contiguous data are transferred in a single memory transaction and provides coalesced accesses to threads. Finally, the code for the rule fitness kernel is shown in Listing 2. The input of the kernel is the array of matching results, and returns an array of fitness values. The 2-level parallel reduction takes advantage of GPU shared memory, in which threads within a block collaborate to compute partial counts of the confusion matrix values. Each thread is responsible to count the results from the *base* index to the *top* index. Therefore, contiguous threads address contiguous memory indexes, achieving maximum throughput.

Listing 2. Rules fitness kernel.

```
__global__ void rulesFitnessKernel(float* fitness, unsigned char* result)
{
  __shared__ int confusionMatrix[512];
  int base = (blockDim.z*blockIdx.x+blockIdx.z)*numberInstances+threadIdx.y;
  int top =  (blockDim.z*blockIdx.x+blockIdx.z+1)*numberInstances-base;

  confusionMatrix[threadIdx.y] = confusionMatrix[threadIdx.y+128] = 0;
  confusionMatrix[threadIdx.y+256] = confusionMatrix[threadIdx.y+384] = 0;

  // Performs the first level reduction of the thread corresponding values
  for(int i = 0; i < top; i+=128)
     confusionMatrix[threadIdx.y*4 + result[base + i]]++;

  __syncthreads();

  if(threadIdx.y < 4)
  {
    // Performs the second level reduction of the half of the sums
    for(int i = 4; i < 512; i+=4)
    {
      confusionMatrix[0] += confusionMatrix[i];   // # true positives
      confusionMatrix[1] += confusionMatrix[i+1]; // # true negatives
      confusionMatrix[2] += confusionMatrix[i+2]; // # false positives
      confusionMatrix[3] += confusionMatrix[i+3]; // # false negatives
    }

    if(threadIdx.y == 0)
    {
      int tp = MC[0], tn = MC[1], fp = MC[2], fn = MC[3];

      float sensitivity, specificity;

      sensitivity = tp / (float) (tp + fn);
      specificity = tn / (float) (tn + fp);

      fitness[blockIdx.z][blockIdx.x] = sensitivity * specificity;
    }
  }
}
```

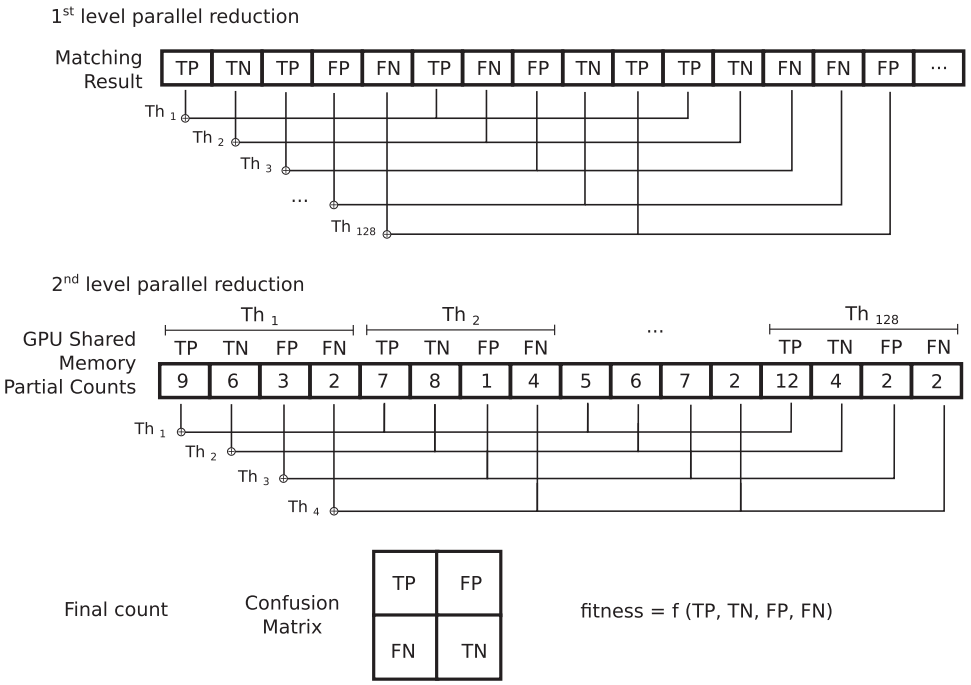


Fig. 4. 2-level parallel reduction with sequential addressing.

3.3. Evaluation of rule sets

Pittsburgh individuals encode sets of rules as complete solutions to the classification problem (classifiers). Many performance measures of a classifier can be evaluated using the ion matrix. The standard performance measure for classification is the accuracy rate, which is the number of successful predictions relative to the total number of classifications. The evaluation of the classifiers is divided into two steps, which are implemented in two GPU kernels. The first one, the classification kernel, performs the class prediction for the instances of the data set. The second one, the rule set fitness kernel, performs a reduction count of the classifier predictions to compute the confusion matrix, from which the fitness metrics for a classifier can be obtained.

3.3.1. Rule set classification kernel

The rule set classification kernel performs the class prediction for the instances of the data set using the classification rules, which are linked as a decision list. An instance is predicted to the class pointed out by the consequent of the first rule which satisfies the conditions of the antecedent. If no rule covers the instance, it is classified using the default class.

In order to save time, the classification kernel reuses the matching results from the rule coverage kernel, and therefore, the rules do not need to be interpreted again. The classifier follows the decision list inference procedure to perform the class prediction. Notice that the class prediction is only triggered when the rule is known to cover the instance (true positive or false positive).

Listing 3. Rule set classification kernel.

```
__global__ void classificationKernel(unsigned char* result, int* Class,
                                      int defaultClass)
{
  int instance = blockDim.y * blockIdx.y + threadIdx.y;
  int index = blockIdx.x * maxRules * numberInstances + instance;

  for(int i = 0; i < maxRules; i++)
  {
    if(result[index + i*numberInstances] == 0)  // TRUE POSITIVE
    {
      result[index] = 1; // Correctly classified
      return;
    }
    else if(result[index + i*numberInstances] == 2) // FALSE POSITIVE
    {
      result[index] = 0; // Misclassified
      return;
    }
  }

  // If none of the rules covers the instance, apply the default hypothesis
  if(Class[instance] == defaultClass)
    result[index] = 1;  // Correctly classified
  else
    result[index] = 0;  // Misclassified
}
```

The classification kernel is implemented using a 2D grid of thread blocks, whose size depends on the number of individuals (width) and instances (height). The kernel setup is similar to the rule coverage kernel. The number of threads per block is also 192, to maximise the occupancy of the streaming multiprocessors. Listing 3 shows the code for the classification kernel. The input of the kernel is the array of matching results, an array with information about the instance class and the default class, which applies when none of the rules covers the instance (default hypothesis).

3.3.2. Rule set fitness kernel

Listing 4. Rule sets fitness kernel.

```
__global__ void ruleSetFitnessKernel(float* fitness, unsigned char* result)
{
  __shared__ int shmCount[128];

  shmCount[threadIdx.y] = 0;

  int base = blockIdx.x*numberInstances*maxRules + threadIdx.y;
  int top =  blockIdx.x*numberInstances*maxRules + numberInstances - base;

  // Performs the reduction of the thread corresponding values
  for(int i = 0; i < top; i+=128)
  {
    shmCount[threadIdx.y] += result[base + i];
  }

  __syncthreads();

  // Calculates the final amount and the accuracy
  if(threadIdx.y == 0)
  {
    int correctPredictions = shmCount[0];

    for(int i = 1; i < 128; i++)
      correctPredictions += shmCount[i];

    // Compute the accuracy of the classifier
    fitness[blockIdx.x] = correctPredictions / (float) numberInstances;
  }
}
```

The rule set fitness kernel performs a reduction operation over the classifier predictions to count the number of successful predictions. The reduction operation is similar to the one from the rule fitness kernel from Section 3.2.2 and counts the number of correctly classified instances to compute the accuracy of the classifier. The settings for the kernel and the reduction operation are the same. The kernel is implemented using a 1D grid of thread blocks whose length depends only on the number of individuals. The code for the rule set fitness kernel is shown in Listing 4. The kernel receives as input the array of prediction results from the rule set classification kernel, and returns an array of fitness values which defines the accuracy of the classifiers. Similarly than the rules fitness kernel, shared memory is employed to count partial results and guarantee contiguous and coalesced memory accesses.

4. Experimental setup

This section describes the experimental study setup, the hardware configuration, and the experiments designed to evaluate the efficiency of the GPU model.

4.1. Hardware configuration

The experiments were run on a cluster of machines equipped with dual Intel Xeon E5645 processors running at 2.4 GHz and 24 GB of DDR3 host memory. The GPUs employed were two NVIDIA GTX 480 video cards equipped with 1.5 GB of GDDR5 video RAM. The GTX 480 GPU comprised 15 multiprocessors and 480 CUDA cores. The host operating system was a GNU/Linux Rocks cluster 5.4.3 64 bit together with CUDA 4.1 runtime.

4.2. Problem domains

The performance of the GPU model was evaluated on a series of data sets collected from the UCI machine learning repository [56] and the KEEL data sets repository [57]. These data sets are very varied, with different degrees of complexity. Thus, the number of instances ranges from the simplest, containing 150 instances, to the most complex, containing one million instances. The number of attributes and classes also differ significantly to represent a wide variety of real word data problems. This information is summarised in Table 2. The wide variety of data sets allowed us to evaluate the model performance on problems of both low and high complexities.

4.3. Experiments

The experimental study comprises three experiments designed to evaluate the performance and efficiency of the model. Firstly, the performance of the rules interpreter was evaluated. Then, the times required for evaluating individuals by CPU and GPU were compared. Finally, the efficiency of the model was analysed regarding performance and power consumption.

4.3.1. Rule interpreter performance

The efficiency of rule interpreters is often reported by means of the number of primitives interpreted by the system per second,

Table 2
Complexity of the data sets.

Data set	# Instances	# Attributes	# Classes
Iris	150	4	3
New-thyroid	215	5	3
Ecoli	336	7	8
Contraceptive	1473	9	3
Thyroid	7200	21	3
Penbased	10,992	16	10
Shuttle	58,000	9	7
Connect-4	67,557	42	3
KDDcup	494,020	41	23
Poker	1,025,010	10	10

similar to Genetic Programming interpreters, which determine the number of Genetic Programming operations per second (GPops/s) [31,53,54].

In this experiment, the performance of the rules interpreter was evaluated by running the interpreter with a different number of rules over data sets with varied number of instances and attributes. Thus, the efficiency of the interpreter was analysed regarding its scalability to larger numbers of rules and instances.

4.3.2. Individual evaluation performance

The second experiment evaluated the performance of the evaluation of the individuals and their rules in order to compute their fitness values. This experiment compared the execution times (these times consider in the case of CPU cluster, data transfers between compute nodes and the GPU times, the data transfer between host and GPU memory) dedicated to evaluate different population sizes over the data sets. The range of population sizes varies from 10 to 100 individuals. This range of population sizes is commonly used in most of the classification problems and algorithms, and represents a realistic scenario for real world data. The number of rules of each individual is equal to the number of classes of the data set, and the length of the rules varies stochastically regarding the number of attributes of the data set, i.e., rules are created adapted to the problem complexity. Thus, the experiments are not biased for unrealistic more complex rules and individuals which would obtain better speedups. The purpose of this experiment was to obtain the speedups of the GPU model and check its scalability to large data sets and multiple GPU devices. Extension to multiple GPUs is simple, the population is divided into as many GPUs as available, and each GPU is responsible for evaluating a subset of the population. Therefore, the scalability is guaranteed to larger population sizes and further number of GPU devices.

4.3.3. Performance per Watt

Power consumption has increasingly become a major concern for high-performance computing, due not only to the associated electricity costs, but also to environmental factors [58]. The power efficiency is analysed based on the throughput results on the evaluated cases. To simplify the estimates, it is assumed that the devices work at their full occupancy, that is, at maximum power consumption [31]. One NVIDIA GTX 480 GPU consumes up to 250 W, whereas one Intel Xeon E5645 consumes up to 80 W. The efficiency of the model is evaluated regarding the performance per Watt (GPops/s/W). The power consumption is reported to the CPU or GPU itself and it does not take into account the base system power consumption. We followed this approach because it is the commonly accepted way both in academia and industry [31] to report the performance per Watt efficiency.

5. Results

Table 3 shows the rule interpreter execution times and performance in terms of the number of primitives interpreted per second (GPops/s). Each row represents the case of a stack-based interpretation of the rules from the population over the instances of the data sets. The number of rules of each individual is equal to the number of classes of the data set. The number of primitives, Genetic Programming operations (GPops), reflects the total number of primitives to be interpreted for that case, which depends on the variable number of rules, their length, and the number of instances, representing the natural variable length of Pittsburgh problems.

The single-threaded CPU interpreter achieves a performance of up to 9.63 million GPops/s, whereas multi-threading with 4 CPU threads brings the performance up to 34.70 million GPops/s. The dual socket cluster platform allows two 6-core CPUs and a total of 12 CPU threads, which are capable of running up to 92.06 million GPops/s in parallel.

On the other hand, the GPU implementation obtains great performance in all cases, especially over large scale data sets with a higher number of instances. One GPU obtains up to 31 billion GPops/s, whereas scaling to two GPU devices enhances the interpreter performance up to 64 billion GPops/s. The best scaling is achieved when a higher number of instances and individuals are considered, i.e., the GPU achieves its maximum performance and occupancy when there are enough threads to fill the GPU multiprocessors. Fig. 5 shows the GPops/s scaling achieved by the GPU model regarding the number of nodes to interpret. The higher the number of nodes to interpret, the higher the occupancy of the GPU and thus, the higher the efficiency.

Table 4 shows the evaluation times and the speedups of the GPUs versus the single-threaded and 12-threaded CPU implementations. The GPU model has high performance and efficiency, which increase as the number of individuals and instances increase. The highest speed up over the single-threaded CPU version is achieved for the Connect-4 data set using 100 individuals (1.880 × using one GPU and 3.461 × using two GPU devices). On the other hand, compared to the parallel 12-threaded CPU version, the highest speedup is 933 × using one GPU and 1.311 × using two GPUs. The evaluation times for the Poker data set using 100 individuals are reduced from 818 s (13 min and 38 s) to 0.2390 s using two NVIDIA GTX 480 GPUs. Since evolutionary algorithms perform the evaluation of the population each generation, the total amount of time dedicated to evaluate individuals along generations becomes a major concern. GPU devices allow greatly speeding up the evaluation process and save much time.

Fig. 6 shows the speedup obtained by comparing the evaluation time when using two NVIDIA GTX 480 GPUs and the single-threaded CPU evaluator. The figure represents the speedup over the four largest data sets with the higher number of instances. The higher the number of instances, the more the number of parallel and concurrent threads to evaluate and thus, the higher the occupancy of the GPU.

Finally, Table 5 shows the efficiency of the model regarding the computing devices, their power consumption, and their performance in terms of GPops/s. Parallel threaded CPU solutions increase their performance as more threads are employed. However, their efficiency per Watt is decreased as more CPU cores are used. On the other hand, GPUs require many Watts but their performance is justified by a higher efficiency per Watt. Specifically, the single-threaded CPU performs around 0.7 million GPops/s/W whereas using two GPUs increases its efficiency up to 129.96 million GPops/s, which is higher than the efficiency

Table 3
Rule interpreter performance.

Data set	Population	GPops	Interpreter time (s)					GPops/s (million)				
			1 CPU	4 CPU	12 CPU	1 GPU	2 GPU	1 CPU	4 CPU	12 CPU	1 GPU	2 GPU
Iris	10	88,560	12	10	8	0.0272	0.0201	7.38	8.86	11.07	3259.72	4406.85
	25	225,450	25	12	11	0.0390	0.0276	9.02	18.79	20.50	5774.85	8163.75
	50	458,460	49	19	15	0.0526	0.0295	9.36	24.13	30.56	8719.95	15,522.07
	100	929,340	100	34	21	0.0988	0.0491	9.29	27.33	44.25	9410.85	18,919.79
New-thyroid	10	126,608	15	12	10	0.0274	0.0204	8.44	10.55	12.66	4627.49	6211.15
	25	322,310	35	19	11	0.0400	0.0265	9.21	16.96	29.30	8064.20	12,164.48
	50	655,428	71	22	13	0.0505	0.0340	9.23	29.79	50.42	12,988.03	19,268.23
	100	1,328,612	144	41	24	0.0967	0.0535	9.23	32.41	55.36	13,734.41	24,832.01
Ecoli	10	539,976	58	18	13	0.0516	0.0392	9.31	30.00	41.54	10,461.41	13,763.66
	25	1,354,168	152	44	23	0.0972	0.0533	8.91	30.78	58.88	13,929.48	25,416.07
	50	2,830,344	299	89	41	0.1813	0.1075	9.47	31.80	69.03	15,607.60	26,339.56
	100	5,658,272	587	166	69	0.3471	0.1708	9.64	34.09	82.00	16,299.87	33,118.75
Contraceptive	10	869,200	96	30	22	0.0550	0.0322	9.05	28.97	39.51	15,801.34	27,000.50
	25	2,212,750	243	70	32	0.1107	0.0625	9.11	31.61	69.15	19,990.88	35,424.40
	50	4,499,700	499	134	102	0.1974	0.1107	9.02	33.58	44.11	22,793.91	40,640.35
	100	9,121,300	989	295	112	0.3857	0.1950	9.22	30.92	81.44	23,646.97	46,773.98
Thyroid	10	4,250,880	488	159	74	0.1625	0.0903	8.71	26.74	57.44	26,165.06	47,089.68
	25	10,821,600	1204	312	153	0.3810	0.2024	8.99	34.68	70.73	28,406.13	53,474.86
	50	22,006,080	2394	703	308	0.7508	0.3780	9.19	31.30	71.45	29,308.30	58,219.61
	100	44,608,320	4824	1403	616	1.5396	0.7536	9.25	31.79	72.42	28,974.27	59,196.14
Penbased	10	22,712,032	2422	1001	550	0.7889	0.3928	9.38	22.69	41.29	28,789.64	57,820.86
	25	55,672,176	6233	1829	777	1.9124	1.0215	8.93	30.44	71.65	29,110.43	54,498.50
	50	115,617,696	12,259	3332	1371	3.9284	1.9120	9.43	34.70	84.33	29,431.36	60,470.52
	100	228,030,384	24,292	7447	2477	7.7361	3.9117	9.39	30.62	92.06	29,476.28	58,294.27
Shuttle	10	80,804,052	9079	3310	2416	2.6057	1.3069	8.90	24.41	33.45	31,010.93	61,826.71
	25	204,515,682	23,116	7325	3332	6.5364	3.4067	8.85	27.92	61.38	31,288.50	60,033.02
	50	424,691,064	45,834	14,661	5839	13.4832	6.5357	9.27	28.97	72.73	31,497.72	64,980.39
	100	852,514,068	90,649	71,206	11,197	27.0089	13.4812	9.40	11.97	76.14	31,564.16	63,237.33
Connect-4	10	39,886,112	5206	2026	1772	1.3552	0.7188	7.66	19.69	22.51	29,431.90	55,491.10
	25	101,539,340	13,164	4455	2472	3.3903	1.7277	7.71	22.79	41.08	29,949.92	58,770.99
	50	206,483,592	25,123	8250	3714	6.8567	3.3941	8.22	25.03	55.60	30,114.12	60,835.82
	100	418,560,968	54,029	37,498	7251	13.8397	6.8558	7.75	11.16	57.72	30,243.40	61,051.74
Kddcup	10	2,297,785,824	293,657	99,679	64,722	73.1540	37.1375	7.82	23.05	35.50	31,410.28	61,872.44
	25	5,985,447,516	733,670	208,969	86,570	189.2256	96.7096	8.16	28.64	69.14	31,631.28	61,890.96
	50	11,748,586,032	1,466,624	389,555	145,077	372.2638	189.2589	8.01	30.16	80.98	31,559.84	62,076.80
	100	23,408,248,464	2,900,167	1,926,780	290,873	742.0416	372.2767	8.07	12.15	80.48	31,545.73	62,878.63
Poker	10	2,118,078,368	237,524	104,783	73,069	70.1586	33.9806	8.92	20.21	28.99	30,189.86	62,331.92
	25	5,191,875,024	616,642	191,831	97,471	172.5495	91.4097	8.42	27.06	53.27	30,089.19	56,797.84
	50	10,782,273,504	1,222,919	376,896	162,384	356.6680	172.6337	8.82	28.61	66.40	30,230.56	62,457.54
	100	21,265,654,416	2,404,491	1,649,626	284,182	704.3908	356.5822	8.84	12.89	74.83	30,190.13	59,637.45

reported in related works [31], which achieve a performance up to 52.7 million GPops/s per Watt.

6. Conclusions

In this paper we have presented a high-performance and efficient evaluation model for individual=rule set (Pittsburgh) genetic rule-based algorithms. The rule interpreter and the GPU kernels have been designed to maximise the GPU occupancy and throughput, reducing the evaluation time of the rules and rule sets. The experimental study has analysed the performance and scalability of the model over a series of varied data sets with different numbers of instances. It is concluded that the GPU implementation is highly efficient, scalable to multiple GPU devices. The best performance was achieved when the number of instances or the population size was large enough to fill the GPU multiprocessors. The speedup of the model was up to 3.461 × when addressing large scale classification problems with two GPUs, significantly higher than the speed-up achieved by the CPU parallel 12-threaded solution.

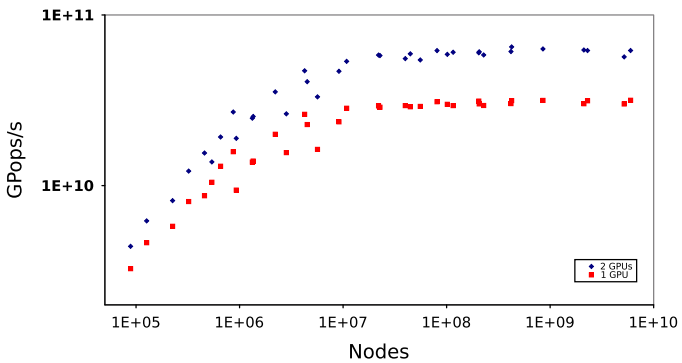


Fig. 5. GPU model GPops/s scaling.

Table 4
Individual evaluation performance.

Data set	Population	Evaluation time (s)					Speedup vs 1 CPU		Speedup vs 12 CPU	
		1 CPU	4 CPU	12 CPU	1 GPU	2 GPU	1 GPU	2 GPU	1 GPU	2 GPU
Iris	10	0.0096	0.0093	0.0090	0.0010	0.0007	9.60	13.71	9.00	12.86
	25	0.0135	0.0122	0.0106	0.0014	0.0009	9.64	15.00	7.57	11.78
	50	0.0203	0.0200	0.0191	0.0016	0.0010	12.69	20.30	11.94	19.10
	100	0.0420	0.0369	0.0247	0.0019	0.0013	22.11	32.31	13.00	19.00
New-thyroid	10	0.0094	0.0090	0.0088	0.0008	0.0006	11.75	15.67	11.00	14.67
	25	0.0166	0.0129	0.0167	0.0012	0.0008	13.83	20.75	13.92	20.88
	50	0.0300	0.0280	0.0188	0.0017	0.0010	17.65	30.00	11.06	18.80
	100	0.0611	0.0533	0.0294	0.0027	0.0012	22.63	50.92	10.89	24.50
Ecoli	10	0.0323	0.0182	0.0122	0.0015	0.0013	21.53	24.85	8.13	9.38
	25	0.0543	0.0475	0.0303	0.0021	0.0014	25.86	38.79	14.43	21.64
	50	0.1026	0.0818	0.0428	0.0029	0.0025	35.38	41.04	14.76	17.12
	100	0.2090	0.1596	0.0717	0.0042	0.0031	49.76	67.42	17.07	23.13
Contraceptive	10	0.0459	0.0419	0.0328	0.0010	0.0008	45.90	57.38	32.80	41.00
	25	0.1002	0.0828	0.0474	0.0013	0.0010	77.08	100.20	36.46	47.40
	50	0.2036	0.1774	0.1046	0.0017	0.0011	119.76	185.09	61.53	95.09
	100	0.4177	0.3415	0.1617	0.0020	0.0017	208.85	245.71	80.85	95.12
Thyroid	10	0.2112	0.1999	0.1691	0.0010	0.0007	211.20	301.71	169.10	241.57
	25	0.4933	0.4162	0.2185	0.0012	0.0010	411.08	493.30	182.08	218.50
	50	1.0148	0.8749	0.3709	0.0015	0.0011	676.53	922.55	247.27	337.18
	100	2.0318	1.5358	0.6768	0.0029	0.0013	700.62	1562.92	233.38	520.62
Penbased	10	0.7738	0.7118	0.4548	0.0017	0.0011	455.18	703.45	267.53	413.45
	25	2.0064	1.5883	0.7367	0.0030	0.0019	668.80	1056.00	245.57	387.74
	50	4.0047	3.0185	1.3909	0.0050	0.0032	800.94	1251.47	278.18	434.66
	100	8.5293	6.2505	2.4959	0.0094	0.0049	907.37	1740.67	265.52	509.37
Shuttle	10	2.6452	2.7268	2.2034	0.0028	0.0018	944.71	1469.56	786.93	1224.11
	25	7.2141	5.8981	3.4525	0.0057	0.0034	1265.63	2121.79	605.70	1015.44
	50	15.6694	12.8749	5.4326	0.0100	0.0057	1566.94	2749.02	543.26	953.09
	100	32.1477	24.6465	9.8108	0.0200	0.0111	1607.38	2896.19	490.54	883.86
Connect-4	10	2.2684	1.9459	1.2467	0.0020	0.0014	1134.20	1620.29	623.35	890.50
	25	4.9263	4.2588	2.3661	0.0034	0.0021	1448.91	2345.86	695.91	1126.71
	50	10.3063	8.4253	4.7221	0.0064	0.0036	1610.36	2862.86	737.83	1311.69
	100	21.8092	16.0537	5.6394	0.0116	0.0063	1880.10	3461.78	486.16	895.14
Kddcup	10	84.7511	78.6884	31.2643	0.0515	0.0267	1645.65	3174.20	607.07	1170.95
	25	208.7099	161.4730	73.0745	0.1236	0.0646	1688.59	3230.80	591.22	1131.18
	50	426.7217	291.6318	123.8365	0.2471	0.1246	1726.92	3424.73	501.16	993.87
	100	841.4498	578.6901	213.8102	0.4850	0.2447	1734.95	3438.70	440.85	873.76
Poker	10	74.0020	69.8736	46.0978	0.0494	0.0277	1498.02	2671.55	933.15	1664.18
	25	193.5917	162.2887	81.8815	0.1203	0.0648	1609.24	2987.53	680.64	1263.60
	50	385.2833	293.4088	126.4694	0.2325	0.1189	1657.13	3240.40	543.95	1063.66
	100	818.7177	591.6098	229.6941	0.4749	0.2390	1723.98	3425.60	483.67	961.06

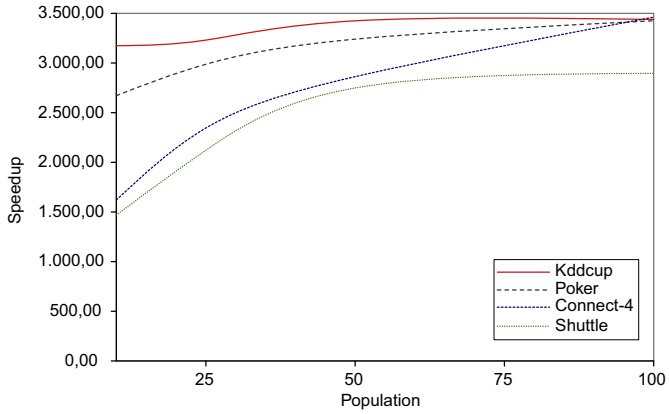


Fig. 6. Model speedup using two GPUs. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.).

Table 5
Performance per Watt.

Compute device	Watts (W)	GPops/s (million)	GPops/s/W (million)
Intel Xeon E5645/1 CPU/1 core	12.5	9.63	0.77
Intel Xeon E5645/1 CPU/4 cores	50	34.70	0.69
Intel Xeon E5645/2 CPU/12 cores	160	92.06	0.58
NVIDIA GTX 480/1 GPU	250	31,631.28	126.52
NVIDIA GTX 480/2 GPU	500	64,980.39	129.96

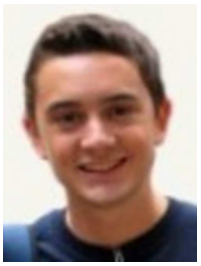
The rule interpreter obtained a performance above 64 billion GPops/s and even the efficiency per Watt is up to 129 million - GPops/s/W.

Acknowledgement

This work was supported by the Regional Government of Andalusia and the Ministry of Science and Technology, projects P08-TIC-3720 and TIN-2011-22408, FEDER funds, and Ministry of Education FPU Grant AP2010-0042.

References

[1] A. Ghosh, L. Jain (Eds.), Evolutionary Computation in Data Mining, Studies in Fuzziness and Soft Computing, vol. 163, Springer, 2005.
[2] A. Abraham, E. Corchado, J.M. Corchado, Hybrid learning machines, Neurocomputing 72 (2009) 2729–2730.
[3] E. Corchado, M. Graña, M. Wozniak, New trends and applications on hybrid artificial intelligence systems, Neurocomputing 75 (2012) 61–63.
[4] E. Corchado, A. Abraham, A. de Carvalho, Hybrid intelligent algorithms and applications, Information Sciences 180 (2010) 2633–2634.
[5] W. Pedrycz, R.A. Aliev, Logic-oriented neural networks for fuzzy neurocomputing, Neurocomputing 73 (2009) 10–23.
[6] S.B. Kotsiantis, Supervised machine learning: a review of classification techniques, Informatica (Ljubljana) 31 (2007) 249–268.
[7] A.A. Freitas, Data Mining and Knowledge Discovery with Evolutionary Algorithms, Springer, 2002.
[8] J. Bacardit, N. Krasnogor, Performance and efficiency of memetic Pittsburgh learning classifier systems, Evolutionary Computation 17 (2009) 307–342.
[9] J. Bacardit, D.E. Goldberg, M.V. Butz, X. Llor, J.M. Garrell, Speeding-up Pittsburgh Learning Classifier Systems: Modeling Time and Accuracy, in: Lecture Notes in Computer Science, vol. 3242, 2004, pp. 1021–1031.
[10] J. Bacardit, X. Llor, Large scale data mining using genetics-based machine learning, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2011, pp. 1285–1309.
[11] S. Zhang, X. Wu, Large scale data mining based on data partitioning, Applied Artificial Intelligence 15 (2001) 129–139.
[12] E. Cantu-Paz, Efficient and Accurate Parallel Genetic Algorithms, Kluwer Academic Publishers, Norwell, MA, USA, 2000.
[13] E. Alba, M. Tomassini, Parallelism and evolutionary algorithms, IEEE Transactions on Evolutionary Computation 6 (2002) 443–462.
[14] M.J. Zaki, C.-T. Ho, Large-Scale Parallel Data Mining, in: Lecture Notes in Computer Science, vol. 1759, 2000, Springer.
[15] J.J. Weinman, A. Lidaka, S. Aggarwal, Large-scale machine learning, in: GPU Gems Emerald Edition, Morgan Kaufman, 2011, pp. 277–291.
[16] P.E. Srokosz, C. Tran, A distributed implementation of parallel genetic algorithm for slope stability evaluation, Computer Assisted Mechanics and Engineering Sciences 17 (2010) 13–26.
[17] M. Rodríguez, D.M. Escalante, A. Peregrín, Efficient distributed genetic algorithm for rule extraction, Applied Soft Computing 11 (2011) 733–743.
[18] S. Dehuri, A. Ghosh, R. Mall, Parallel multi-objective genetic algorithm for classification rule mining, IETE Journal of Research 53 (2007) 475–483.
[19] A. Flling, C. Grimme, J. Lepping, A. Papaspyrou, Connecting Community-Grids by supporting job negotiation with coevolutionary Fuzzy-Systems, Soft Computing 15 (2011) 2375–2387.
[20] P. Switalski, F. Seredynski, An Efficient Evolutionary Scheduling Algorithm for Parallel Job Model in Grid Environment, in: Lecture Notes in Computer Science, vol. 6873, 2011, pp. 347–357.
[21] J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krger, A.E. Lefohn, T. J. Purcell, A survey of general-purpose computation on graphics hardware, Computer Graphics Forum 26 (2007) 80–113.
[22] J.D. Owens, M. Houston, D. Luebke, S. Green, J.E. Stone, J.C. Phillips, GPU computing, Proceedings of the IEEE 96 (2008) 879–899.
[23] S. Che, M. Boyer, J. Meng, D. Tarjan, J.W. Sheaffer, K. Skadron, A performance study of general-purpose applications on graphics processors using CUDA, Journal of Parallel and Distributed Computing 68 (2008) 1370–1380.
[24] D.M. Chitty, Fast parallel genetic programming: multi-core CPU versus many-core GPU, Soft Computing 16 (2012) 1795–1814.
[25] S.N. Omkar, R. Karanth, Rule extraction for classification of acoustic emission signals using Ant Colony Optimisation, Engineering Applications of Artificial Intelligence 21 (2008) 1381–1388.
[26] K.L. Fok, T.T. Wong, M.L. Wong, Evolutionary computing on consumer graphics hardware, IEEE Intelligent Systems 22 (2007) 69–78.
[27] L. Jian, C. Wang, Y. Liu, S. Liang, W. Yi, Y. Shi, Parallel data mining techniques on Graphics Processing Unit with Compute Unified Device Architecture (CUDA), The Journal of Supercomputing (2011) 1–26.
[28] A. Cano, A. Zafra, S. Ventura, A Parallel Genetic Programming Algorithm for Classification, in: Lecture Notes in Computer Science, vol. 6678, 2011, pp. 172–181.
[29] M.A. Franco, N. Krasnogor, J. Bacardit, Speeding up the evaluation of evolutionary learning systems using GPGPUs, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2010, pp. 1039–1046.
[30] A. Cano, A. Zafra, S. Ventura, Speeding up the evaluation phase of GP classification algorithms on GPUs, Soft Computing 16 (2012) 187–202.
[31] D.A. Augusto, H. Barbosa, Accelerated parallel genetic programming tree evaluation with OpenCL, Journal of Parallel and Distributed Computing 73 (2013) 86–100.
[32] P.L. Lanzi, Learning classifier systems: then and now, Evolutionary Intelligence 1 (2008) 63–82.
[33] M.V. Butz, T. Kovacs, P.L. Lanzi, S.W. Wilson, Toward a theory of generalization and learning in XCS, IEEE Transactions on Evolutionary Computation 8 (2004) 28–46.
[34] E. Bernadó-Mansilla, J.M. Garrell-Guiu, Accuracy-based learning classifier systems: models, analysis and applications to classification tasks, Evolutionary Computation 11 (2003) 209–238.
[35] J. Casillas, B. Carse, L. Bull, Fuzzy-XCS: a Michigan genetic fuzzy system, IEEE Transactions on Fuzzy Systems 15 (2007) 536–550.
[36] A. Orriols-Puig, J. Casillas, E. Bernadó-Mansilla, Fuzzy-UCS: a Michigan-style learning fuzzy-classifier system for supervised learning, IEEE Transactions on Evolutionary Computation 13 (2009) 260–283.
[37] A. González, R. Pérez, SLAVE: a genetic learning system based on an iterative approach, IEEE Transactions on Fuzzy Systems 7 (1999) 176–191.
[38] G. Venturini, SIA: a supervised inductive algorithm with genetic search for learning attributes based concepts, in: European Conference on Machine Learning, 1993, pp. 280–296.
[39] J.S. Aguilar-Ruiz, R. Giraldez, J.C. Riquelme, Natural encoding for evolutionary supervised learning, IEEE Transactions on Evolutionary Computation 11 (2007) 466–479.
[40] F.J. Berlanga, A.J.R. Rivas, M.J. del Jesús, F. Herrera, GP-COACH: Genetic Programming-based learning of Compact and Accurate fuzzy rule-based classification systems for high-dimensional problems, Information Sciences 180 (2010) 1183–1200.
[41] D.P. Greene, S.F. Smith, Competition-based induction of decision models from examples, Machine Learning 13 (1994) 229–257.
[42] R. Axelrod, The Complexity of Cooperation: Agent-based Models of Competition and Collaboration, Princeton University Press, 1997.
[43] A. Palacios, L. Sánchez, I. Couso, Extending a simple genetic cooperative-competitive learning fuzzy classifier to low quality datasets, Evolutionary Intelligence 2 (2009) 73–84.
[44] T. Kovacs, Genetics-based machine learning, in: G. Rozenberg, T. Bäck, J. Kok (Eds.), Handbook of Natural Computing: Theory, Experiments, and Applications, Springer Verlag, 2011.
[45] W.B. Langdon, Fitness Causes Bloat in Variable Size Representations, Technical Report CSRP-97-14, University of Birmingham, School of Computer Science, 1997.
[46] J. Rissanen, Minimum description length principle, in: Encyclopedia of Machine Learning, 2010, pp. 666–668.
[47] J. Bacardit, Pittsburgh Genetics-Based Machine Learning in the Data Mining Era: Representations, Generalization, and Run-time, Ph.D. Thesis, Ramon Llull University, Barcelona, Spain, 2004.
[48] NVIDIA Corporation, NVIDIA CUDA Programming and Best Practices Guide, ⟨http://www.nvidia.com/cuda⟩, 2012.
[49] M. Garland, S. Le Grand, J. Nickolls, J. Anderson, J. Hardwick, S. Morton, E. Phillips, Y. Zhang, V. Volkov, Parallel computing experiences with CUDA, IEEE Micro 28 (2008) 13–27.
[50] R.L. Rivest, Learning decision lists, Machine Learning 2 (1987) 229–246.
[51] J. Quinlan, C4.5: Programs for Machine Learning, , 1993.
[52] J.E. Hopcroft, R. Motwani, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, Context-Free Grammars, Addison-Wesley, 2006 (Chapter 4).
[53] W.B. Langdon, W. Banzhaf, A SIMD Interpreter for Genetic Programming on GPU Graphics Cards, in: Lecture Notes in Computer Science, vol. 4971, 2008, pp. 73–85.
[54] W.B. Langdon, A many threaded cuda interpreter for genetic programming, Lecture Notes in Computer Science 6021 (2010) 146–158.
[55] G. Rivera, C.W. Tseng, Data transformations for eliminating conflict misses, ACM SIGPLAN 33 (1998) 38–49.
[56] D.J. Newman, A. Asuncion, UCI machine learning repository, 2007.
[57] J. Alcalá-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. García, L. Sánchez, F. Herrera, KEEL data-mining software tool: data set repository, integration of algorithms and experimental analysis framework, Journal of Multiple-Valued Logic and Soft Computing 17 (2011) 255–287.
[58] W.-C. Feng, X. Feng, R. Ge, Green supercomputing comes of age, IT Professional 10 (2008) 17–23.



Alberto Cano was born in Cordoba, Spain, in 1987. He is currently a Ph.D. student with an M.Sc. in Computer Science. His research is performed as a member of the Knowledge Discovery and Intelligent Systems Research Laboratory, and is focussed on general purpose GPU systems, parallel computing, soft computing, machine learning, data mining and its applications.

Amelia Zafra is an Associate Professor of Computer Sciences and Artificial Intelligence at the University of Cordoba. She received the B.S. degree and Ph.D. degrees in Computer Science from the University of Granada, Granada, Spain, in 2005 and 2009, respectively. Her research is performed as a member of the Knowledge Discovery and Intelligent Systems Research Laboratory, and is focussed on soft computing, machine learning, and data mining and its applications.

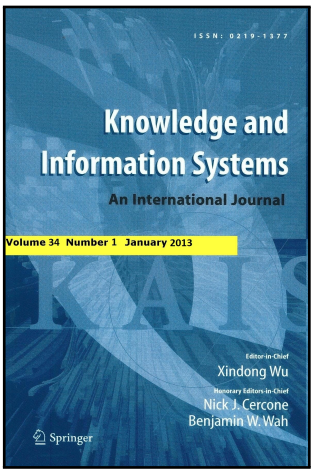
Sebastián Ventura received the B.Sc. and Ph.D. degrees in sciences from the University of Córdoba, Córdoba, Spain, in 1989 and 1996, respectively. He is currently an Associate Professor in the Department of Computer Science and Numerical Analysis, University of Córdoba, where he heads the Knowledge Discovery and Intelligent Systems Research Laboratory. He has authored or coauthored more than 120 international publications, 35 of them published in international journals. He has also worked on eleven research projects (being the Coordinator of three of them) that were supported by the Spanish and Andalusian governments and the European Union. His research interests include soft computing, machine learning, data mining, and its applications. He is a Senior Member of the IEEE Computer Society, the IEEE Computational Intelligence Society, the IEEE Systems, Man, and Cybernetics Society, and the Association of Computing Machinery.

TITLE:

Speeding up Multiple Instance Learning Classification Rules on GPUs

AUTHORS:

A. Cano, A. Zafra, and S. Ventura



Knowledge and Information Systems, *Submitted, 2013*

RANKING:

Impact factor (JCR 2011): 2.225

Knowledge area:

Computer Science, Artificial Intelligence: 21/111

Computer Science, Information Systems: 18/135

Noname manuscript No.
(will be inserted by the editor)

Speeding up Multiple Instance Learning Classification Rules on GPUs

Alberto Cano · Amelia Zafra · Sebastián Ventura

Received: Apr XX, 2013 / Revised: YY XX, 2013 / Accepted: YY XX, 2013

Abstract Multiple instance learning is a challenging task in supervised learning and data mining. However, algorithm performance becomes slow when learning from large-scale and high-dimensional data sets. Algorithms from a considerable number of areas of knowledge are reducing their computing time using graphics processing units (GPUs) and the compute unified device architecture (CUDA) platform. Similarly, application of this technology to the multiple instance learning problem could prove to be highly advantageous. This paper presents an implementation of the G3P-MI algorithm in CUDA for solving multiple instance problems using classification rules. The GPU model proposed is distributable to multiple GPU devices, seeking for its scalability across large-scale and high-dimensional data sets. The proposal is evaluated and compared to the multi-threaded CPU algorithm over a series of real-world and artificial multi-instance data sets. Experimental results report that the computation time can be significantly reduced and its scalability improved. Specifically, an speedup up to 450× can be achieved over the multi-threaded CPU algorithm when using four GPUs, and the rules interpreter achieves great efficiency and runs over 108 billion Genetic Programming operations per second (GPops/s).

Keywords Multi-instance learning · classification · parallel computing · GPU

1 Introduction

Multiple instance learning (MIL) is a generalization of traditional supervised learning that has received a significant amount of attention over the last few years [8,13,17,41]. Unlike traditional learning, in multi-instance learning, an example is called a bag and it represents a set of non-repeated instances. The bag is associated with a single class label, although the labels of the instances are unknown. The way in which bags are labelled depends on the multi-instance hypothesis or assumption. The standard hypothesis, introduced by Dietterich et al. [13], assumes a bag to be positive if it contains at least one positive instance. More recently, other generalized multi-instance models have been formalized [17,41]. According to its own definition, MIL is highly suitable for parallelization due to the fact that learners receive a set of bags composed by instances rather than a set of instances directly. With this data structure, the different bags and instances could be evaluated in a parallel way to reduce the execution time of the algorithms.

Alberto Cano · Amelia Zafra · Sebastián Ventura
Department of Computer Science and Numerical Analysis, University of Cordoba
Rabanales Campus, 14071 Cordoba, Spain
Tel.: +34957212218
Fax: +34957218630
E-mail: {acano, azafra, sventura}@uco.es

Multi-instance learning has received much attention in the machine learning community because many real-world problems can be represented as multi-instance problems. It has been applied successfully to several problems such as text categorization [2], content-based image retrieval [27,53] and image annotation [40], drug activity prediction [37,56], web index page recommendation [52,57], video concept detection [22,25], semantic video retrieval [9] and predicting student performance [51,54].

Similarly, there are many machine learning methods available to solve these problems, such as multi-instance lazy learning algorithms [46], multi-instance tree learners [10], multi-instance rule inducers [11], multi-instance bayesian approaches [29], multi-instance neural networks [7], multi-instance kernel methods [24,40], Markov chain-based [49], multi-instance ensembles [48,56], and evolutionary algorithms [53, 55]. However, most of the MIL algorithms are very slow and cannot be applied to large data sets. The main problem is that the MIL problem is more complex than the traditional supervised learning problem. Therefore, this type of algorithms over MIL still causes an increase in computation time, especially for high-dimensional and large-scale input data. Since real applications often work under time constraints, it is convenient to adapt the learning process in order to complete it in a reasonable time. There are several works that try to optimize the computation time of some algorithms in MIL [5,18,36,39,42], showing the growing interest in this area.

In this context, graphics processing units (GPUs) have demonstrated efficient performance in traditional rule-based classification [6,19]. Moreover, GPU systems have been widely used in many other evolutionary algorithms and data mining techniques in recent years [16,26,33,35]. However, we have not been able to find any proposal based on GPU implementation of MIL. Therefore, we think that a GPU-based model for MIL could be an interesting alternative to reduce the excessive execution time inherent to this learning framework. Especially, we seek the scalability of MIL algorithms to large-scale and high-dimensional problems in which larger population sizes should be employed to achieve accurate results.

This paper presents a GPU-based parallel implementation of the G3P-MI [53] algorithm using CUDA, which allows the acceleration of the learning process. G3P-MI is an evolutionary algorithm based on classification rules which has proven itself to be a suitable model because of its flexibility, rapid adaptation, excellent quality of representation, and competitive results. However, its performance becomes slow when learning form large-scale and high-dimensional data sets. The proposal presented here aims to be a general purpose model for evaluating multi-instance classification rules on GPUs, which is independent to algorithm behaviour and applicable to any of the multi-instance hypotheses. The proposal addresses the computational time problem of evolutionary rule-based algorithms when evaluating the rules on multi-instance data sets, especially when the number of rules is high, or when the dimensionality and complexity of the data increase. The design of the model comprises three different GPU kernels which implement the functionality to evaluate the classification rules over the examples in the data set. The interpreter of the rules is carefully designed to maximize efficiency, performance, and scalability. The GPU model is distributable to multiple GPU devices, providing transparent scalability to multiple GPUs. Moreover, scalability to multiple GPUs allow to extend the application of MIL algorithms across large-scale and high-dimensional data sets.

The proposal is evaluated over a series of real-world and artificial multi-instance data sets and its execution times are compared with the multi-threaded CPU ones, in order to analyze its efficiency and scalability to larger data sets having different population sizes. Experimental results show the great performance and efficiency of the model, achieving an speedup of up to 450× when compared to the multi-threaded CPU implementation. The efficient rules interpreter demonstrates the ability to run up to 108 billion Genetic Programming operations per second (GPops/s) whereas the multi-threaded CPU interpreter runs up to 98 million GPops/s. Moreover, it has shown great scalability to two and four GPUs. This means that more complex multi-instance problems with larger number of examples can be addressed within reasonable time, which was not previously possible without GPU parallelization.

This paper is organized as follows. The next section defines the multi-instance classification problem and presents the rule-based approach to the multi-instance problem. Section 3 introduces the CUDA programming model. Section 4 presents a computational analysis of the multi-instance algorithm. Section 5 presents the GPU implementation of the multi-instance model. Section 6 describes the experimental study, whose results are discussed in Section 7. Finally, Section 8 presents the conclusions.

2 Multi-instance classification

This section defines the multi-instance learning problem and presents the basis of multi-instance rule-based models.

2.1 Problem definition

Standard classification consists in predicting the class membership of uncategorized examples, whose label is not known, using the properties of the examples. An example (instance) is represented using a feature vector $\bar{x}$, which is associated with a class label C. Traditional classification models induct a prediction function $f(\bar{x}) \rightarrow C$.

On the other hand, multi-instance classification examples are called bags, and represent a set of instances. The class is associated with the whole bag although the instances are not explicitly associated with any particular class. Therefore, multi-instance models induct a prediction function $f(bag) \rightarrow C$ where the bag is a set of instances $\{\bar{x_1}, \bar{x_2}, ..., \bar{x_n}\}$.

The way in which a bag is classified as positive or negative depends on the multi-instance hypotheses. In the early years of multi-instance learning research, all multi-instance classification work was based on the standard or Dietterich hypothesis [13]. The standard hypothesis assumes that if the result observed is positive, then at least one of the instances from the bag must have produced that positive result. However, if the result observed is negative, then none of the instances from the bag could have produced a positive result. Therefore, a bag is positive if and only if at least one of its instances is positive. This can be modelled by introducing a second function $g(bag, j)$ that takes a single variant instance j and produces a result. The externally observed result $f(bag)$ can be defined as follows:

$$f(bag) = \begin{Bmatrix} 1 \; if \; \exists \; j & | & g(bag, j) = 1 \\ 0 & & otherwise \end{Bmatrix} \tag{1}$$

More recently, generalized multi-instance models have been formulated where a bag is qualified to be positive if the instances in the bag satisfy more sophisticated constraints than simply having at least one positive instance. Firstly, Weidmann et al. [47] defined three kinds of generalized multi-instance problems, based on employing different assumptions of how the classification of instances determines the bag label. These definitions are presence-based, threshold-based, and count-based.

- Presence-based is defined in terms of the presence of at least one instance of each concept in a bag (the standard hypothesis is a special case of this assumption which considers just one underlying concept).
- Threshold-based requires a certain number of instances of each concept in a bag.
- Count-based requires a maximum and a minimum number of instances of a certain concept in a bag.

Independently, Scott et al. [43] defined another generalized multi-instance learning model in which a bag label is not based on the proximity of one single instance to one single target point. In this model, a bag is positive if and only if it contains a collection of instances, each near one of a set of target points.

Regardless of the multi-instance hypothesis, MIL algorithms generally demand significant resources, taking excessive computing time when data size increases. The high computational cost of MIL algorithms prevent their application in large-scale and high-dimensional real world problems within reasonable time, such as text categorization, image annotation, or web index page recommendation on large databases. However, the multiple instances learning process using data structures representation with bags and instances is inherently parallel. Therefore, it is essential to take advantage of the parallel capabilities of modern hardware to speed up the learning process and reduce computing time.

2.2 Rule-based models

Rule-based models are white box classification techniques which add comprehensibility and clarity to the knowledge discovery process, expressing information in the form of IF-THEN classification rules. The comprehensibility of the knowledge discovered has been an area of growing interest and this comprehensibility is currently considered to be just as important as obtaining high predictive accuracy [3,30].

The evaluation of the classification rules over a data set requires the interpreting of the conditions expressed in the antecedent of the rule and checking whether the data examples satisfy them. The rule interpreter has usually been implemented in a stack-based manner, i.e., operands are pushed onto the stack, and when an operation is performed, its operands are removed from the stack and its result pushed back on. Therefore, its performance and the amount of time taken up depend on the number of examples, the number of rules, and their complexity (i.e. the number of conditions of the rule to evaluate).

Evolutionary Algorithms [20,21], and specifically, Genetic Programming [14], have been successfully employed for obtaining classification rules over a wide range of problems, including multi-instance classification. The performance impact of rule evaluation is increased in these algorithms since each generation of the algorithm, a population of solutions (rules or set of rules) must be evaluated according to a fitness function. Thus, the algorithms perform slowly and their scalability is severely limited to the problem dimensionality and complexity. The use of GPUs for the evaluation of individuals in an evolutionary computation environment has demonstrated high performance and efficient results in multiple heuristics and tasks in many studies. These studies include using genetic programming for stock trading [38], classification rules [6,19], differential evolution [12,45], image clustering [31], or optimization problems [15]. However, to the best of our knowledge there are no GPU-based implementations of multi-instance classification rules algorithms to date.

G3P-MI [53] is a Grammar-Guided Genetic Programming (G3P) [28] algorithm for multi-instance learning. It is based on the presence-based hypothesis and has demonstrated accurate classification results when applied in varied application domains, as well as better performance than many other multi-instance classification techniques. However, the large population required and the highly complex rules generated prevent the algorithm running as fast as desired, especially across large data sets. Therefore, a GPU-based parallel implementation of the algorithm which reduces execution time significantly makes for a very appealing and valuable proposal. Moreover, the GPU-based model to speed up the learning process is applicable to any other multi-instance rule-based method with any of the multi-instance hypotheses.

3 CUDA programming model

Computer unified device architecture (CUDA) [1] is a parallel computing architecture developed by NVIDIA that allows programmers to take advantage of the parallel computing capacity of NVIDIA GPUs in a general purpose manner. The CUDA programming model executes kernels as batches of parallel threads. These kernels comprise thousands or even millions of lightweight GPU threads per each kernel invocation.

CUDA's threads are organized into threads blocks in the form of a grid. Thread blocks are executed by streaming multiprocessors. A stream multiprocessor can perform zero overhead scheduling to interleave warps (a warp is a group of threads that execute together) and hide the overhead of long-latency arithmetic and memory operations. GPU's architecture was rearranged from SIMD (Single Instruction, Multiple Data) to MIMD (Multiple Instruction, Multiple Data), which runs independent separate program codes. Thus, up to 16 kernels can be executed concurrently as long as there are available multiprocessors. Moreover, asynchronous data transfers can be performed concurrently with kernel executions. These two features allow a speedup in execution compared to the sequential kernel pipeline and synchronous data transfers from previous GPU architectures.

There are four different specialized memory spaces with different access times, lifetimes and output limitations.

- Global memory: is a large long-latency memory that exists physically as an off-chip dynamic device memory. Threads can read and write global memory to share data and must write the kernel's output to be readable after the kernel terminates. However, a better way to share data and improve performance is to take advantage of shared memory.
- Shared memory: is a small low-latency memory that exists physically as on-chip registers. Its contents are only maintained during thread block execution and are discarded when the thread block completes. Kernels which read or write a known range of global memory with spatial or temporal locality can employ shared memory as a software-managed cache. Such caching potentially reduces global memory bandwidth demands and improves overall performance.

- Local memory: each thread also has its own local memory space as registers, so the number of registers a thread uses determines the number of concurrent threads executed in the multiprocesor, which is called multiprocessor occupancy. To avoid wasting hundreds of cycles while a thread waits for a long-latency global-memory load or store to complete, a common technique is to execute batches of global accesses, one per thread, exploiting the hardware's warp scheduling to overlap the threads' access latencies.
- Constant memory: is specialized for situations in which many threads will read the same data simultaneously. This type of memory stores data written by the host thread, is accessed constantly and does not change during the execution of the kernel. A value read from the constant cache is broadcast to all threads in a warp, effectively serving all loads from memory with a single-cache access. This enables a fast, single-ported cache to feed multiple simultaneous memory accesses.

There are some recommendations for improving the performance on the GPU [23]. Memory accesses must be coalesced as with accesses to global memory. Global memory resides in device memory and is accessed via 32, 64, or 128-byte segment memory transactions. It is recommended to perform fewer but larger memory transactions. When a warp executes an instruction which accesses global memory, it coalesces the memory accesses of the threads within the warp into one or more of these memory transactions depending on the size of the word accessed by each thread and the distribution of memory addresses across the threads. In general, the more transactions necessary, the more unused words are transferred in addition to the words accessed by the threads, reducing the instruction throughput accordingly.

To maximize global memory throughput, it is therefore important to maximize coalescing by following optimal access patterns, using data types which meet the size and alignment requirement or padding data. For these accesses to be fully coalesced, both the width of the thread block and the width of the array must be a multiple of the warp size.

4 Computational analysis of the multi-instance algorithm

This section analyzes the computational cost of the multi-instance algorithm in order to identify computational bottlenecks and to propose parallelization strategies to overcome these issues.

G3P-MI is a Genetic Programming algorithm that consists of the traditional stages of an evolutionary-based algorithm: initialization, selection, genetic operators, evaluation, and replacement. The initialization process creates a randomly-initialized population of rules by means of a context-free grammar that conducts the generation of the rule syntaxes. The selection process selects the parent candidates from the population, on which the genetic operators (crossover and mutation) will be applied. The evaluation process checks the fitness of the new rules (offspring) resulted from the application of the genetic operators. The replacement process selects the best rules from the parent population and the offspring, keeping the population size constant and leading the population to better fitness landscapes. This process is iteratively repeated along a given number of generations, after which the best rules from the population are selected to build the classifier. However, it is well-known and it has been demonstrated in several studies [6, 19, 35] that the evaluation phase is the one that demands most of the computational cost of the algorithm, requiring from 90% to 99% of the execution time, which increases as the data set becomes bigger. Thereby, significant effort should be focus on speeding up this stage. Thus, we analyze the computational cost of the evaluation function.

The evaluation process consists on predicting the class membership of the examples of the data set and to compare the predicted class with the actual class to measure the prediction error. Specifically, it is measured the number of true positives (t_p), true negatives (t_n), false positives (f_p) and false negatives (f_n). These values are used to build the confusion matrix, from which any classification performance metric is obtained (sensitivity, specificity, accuracy, etc). Each individual from the population represents a classification rule which comprises several attribute–value conditions combined using logical operators. The evaluation process is usually implemented using two nested loops. Thereby, the algorithmic complexity of the fitness function is *O(population size* $\times$ *number of examples)*. This makes the algorithm to perform slow when the population size and the number of examples of the data set increase. The multiple instance problem representation particularizes an example as a bag containing a set of instances. Therefore,

the computational complexity increases as the examples (bags) contain higher number of instances. The pseudo-code of the fitness function is shown in Algorithm 1, particularized for the Dietterich hypothesis (a single positive instance makes the whole bag prediction as positive), and Algorithm 2, particularized for the generalized hypothesis. It is noted that for the Dietterich hypothesis, the inner loop must be stopped as soon as one instance is covered, whereas for the generalized hypothesis it is necessary to evaluate all the instances of the bag. Therefore, performance on data sets having large bag sizes is penalized.

Algorithm 1 Evaluation: Dietterich hypothesis

Input: population_size, number_examples

```
1: for each individual within the population do
2:    tp ← 0, fp ← 0, tn ← 0, fn ← 0
3:    for each example from the dataset do
4:       for each instance from the example's bag do
5:          if individual's rule covers actual instance then
6:             if the bag is labelled as positive then
7:                tp++
8:             else
9:                fp++
10:            end if
11:            continue with the next example
12:         end if
13:      end for
14:      // None of the instances were covered
15:      if the bag is labelled as positive then
16:         fn++
17:      else
18:         tn++
19:      end if
20:   end for
21:   fitnessValue ← fitnessMetric(tp,tn,fp,fn)
22: end for
```

Algorithm 2 Evaluation: Generalized hypothesis

Input: population_size, number_examples

```
1: for each individual within the population do
2:    tp ← 0, fp ← 0, tn ← 0, fn ← 0
3:    for each example from the dataset do
4:       coverCount ← 0
5:       for each instance from the example's bag do
6:          if individual's rule covers actual instance then
7:             coverCount++
8:          end if
9:       end for
10:      if coverCount ≥ minimumCount && coverCount ≤ maximumCount then
11:         if the bag is labelled as positive then
12:            tp++
13:         else
14:            fp++
15:         end if
16:      else
17:         if the bag is labelled as positive then
18:            fn++
19:         else
20:            tn++
21:         end if
22:      end if
23:   end for
24:   fitnessValue ← fitnessMetric(tp,tn,fp,fn)
25: end for
```

The evaluation process is noted to have three main stages: rule-instance coverage, bag class prediction, and confusion matrix building by means of counting the number of t_p, t_n, f_p, and f_n values. As seen, the complexity of the fitness function lies in the population size (number of rules) and the data set size (number of bags and instances). Evaluating them along high number of generations is the reason for the high computational cost of MIL algorithms, and motivates their parallelization on GPUs. Fortunately, the evaluation process of each rule from the population is an individual computation problem that can be solved independently (population parallel approach). Moreover, the coverage of a rule against all the examples of the dataset is also an independent task that can be performed concurrently (data parallel approach). Therefore, multiple parallelization strategies can be employed to take advantage of the parallel capabilities of the evaluation process.

The parallelization of the evaluation function in the CPU is straightforward by means of a population-parallel approach. The algorithm can take advantage of multi-core CPUs and create as many CPU threads as number of cores, evaluating independently and concurrently each of the individuals of the population. However, the hardware industry provide today 4-cores desktop processors, which limit the parallelism of this approach. Nevertheless, fortunate users having a CPU cluster or a grid environment may exploit this parallel approach more significantly, having multiple nodes connected through a local area network. Furthermore, the data-parallel approach may be also employed for distributing rule evaluation among multiple hosts. However, this complicates the evaluator code by means of including more complex message transfer between the hosts of the network, which eventually reduces the absolute efficiency of the process. On the other hand, GPUs have demonstrated to achieve high performance on similar computational tasks and avoid many of the problems of distributed computing systems. Therefore, the next section presents the GPU model proposed to address this highly parallelizable evaluation process.

5 Multi-instance rules evaluation on GPUs

This section presents the GPU-based evaluation model of the multi-instance rules. According to the computational analysis presented in the previous section, the evaluation of the rules is divided into three steps, which are implemented through three GPU kernels. The first one, the coverage kernel, checks the coverage of the rules over the instances within the bags. The second one, the hypothesis kernel, implements the bag class prediction regarding to the bag instances which satisfy the concepts of the rule. The three hypotheses are considered to provide generalization to any multi-instance approach. Finally, the fitness computation kernel builds the confusion matrix from the predicted bag class and actual bag class values, and computes the fitness (quality metrics) of the rules. The data and computation flow is overviewed in Fig. 1, whereas the GPU evaluation model is shown in Fig. 2 and it is described in the following sections. Data memory transactions between CPU and GPU memories are shown dashed and light gray, whereas GPU kernels computation are shown dotted and black gray.

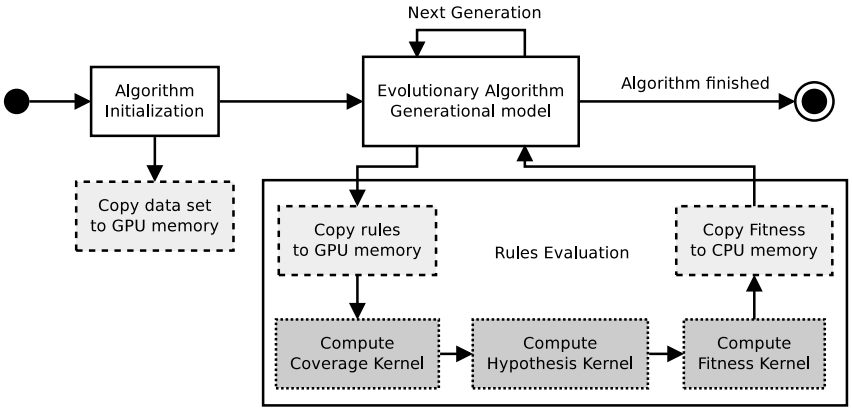


Fig. 1 Data & computation flow overview

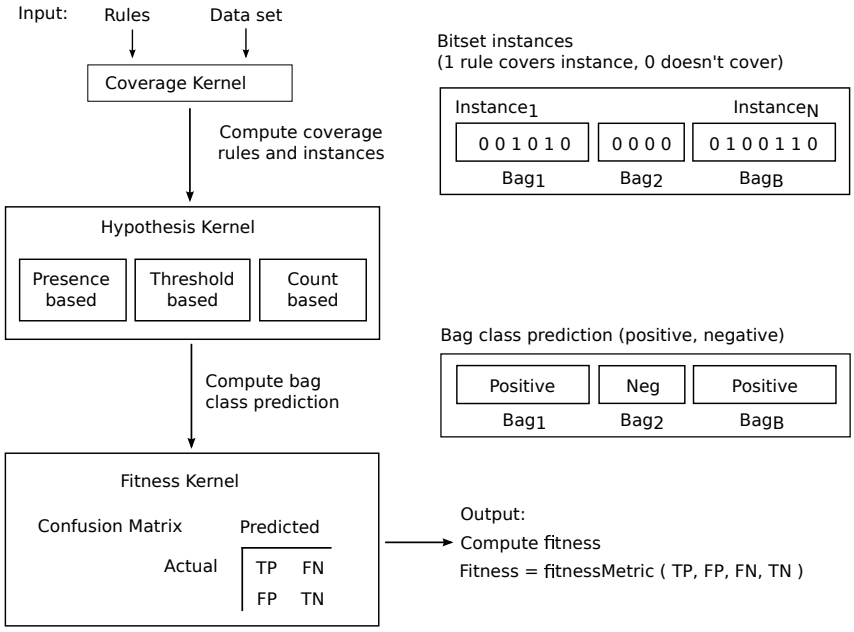


Fig. 2 GPU evaluation model using three kernels

The data set values are stored in the GPU global memory using an array whose length is (number of attributes × number of instances), and it is allocated transposed to facilitate memory coalescing when GPU threads request instances values. Data set values are copied at the beginning of the algorithm's execution and they can be transferred asynchronously while the population is being initialized. This means that no delay is introduced to the algorithm execution due to data set transfer to GPU memory. Rules and fitness are also stored in global memory, but they require synchronous transfers, meaning that the evaluation process cannot begin until the rules are copied to the GPU memory, and the evolutionary process cannot continue until the fitness values are copied back on the host memory. Fortunately, the data size for both elements is very small and memory transfers complete within few nanoseconds, as measured by the NVIDIA Visual Profiler tool. Both data structures facilitate to maximize the coalescing and minimize the effective number of memory transactions.

5.1 Coverage kernel

The coverage kernel interprets the rules and checks whether the instances of the data set satisfy the conditions of the rules. These rules are very clear and comprehensible since they provide a natural extension for knowledge representation. For example:

$$\text{IF } [(At_1 \geq V_1 \text{ AND } At_2 < V_2) \text{ OR } At_3 > V_3] \text{ THEN } Class_1$$

where At_1, At_2, and At_3 are attributes of the data set, and V_1, V_2, and V_3 are specific values in the range of each attribute.

Rules are easily represented in prefix notation on a computer, which is a form of notation that has been widely used in many applications and computer languages such as Lisp. Prefix notation is popular with stack-based operations due to its ability to distinguish the order of operations without the need for parentheses. The example rule from above is represented in prefix notation as:

$$\text{IF } [\text{OR} > At_3\ V_3 \text{ AND} \geq At_1\ V_1 < At_2\ V_2] \text{ THEN } Class_1$$

The interpreter we first used for traditional classification in [6] employed these kind of expressions. The implementation of a prefix or postfix interpreter require the usage of a stack and every operand/operator

perform push/pop operations. Thus, the number of reads and writes over the stack increases with the length of the rule. However, seeking for an efficient implementation on GPUs is not straightforward. GPUs are not especially designed for stack-based memory operations. Therefore, we propose to employ an intermediate rule representation to take advantage of the flexibility of the stack-based operations and minimize as well the number of operations over the stack. The conditions are internally represented in prefix notation whereas the rule set is written in postfix. The antecedent of the rule is rewritten as:

$$< At_2\ V_2 \geq At_1\ V_1 \text{ AND} > At_3\ V_3 \text{ OR}$$

Fig. 3 shows the performance differences of the prefix interpreter and the intermediate interpreter as the interpreter finds tokens and computes given actions. The intermediate interpreter reads the rule from the left to the right. It first finds the $<$ operand, which is known to have two operands, so it knows it has to compute $< At_2\ V_2$ and pushes only the result into the stack. The rule pointer must be placed to the right of the number of operands. Similarly, it operates with the other operations. Finally, the OR operator pops the two operands and returns their OR operation.

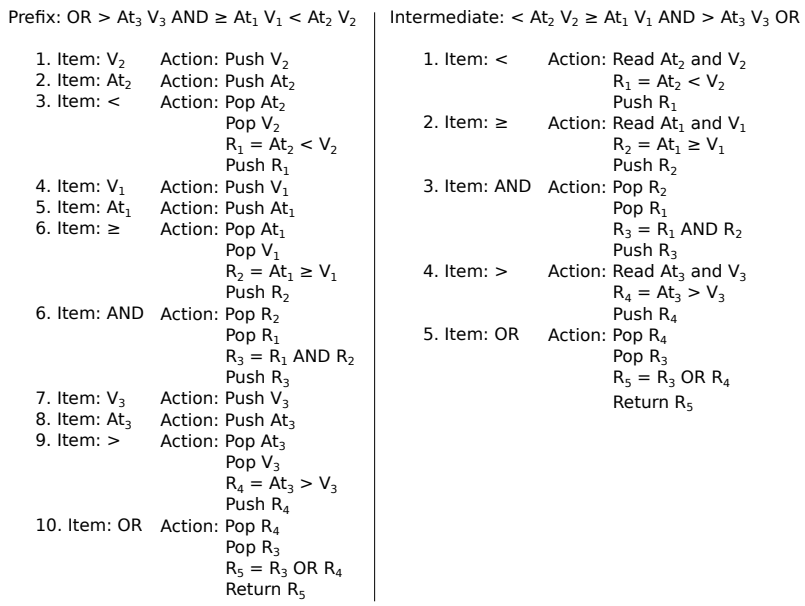


Fig. 3 Prefix and intermediate rule interpreter

This representation allows the interpreter performance to speed up, and minimizes the number of push and pop operations on the stack. For instance, the traditional interpreter requires 10 push and 10 pop operations whereas the proposed representation requires only 4 push and 4 pop operations, which reduces memory accesses and increases interpreter performance. The CPU interpreter loops the interpretation of every rule within the algorithm's population on another loop for every single instance within the data set, which notes its high computational complexity.

Fortunately, the evaluation of every single rule over every single instance can be parallelized on the GPU using a two dimensional matrix of threads. The first kernel checks the coverage of the rules over the instances of the data set, and stores the results of the coverage matching into a bitset (array of bits). Each thread is responsible for the coverage of a single rule over a single instance. Threads are grouped into a 2D grid of thread blocks, whose size depends on the number of rules (width) and instances (height). Eventually, a warp (group of threads that are evaluated concurrently at a given time in the multiprocessor) represents the evaluation of a given rule over multiple data, following a SIMD model. Thereby, there is no divergence in the instruction path of the kernel, which is one of the known main reasons for decreasing performance and we avoid this issue. Moreover, reading from multiple data is guaranteed to be coalesced

```
__global__ void coverageKernel(float* rules, unsigned char* bitset, int numberInstances)
{
  int instance = blockDim.y * blockIdx.y + threadIdx.y;

  bitset[blockIdx.x * numberInstances + instance] = covers(&rules[blockIdx.x], instance);
}

__device__ unsigned char covers(float* rule, int instance)
{
  ...
  for(int ptr = 0; ptr < ruleLength; )
  {
    switch(rule[ptr]) {
    ...
    case GREATER:
      attribute = expr[prt+1];
      op1 = instancesData[instance + numberInstances * attribute];
      op2 = expr[prt+2];
      if (op1 > op2)       push(1, stack);
      else                 push(0, stack);
      ptr += 3;
      break;
    ...
    case AND:
      op1 = pop(stack);
      op2 = pop(stack);
      if (op1 * op2 == 1)  push(1, stack);
      else                 push(0, stack);
      ptr++;
      break;
    ...
    }
  }

  return (unsigned char) pop(stack);
}
```

Fig. 4 Coverage kernel and rules interpreter

since threads are responsible of handling the adjacent memory addresses. Bitset storage of coverage result is also coalesced by addressing adjacent memory addresses. Coalescing avoids memory addressing conflicts and permits to improve efficiency of memory transactions. The code for the coverage kernel and the rule interpreter using the intermediate representation is shown in Fig. 4.

The kernel parameters configuration is essential to maximize occupancy and performance. The number of vertical blocks depends on the number of instances and the number of threads per block, which is recommended to be a multiple of the warp size, usually being 128, 256 or 512 threads per block. This number is important as it concerns the scalability of the model in future devices. NVIDIA recommends running at least twice as many thread blocks as the number of multiprocessors in the GPU. 256 threads per block are used since it provides both maximum occupancy and more active threads blocks per multiprocessor to hide latency arising from register dependencies and, therefore, a wider range of possibilities is given to the dispatcher to issue concurrent blocks to the execution units. Moreover, it provides better scalability for future GPU devices, with more multiprocessors, and capable of handling more active blocks. Details about the setup settings of the kernels are shown in Section 6.

5.2 Hypothesis kernel

The hypothesis kernel performs the class prediction for the bags, using the coverage results from their instances and the coverage kernel. Three functions implement the different hypothesis kernels concerning their requirements based on the presence-based, threshold-based, or count-based multiple instance learning hypotheses. The presence-based kernel only requires that one instance is active in order to predict the bag to be positive (Dietterich hypothesis) as previously shown in Algorithm 1. The threshold-based kernel computes first the count of the number of instances from the bag that satisfy the concepts of the rule. It predicts as positive if the count is higher than a minimum number of instances. The count-based

```
__global__ void presenceHypothesis(unsigned char* bitset, int* bagPrediction, int numInstances,
                                   int numBags, int* bag) {
  int instance = blockDim.y * blockIdx.y + threadIdx.y;

  if(bitset[blockIdx.x * numInstances + instance] == 1)
     bagPrediction[blockIdx.x * numBags + bag[instance]] = 1;
}

__global__ void thresholdHypothesis(unsigned char* bitset, int* bagPrediction, int numInstances
                                    int numBags, int minimumCount, int* firstInstanceBag,
                                    int* lastInstanceBag) {
  int bag = blockDim.y * blockIdx.y + threadIdx.y;
  int begin = firstInstanceBag[bag], end = lastInstanceBag[bag];
  int coverCount = 0;

  for(int i = begin; i < end; i++)
    if(bitset[blockIdx.x * numInstances + i] == 1)
       coverCount++;

  if(coverCount >= minimumCount)
     bagPrediction[blockIdx.x * numBags + bag] = 1;
}

__global__ void countHypothesis(unsigned char* bitset, int* bagPrediction, int numInstances,
                                int numBags, int minimumCount, int maximumCount,
                                int* firstInstanceBag, int* lastInstanceBag) {
  int bag = blockDim.y * blockIdx.y + threadIdx.y;
  int begin = firstInstanceBag[bag], end = lastInstanceBag[bag];
  int coverCount = 0;

  for(int i = begin; i < end; i++)
    if(bitset[blockIdx.x * numInstances + i] == 1)
       coverCount++;

  if(coverCount >= minimumCount && coverCount <= maximumCount)
     bagPrediction[blockIdx.x * numBags + bag] = 1;
}
```

Fig. 5 Hypothesis kernels

kernel counts the number of active instances as does the threshold-based, but its prediction depends on a minimum and maximum number of active instances. The threshold-based and count-based are known as the generalized hypothesis which was shown in Algorithm 2. Counting is a reduction operation and there are several parallel ways to count using the GPU. However, the average bag size of the multiple instances datasets available is relatively very small, usually having less than 10 instances per bag. Therefore, a parallel reduction scheme of the counting process would be excessive and inefficient for such small number of values. Thereby, it is more efficient to propose a parallel prediction model in which a thread is responsible of the prediction of a single bag, and the thread iterates among the instances from the bag to check the number of covered instances. The code for the different hypotheses is shown in Fig. 5.

It is very important to note that the Dietterich hypothesis only requires one instance to be active to predict the bag class as positive. Therefore, as soon as one instance is found to be active, there is no need to check the remaining instances in the bag. This allows a significantly amount of time to be saved, especially when the size of the bag increases. The CPU evaluator checks the instances from the bag from the first to the last, and it is not known a priori if the positive instance is going to happen earlier or later. Thus, the average number of instance checks required to find the positive instance is said to be the half of the bag size. On the other hand, the GPU model checks in parallel all the instances at a given time. Therefore, the GPU checking process always requires a single scan to find the positive instance.

Furthermore, the generalized hypotheses (threshold-based and count-based) require all of the instances to be processed in order to predict the bag class using the count values. This means that the runtime of the CPU process is inevitably increased as the number of instances increases, not existing the possibility of an earlier loop breaking. On the other hand, the GPU approach keeps its efficiency and is capable of counting the number of positives instances for all bags in a single call. Therefore, it is expected to have better efficiency and speedups on the generalized hypotheses.

5.3 Fitness kernel

The fitness computation kernel evaluates the fitness of the rule, i.e., its ability to perform accurate classifications. It computes the confusion matrix values: true positives (t_p), true negatives (t_n), false positives (f_p), and false negatives (f_n) to calculate some well-known indicators in classification such as sensitivity, specificity, precision, recall, F-Measure, etc. For instance, the goal of the G3P-MI [53] algorithm is to maximize both sensitivity and specificity, computing fitness as their product.

The confusion matrix values result from a reduction operation [50] of the bag class predictions and the actual values. The naive reduction operation is conceived as an iterative and sequential process. However, there are many ways to perform this operation in parallel. For instance, NVIDIA provides six different ways of optimizing parallel reduction in CUDA [1]. The one which performs best in this problem for reducing the confusion matrix values is the 2-level parallel reduction with shared memory storage, and it is illustrated in Fig. 6. The first level of the reduction reads the bag class prediction and compares with the actual bag class, producing one of the four possible values of the confusion matrix. To compute the fitness of a rule, 128 threads are used to read bag class predictions and match with the actual class labels. Therefore, each thread is responsible of processing (numberBags / 128) bags. The reason for using 128 threads is that threads in a thread block must share shared memory resources, which is limited in many architectures to 16 KB. Since a thread must employ 4 32-bit integers to count the confusion matrix values, it gives that a thread block requires 512 integers (2 KB) on shared memory. This limits the number of concurrent blocks to 8 per multiprocessor. Doubling the number of threads would only reduce the number of concurrent blocks capable of running in a multiprocessor, which reduces effective performance. Fortunately, this is parametrizable to adapt the number of threads to the hardware capacities of the GPU, adapting automatically the configuration parameters in order to be capable of achieving maximum performance.

The key point of the reduction operations is that accesses are fully coalesced to avoid memory addressing divergences as shown in the figure. Temporary results are stored in the shared memory of the GPU multiprocessors, which provides fast and low latency access. The second level of the reduction operation reads the partial sums of the counting and computes the final count of the confusion matrix values. Finally, the fitness metrics are calculated. This process is performed in parallel for all the rules of the population. The code for the fitness kernel, including the reduction operation, is shown in Fig. 7.

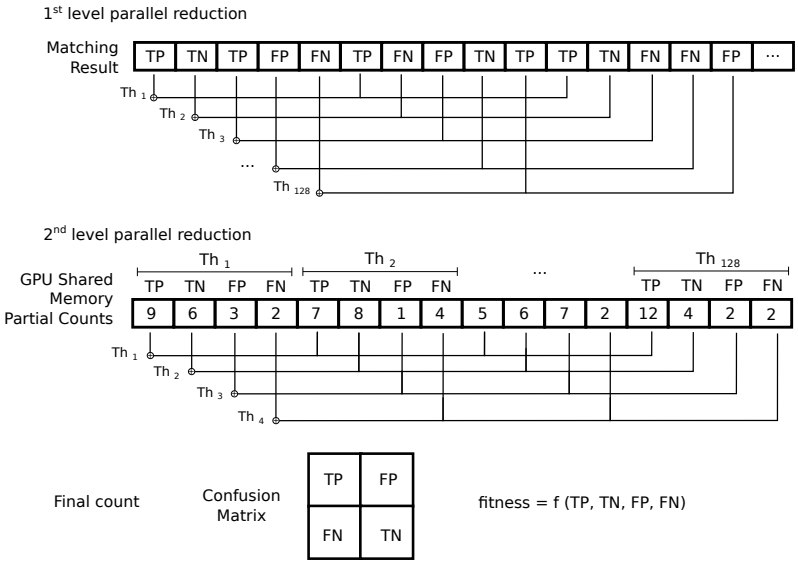


Fig. 6 Fitness computation using 2-level parallel reduction

```
__global__ void fitnessKernel(float* fitness, int* bagPrediction, int* bagClass, int numBags) {
  __shared__ int MC[512];
  int base = blockIdx.x * numBags + threadIdx.y;
  int top =  blockIdx.x * numBags + numBags - base;

  MC[threadIdx.y] = MC[threadIdx.y+128] = MC[threadIdx.y+256] = MC[threadIdx.y+384] = 0;

  // Performs the reduction of the confusion matrix values
  for(int i = 0; i < top; i+=128) {
    if(bagClass[threadIdx.y + i] == 1 && bagPrediction[base + i] == 1)
      MC[threadIdx.y*4]++;       // True positive
    else if(bagClass[threadIdx.y + i] == 0 && bagPrediction[base + i] == 1)
      MC[threadIdx.y*4 + 2]++;   // False positive
    else if(bagClass[threadIdx.y + i] == 1 && bagPrediction[base + i] == 0)
      MC[threadIdx.y*4 + 3]++;   // False Negative
    else
      MC[threadIdx.y*4 + 1]++;   // True negative
  }

  __syncthreads();

  if(threadIdx.y < 4) {
    for(int i = 4; i < 512; i+=4) {
      MC[0] += MC[threadIdx.y];     // Number of true positives
      MC[1] += MC[threadIdx.y+1];   // Number of true negatives
      MC[2] += MC[threadIdx.y+2];   // Number of false positives
      MC[3] += MC[threadIdx.y+3];   // Number of false negatives
    }

    if(threadIdx.y == 0) { // Perform final reduction and fitness computation
      float sensitivity, specificity;
      int tp = MC[0], tn = MC[1], fp = MC[2], fn = MC[3];

      sensitivity = tp / (float) (tp + fn);
      specificity = tn / (float) (tn + fp);

      // Fitness function
      fitness[blockIdx.x] = sensitivity * specificity;
    }
}
```

Fig. 7 Fitness kernel with 2-level reduction

6 Experimental Setup

This section presents the experimental study setup, the hardware configuration and the different experiments designed to evaluate the performance and efficiency of the GPU-based model.

6.1 Hardware configuration

The experiments were run on a machine equipped with an Intel Core i7 quad-core processor running at 3.0 GHz and 12 GB of DDR3-1600 host memory. The video cards used were two dual-GPU NVIDIA GTX 690 equipped with 4 GB of GDDR5 video RAM. Each GTX 690 video card had two GPUs with 1,536 CUDA cores. In total there were 4 GPUs and 6,144 CUDA cores at default clock speeds. The host operating system was GNU/Linux Ubuntu 12.10 64 bit along with CUDA runtime 5.0, NVIDIA drivers 310.40, and GCC compiler 4.6.3 (O3 optimization level).

6.2 Configuration settings

The G3P-MI algorithm is implemented in the JCLEC software [44] and its main parameters (collected from the author's proposal) are shown in Table 1.

On the other hand, the GPU kernels require two parameters which define the number of threads per block and the number of blocks per grid. The compiler and hardware threads scheduler will schedule

Table 1 G3P-MI parameter configuration

Parameter	Value
Population size	1000
Number of generations	100
Crossover probability	0.95
Mutation probability	0.3
Elitist probability	0.05
Parent selector	Binary Tournament
Maximum tree depth	50

Table 2 Threads per block and multiprocessor occupancy

Threads per block	64	128	256	512	1024
Active Threads per Multiprocessor	768	1536	2048	2048	2048
Active Warps per Multiprocessor	24	48	64	64	64
Active Thread Blocks per Multiprocessor	12	12	8	4	2
Occupancy of each Multiprocessor	38%	75%	100%	100%	100%

instructions as optimally as possible to avoid register memory conflicts. The CUDA GPU occupancy calculator provides information about the GPU multiprocessors occupancy regarding to the device capability, the number of threads per block, the registers used per thread and the shared memory employed by the blocks. One of the keys to achieve optimum performance is to keep the device multiprocessors as busy as possible. Configuration settings where work is poorly balanced across the multiprocessors will result in suboptimal performance.

Table 2 shows the GPU occupancy data for the different number of threads per block. Occupancy is the ratio of the number of active warps per multiprocessor to the maximum number of possible active warps. 64 and 128 threads per block reports a multiprocessor occupancy of 38% and 75% respectively, limited by the shared memory per multiprocessor. 256, 512, and 1024 threads per block report an occupancy of 100%. In both cases the number of active threads per multiprocessor is 2048, but they differ in the number of active thread blocks per multiprocessor. Best choice is 256 threads per block since it provides full occupancy and a trade-off between the number of active thread blocks per multiprocessor and the total number of thread blocks. This guarantees the scalability of the model design for future GPUs with larger number of CUDA multiprocessors.

6.3 Experiments

The experimental study comprises two experiments. Firstly, the performance and efficiency of the rules interpreter is evaluated. Secondly, the rule-based classification performance is evaluated over a series of data sets. Detailed information about the data sets is provided as additional material at link[1].

6.3.1 Rules interpreter performance

The efficiency of rules interpreters is often reported using the number of primitives interpreted by the system per second, similarly to Genetic Programming (GP) interpreters, which determine the number of GP operations per second (GPops/s) [4,32–34]. GP interpreters evaluate expression trees, which represent solutions to perform a user-defined task. In this experimental stage, the performance of the rules interpreter is evaluated by running over different number of instances and rules. This way, it achieves a sensitivity analysis of the effect of these parameters on the speed of the interpreter in terms of GPops/s.

[1]The collection of multi-instance data sets are publicly available to facilitate the replicability of the experiments and future comparisons at:

`http://www.uco.es/grupos/kdis/kdiswiki/MIL-GPU`

6.3.2 Rule-based classification performance

The second experiment evaluates the performance of the proposal across a series of 14 real-world multi-instance data sets. However, these real-world data sets may be categorized as medium size. In order to evaluate larger data, we generated 7 artificial data sets which comprise a wide dimensionality range, containing from 100 to 1 million instances. The objective of this experiment was to analyze the scalability of the GPU model regarding the data sets' complexity and dimensionality (number of bags, instances and attributes). Moreover, the use of one, two, and four GPUs will provide useful information about its scalability to big data and multiple GPU devices. All experiments were run 100 times using 10 × 10-fold cross-validation scheme and the average runtimes are reported.

7 Results

This section presents and discusses the experimental results obtained from different experimental studies.

7.1 Rules interpreter performance

Table 3 shows the rules interpreter performance regarding the number of rules to evaluate and the number of instances of the data set, which determine the total number of GP operations (GPops) to be interpreted by the evaluator. The table provides the evaluation times expressed in milliseconds for the multi-threaded CPU and the GPU-based interpreter using one, two, and four GPUs.

The number of GP operations per second (GPops/s) is calculated using the number of GPops evaluated and the time required. The multi-threaded CPU interpreter achieves up to 98 million GPops/s, whereas the GPU interpreter achieves up to 108 billion GPops/s when distributing the computation into four GPUs. Maximum performance is achieved when the number of rules or the number of instances are high enough to fill the GPU multiprocessors with enough thread blocks. In other words, maximum performance is achieved when the GPU cores are fully occupied by a large number of threads.

On the other hand, performance is reduced when the number of rules or instances is small, i.e, there are less threads to compute. Nevertheless, even with a low number of rules or instances, the performance of the GPU-based interpreter is still significantly better than the multi-threaded CPU. The increasing

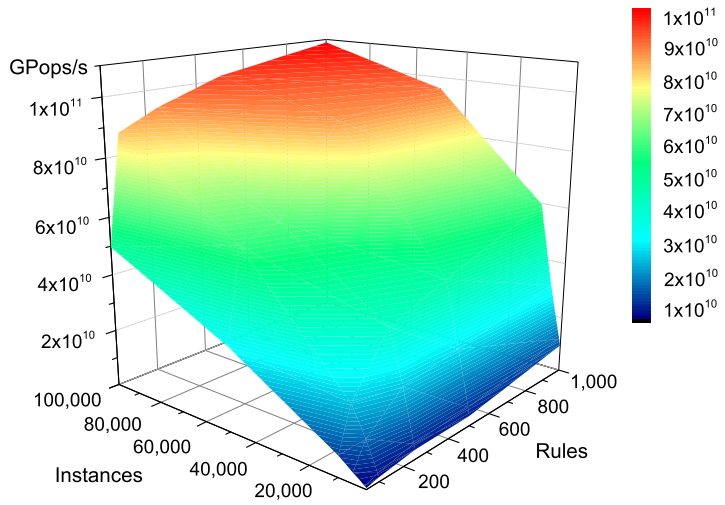


Fig. 8 Interpreter performance regarding the number of instances and rules

Table 3 Rules interpreter performance

			Runtime (ms)				GPops/s (Million)			
Rules	Instances	GPops	4 CPUs	1 GPU	2 GPUs	4 GPUs	4 CPUs	1 GPU	2 GPUs	4 GPUs
50	500	1.23×10^6	25	6.64	3.55	2.55	49	185	347	483
	1,000	2.46×10^6	49	6.94	4.23	2.83	50	355	581	869
	5,000	1.23×10^7	139	7.42	5.93	3.43	89	1,660	2,076	3,585
	50,000	1.23×10^8	1352	9.26	7.35	4.30	91	13,295	16,752	28,643
	100,000	2.46×10^8	2783	11.30	9.27	4.95	88	21,791	26,566	49,757
100	500	2.45×10^6	44	4.72	2.74	1.64	56	519	892	1,489
	1,000	4.90×10^6	76	5.10	2.89	1.91	64	961	1,692	2,561
	5,000	2.45×10^7	289	5.54	3.75	2.08	85	4,415	6,534	11,760
	50,000	2.45×10^8	2721	13.65	7.23	4.09	90	17,931	33,858	59,834
	100,000	4.90×10^8	5556	20.84	11.66	5.02	88	23,489	41,987	97,555
250	500	6.10×10^6	104	7.50	2.11	1.89	59	814	2,888	3,225
	1,000	1.22×10^7	124	7.84	4.32	2.49	98	1,557	2,826	4,900
	5,000	6.10×10^7	620	7.89	5.32	2.77	98	7,734	11,474	22,027
	50,000	6.10×10^8	6616	25.03	13.86	9.59	92	24,390	44,061	63,672
	100,000	1.22×10^9	13880	48.86	24.66	14.55	88	24,991	49,505	83,910
500	500	1.20×10^7	142	9.38	5.52	3.52	85	1,283	2,180	3,423
	1,000	2.41×10^7	261	11.51	6.26	3.85	92	2,091	3,849	6,248
	5,000	1.20×10^8	1230	14.64	7.91	4.83	98	8,222	15,214	24,924
	50,000	1.20×10^9	12860	47.22	25.29	13.09	94	25,491	47,596	91,962
	100,000	2.41×10^9	27571	90.11	46.99	23.73	87	26,717	51,239	101,465
1,000	500	2.43×10^7	255	11.50	7.79	2.56	95	2,111	3,118	9,481
	1,000	4.85×10^7	505	15.59	8.17	4.10	96	3,114	5,941	11,836
	5,000	2.43×10^8	2486	16.13	11.60	7.83	98	15,047	20,923	30,999
	50,000	2.43×10^9	26372	93.79	47.63	25.31	92	25,881	50,960	95,917
	100,000	4.85×10^9	53434	179.43	89.38	44.58	91	27,057	54,315	108,908

performance of the interpreter is shown in Fig. 8, which illustrates the interpreter performance regarding the number of rules to evaluate and the number of instances of the data set.

Moreover, Table 3 also shows the good scalability of the model from one to two and four GPUs, regardless the number of instances and rules to evaluate. In best performance scenarios with a high number of rules and instances, doubling the number of GPU devices doubles the interpreter's performance.

7.2 Rule-based classification performance

Table 4 shows the classification performance of the G3P-MI algorithm using the Dietterich hypothesis. The table shows the time dedicated evaluating the population of classification rules over a series of multi-instance data sets, considering different number of attributes, bags and instances. The speedup is the ratio of multi-threaded CPU time to GPU time. Similarly to the interpreter performance, classification results indicate that the higher the dimensionality of the data with a larger number of instances, the better the speedup achieved. The best performance scenario in which the highest speedup is achieved (408×) corresponds with the *process* data set when using four GPUs, which allows the evaluation time to be reduced from 57 minutes to only 8.43 seconds, which is a significant reduction.

Table 4 UCI data sets evaluation performance (Dietterich Hypothesis)

				Evaluation Time (s)				Speedup		
Data set	Atts	Bags	Instances	4 CPUs	1 GPU	2 GPUs	4 GPUs	1 GPU	2 GPUs	4 GPUs
Component	201	3,130	36,894	1,024	11.15	5.85	3.92	91.94	175.14	261.51
EastWest	25	20	213	5.15	2.25	2.06	1.00	2.29	2.50	5.17
Elephant	231	200	1,391	34.38	1.76	1.36	1.15	19.51	25.35	30.00
Fox	231	200	1,320	33.77	1.90	1.69	1.17	17.81	19.94	28.89
Function	201	5,242	55,536	1,555	15.33	8.62	4.58	101.52	180.45	339.49
Musk1	167	92	476	11.29	1.77	1.01	1.13	6.37	11.19	10.01
Musk2	167	102	6,598	89.77	2.26	1.71	1.21	39.71	52.58	74.33
Mut-atoms	11	188	1,618	36.82	2.17	1.44	1.61	16.99	25.55	22.82
Mut-bonds	17	188	3,995	74.90	2.06	1.91	1.75	36.39	39.14	42.85
Mut-chains	25	188	5,349	96.98	2.55	1.35	1.07	38.04	71.67	90.52
Process	201	11,718	118,417	3,448	29.99	16.25	8.43	114.99	212.24	408.93
Suramin	21	11	2,378	41.76	3.53	2.09	1.04	11.84	19.95	40.13
Tiger	231	200	1,220	31.15	2.48	1.67	1.08	12.58	18.60	28.75
Trx	9	193	26,611	482.8	6.81	4.36	2.01	70.90	110.78	240.21
Artificial1	100	10	1,000	6.28	2.90	2.15	1.42	2.16	2.93	4.43
Artificial2	100	100	1,000	20.45	3.16	1.62	1.85	6.48	12.60	11.04
Artificial3	100	100	1×10^4	54.89	4.49	2.88	1.84	12.22	19.09	29.89
Artificial4	100	1,000	1×10^4	175.6	4.55	2.52	1.51	38.56	69.59	116.22
Artificial5	100	1,000	1×10^5	551.3	20.42	10.50	6.10	27.00	52.49	90.40
Artificial6	100	1×10^4	1×10^5	1,850	25.07	13.45	6.55	73.81	137.55	282.48
Artificial7	100	1×10^5	1×10^6	18,431	213.2	109.3	57.38	86.45	168.51	321.23

On the other hand, good speedups are also achieved over small data sets with a low number of instances. It is important to note that even with a small data set, the speedup is greater than one, i.e., GPU runtimes are always lower than CPU ones. Thus, we can conclude that we recommend the use of GPU evaluation regardless of the size of the data set. This is non-trivial because in some computation problems with GPUs, it is only recommended to use GPUs when the problem size is higher than a certain threshold, whereas GPU times for small problems are higher than CPU ones.

Table 5 shows the classification performance of the G3P-MI algorithm using the generalized hypothesis (threshold-based or count-based). Similarly to the presence-based hypothesis, the speedup follows the same trend line when the data set size is increased and also shows good scalability to multiple GPU devices. However, there is a significant difference which can be observed when analyzing performance over the artificial data sets. Specifically, note and compare the CPU evaluation times for the *Artificial5* and *Artificial6* data sets in Tables 4 and 5. These data sets have the same number of instances, but the *Artificial6* data set has 10 times the number of bags. For the presence-based hypothesis, *Artificial6* triples the evaluation times of the *Artificial5* data set, whereas for the generalized hypothesis their times are similar. This difference in the runtime is due to the Dietterich hypothesis behaviour. As described in Section 5.2, the presence-based hypothesis allows instances matching to be stopped as soon as there is one instance that meets the rule conditions. However, for the generalized hypotheses, it is necessary to complete rule matching with all the instances of the bag to compute counts. Therefore, evaluation times for the generalized hypotheses depend on the number of instances rather than the number of bags. Thus, the GPU evaluation model benefits from the behaviour of the generalized hypotheses, achieving significantly higher speedup values.

Table 5 UCI data sets evaluation performance (Generalized Hypothesis)

				Evaluation Time (s)				Speedup		
Data set	Atts	Bags	Instances	4 CPUs	1 GPU	2 GPUs	4 GPUs	1 GPU	2 GPUs	4 GPUs
Component	201	3,130	36,894	1,874	15.94	8.47	4.32	117.64	221.27	434.38
EastWest	25	20	213	9.96	1.84	1.18	0.94	5.40	8.44	10.63
Elephant	231	200	1,391	59.05	1.73	1.21	0.77	34.19	48.99	76.90
Fox	231	200	1,320	56.17	1.72	1.23	0.73	32.65	45.69	76.90
Function	201	5,242	55,536	2,833	25.36	14.37	7.84	111.72	197.10	361.38
Musk1	167	92	476	20.37	1.55	1.04	0.72	13.12	19.53	28.15
Musk2	167	102	6,598	291.2	2.55	1.47	1.05	114.38	197.50	276.17
Mut-atoms	11	188	1,618	66.37	1.76	1.18	0.79	37.74	56.11	83.82
Mut-bonds	17	188	3,995	163.1	2.17	1.24	0.79	75.21	131.81	207.29
Mut-chains	25	188	5,349	218.5	2.20	1.38	0.94	99.52	157.87	231.95
Process	201	11,718	118,417	6,217	85.03	45.64	24.20	73.12	136.24	256.93
Suramin	21	11	2,378	105.8	2.76	2.07	1.14	38.31	51.04	93.11
Tiger	231	200	1,220	52.00	1.71	1.12	0.87	30.35	46.37	59.53
Trx	9	193	26,611	1,215	6.58	3.98	5.02	184.66	305.70	242.31
Artificial1	100	10	1,000	50.05	2.79	1.65	1.68	17.94	30.38	29.71
Artificial2	100	100	1,000	49.95	2.59	1.61	1.44	19.28	31.02	34.68
Artificial3	100	100	1×10^4	503.4	4.52	2.55	1.41	111.32	197.06	356.36
Artificial4	100	1,000	1×10^4	522.1	4.95	2.67	1.31	105.36	195.73	398.26
Artificial5	100	1,000	1×10^5	5,542	44.93	24.39	12.63	123.37	227.21	439.00
Artificial6	100	1×10^4	1×10^5	5,560	45.97	24.97	12.62	120.96	222.71	440.71
Artificial7	100	1×10^5	1×10^6	55,897	454.7	243.4	124.1	122.94	229.65	450.25

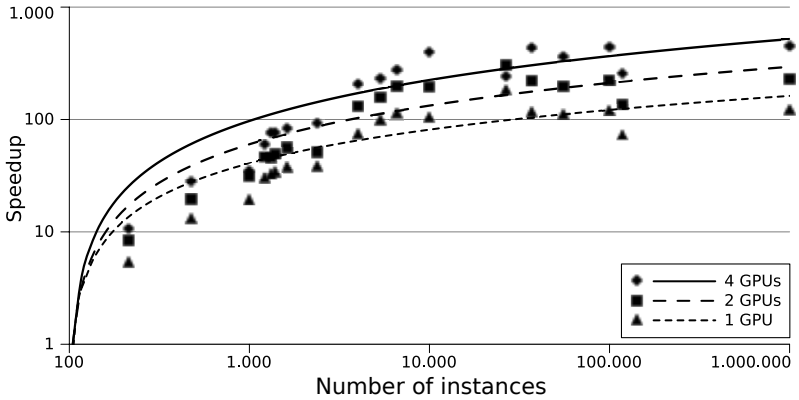


Fig. 9 Analysis of performance scalability using 1, 2, and 4 GPUs

Finally, the scalability of the model performance regarding to the increasing size of the data set is shown in Fig. 9 (log scale), which illustrates and summarizes the speedup values achieved with one, two, and four GPUs when compared to the multi-threaded CPU performance. This figure shows the speedup trend line and proves the great efficiency and scalability of GPUs. The figure indicates that as soon as the data set size is big enough, the high costs of the MIL algorithm makes the GPU parallelization highly recommended to run within reasonable time.

8 Concluding Remarks

Multi-instance classification is a challenging task which has been solved using evolutionary rule-based algorithms such as G3P-MI. Evolutionary rule-based classification algorithms become slow when the dimensionality and complexity of the data set increases. In this paper we proposed a GPU-based model for evaluating multi-instance classification rules to speed up the algorithm process. The GPU evaluation model was evaluated to analyze its efficiency regarding to the rules interpreter and classification performance, and compared to the multi-threaded CPU implementation over a series of real-world and artificial multi-instance data sets. The performance of the GPU rules interpreter achieved up to 108 billion GPops/s. The GPU evaluation model demonstrated efficient performance and to speedup the classification task up to 450×. Moreover, its efficiency increased as the dimensionality of the data. The distributed implementation into multiple GPUs allowed scalability across large-scale and high-dimensional data sets, in which the application of evolutionary rule-based algorithms, until now, was difficult within reasonable time.

Nevertheless, it is noteworthy to mention that highest speedups were obtained when using 4 high-performance GPUs on very big data sets, where GPUs are capable to show their full power. These scenarios are ideal for GPU parallelization (millions of parallel threads), while on other data sets with average size, GPUs show not such impressive speedups. Indeed, when comparing single GPU performance, it was achieved similar performance than other related works in literature. Using 4 high-performance GPUs is a non-trivial contribution while most of related works focus only on single-GPU designs, which prevent their scalability to multiple GPU devices.

Acknowledgements This work was supported by the Regional Government of Andalusia and the Ministry of Science and Technology, projects P08-TIC-3720 and TIN-2011-22408, and FEDER funds. This research was also supported by the Spanish Ministry of Education under FPU grant AP2010-0042.

References

1. NVIDIA Programming and Best Practices Guide.
2. S. Andrews, I. Tsochantaridis, and T. Hofmann. Support vector machines for multiple-instance learning. In *Proceedings of the Neural Information Processing System*, pages 561–568, 2002.
3. V. Balachandran, D. P, and D. Khemani. Interpretable and reconfigurable clustering of document datasets by deriving word-based rules. *Knowledge and Information Systems*, 32(3):475–503, 2012.
4. W. Banzhaf, S. Harding, W. B. Langdon, and G. Wilson. Accelerating Genetic Programming through Graphics Processing Units. In *Genetic Programming Theory and Practice VI*, Genetic and Evolutionary Computation, pages 1–19. 2009.
5. C. Bergeron, G. Moore, J. Zaretzki, C. Breneman, and K. Bennett. Fast bundle algorithm for multiple-instance learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(6):1068–1079, 2012.
6. A. Cano, A. Zafra, and S. Ventura. Speeding up the evaluation phase of GP classification algorithms on GPUs. *Soft Computing*, 16:187–202, 2012.
7. Y. Chai and Z. Yang. A multi-instance learning algorithm based on normalized radial basis function network. In *Proceedings of the 4th International Symposium on Neural Networks*, volume 4491 of *Lecture Notes in Computer Science*, pages 1162–1172, 2007.
8. S. Chen and L. Jiang. An empirical study on multi-instance learning. *Advances in Information Sciences and Service Sciences*, 4(6):193–202, 2012.
9. X. Chen, C. Zhang, S. . Chen, and S. Rubin. A human-centered multiple instance learning framework for semantic video retrieval. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, 39(2):228–233, 2009.
10. Y. Chevaleyre, N. Bredeche, and J. Zucker. Learning rules from multiple instance data: Issues and algorithms. In *Proceedings of 9th Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 455–459, 2002.
11. Y. Chevaleyre and J. Zucker. Solving multiple-instance and multiple-part learning problems with decision trees and decision rules. Application to the mutagenesis problem. In *Proceedings of the 14th of the Canadian Society for Computational Studies of Intelligence*, volume 2056 of *Lecture Notes in Computer Science*, pages 204–214, 2001.
12. F. B. De Oliveira, D. Davendra, and F. G. Guimarães. Multi-objective differential evolution on the GPU with C-CUDA. *Advances in Intelligent Systems and Computing*, 188:123–132, 2013.
13. T. G. Dietterich, R. H. Lathrop, and T. Lozano-Pérez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89:31–71, 1997.

14. P. G. Espejo, S. Ventura, and F. Herrera. A survey on the application of genetic programming to classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 40(2):121–144, 2010.
15. F. Fabris and R. A. Krohling. A co-evolutionary differential evolution algorithm for solving min-max optimization problems implemented on GPU using C-CUDA. *Expert Systems with Applications*, 39(12):10324–10333, 2012.
16. K. L. Fok, T. T. Wong, and M. L. Wong. Evolutionary computing on consumer graphics hardware. *IEEE Intelligent Systems*, 22(2):69–78, 2007.
17. J. Foulds and E. Frank. A review of multi-instance learning assumptions. *Knowledge Engineering Review*, 25(1):1–25, 2010.
18. J. R. Foulds and E. Frank. Speeding up and boosting diverse density learning. In *Proceedings of the 13th international conference on Discovery science*, pages 102–116, 2010.
19. M. A. Franco, N. Krasnogor, and J. Bacardit. Speeding up the evaluation of evolutionary learning systems using GPGPUs. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1039–1046, 2010.
20. A. A. Freitas. *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer-Verlag, 2003.
21. A. A. Freitas. *A Review of Evolutionary Algorithms for Data Mining*, pages 61–93. 2007.
22. S. Gao and Q. Suna. Exploiting generalized discriminative multiple instance learning for multimedia semantic concept detection. *Pattern Recognition*, 41(10):3214–3223, 2008.
23. M. Garland, S. Le Grand, J. Nickolls, J. Anderson, J. Hardwick, S. Morton, E. Phillips, Y. Zhang, and V. Volkov. Parallel computing experiences with CUDA. *IEEE Micro*, 28(4):13–27, 2008.
24. T. Gartner, P. A. Flach, A. Kowalczyk, and A. J. Smola. Multi-instance kernels. In *Proceedings of the 19th International Conference on Machine Learning*, pages 179–186, 2002.
25. Z. Gu, T. Mei, J. Tang, X. Wu, and X. Hua. MILC2: A multi-layer multi-instance learning approach to video concept detection. In *Proceedings of the 14th International Conference of Multimedia Modeling*, pages 24–34, 2008.
26. S. Harding and W. Banzhaf. Fast genetic programming on GPUs. *Lecture Notes in Computer Science*, 4445:90–101, 2007.
27. G. Herman, G. Ye, J. Xu, and B. Zhang. Region-based image categorization with reduced feature set. In *Proceedings of the 10th IEEE Workshop on Multimedia Signal Processing*, pages 586–591, 2008.
28. R. I. Hoai, N. X. Whigham, P. A. Shan, Y. O'neill, and M. McKay. Grammar-based Genetic programming: A survey. *Genetic Programming and Evolvable Machines*, 11(3-4):365–396, 2010.
29. H. Huang and C. Hsu. Bayesian classification for data from the same unknown class. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 32(2):137–145, 2002.
30. J. Huysmans, K. Dejaeger, C. Mues, J. Vanthienen, and B. Baesens. An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models. *Decision Support Systems*, 51:141–154, 2011.
31. D. Konieczny, M. Marcinkowski, and P. . Myszkowski. GPGPU implementation of evolutionary algorithm for images clustering. *Studies in Computational Intelligence*, 457:219–238, 2013.
32. W. B. Langdon. A many threaded cuda interpreter for genetic programming. *Lecture Notes in Computer Science*, 6021:146–158, 2010.
33. W. B. Langdon. Graphics processing units and genetic programming: An overview. *Soft Computing*, 15(8):1657–1669, 2011.
34. W. B. Langdon and W. Banzhaf. A SIMD interpreter for genetic programming on GPU graphics cards. *Lecture Notes in Computer Science*, 4971:73–85, 2008.
35. W. B. Langdon and A. P. Harrison. GP on SPMD parallel graphics hardware for mega bioinformatics data mining. *Soft Computing*, 12(12):1169–1183, 2008.
36. C. H. Li, I. Gondra, and L. Liu. An efficient parallel neural network-based multi-instance learning algorithm. *Journal of Supercomputing*, 62(2):724–740, 2012.
37. O. Maron and T. Lozano-Pérez. A framework for multiple-instance learning. In *Proceedings of Neural Information Processing System*, pages 570–576, 1997.
38. D. McKenney and T. White. Stock trading strategy creation using GP on GPU. *Soft Computing*, 16(2):247–259, 2012.
39. D. Nguyen, C. Nguyen, R. Hargraves, L. Kurgan, and K. Cios. mi-ds: Multiple-instance learning algorithm. *IEEE Transactions on Cybernetics*, 43(1):143–154, 2013.
40. X. Qi and Y. Han. Incorporating multiple svms for automatic image annotation. *Pattern Recognition*, 40(2):728–741, 2007.
41. S. Sabato and N. Tishby. Multi-instance learning with any hypothesis class. *Journal of Machine Learning Research*, 13:2999–3039, 2012.
42. J. Santner, C. Leistner, A. Saffari, T. Pock, and H. Bischof. PROST: Parallel robust online simple tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 723–730, 2010.
43. S. Scott, J. Zhang, and J. Brown. On Generalized Multiple-instance Learning. *International Journal of Computational Intelligence and Applications*, 5(1):21–36, 2005.
44. S. Ventura, C. Romero, A. Zafra, J. A. Delgado, and C. Hervás. JCLEC: a Java framework for Evolutionary Computation. *Soft Computing*, 12(4):381–392, 2007.
45. H. Wang, S. Rahnamayan, and Z. Wu. Parallel differential evolution with self-adapting control parameters and generalized opposition-based learning for solving high-dimensional optimization problems. *Journal of Parallel and Distributed Computing*, 73(1):62–73, 2013.

46. J. Wang and J.-D. Zucker. Solving the multiple-instance problem: A lazy learning approach. In *Proceedings of the 17th International Conference on Machine Learning*, pages 1119–1126, 2000.
47. N. Weidmann, E. Frank, and B. Pfahringer. A Two-level Learning Method for Generalized Multi-instance Problems. In *Proceedings of the 14th European Conference on Machine Learning*, pages 468–479, 2003.
48. I. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. 2nd Edition. Morgan Kaufmann, 2005.
49. Q. Wu, M. Ng, and Y. Ye. Markov-Miml: A Markov chain-based multi-instance multi-label learning algorithm. *Knowledge and Information Systems*, pages 1–22, 2012.
50. X. L. Wu, N. Obeid, and W. M. Hwu. Exploiting more parallelism from applications having generalized reductions on GPU architectures. In *Proceedings of the 10th IEEE International Conference on Computer and Information Technology*, pages 1175–1180, 2010.
51. A. Zafra, C. Romero, and S. Ventura. Multiple instance learning for classifying students in learning management systems. *Expert System Applications*, 38(12):15020–15031, 2011.
52. A. Zafra, C. Romero, S. Ventura, and E. Herrera-Viedma. Multi-instance genetic programming for web index recommendation. *Expert Systems with Applications*, 36:11470–11479, 2009.
53. A. Zafra and S. Ventura. G3P-MI: A genetic programming algorithm for multiple instance learning. *Information Sciences*, 180:4496–4513, 2010.
54. A. Zafra and S. Ventura. Multi-instance genetic programming for predicting student performance in web based educational environments. *Applie Soft Computing*, 12(8):2693–2706, 2012.
55. A. Zafra and S. Ventura. Multi-objective approach based on grammar-guided genetic programming for solving multiple instance problems. *Soft Computing*, 16:955–977, 2012.
56. Z. Zhou and M. Zhang. Solving multi-instance problems with classifier ensemble based on constructive clustering. *Knowledge and Information Systems*, 11(2):155–170, 2007.
57. Z.-H. Zhou, K. Jiang, and M. Li. Multi-instance learning based web mining. *Applied Intelligence*, 22(2):135–147, 2005.

TITLE:

An Interpretable Classification Rule Mining Algorithm

AUTHORS:

A. Cano, A. Zafra, and S. Ventura

Information Sciences, *Volume 240, pp. 1-20, 2013*

RANKING:

Impact factor (JCR 2012): 3.643

Knowledge area:

Computer Science, Information Systems: 6/132

DOI: 10.1016/J.INS.2013.03.038

Information Sciences 240 (2013) 1–20

Contents lists available at SciVerse ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

An interpretable classification rule mining algorithm

Alberto Cano, Amelia Zafra, Sebastián Ventura *

Department of Computer Science and Numerical Analysis, University of Córdoba, Spain

ARTICLE INFO

Article history:
Received 9 October 2011
Received in revised form 14 February 2013
Accepted 16 March 2013
Available online 6 April 2013

Keywords:
Classification
Evolutionary programming
Interpretability
Rule mining

ABSTRACT

Obtaining comprehensible classifiers may be as important as achieving high accuracy in many real-life applications such as knowledge discovery tools and decision support systems. This paper introduces an efficient Evolutionary Programming algorithm for solving classification problems by means of very interpretable and comprehensible IF-THEN classification rules. This algorithm, called the Interpretable Classification Rule Mining (ICRM) algorithm, is designed to maximize the comprehensibility of the classifier by minimizing the number of rules and the number of conditions. The evolutionary process is conducted to construct classification rules using only relevant attributes, avoiding noisy and redundant data information. The algorithm is evaluated and compared to nine other well-known classification techniques in 35 varied application domains. Experimental results are validated using several non-parametric statistical tests applied on multiple classification and interpretability metrics. The experiments show that the proposal obtains good results, improving significantly the interpretability measures over the rest of the algorithms, while achieving competitive accuracy. This is a significant advantage over other algorithms as it allows to obtain an accurate and very comprehensible classifier quickly.

1. Introduction

Discovering knowledge in large amounts of data collected over the last decades has become significantly challenging and difficult, especially in large-scale databases. Data mining (DM) [60] involves the use of data analysis tools to discover previously unknown, valid patterns and relationships in large data sets. Classification and regression are two forms of data analysis which can be used to extract models describing important data classes or to predict future data trends. Classification predicts categorical labels whereas regression models predict continuous-valued functions.

The data analysis tools used for DM include statistical models, mathematical methods, and machine learning algorithms. Classification is a common task in supervised machine learning with the search for algorithms that learn from training examples to produce predictions about future examples.

Classification has been successfully solved using several approaches [26]. On the one hand, there are approaches such as artificial neural networks (ANN) [46], support vector machines (SVM) [16], and instance-based learning methods [2]. These approaches obtain accurate classification models but they must be regarded as black boxes, i.e., they are opaque to the user. Opaque predictive models prevent the user from tracing the logic behind a prediction and obtaining interesting knowledge previously unknown from the model. These classifiers do not permit human understanding and inspection, they are not directly interpretable by an expert and it is not possible to discover which are the relevant attributes to predict the class of an example. This opacity prevents them from being used in many real-life knowledge discovery applications where both accuracy and comprehensibility are required, such as medical diagnosis [55], credit risk evaluation [42], and decision support systems [6], since the prediction model must explain the reasons for classification.

* Corresponding author.
E-mail addresses: acano@uco.es (A. Cano), azafra@uco.es (A. Zafra), sventura@uco.es (S. Ventura).

0020-0255/$ - see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ins.2013.03.038

On the other hand, there are machine learning approaches which overcome this limitation and provide transparent and comprehensible classifiers such as decision trees [62] and rule-based systems [49]. Evolutionary Algorithms [65], and specifically Evolutionary Programming (EP) [13,64] and Genetic Programming (GP) [25], have been successfully applied to build decision trees and rule-based systems easily. Rule-based systems are especially user-friendly and offer compact, understandable, intuitive and accurate classification models. To obtain comprehensibility, accuracy is often sacrificed by using simpler but transparent models, achieving a trade-off between accuracy and comprehensibility. Even though there are many rule based classification models, it has not been until recently that the comprehensibility of the models is becoming a more relevant objective. Proof of this trend is found in recent studies of issue [18,27,34,57], i.e, the comprehensibility of the models is a new challenge as important as accuracy. This paper focuses on the interpretability, trying to reach more comprehensible models than most of the current proposals and thus covering the needs of many application domains that require greater comprehensibility than the provided by current methods.

This paper presents an EP approach applied to classification problems to obtain comprehensible rule-based classifiers. This algorithm, called ICRM (Interpretable Classification Rule Mining), is designed to obtain a base of rules with the minimum number of rules and conditions, in order to maximize its interpretability, while obtaining competitive accuracy results. The algorithm uses an individual = rule representation, following the Iterative Rule Learning (IRL) model. Individuals are constructed by means of a context-free grammar [33,61], which establishes a formal definition of the syntactical restrictions of the problem to be solved and its possible solutions, so that only grammatically correct individuals are generated. Next, the most important characteristics of the algorithm are detailed. Firstly, the algorithm guarantees obtaining the minimum number of rules. This is possible because it generates one rule per class, together with a default class prediction, which is assigned when none of the available rules are triggered. Moreover, it is guaranteed that there are no contradictory or redundant rules, i.e., there is no pair of rules with the same antecedents and different consequents. Finally, it also guarantees the minimum number of conditions forming the antecedents of these rules, which is achieved by selecting only the most relevant and discriminating attributes that separate the classes in the attribute domains.

The experiments carried out on 35 different data sets and nine other algorithms show the competitive performance of our proposal in terms of predictive accuracy and execution time, obtaining significantly better results than all the other algorithms in terms of all the interpretability measures considered: the minimum number of rules, minimum number of conditions per rule, and minimum number of conditions of the classifier. The experimental study includes a statistical analysis based on the Bonferroni–Dunn [24] and Wilcoxon [59] non-parametric tests [28,29] in order to evaluate whether there are statistically differences in the results of the algorithms.

This paper is structured as follows. Section 2 briefly reviews the related background works. Section 3 describes the ICRM algorithm. Section 4 describes the experimental study whose results are discussed in Section 5. Finally, Section 6 draws some conclusions raised from the work.

2. Background

This section introduces the accuracy vs interpretability problem and discusses the interpretability definition and metrics. Finally, it briefly reviews the most important works related to genetic rule-based classification systems in recent years.

2.1. Accuracy vs interpretability

Classification with rule-based systems comes with two contradictory requirements in the obtained model: the interpretability, capability the behavior of the real system in an understandable way, and the accuracy, capability to faithfully represent the real system. Obtaining high degrees of interpretability and accuracy is a contradictory purpose and, in practice, one of the two properties prevails over the other. To find the best trade-off between them is an optimization problem that is very difficult to solve efficiently [36].

In contrast, when looking for interpretability, fuzzy-based systems are usually considered [5,41]. These systems comprise very interpretable rules since they employ linguistic variables to address the vagueness of human language [44]. One of these systems is the algorithm by *González and Pérez*, SLAVE (Structural Learning Algorithm on Vague Environment) [30], which is a genetic learning algorithm that uses the iterative approach to learn fuzzy rules. Another fuzzy rule-based algorithm is GFS-GP [52], which combines genetic programming operators with simulated annealing search. The use of evolutionary programming, genetic programming and grammars to construct classification rules has been widely used [7,20,25,66]. However, this interpretability perspective is not related to the search for simpler classifiers in terms of fewer number of rules and conditions, in which we are really interested.

2.2. Interpretability metrics

There is no well-established definition of the interpretability of a rule-based system. The interpretability of a rule set is very important, due to the fact that very large sets of rules or very complex rules are rather lacking in interest. In fact, studies have focused on reducing the complexity of rule-based classifiers [35,53]. Nevertheless, there are some indicators that allow us to estimate the interpretability and comprehensibility of a rule-based classifier, which are described by García et al. [27]:

these are the number of rules and the number of conditions. A complexity value of a rule-based classifier was provided by Nauck [43], who proposes an interpretability measure related to the number of conditions of the classifier and the number of classes to be covered:

$$complexity = \frac{m}{\sum_{i=1}^{r} n_i} \tag{1}$$

where m is the number of classes, r is the number of rules, and n_i is the number of conditions used in the ith rule. This measure is 1 if the classifier contains only one rule per class using one condition each and it approaches 0 the more rules and conditions are used. However, this measure can be up to 2 for a two class data set where the algorithm learns a classifier containing one rule with one condition and another default rule with no conditions predicting the default class.

2.3. Genetic rule-based systems

Genetic algorithms (GAs) evolve a population of individuals which correspond to candidate solutions to a problem. GAs have been used for learning rules (Genetic rule-based systems) [21], including crisp and fuzzy rules, and they follow two approaches for encoding rules within a population.

The first one represents an individual as a single rule (individual = rule). The rule base is formed by combining several individuals from the population (rule cooperation) or via different evolutionary runs (rule competition). This representation results in three approaches:

- Michigan: they employ reinforcement learning and the GA is used to learn new rules that replace the older ones via competition through the evolutionary process. These systems are usually called learning classifier systems [14,40], such as XCS [15] and UCS [12].
- Iterative Rule Learning (IRL): individuals compete to be chosen in every GA run. The rule base is formed by the best rules obtained when the algorithm is run multiple times. SLAVE [30] and HIDER [1] are examples which follow this model.
- Genetic Cooperative-Competitive Learning (GCCL): the whole population or a subset of individuals encodes the rule base. In this model, the individuals compete and cooperate simultaneously. This approach makes it necessary to introduce a mechanism to maintain the diversity of the population in order to avoid a convergence of all the individuals in the population. GP-COACH [11] or COGIN [31] follow this approach.

The second one represents an individual as a complete set of rules (individual = set of rules), which is also known as the Pittsburgh approach. The main advantage of this approach compared to the first one is that it allows of addressing the cooperation–competition problem, involving the interaction between rules in the evolutionary process [8,45]. Pittsburgh systems are generally slower, since they evolve more complex structures [39]. Therefore, studies have focused on improving the effectiveness and efficiency of the rule structure exploration [63]. Moreover, one of their main problems is controlling the number of rules, which increases the complexity of the individuals, adding computational cost to their evaluation and becoming an unmanageable problem. This problem is known as the bloat effect [9], i.e., a growth without control of the size of the individuals.

One method based on this approach is the Memetic Pittsburgh Learning Classifier System (MPLCS) [10], which evaluates several local search mechanisms that heuristically edit classification rules and rule sets to improve their performance. In order to avoid the bloat effect, they employ a rule deletion operator and a fitness function based on the minimum description length [9,50], which balances the complexity and accuracy of the rule set. Moreover, this system uses a windowing scheme that reduces the run-time of the system by dividing the training set into many non-overlapping subsets over which the fitness is computed at each GA iteration. Another Pittsburgh style method is ILGA [32], which fixes the number of rules a priori, i.e., the complexity regarding to the number of rules is let to the user.

The order of the rules in the rule base is critical when dealing with decision lists [51]. A decision list is an ordered list of conjunctive rules. The classifier predicts the class membership of the highest priority rule which matches the antecedent of the rule and the data instance. Some authors have used evolutionary algorithms to sort the rules a posteriori, such as Tan et al. and their the CO-Evolutionary Rule Extractor (CORE) algorithm [54]. This GCCL model uses a gene to represent the rule index and the chromosome represents the ordering of the rules. Thus, the ordering of the rules is optimized.

There are also hybrid models such as DTGA [19] by Carvalho and Freitas, which is a hybrid decision tree/genetic algorithm method. The idea of this hybrid method involves the concept of small disjuncts. A set of classification rules can be regarded as a logical disjunction of rules, so that each rule can be regarded as a disjunct. A small disjunct is a rule covering a small number of examples. However, although each small disjunct covers just a few examples, the set of all small disjuncts can cover a large number of examples.

3. The ICRM algorithm

This section describes the most relevant features and the execution model of the ICRM algorithm. This paper presents a comprehensive and extended version of the ICRM algorithm, whose initial results were reported in [17]. The algorithm

consists of three phases. In the first phase, the algorithm creates a pool of rules that explore the attribute domains. In the second phase, the algorithm iterates to find classification rules and builds the classifier. Finally, the third phase optimizes the accuracy of the classifier. Fig. 1 shows the workflow of the algorithm. The following three sections describe the phases of the algorithm.

3.1. Phase 1: exploring the attribute domains

This phase explores the attribute domains, whose output is a pool of rules composed of a single attribute–value comparison. The idea is to find the best attribute–value comparisons to classify each of the classes. Fig. 2 shows the grammar used to represent single attribute–value comparisons.

The point is to ensure the exploration of the entire range of attribute domains. The domains can be categorical or numerical. On the one hand, categorical domains are constrained to a limited number of labels or nominal values. Therefore, finding a relevant label that best classifies a class is simple and it is a combinatorial problem with a relatively low number of solutions. The algorithm creates a single attribute–label comparison for each label and nominal relational operator.

On the other hand, numerical domains have infinite solutions. Therefore, a method to explore numerical domains is described now. The idea is to initially explore the range of the numerical domain at multiple points uniformly distributed (then, the rules' numerical values are optimized in Phase 2). The algorithm creates a single attribute–value comparison rule for each of these points and numerical relational operators. All these rules are stored in a pool and they will be used in Phase 2. The pseudo-code of the creation procedure of these single attribute–value comparison is shown in Algorithm 1.

Algorithm 1. Pool creation procedure

```
1: for i = 1 to numberAttributes do
2:   if attribute (i) is numerical then
3:     step ← domain (i)/numberPoints
4:     left ← minValueDomain (i)
5:     for j = 1 to numberPoints-1 do
6:       value ← left + j * step
7:       createComparison (⩾attribute (i) value)
8:       createComparison (⩽attribute (i) value)
9:     end for
10:  else if attribute (i) is categorical then
11:    for all label in domain(i) do
12:      createComparison (=attribute (i) label)
13:      createComparison (≠attribute (i) label)
14:    end for
15:  end if
16: end for
```

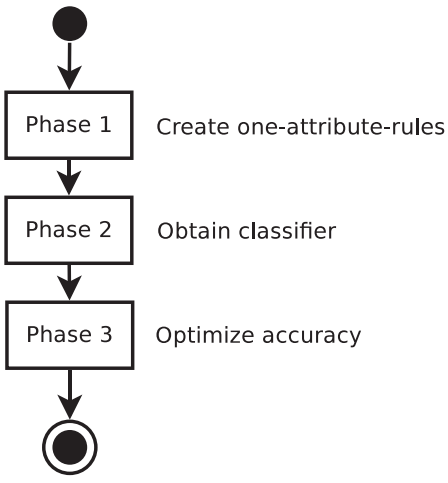


Fig. 1. Algorithm's workflow.

$$\langle S \rangle \rightarrow \langle cmp \rangle$$
$$\langle cmp \rangle \rightarrow \langle op\_num \rangle \ \langle variable \rangle \ \langle value \rangle$$
$$\langle cmp \rangle \rightarrow \langle op\_cat \rangle \ \langle variable \rangle \ \langle value \rangle$$
$$\langle op\_num \rangle \rightarrow \geq \mid \leq$$
$$\langle op\_cat \rangle \rightarrow = \mid \neq$$
$$\langle variable \rangle \rightarrow Any\ valid\ attribute\ in\ data\ set$$
$$\langle value \rangle \rightarrow Any\ valid\ value$$

Fig. 2. Grammar used to create single attribute–value comparisons.

3.2. Phase 2: obtaining the classifier

The objective of this phase is to obtain interpretable, accurate and complete classification rules. The input of this phase is the pool of single attribute–value comparisons created in the first phase and the output is a set of classification rules forming a classifier. Fig. 3 shows the workflow of this phase. The following sections detail the procedure to obtain the rules, the encoding of the individuals, the genetic operator, and the fitness function.

3.2.1. Obtaining classification rules

The goal of Phase 2 is to obtain a complete base of rules with as many rules as classes. Each rule predicts a single class and each class is predicted by a single rule. In each iteration of this procedure a classification rule is obtained. The rule predicts a class that has not been covered yet and which is the easiest class to classify among the remaining classes. To do so, it evolves in parallel as many evolutionary processes as classes to be covered. The output of each evolutionary process is a classification rule that predicts a unique class. The best rule among all the outputs from the evolutionary processes is selected and returned to be inserted into the classifier. The class predicted by this rule is set as covered and the algorithm removes the instances of that class from the training set. This process is repeated as many times as the number of classes but one, and the last class is set as the default class of the classifier.

This iterative procedure implies to evolve as many evolutionary algorithms as number of classes, i.e., that for a data set with N classes, the first iteration evolves N evolutionary algorithms, the second, $N-1$, the third, $N-2$, and so on, until there is only one remaining class to be covered. This may seem to require a high computational cost. However, in every iteration the number of classes to discern and the number of instances is lower than in the previous iteration, reducing the dimensionality of the problem (number of instances and classes to cover). Consequently, each iteration runs faster than the previous one. The experimental results and the execution times shown in Section 5.6 demonstrate the great efficiency of the algorithm.

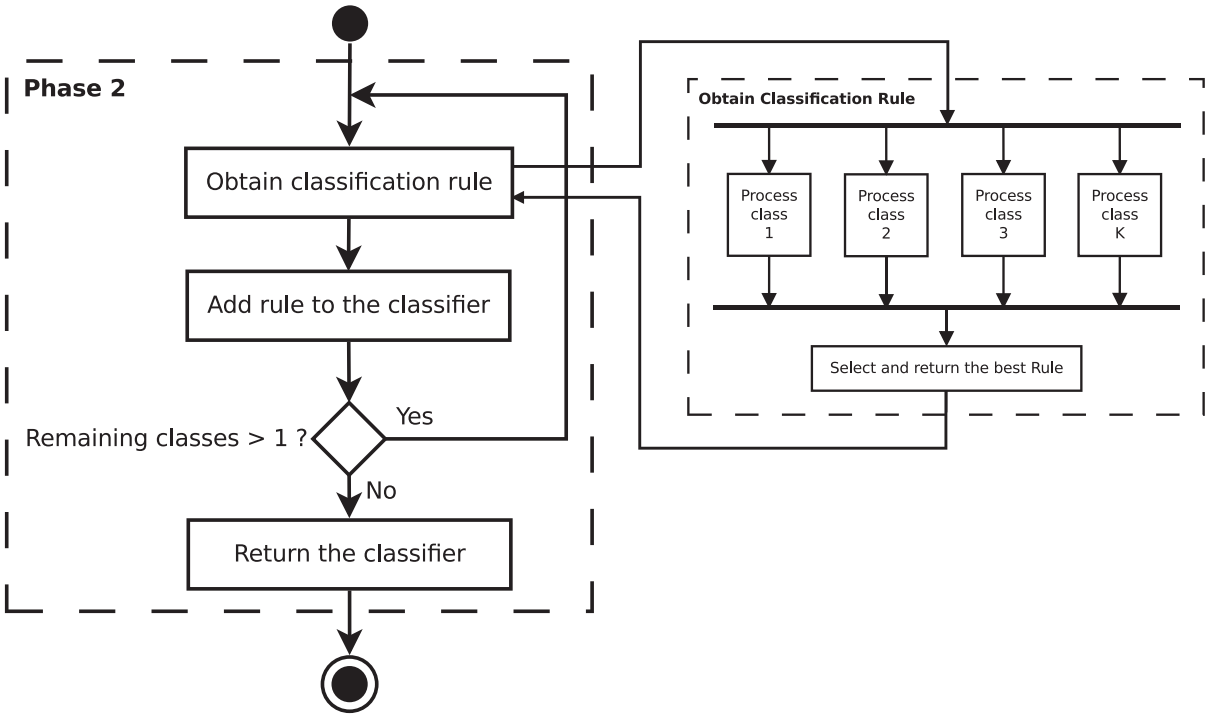


Fig. 3. Phase 2 workflow.

The following sections describe the individual representation, the overview of the algorithm, the genetic operator and the fitness function of the evolutionary algorithm employed to obtain the optimized classification rules which will comprise the classifier.

3.2.2. Individual representation

Phase 2 uses an individual = rule representation to learn interpretable IF-THEN rules that are inserted into a decision list forming a classifier. This representation provides greater efficiency, and addresses the cooperation–competition problem by dealing with the order of the rules in the evolutionary process. The antecedent of the rules represents a conjunction of attribute–value comparisons and the rule consequent contains the predicted class for the concepts satisfied by the antecedent of the rule. Fig. 4 shows the grammar used to represent the rules, i.e., a rule is a conjunction of one or more attribute–value comparisons.

3.2.3. Evolutionary rule generation

The learning process of the algorithm performs as a generational and elitist evolutionary algorithm. The evolutionary process will find the conjunction of attribute–value comparisons over the relevant attributes to discriminate a class from the other classes, by means of the genetic operator. For each rule, the genetic operator employs the not-yet covered attributes of the individual's rule to find the best condition over any other attribute that, appended to the rule, improves its accuracy. Therefore, the maximum number of generations is set as the number of attributes. The survival of the best individuals is guaranteed by copying them to the next generation. Fig. 5 shows the workflow of the evolutionary algorithm.

The genetic operator adds new clauses to the rule. When a new condition is added, the search area and the number of instances covered by the rule are both reduced. Thus, rather than search new conditions on the entire training set, the

$$\langle S\rangle \rightarrow \langle cmp\rangle \mid \langle S\rangle\ AND\ \langle cmp\rangle$$
$$\langle cmp\rangle \rightarrow \langle op\_num\rangle\ \langle variable\rangle\ \langle value\rangle$$
$$\langle cmp\rangle \rightarrow \langle op\_cat\rangle\ \langle variable\rangle\ \langle value\rangle$$
$$\langle op\_num\rangle \rightarrow \geq \mid \leq$$
$$\langle op\_cat\rangle \rightarrow = \mid \neq$$
$$\langle variable\rangle \rightarrow Any\ valid\ attribute\ in\ data\ set$$
$$\langle value\rangle \rightarrow Any\ valid\ value$$

Fig. 4. Rule = conjunction of attribute–value comparisons.

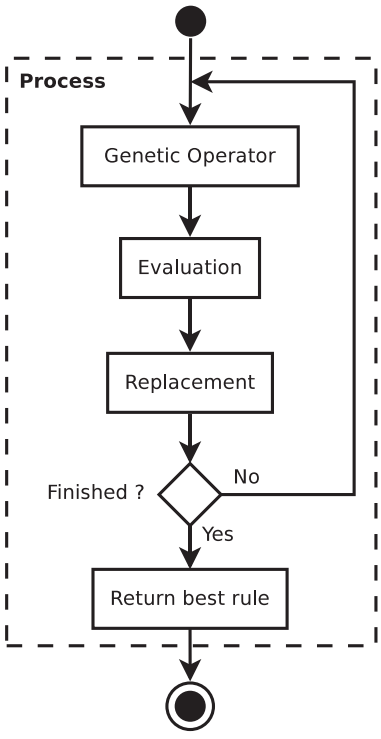


Fig. 5. Rule generation workflow.

genetic operator focuses on finding which attribute is now relevant in the training subset of instances covered by the rule. The genetic operator will return a condition to be appended to the rule, creating a new individual. If the genetic operator did not find a relevant attribute that improves the accuracy of the rule, the individual is marked as unbeatable. Moreover, once an individual is constrained to all the attributes, it is also marked as unbeatable (all the attributes have been covered).

The evaluation method computes the fitness value for each individual created by the genetic operator. The higher the fitness value is, the better the rule classifies.

Finally, the replacement method selects the best individuals from the previous generation and the offspring as the population for the next generation, keeping the population size constant. The algorithm finishes when the maximum number of generations is reached or all the individuals from the population are marked as unbeatable. Finally, the best individual is returned, which maximizes the fitness function and is constrained only to relevant attributes.

3.2.4. Genetic operator

This section describes the genetic operator designed to find the relevant attributes that, appended to a parent rule, improves its fitness. The parameters of this operator are the pool of single attribute–value comparisons, the class to predict, and the training subset of instances covered by the parent rule. The operator executes a hill-climbing microgenetic algorithm (MGA) [37] to optimize the numerical values of the attribute–value comparisons over the instances subset. The operator returns a new attribute–value comparison that best discriminates the predicted class among the other classes. This new attribute–value comparison is appended to the parent rule, creating a new individual.

The optimization of the numeric values of the attribute–value comparisons is treated as an optimization problem of a continuous variable real function. The MGA uses a mutation operator to approximate the numerical values of the rules to the maximum of the function. Each rule mutates the value of the comparison within a close range $[-step, +step]$ using the *step* size described in Algorithm 1. The selection procedure is responsible for ensuring the survival of the best individuals and after a small number of generations, the individuals converge to the value that best classifies the target class.

3.2.5. Fitness function

The results of the matching of the rules and the instances from the data set are used to build the confusion matrix, which is a table with two rows and two columns that reports the number of true positives (T_P), false positives (F_P), true negatives (T_N), and false negatives (F_N). The algorithm fitness function uses these values and combines two indicators that are commonplace in the domain, namely the sensitivity (Se) and the specificity (Sp). Sensitivity, also called recall in some fields, measures the proportion of actual positives which are correctly identified as such, whereas specificity measures the proportion of negatives which are correctly identified.

$$Se = \frac{T_P}{T_P + F_N} \qquad Sp = \frac{T_N}{T_N + F_P}$$

The fitness value is computed by means of the sensitivity and the specificity in order to maximize both metrics. In addition, to ensure the better interpretability of the rules represented by the individuals, for equally accurate rules, the simplest one with the lowest number of conditions prevails.

$$Fitness = Se * Sp$$

3.3. Phase 3: optimizing the accuracy

This section describes the evolutionary process for optimizing the classifier obtained in Phase 2. The rules of the classifier have been selected and optimized individually to maximize the product of sensitivity and specificity. However, the values from the numerical comparisons of the rules can be slightly tuned to improve the overall accuracy of the classifier. Therefore, Phase 3 runs an evolutionary process which improves the accuracy of the classifier.

Phase 3 uses an individual = classifier representation. The population is initialized using the classifier provided by Phase 2 as the seed. The algorithm iterates mutating the rules of the classifiers and selecting the best offspring until a certain number of generations have been performed. The population size and the maximum number of generations are parameters specified in Section 4.3.

The mutation operator is applied to every individual within the population. Given a parent individual, it creates as many new individuals as there are rules in the parent classifier in order to tune the comparisons of the rules to improve the overall performance of the classifier. Every new individual is a copy of the parent, but one rule has been mutated. The mutation of a rule performs as a local search and tunes the values from the attribute–value comparisons within a close range $[-step, +step]$ using the *step* size described in Algorithm 1, similar to the MGA described in the genetic operator from Phase 2.

The fitness function computes the fitness value of the individuals from the population. In Phase 3, the fitness of an individual is the accuracy of the classifier, since all the individuals have the same complexity. The accuracy is the ratio of successfully classified instances, i.e., the number of instances where the class prediction was correct compared to the total number of instances.

4. Experimental study

This section describes the details of the experiments performed in various problem domains to evaluate the capabilities of the proposal and compare it to other classification methods.

The experiments carried out compare the results of the ICRM algorithm and nine other classification algorithms over 35 data sets. These data sets were collected from the KEEL repository website [3] and the algorithms are available on the KEEL software tool [4]. The data sets together with their partitions are available to facilitate the replicability of the experiments and future comparisons with previous or new proposals.[1]

First, the application domains and the algorithms used for the comparison are presented. Second, an analysis of the parameters of ICRM algorithm is performed in order to find appropriate parameter settings. Third, a brief description of the experimental methodology and computed metrics is given. Finally, the statistical tests used for validating the results are presented.

4.1. Problem domains

The data sets used in the experiments were collected from the KEEL repository website [3] and are very varied in their degrees of complexity, number of classes, number of attributes and number of instances. The number of classes ranges up to 10, the number of attributes ranges from 3 to 60 and the number of instances ranges from 101 to 10,992. Table 1 summarizes the information about these data sets, which are available through the link 1.

4.2. Classification algorithms

This section describes the algorithms used in our experiments, which are obtained from the KEEL software tool [4]. The most relevant rule-based proposals presented to date are compared to determine whether our evolutionary algorithm is competitive in the different domains with the other approaches. Six of the methods were presented in the background section (MPLCS, ILGA, CORE, SLAVE, GFS-GP, DTGA), but we also consider interesting to compare with three other well-known methods, including one non-evolutionary crisp rule method (RIPPER), the classical decision tree (C4.5) from which extract rules (C45R), and one ant colony method (Ant Miner+).

- MPLCS [10]: a Memetic Pittsburgh Learning Classifier System that combines LS operators and policies.
- ILGA [32]: an Incremental Learning approach to Genetic Algorithms, with different initialization schemes based on different initializations of the GA population.
- CORE [54]: COevolutionary Rule Extractor that coevolves rules and rule sets concurrently in two cooperative populations to confine the search space and to produce good rule sets that are comprehensive.
- SLAVE [30]: Structural Learning Algorithm on Vague Environment is a genetic learning algorithm that uses the iterative approach to learn fuzzy rules.
- GFS-GP [52]: GP algorithm used to learn fuzzy rule-based classifiers.
- DTGA [19]: hybrid decision tree/ genetic algorithm discovering rules on small disjuncts.
- Ant Miner+ [47]: Ant Colony-based data miner to extract classification rules inspired by the research on the behavior of real ant colonies.
- RIPPER [22]: improvement of the efficient incremental reduced error pruning (IREP) algorithm.
- C45R [48]: C45Rules reads the decision tree or trees produced by C4.5 and generates a set of production rules from each tree and from all trees together.

Many different configurations have been established by the authors of each paper for the different techniques. The parameters used for the experimental study in all classification methods are the optimal values provided by their authors.

4.3. ICRM parameters

This section discusses the ICRM parameter settings to determine the influence of each parameter and their combinations in terms of accuracy and computational cost. ICRM has been implemented in the JCLEC software [56] and its main parameters are shown in Table 2. The three different phases of the algorithm have some parameters whose optimal values are shown in Table 2. Phase 1 requires the number of points in which the domains are initially explored. Phase 2 requires a selection pressure value in $(0,1]$, which determines the number of relevant attributes to explore regarding to the number of attributes. The genetic operator from Phase 2 requires two parameters: the population size and the number of generations of the MGA which optimizes the rules. Phase 3 also requires two parameters: the population size and the number of generations of the algorithm that optimizes the full classifier. The effect of these parameters has no relevance in the accuracy results, but is

[1] http://www.uco.es/grupos/kdis/kdiswiki/ICRM.

Table 1
General information about the data sets.

Data set	# Instances	# Attributes	# Classes
Appendicitis	106	7	2
Australian	690	14	2
Balance	625	4	3
Breast	286	9	2
Bupa	345	6	2
Car	1728	6	4
Chess	3196	36	2
Contraceptive	1473	9	3
Dermatology	366	34	6
Ecoli	336	7	8
Flare	1066	11	6
German	1000	20	2
Glass	214	9	7
Haberman	306	3	2
Heart	270	13	2
Ionosphere	351	33	2
Iris	150	4	3
Lymphography	148	18	4
Monk-2	432	6	2
New-thyroid	215	5	3
Page-blocks	5472	10	5
Penbased	10,992	16	10
Pima	768	8	2
Saheart	462	9	2
Satimage	6435	36	7
Segment	2310	19	7
Sonar	208	60	2
Tae	151	5	3
Thyroid	7200	21	3
Tic–tac–toe	958	9	2
Twonorm	7400	20	2
Vehicle	846	18	4
Wine	178	13	3
Yeast	1484	8	10
Zoo	101	16	7

Table 2
Parameter of ICRM algorithm.

	Parameter	Value
Phase 1	Number points	10
	Selection pressure	0.5
Phase 2	MGA population	10
	MGA generations	10
Phase 3	Fitting population	5
	Fitting generations	100

significant as to the computational time, increasing it considerably as long as the population size or the number of generations increases.

4.4. Experimental settings

The experiments consider the following metrics: the predictive accuracy, the number of rules of the classifier, the number of conditions per rule, the number of conditions per classifier, the complexity measure from Eq. (1), and the execution time. All experiments are repeated with 10 different seeds for stochastic methods and the average results are shown in the results tables. All algorithms are tested using 10-fold stratified cross validation [38,58] on all data sets. Experiments were run on a PC equipped with an Intel Core i7 quad-core processor running at 2.66 GHz and 12 GB of DDR3 memory. The host operating system was GNU/Linux Ubuntu 12.10 64 bit.

4.5. Statistical analysis

In order to analyse the results from the experiments, some non-parametric statistical tests are used to validate the results and conclusions [28,29]. To evaluate whether there are significant differences in the results of the different algorithms, the

Iman and Davenport test is performed. This useful non-parametric test, recommended by Demsar [23], is applied to rank the K algorithms over the N data sets (in our case there re 10 algorithms and 35 data sets) according to the F-distribution. When the Iman and Davenport test indicates that the results are significantly different, the Bonferroni–Dunn post hoc test [24] is used to find the significant differences occurring between algorithms in the multiple comparison. It assumes that the performance of two classifiers is significantly different if the corresponding average ranks differ by at least a critical difference value. Finally, the Wilcoxon rank-sum test [59] is used to perform multiple pairwise comparisons among the algorithms. The Wilcoxon rank-sum test statistic is the sum of the ranks for observations from one of the samples.

5. Results

This section discusses the experimental results and compares our method to different algorithms. In order to demonstrate the effectiveness and efficiency of our model, the accuracy, the execution time, and the different interpretability measures are evaluated.

5.1. Predictive accuracy

The predictive accuracy determines the ratio of successfully classified patterns by the model. Obviously, this is one of the most relevant factors when designing a classification algorithm, since it is desired to be as accurate as possible when making the predictions.

Table 3 shows the average accuracy results for the test folds and the ranking of the algorithms. The best values for accuracy, at a ranking value of 2.4286, are obtained by the MPLCS algorithm. The Iman and Davenport statistic for average accuracy distributed according to F-distribution with $K-1=9$ and $(K-1)(N-1)=306$ degrees of freedom is 18.8472. The test establishes an F-distribution value = 2.4656 for a significance level of alpha = 0.01. This value is lower than the

Table 3
Accuracy results from the test folds of the 10-fold cross-validation.

Accuracy	ICRM	C45R	RIPPER	MPLCS	AntMin+	CORE	ILGA	SLAVE	DTGA	GFS-GP
Appendicitis	0.8551	0.8517	0.8300	**0.8623**	0.8565	0.8415	0.8498	0.7063	0.8327	0.8518
Australian	0.8551	0.8517	0.8300	**0.8623**	0.8565	0.8415	0.8498	0.7063	0.8609	0.8449
Balance	0.7985	**0.8128**	0.5168	0.8043	0.7041	0.6460	0.5003	0.7460	0.7520	0.7234
Breast	0.7162	0.6970	0.6589	0.7186	0.7269	0.7234	0.7404	0.7080	0.7546	**0.7691**
Bupa	0.5961	0.6505	0.6232	0.6226	0.4154	0.5912	0.5540	0.5877	**0.6589**	0.5737
Car	0.8078	0.8794	0.8957	**0.9890**	0.8080	0.7982	0.7002	0.7002	0.8310	0.7882
Chess	0.7741	**0.9945**	0.9931	0.9927	0.8673	0.5438	0.9662	0.5222	0.9900	0.7650
Contraceptive	0.5119	0.5114	0.5169	**0.5484**	0.4626	0.4628	0.4417	0.4368	0.5275	0.4847
Dermatology	**0.9635**	0.9436	0.9300	0.9561	0.8710	0.3594	0.5877	0.9165	0.9408	0.6667
Ecoli	0.7774	0.7767	0.7482	0.8089	0.7409	0.6549	0.6352	**0.8444**	0.7709	0.7022
Flare	0.6857	0.6839	0.6720	0.7296	0.7049	0.6592	0.6273	0.0000	**0.7402**	0.5826
German	0.6920	0.7000	0.6610	**0.7250**	0.6840	0.6930	0.7100	0.7050	0.7120	0.7100
Glass	**0.6855**	0.6805	0.6397	0.6541	0.4936	0.5026	0.5013	0.5961	0.6117	0.5515
Haberman	**0.7459**	0.7119	0.4782	0.7349	0.7049	0.7394	0.7200	0.7178	0.7348	0.7156
Heart	0.7481	0.7901	0.7568	**0.8136**	0.8037	0.7198	0.6543	0.7975	0.7778	0.7370
Ionosphere	0.8578	0.9005	0.8606	0.9173	0.8550	0.5810	0.8206	**0.9259**	0.8890	0.8091
Iris	**0.9707**	0.9667	0.9378	0.9578	0.9356	0.9467	0.9289	0.9533	0.9600	0.9333
Lymphography	0.8031	0.7360	0.7475	**0.8074**	0.7420	0.6332	0.7769	0.6875	0.7425	0.7187
Monk-2	0.9727	**1.0000**	**1.0000**	0.9955	0.9727	0.9284	0.5437	0.9727	**1.0000**	0.9471
New-thyroid	0.9275	**0.9353**	0.9320	0.9242	0.8916	0.9136	0.9166	0.9013	0.9208	0.8374
Page-blocks	0.9502	0.9551	0.9646	0.9434	0.9376	0.9038	0.9278	0.9355	**0.9662**	0.9331
Penbased	0.7533	0.9490	**0.9648**	0.9420	0.4728	0.1569	0.5325	0.9322	0.9377	0.5330
Pima	0.7411	0.7183	0.6923	**0.7483**	0.7210	0.7297	0.7336	0.7337	0.7331	0.7293
Saheart	**0.6919**	0.6804	0.6002	0.6870	0.6795	0.6883	0.6730	0.6406	0.6773	0.6861
Satimage	0.7587	0.8406	0.8552	**0.8612**	0.7063	0.3978	0.7293	0.6566	0.8281	0.7346
Segment	0.9177	0.9515	**0.9537**	0.9532	0.8121	0.4294	0.8264	0.8784	0.9494	0.6571
Sonar	0.6939	0.7055	0.7528	**0.7733**	0.7463	0.5338	0.7080	0.7363	0.6576	0.6969
Tae	0.4918	0.4549	0.4929	**0.5706**	0.3312	0.4500	0.4251	0.4747	0.3783	0.4629
Thyroid	0.9879	**0.9960**	0.9947	0.9457	0.9826	0.7589	0.9405	0.9304	0.9950	0.9301
Tic–tac–toe	0.8993	0.8423	0.9760	**1.0000**	0.9833	0.6986	0.7811	0.6535	0.7880	0.7599
Twonorm	0.8607	0.8677	0.9142	0.8797	0.7878	0.6811	**0.9284**	0.8516	0.8315	0.7939
Vehicle	0.6312	0.6666	0.6915	0.7053	0.6332	0.3853	0.5853	0.6369	**0.7174**	0.5224
Wine	0.9356	**0.9490**	0.9377	0.9227	0.8690	0.9439	0.8893	0.9211	**0.9490**	0.8925
Yeast	0.4428	0.5594	0.5054	**0.5755**	0.4324	0.3767	0.4252	0.5013	0.4812	0.4628
Zoo	0.9583	0.9281	0.9269	**0.9656**	0.4990	0.9325	0.8587	0.0000	0.8561	0.8236
Avg. values	0.7845	0.8040	0.7843	**0.8256**	0.7340	0.6528	0.7140	0.7033	0.7930	0.7237
Avg. ranks	4.2857	3.9000	4.7714	**2.4286**	6.7143	7.7429	7.2000	6.6429	4.1286	7.1857

statistic critical value 18.8472. Thus, the test rejects the null hypothesis and therefore it can be said that there are statistically significant differences between the accuracy results of the algorithms.

Fig. 6 shows the application of the Bonferroni–Dunn test to accuracy for alpha = 0.01, whose critical difference is 2.3601. This graph is a bar chart, whose values are proportional to the mean rank obtained from each algorithm. The critical difference value is represented as a thicker horizontal line and those values that exceed this line are algorithms with significantly different results than the control algorithm, which is the one with the lowest rank value. Therefore, the algorithms right beyond the critical difference are significantly worse than the control algorithm. For accuracy, MPLCS is the best ranked algorithm and therefore, it is the control algorithm. Observing this figure, C45R, DTGA, ICRM and RIPPER achieve statistically similar accuracy results than MPLCS. On the other hand, AntMiner+, CORE, ILGA, SLAVE, and GFS-GP perform statistically significantly worse.

Table 4 shows the results of the Wilcoxon rank-sum test for the accuracy to compute multiple pairwise comparisons among the ICRM algorithm and the other methods. The p-values reported indicate that no significant differences can be found when comparing ICRM vs C45R, RIPPER, MPLCS, and DTGA. However, these differences are significant when compared to AntMiner+, CORE, ILGA, SLAVE, and GFS-GP, achieving p-values lower than 0.01, i.e., a statistical confidence higher than 99%.

5.2. Number of rules

The number of rules determines the complexity of the model. The greater the number of rules, the greater probability of conflicts between them and the greater difficulty in understanding the conditions necessary to predict a particular class.

Table 5 shows the average number of rules of the classifiers and the ranking of the algorithms. The best values are the lowest number of rules, at a ranking value of 1.63, ICRM performs better than the others and it is followed by CORE and ANTMiner+. The Iman and Davenport statistic for the number of rules distributed according to an F-distribution with $K-1=9$ and $(K-1)(N-1)=306$ degrees of freedom is 104.4333. The test establishes an F-distribution value = 2.4656 for a significance level of alpha = 0.01. This value is lower than the statistic critical value 104.4333. Thus, the test rejects the null hypothesis and therefore it can be said that there are statistically significant differences between the number of rules of the algorithms.

Fig. 7 shows the application of the Bonferroni–Dunn test to the number of rules for alpha = 0.01, whose critical difference is 2.3601. The algorithms right beyond the critical difference from the ICRM rank are significantly worse than ICRM with a confidence level higher than 99%. Observing this figure, all the algorithms but ANTMiner + and CORE are significantly worse than ICRM, whereas MPLCS is borderline.

Table 6 shows the results of the Wilcoxon rank-sum test for the number of rules to compute multiple pairwise comparisons among the proposal and the other methods. The p-values reported indicate significant differences in the number of rules of the algorithms with a confidence level higher than 99%. All methods are shown to obtain classifiers with significantly higher number of rules.

5.3. Conditions per rule

The number of conditions in a rule determines the length and complexity of a rule. The greater the number of conditions, the lower the interpretability of the rule.

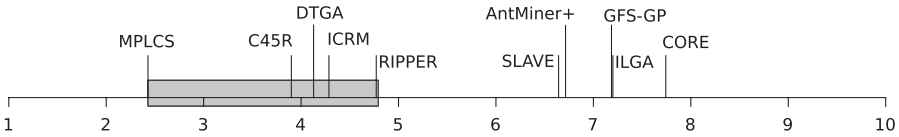


Fig. 6. Bonferroni-Dunn for accuracy.

Table 4
Wilcoxon test for accuracy.

ICRM vs	R^+	R^-	p-value
C45R	224.0	406.0	$\geqslant$0.2
RIPPER	296.0	334.0	$\geqslant$0.2
MPLCS	103.0	527.0	$\geqslant$0.2
AntMiner+	471.5	123.5	0.002256
CORE	618.0	12.0	4.074E−9
ILGA	555.0	75.0	2.612E−5
SLAVE	465.0	130.0	0.003394
DTGA	272.0	358.0	$\geqslant$0.2
GFS-GP	588.0	42.0	6.602E−7

Table 5
Number of rules.

# Rules	ICRM	C45R	RIPPER	MPLCS	AntMin+	CORE	ILGA	SLAVE	DTGA	GFS-GP
Appendicitis	**2.00**	11.07	23.53	4.83	5.50	4.03	30.00	6.97	3.00	219.40
Australian	**2.00**	11.07	23.53	4.83	5.50	4.03	30.00	6.97	27.70	197.75
Balance	**3.00**	34.80	37.47	14.83	6.13	4.23	30.00	24.10	53.70	83.10
Breast	2.00	14.60	35.60	17.90	7.50	5.20	30.00	**1.00**	11.00	175.25
Bupa	**2.00**	8.57	23.80	6.60	**2.00**	3.43	30.00	6.30	30.10	174.90
Car	4.00	76.87	68.67	19.73	25.13	3.33	30.00	**1.00**	131.80	82.55
Chess	2.00	30.40	22.50	9.70	16.20	14.00	30.00	**1.00**	34.00	153.85
Contraceptive	**3.00**	31.00	64.60	5.67	3.13	8.93	30.00	44.90	138.80	82.85
Dermatology	**6.00**	9.20	14.27	7.77	6.70	6.47	30.00	10.57	10.30	118.10
Ecoli	8.00	11.40	35.50	8.90	7.63	**6.33**	30.00	12.77	27.70	79.10
Flare	6.00	27.90	101.60	16.67	20.30	4.80	30.00	**0.00**	49.60	110.65
German	**2.00**	27.20	36.30	12.40	14.80	3.30	30.00	8.80	84.10	71.05
Glass	7.00	10.07	22.20	8.27	**4.57**	7.03	30.00	16.27	39.50	176.25
Haberman	**2.00**	4.30	17.67	4.37	**2.00**	4.13	30.00	5.63	2.60	230.40
Heart	**2.00**	10.83	13.97	6.90	4.87	5.47	30.00	7.47	17.70	100.30
Ionosphere	**2.00**	12.30	11.60	4.70	9.80	2.30	30.00	4.30	14.80	153.25
Iris	3.00	5.00	6.30	4.07	**2.97**	3.63	30.00	3.00	4.70	85.95
Lymphography	4.00	11.60	13.10	7.57	**3.70**	6.43	30.00	4.37	17.90	84.75
Monk-2	**2.00**	6.00	4.00	4.00	3.00	3.57	30.00	3.00	5.00	121.05
New-thyroid	**3.00**	6.80	6.93	5.43	3.57	3.30	30.00	5.50	8.70	185.65
Page-blocks	**5.00**	22.50	51.23	7.10	14.70	5.43	30.00	9.90	51.60	65.85
Penbased	10.00	125.50	109.90	47.70	82.70	**6.00**	30.00	40.20	201.10	56.00
Pima	**2.00**	8.40	25.10	5.43	12.30	3.13	30.00	8.57	21.10	156.10
Saheart	**2.00**	6.70	24.57	6.23	3.70	6.37	30.00	11.30	11.90	112.95
Satimage	7.00	75.10	109.00	30.90	69.70	**1.00**	30.00	36.60	334.30	85.45
Segment	7.00	26.30	33.20	16.90	35.00	**4.40**	30.00	18.80	50.60	137.75
Sonar	2.00	8.70	8.13	6.17	4.83	**1.00**	30.00	8.37	20.20	97.25
Tae	3.00	7.57	28.43	5.57	**2.00**	6.47	30.00	11.03	27.30	224.60
Thyroid	**3.00**	10.33	11.43	4.03	24.17	3.97	30.00	6.93	19.60	108.30
Tic–tac–toe	2.00	50.40	16.00	9.00	9.00	3.43	30.00	**1.00**	83.20	211.15
Twonorm	**2.00**	57.70	56.60	23.20	39.10	7.10	30.00	35.80	287.70	96.30
Vehicle	**4.00**	19.80	46.70	15.17	25.97	6.77	30.00	34.77	69.30	156.50
Wine	**3.00**	5.00	5.47	4.47	3.77	3.07	30.00	4.03	5.70	96.35
Yeast	10.00	36.40	140.10	13.70	36.00	**7.20**	30.00	23.20	174.70	121.70
Zoo	7.00	8.70	9.10	7.00	4.20	7.03	30.00	**0.00**	13.00	80.15
Avg. values	**3.89**	23.72	35.95	10.79	14.92	5.04	30.00	12.13	59.54	128.36
Avg. ranks	**1.63**	6.23	7.44	3.99	3.83	2.54	7.54	4.20	7.94	9.66

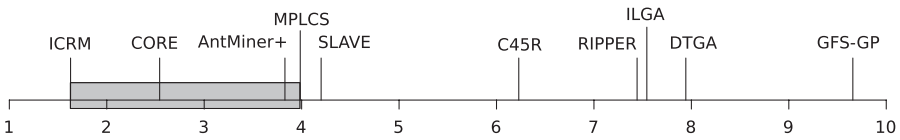


Fig. 7. Bonferroni-Dunn for the number of rules.

Table 6
Wilcoxon test for the number of rules.

ICRM vs	R^+	R^-	p-value
C45R	630.0	0.0	5.82E−11
RIPPER	630.0	0.0	5.82E−11
MPLCS	595.0	0.0	1.16E−10
AntMiner+	577.0	53.0	2.584E−6
CORE	464.0	166.0	0.013682
ILGA	630.0	0.0	5.82E−11
SLAVE	533.5	61.5	1.325E−5
DTGA	630.0	0.0	5.82E−11
GFS-GP	630.0	0.0	5.82E−11

Table 7 shows the average number of conditions per rule of the classifiers and the ranking of the algorithms. The best values are the lowest number of conditions, at a ranking value of 1.31, our method performs better than the others and it is followed distantly by AntMiner+. The Iman and Davenport statistic for the number of conditions per rule distributed

Table 7
Conditions per rule.

# Cond/ Rule	ICRM	C45R	RIPPER	MPLCS	AntMin+	CORE	ILGA	SLAVE	DTGA	GFS-GP
Appendicitis	**0.50**	3.15	3.22	3.49	4.00	2.21	8.36	3.52	1.43	3.45
Australian	**0.50**	3.15	3.22	3.49	4.00	2.21	8.36	3.52	3.66	2.86
Balance	1.27	3.12	3.33	2.39	2.49	**0.99**	1.95	2.75	3.44	3.73
Breast	0.55	3.40	3.28	5.63	1.01	1.86	7.51	**0.00**	1.59	2.54
Bupa	**0.90**	2.48	3.28	3.25	1.50	1.83	2.81	3.40	3.23	3.97
Car	1.95	3.98	4.40	3.60	3.59	2.42	5.20	**0.00**	2.99	4.48
Chess	1.00	3.74	3.37	5.82	1.07	10.64	29.18	**0.00**	9.05	3.24
Contraceptive	**1.00**	5.04	5.93	2.92	3.14	1.76	4.09	4.19	6.17	5.99
Dermatology	**1.43**	2.56	2.44	3.38	8.30	9.15	14.12	1.98	4.73	3.25
Ecoli	**1.89**	2.98	2.43	2.47	2.82	2.69	3.18	3.30	5.13	3.75
Flare	1.23	3.02	5.20	6.20	4.71	3.42	9.14	**0.00**	3.00	3.58
German	**0.50**	3.81	3.79	8.73	1.36	3.41	12.75	3.25	3.18	7.84
Glass	**2.06**	3.17	2.46	2.67	2.65	3.21	4.16	3.46	5.68	3.46
Haberman	**0.50**	1.31	2.63	1.54	0.65	0.94	1.47	2.19	1.80	1.80
Heart	**1.00**	2.68	2.52	3.97	4.07	2.03	5.48	3.59	2.73	4.09
Ionosphere	**1.00**	2.28	1.70	4.60	1.11	6.20	13.69	3.35	3.03	3.31
Iris	1.00	1.14	1.53	**0.87**	1.29	1.28	1.87	1.36	2.17	66.75
Lymphography	**1.25**	2.06	2.14	4.61	3.61	3.96	13.13	1.97	2.91	3.66
Monk-2	1.00	1.67	1.25	1.31	2.31	0.98	2.87	**0.67**	2.10	2.55
New-thyroid	**1.29**	1.87	1.55	1.69	1.51	2.38	2.34	1.67	2.35	21.51
Page-blocks	**1.43**	3.46	3.31	2.56	3.36	2.72	4.40	3.61	5.03	3.01
Penbased	1.70	5.67	4.05	4.55	**1.05**	3.61	6.61	7.21	5.11	2.99
Pima	**0.50**	2.34	4.06	3.35	4.72	1.50	3.60	3.42	3.78	3.76
Saheart	**0.50**	1.97	3.48	3.38	3.94	1.70	4.40	3.83	2.78	3.85
Satimage	1.39	5.76	4.75	4.79	**1.00**	2.00	16.55	6.54	10.46	6.55
Segment	1.41	3.78	2.96	3.42	**1.03**	4.26	8.28	4.22	4.65	3.75
Sonar	**0.89**	2.66	1.93	6.43	14.82	15.00	25.09	6.07	39.26	3.97
Tae	**1.17**	2.17	2.91	2.50	**1.17**	1.37	2.27	2.94	3.84	3.15
Thyroid	**1.38**	2.70	3.97	1.94	10.34	2.98	9.43	3.12	5.53	4.09
Tic–tac–toe	0.50	3.31	3.68	2.67	2.67	1.69	7.60	**0.00**	3.72	3.46
Twonorm	**0.55**	5.14	4.92	5.47	1.00	4.32	9.42	5.74	5.10	5.68
Vehicle	**1.25**	3.60	3.06	3.72	6.60	3.46	8.03	5.96	4.24	3.50
Wine	**1.47**	1.62	1.72	2.18	2.26	4.34	5.71	3.14	2.60	4.11
Yeast	**1.18**	4.74	4.21	2.49	1.53	3.31	3.77	3.56	6.08	3.55
Zoo	1.14	2.11	1.94	1.39	3.26	4.85	13.53	**0.00**	5.31	14.30
Avg. values	**1.09**	3.08	3.16	3.53	3.25	3.45	8.01	2.96	5.08	6.44
Avg. ranks	**1.31**	5.11	5.29	5.39	4.94	4.57	8.63	5.50	7.16	7.10

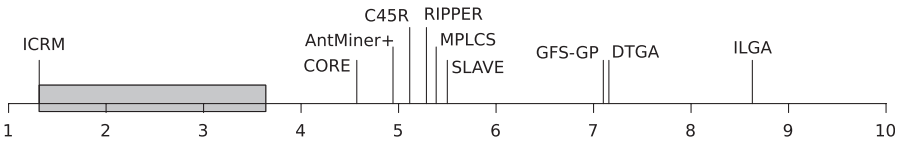


Fig. 8. Bonferroni–Dunn for the number of conditions per rule.

according to an F-distribution with $K-1=9$ and $(K-1)(N-1)=306$ degrees of freedom is 23.8287. The test establishes an F-distribution value = 2.4656 for a significance level of alpha = 0.01. Thus, the test rejects the null hypothesis and therefore it can be said that there are statistically significant differences between the number of conditions per rule of the algorithms.

Fig. 8 shows the application of the Bonferroni-Dunn test to the number of conditions per rule for alpha = 0.01, whose critical difference is 2.3601. The algorithms right beyond the critical difference from the ICRM value are significantly worse than ICRM. All the algorithms show significant differences compared to our proposal, which is the one that obtains the best results. Moreover, our proposal employs an attribute only once at most, performing a selection of relevant features. Looking at the results of Table 7, there are significant differences, e.g., between our proposal and ILGA over the two classes and 60 attributes Sonar data set. While our proposal uses on average 0.89 conditions per rule, i.e., almost one rule with one condition and another default class rule, ILGA requires 25.09 conditions per rule. Therefore, ICRM considers only one of the 60 attributes as relevant to discriminate between the two classes, while ILGA considers 25. In this case the predictive accuracy difference between ICRM and ILGA is only 1.4%.

Table 8 shows the results of the Wilcoxon rank-sum test for the number of conditions per rule to compute multiple pairwise comparisons among the proposal and the other methods. The p-values reported indicate significant differences with a confidence level higher than 99% from ICRM versus all.

Table 8
Wilcoxon test for the number of conditions per rule.

ICRM vs	R^+	R^-	p-value
C45R	630.0	0.0	5.82E−11
RIPPER	630.0	0.0	5.82E−11
MPLCS	629.0	1.0	1.16E−10
AntMiner+	567.0	28.0	1.726E−7
CORE	625.5	4.5	4.94E−10
ILGA	630.0	0.0	5.82E−11
SLAVE	574.5	55.5	3.451E−6
DTGA	630.0	0.0	5.82E−11
GFS-GP	630.0	0.0	5.82E−11

5.4. Conditions per classifier

The total number of conditions of the classifier determines its overall complexity, regardless of the number of rules, since it measures the number of conditions that must be evaluated to classify instances. The greater the number of conditions, the lower the interpretability of the classifier and the more time required to evaluate the conditions.

Table 9 shows the average number of conditions of the classifiers and the ranking of the algorithms. The best ranking values are the lowest number of conditions, at an average ranking value of 1.27, our method performs better than the others and it is followed distantly by CORE. The Iman and Davenport statistic for the number of conditions per classifier distributed according to an F-distribution with $K-1=9$ and $(K-1)(N-1)=306$ degrees of freedom is 86.5088. The test establishes an F-distribution value = 2.4656 for a significance level of alpha = 0.01. Thus, the test rejects the null hypothesis and therefore it can be said that there are statistically significant differences between the number of conditions per classifier of the algorithms.

Table 9
Conditions per classifier.

# Cond/ Clrf	ICRM	C45R	RIPPER	MPLCS	AntMin+	CORE	ILGA	SLAVE	DTGA	GFS-GP
Appendicitis	**1.00**	35.17	75.93	17.20	23.20	10.50	250.83	24.70	4.70	761.60
Australian	**1.00**	35.17	75.93	17.20	23.20	10.50	250.83	24.70	102.10	544.70
Balance	**3.80**	108.60	124.83	35.27	15.80	4.37	58.40	66.53	182.50	316.80
Breast	1.10	49.70	117.10	101.40	7.60	10.20	225.30	**0.00**	19.20	438.70
Bupa	**1.80**	21.37	78.47	21.60	3.00	6.03	84.17	21.60	97.80	619.50
Car	7.80	306.20	302.70	71.30	90.23	8.00	155.90	**0.67**	393.60	371.20
Chess	2.00	113.80	76.20	56.90	17.40	149.00	875.50	**0.00**	301.90	422.50
Contraceptive	**3.00**	156.43	382.50	17.27	11.73	16.77	122.70	188.00	846.70	442.10
Dermatology	**8.60**	23.60	34.90	26.23	55.17	62.27	423.73	21.20	47.20	381.10
Ecoli	**15.14**	34.10	86.40	21.80	21.60	17.07	95.43	42.03	129.90	322.70
Flare	7.40	84.57	528.40	103.80	95.80	16.23	274.10	**0.00**	147.50	381.50
German	**1.00**	103.80	137.30	108.30	20.10	12.20	382.40	29.50	270.40	301.30
Glass	14.42	31.87	54.67	22.07	**12.23**	22.70	124.67	56.47	204.90	601.10
Haberman	**1.00**	5.80	46.60	6.70	1.30	4.03	44.10	12.30	4.70	728.30
Heart	**2.00**	29.40	35.30	27.47	19.93	12.97	164.50	27.07	48.60	446.20
Ionosphere	**2.00**	28.30	19.70	20.70	10.90	13.70	410.80	13.40	45.10	488.10
Iris	**3.00**	5.70	9.70	3.57	3.83	4.70	55.97	4.07	10.30	539.80
Lymphography	**5.00**	23.97	28.10	34.97	13.47	26.80	393.87	8.40	51.90	309.90
Monk-2	**2.00**	10.00	5.00	5.23	6.93	4.20	86.07	**2.00**	10.50	302.20
New-thyroid	**3.86**	12.70	10.73	9.27	5.33	7.80	70.17	9.10	20.60	770.90
Page-blocks	**7.17**	78.20	169.47	18.50	49.40	14.67	131.97	35.77	253.20	196.10
Penbased	**17.00**	710.90	444.90	220.50	86.70	21.70	198.40	291.20	1,025.9	170.60
Pima	**1.00**	20.10	101.90	18.30	58.03	5.00	108.13	29.83	83.20	563.50
Saheart	**1.00**	13.37	85.53	21.10	14.90	12.03	132.10	43.30	34.20	416.40
Satimage	9.70	433.20	517.70	147.90	69.70	**2.00**	496.40	239.90	3,433.7	439.30
Segment	**9.90**	99.70	98.10	58.30	36.00	18.40	248.40	79.30	232.40	494.30
Sonar	**1.78**	23.10	15.80	39.73	73.73	15.00	752.67	51.13	651.90	443.80
Tae	3.50	16.53	83.13	13.83	**2.33**	9.20	68.23	32.40	106.40	704.20
Thyroid	**4.14**	27.80	45.60	7.77	250.17	14.70	282.83	23.07	106.80	371.20
Tic–tac–toe	1.00	167.50	58.93	24.00	24.00	6.63	227.87	**0.67**	310.40	661.90
Twonorm	**1.10**	296.20	278.30	126.70	39.10	30.20	282.70	203.60	1,467.1	387.40
Vehicle	**5.00**	71.33	143.50	56.70	171.40	28.10	240.97	208.17	297.00	570.50
Wine	**4.42**	8.10	9.37	9.73	8.50	13.27	171.23	12.60	16.10	440.80
Yeast	**11.80**	173.40	590.00	33.90	55.00	23.30	113.20	82.20	1,059.5	456.00
Zoo	8.00	18.40	17.73	9.73	13.93	34.17	405.77	**0.00**	67.30	442.30
Avg. values	**4.96**	96.52	139.73	43.86	40.33	19.10	240.29	53.85	345.29	464.24
Avg. ranks	**1.27**	5.63	6.74	4.37	3.76	3.23	8.23	4.34	8.06	9.37

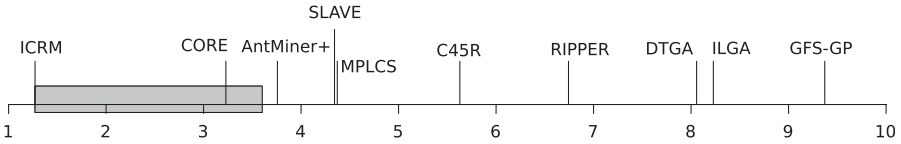


Fig. 9. Bonferroni-Dunn for the number of conditions per classifier.

Table 10
Wilcoxon test for the total number of conditions.

ICRM vs	R^+	R^-	p-value
C45R	630.0	0.0	5.82E−11
RIPPER	630.0	0.0	5.82E−11
MPLCS	630.0	0.0	5.82E−11
AntMiner+	621.0	9.0	1.921E−9
CORE	616.0	14.0	6.402E−9
ILGA	630.0	0.0	5.82E−11
SLAVE	563.0	32.0	3.218E−7
DTGA	630.0	0.0	5.82E−11
GFS-GP	630.0	0.0	5.82E−11

Fig. 9 shows the application of the Bonferroni–Dunn test to the number of conditions of the classifier for alpha = 0.01, whose critical difference is 2.3601. The algorithms right beyond the critical difference from the ICRM value are significantly worse than ICRM, which is again the best algorithm and the most interpretable regarding the total number of conditions of the rule-based classifier. Looking at the values from Table 9, there are significant differences with all the algorithms but CORE regarding the average number of conditions of the classifiers.

Table 10 shows the results of the Wilcoxon rank-sum test for the total number of conditions to compute multiple pairwise comparisons among the proposal and the other methods. The p-values reported indicate significant differences with a confidence level higher than 99% from ICRM with all the other methods.

5.5. Complexity

The complexity metric provided by Nauck [43] permits to compute an interpretability value that combines all the previous results.

Table 11 shows the average complexity values and the ranking of the algorithms. The best values are the higher complexity values, at a ranking value of 1.16, our method performs best. The Iman and Davenport statistic for the complexity distributed according to an F-distribution with $K-1=9$ and $(K-1)(N-1)=306$ degrees of freedom is 74.2908. The test establishes an F-distribution value = 2.4656 for a significance level of alpha = 0.01. Thus, the null hypothesis is rejected and there are significant complexity differences.

Fig. 10 shows the application of the Bonferroni–Dunn test to the complexity for alpha = 0.01, whose critical difference is 2.3601. The algorithms right beyond the critical difference from the ICRM value have significantly worse complexity values than ICRM. Observing this figure, all the algorithms but CORE have significantly worse results than ICRM regarding to the complexity value, i.e., worse interpretability. Interestingly, the results of CORE show it to be the second most interpretable algorithm in the comparison, but its accuracy results are among the worst.

Table 12 shows the results of the Wilcoxon rank-sum test for the complexity metric to compute multiple pairwise comparisons among the proposal and the other methods. The p-values reported indicate significant differences with a confidence level higher than 99% from ICRM versus all. These are the best results achieved by the ICRM method among all the metrics considered.

5.6. Execution time

The execution times for the different algorithms and data sets are shown in Table 13. These values represent the time (in s) taken by an algorithm to learn from the training data, build a classifier and perform the classification over both the training and test data. The best values are the lowest running times, at a ranking value of 2.21, C45R is the fastest algorithm followed by RIPPER. The Iman and Davenport statistic for the execution time distributed according to an F-distribution with $K-1=9$ and $(K-1)(N-1)=306$ degrees of freedom is 86.6229. The test establishes an F-distribution value = 2.4656 for a significance level of alpha = 0.01. Thus, the test rejects the null hypothesis and therefore it can be said that there are significant differences between the execution times of the algorithms.

Fig. 11 shows the application of the Bonferroni–Dunn test to the execution time for alpha = 0.01, whose critical difference is 2.3601. The algorithms right beyond the critical difference from the C45R value are significantly slower. Observing this

Table 11
Complexity.

Complexity	ICRM	C45R	RIPPER	MPLCS	AntMin+	CORE	ILGA	SLAVE	DTGA	GFS-GP
Appendicitis	**2.00**	0.06	0.03	0.12	0.09	0.19	0.01	0.08	0.43	0.00
Australian	**2.00**	0.06	0.03	0.12	0.09	0.19	0.01	0.08	0.02	0.00
Balance	**0.79**	0.03	0.02	0.09	0.19	0.69	0.05	0.05	0.02	0.01
Breast	**1.82**	0.04	0.02	0.02	0.26	0.20	0.01	0.00	0.10	0.00
Bupa	**1.11**	0.09	0.03	0.09	0.67	0.33	0.02	0.09	0.02	0.00
Car	0.51	0.01	0.01	0.06	0.04	0.50	0.03	**6.00**	0.01	0.01
Chess	**1.00**	0.02	0.03	0.04	0.11	0.01	0.00	0.00	0.01	0.00
Contraceptive	**1.00**	0.02	0.01	0.17	0.26	0.18	0.02	0.02	0.00	0.01
Dermatology	**0.70**	0.25	0.17	0.23	0.11	0.10	0.01	0.28	0.13	0.02
Ecoli	**0.53**	0.23	0.09	0.37	0.37	0.47	0.08	0.19	0.06	0.02
Flare	**0.81**	0.07	0.01	0.06	0.06	0.37	0.02	0.00	0.04	0.02
German	**2.00**	0.02	0.01	0.02	0.10	0.16	0.01	0.07	0.01	0.01
Glass	0.49	0.22	0.13	0.32	**0.57**	0.31	0.06	0.12	0.03	0.01
Haberman	**2.00**	0.34	0.04	0.30	1.54	0.50	0.05	0.16	0.43	0.00
Heart	**1.00**	0.07	0.06	0.07	0.10	0.15	0.01	0.07	0.04	0.00
Ionosphere	**1.00**	0.07	0.10	0.10	0.18	0.15	0.00	0.15	0.04	0.00
Iris	**1.00**	0.53	0.31	0.84	0.78	0.64	0.05	0.74	0.29	0.01
Lymphography	**0.80**	0.17	0.14	0.11	0.30	0.15	0.01	0.48	0.08	0.01
Monk-2	**1.00**	0.20	0.40	0.38	0.29	0.48	0.02	**1.00**	0.19	0.01
New-thyroid	**0.78**	0.24	0.28	0.32	0.56	0.38	0.04	0.33	0.15	0.00
Page-blocks	**0.70**	0.06	0.03	0.27	0.10	0.34	0.04	0.14	0.02	0.03
Penbased	**0.59**	0.01	0.02	0.05	0.12	0.46	0.05	0.03	0.01	0.06
Pima	**2.00**	0.10	0.02	0.11	0.03	0.40	0.02	0.07	0.02	0.00
Saheart	**2.00**	0.15	0.02	0.09	0.13	0.17	0.02	0.05	0.06	0.00
Satimage	0.72	0.02	0.01	0.05	0.10	**3.50**	0.01	0.03	0.00	0.02
Segment	**0.71**	0.07	0.07	0.12	0.19	0.38	0.03	0.09	0.03	0.01
Sonar	**1.12**	0.09	0.13	0.05	0.03	0.13	0.00	0.04	0.00	0.00
Tae	0.86	0.18	0.04	0.22	**1.29**	0.33	0.04	0.09	0.03	0.00
Thyroid	**0.72**	0.11	0.07	0.39	0.01	0.20	0.01	0.13	0.03	0.01
Tic–tac–toe	2.00	0.01	0.03	0.08	0.08	0.30	0.01	**3.00**	0.01	0.00
Twonorm	**1.82**	0.01	0.01	0.02	0.05	0.07	0.01	0.01	0.00	0.01
Vehicle	**0.80**	0.06	0.03	0.07	0.02	0.14	0.02	0.02	0.01	0.01
Wine	**0.68**	0.37	0.32	0.31	0.35	0.23	0.02	0.24	0.19	0.01
Yeast	**0.85**	0.06	0.02	0.29	0.18	0.43	0.09	0.12	0.01	0.02
Zoo	**0.88**	0.38	0.39	0.72	0.50	0.20	0.02	0.00	0.10	0.02
Avg. values	**1.11**	0.13	0.09	0.19	0.28	0.38	0.03	0.40	0.07	0.01
Avg. ranks	**1.16**	5.56	6.76	4.19	3.73	3.13	8.09	5.37	7.89	9.14

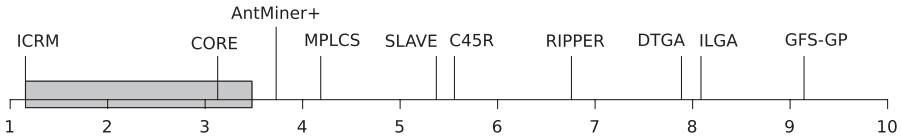


Fig. 10. Bonferroni–Dunn for complexity.

Table 12
Wilcoxon test for complexity.

ICRM vs	R^+	R^-	p-value
C45R	630.0	0.0	5.82E−11
RIPPER	630.0	0.0	5.82E−11
MPLCS	630.0	0.0	5.82E−11
AntMiner+	622.0	8.0	1.455E−9
CORE	595.0	35.0	2.508E−7
ILGA	630.0	0.0	5.82E−11
SLAVE	538.5	56.5	7.672E−6
DTGA	630.0	0.0	5.82E−11
GFS-GP	630.0	0.0	5.82E−11

figure, C45R, RIPPER, DTGA and ICRM have no statistically significant differentes in the execution time of the algorithsm. On the other hand, AntMiner+, SLAVE, MPLCS, CORE, ILGA and GFS-GP do perform significantly slower.

Table 14 shows the results of the Wilcoxon rank-sum test for the execution time to compute multiple pairwise comparisons among the proposal and the other methods. The p-values reported indicate significant differences with a confidence level higher than 99% from ICRM with MPLCS, AntMiner+, CORE, ILGA, SLAVE, and GFS-GP.

Table 13
Execution time.

Time	ICRM	C45R	RIPPER	MPLCS	AntMin+	CORE	ILGA	SLAVE	DTGA	GFS-GP
Appendicitis	2.26	1.00	1.19	18.56	55.93	35.41	56.89	32.26	**0.11**	129.11
Australian	2.26	**1.00**	1.19	18.56	55.93	35.41	56.89	32.26	**1.00**	683.78
Balance	1.15	1.07	**0.67**	19.78	26.11	12.89	9.07	26.56	2.56	412.22
Breast	**0.09**	0.44	0.44	32.33	0.33	8.11	10.44	2.33	0.44	233.44
Bupa	**0.54**	0.59	0.56	8.22	5.56	10.63	8.63	8.00	0.78	482.11
Car	**0.33**	1.04	5.07	69.74	458.85	109.96	29.26	10.81	5.44	1,222.6
Chess	**1.60**	2.33	2.89	470.00	2.00	551.56	1,402.4	67.56	6.22	2,643.9
Contraceptive	**2.94**	11.52	3.59	45.22	66.74	42.89	55.15	194.19	9.00	1,506.0
Dermatology	134.59	0.59	**0.48**	48.70	118.44	60.96	144.59	51.81	0.78	304.11
Ecoli	26.49	0.59	**0.56**	13.30	32.07	22.70	15.41	29.63	1.67	381.89
Flare	1.44	**0.81**	3.52	38.37	178.56	59.67	73.11	17.67	2.78	992.78
German	0.78	1.89	2.33	109.11	**0.67**	134.78	167.44	66.22	1.67	689.89
Glass	24.45	**0.52**	**0.52**	9.30	11.74	13.96	15.52	26.52	2.22	206.44
Haberman	**0.12**	0.30	0.33	4.89	5.48	8.00	3.22	3.70	0.22	441.44
Heart	1.64	0.63	**0.48**	13.70	33.04	13.04	27.00	12.19	0.67	358.44
Ionosphere	2.28	**0.56**	1.00	102.44	1.00	55.11	89.11	16.00	0.67	415.89
Iris	0.71	0.15	**0.11**	4.11	4.07	9.33	2.93	2.41	**0.11**	132.00
Lymphography	3.54	0.30	**0.26**	6.30	12.15	21.15	19.96	10.67	0.56	142.44
Monk-2	0.27	0.33	**0.19**	10.74	12.22	14.44	7.41	4.37	0.22	305.22
New-thyroid	1.49	0.22	**0.19**	5.41	7.52	10.81	5.70	5.11	0.33	265.67
Page-blocks	46.22	**10.26**	14.67	191.30	616.30	225.63	469.74	352.11	22.33	3,044.2
Penbased	62.62	40.78	21.11	9,343.0	**19.11**	1,011.3	4,261.0	4,976.7	62.67	4,969.7
Pima	1.36	**0.96**	1.48	19.11	115.33	24.15	35.63	22.59	3.22	916.78
Saheart	1.08	**0.67**	1.04	11.48	28.78	13.63	33.96	20.74	**0.67**	533.89
Satimage	123.52	39.67	**27.78**	7,040.4	95.67	924.44	2811.1	2,886.4	142.78	6,984.8
Segment	14.70	**3.11**	7.22	794.00	5.33	301.56	508.00	501.11	8.78	1,988.4
Sonar	98.20	**0.70**	0.96	34.63	78.44	34.11	152.74	21.96	3.44	197.00
Tae	0.65	0.33	**0.26**	4.52	2.78	5.85	4.19	6.26	0.56	172.11
Thyroid	71.61	4.37	**4.15**	629.81	9,671.8	893.44	813.78	331.07	9.78	4,988.6
Tic–tac–toe	**0.08**	0.81	0.70	20.37	151.52	27.37	56.07	6.04	5.89	1,096.9
Twonorm	**4.74**	36.33	270.67	2,757.8	7.67	913.33	3761.78	1,454.2	9.67	7,898.0
Vehicle	32.74	**2.00**	2.63	85.85	274.22	73.67	158.52	204.15	2.78	880.22
Wine	13.86	**0.26**	**0.26**	9.41	12.78	16.48	14.59	7.19	0.33	167.56
Yeast	6.08	11.67	**4.56**	232.11	18.17	83.89	97.22	234.22	16.33	1,221.1
Zoo	2.98	**0.19**	**0.19**	3.07	9.59	14.93	16.70	5.19	0.56	95.67
Avg. values	19.70	**5.09**	10.95	635.02	348.45	165.56	439.86	332.86	9.35	1,345.8
Avg. ranks	3.60	**2.21**	2.24	6.51	6.39	7.06	7.54	6.41	3.13	9.90

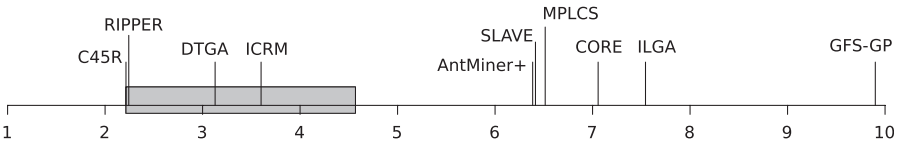


Fig. 11. Bonferroni–Dunn for the execution time.

Table 14
Wilcoxon test for the execution time.

ICRM vs	R^+	R^-	p-value
C45R	139.0	491.0	$\geqslant$0.2
RIPPER	150.0	480.0	$\geqslant$0.2
MPLCS	551.0	79.0	3.796E−5
AntMiner+	500.0	130.0	0.001843
CORE	570.0	60.0	5.686E−6
ILGA	609.0	21.0	2.602E−8
SLAVE	568.0	62.0	7.052E−6
DTGA	234.0	396.0	$\geqslant$0.2
GFS-GP	630.0	0.0	5.82E−11

5.7. Overall comparison

Table 15 summarizes the ranks obtained by the algorithms using the different metrics. The bottom rows show the average rank values and the meta rank (rank of the ranks). This table shows the great overall performance of the ICRM algorithm and

Table 15
Ranks and Meta Rank (Rank of the Ranks).

Meta Rank	ICRM	C45R	RIPPER	MPLCS	AntMin+	CORE	ILGA	SLAVE	DTGA	GFS-GP
Accuracy	4.29	3.90	4.77	**2.43**	6.71	7.74	7.20	6.64	4.13	7.19
Rules	**1.63**	6.23	7.44	3.99	3.83	2.54	7.54	4.20	7.94	9.66
C/R	**1.31**	5.11	5.29	5.39	4.94	4.57	8.63	5.50	7.16	7.10
Conditions	**1.27**	5.63	6.74	4.37	3.76	3.23	8.23	4.34	8.06	9.37
Complexity	**1.16**	5.56	6.76	4.19	3.73	3.13	8.09	5.37	7.89	9.14
Time	3.60	**2.21**	2.24	6.51	6.39	7.06	7.54	6.41	3.13	9.90
Avg. ranks	**2.21**	4.77	5.54	4.48	4.89	4.71	7.87	5.41	6.38	8.73
Meta rank	**2.00**	4.17	5.50	4.50	4.00	4.33	9.00	5.50	6.67	9.33

the bad performance of GFS-GP and ILGA. Ripper and C45R are the fastest algorithms since they are not evolutionary, but they provide classifiers with many rules and significantly more complex. MPLCS provides the most accurate classifier but its execution time is significantly slower. AntMiner + and CORE provide a relatively low number of rules but they run slower. ILGA and GFS-GP are definitely the worst algorithms from the comparison. Specifically, GFS-GP achieves the worst ranks over four of the metrics considered. Our proposal, ICRM, obtains the best rankings regarding the number of rules, the number of conditions per rule, the number of conditions per classifier, and complexity. The accuracy, the complexity of the classifier and the execution time are conflicting objectives on which ICRM has shown best overall performance.

6. Conclusion

In this paper we have proposed an interpretable classification rule mining (ICRM) algorithm, which is an interpretable and efficient rule-based evolutionary programming classification algorithm. The algorithm solves the cooperation–competition problem by dealing with the interaction among the rules during the evolutionary process. The proposal minimizes the number of rules, the number of conditions per rule, and the number of conditions of the classifier, increasing the interpretability of the solutions. The algorithm does not explore already explored search spaces. Once one rule has no better conditions to be appended, the rule is marked as unbeatable and no more searches are performed over that rule, saving computational resources. Moreover, the algorithm stops when all the individuals are marked as unbeatable, i.e., there is no possibility of improving any individual, saving further generations. The population size and the maximum number of generations are equal to the number of features. Thus, it is not necessary to configure or optimize these parameters; it is self-adapting to the problem complexity.

The experiments performed compared our algorithm with other machine learning classification methods, including crisp and fuzzy rules, rules extracted from decision trees, and an ant colony algorithm. The results show the competitive performance of our proposal in terms of predictive accuracy, obtaining significantly better results than AntMiner+, CORE, ILGA, GFS-GP, and SLAVE. ICRM obtains the best results in terms of interpretability, i.e, it minimizes the number of rules, the number of conditions per rule, and the number of conditions of the classifier. These interpretability measures can be summarized using the Nauck complexity metric. Using this measure, ICRM obtains significantly better interpretability results than all the other techniques. The algorithms whose overall performances are closest to ICRM, in accuracy vs interpretability, are C45R and MPLCS. They obtain significantly worse interpretability results but a little improvement in accuracy.

Experimental results have shown the good performance of the algorithm, but it would be honest to note its limitations. The algorithm is capable of finding comprehensible classifiers with low number of rules and conditions, while achieving competitive accuracy. However, the comprehensibility is prioritized between these conflicting objectives. The one rule per class design allows to obtain very interpretable solutions and it is useful to extract fast ''big pictures'' of the data. Nevertheless, the accuracy on very complex data might be lower than the obtained by other algorithms but with much more complex classifiers. There is no classifier which achieves best accuracy and comprehensibility for all data. Thus, this algorithm focuses on the interpretability of the classifier, which is very useful for knowledge discovery and decision support systems.

The algorithm runs quite fast, significantly faster than other evolutionary-based algorithms, according to the experiments. However, there exits linear performance complexity regarding to the number of attributes. Therefore, mining extremely high-dimensional data with thousands of attributes would impact the time performance. The solution to this problem would be the execution of a feature selection algorithm prior to the classification algorithm.

Acknowledgments

This work has been supported by the Regional Government of Andalusia and the Ministry of Science and Technology, projects P08-TIC-3720 and TIN-2011–22408, FEDER funds, and Ministry of Education FPU Grant AP2010–0042.

References

[1] J.S. Aguilar-Ruiz, R. Giraldez, J.C. Riquelme, Natural encoding for evolutionary supervised learning, IEEE Transactions on Evolutionary Computation 11 (2007) 466–479.
[2] D.W. Aha, D. Kibler, M.K. Albert, Instance-based learning algorithms, Machine Learning 6 (1991) 37–66.

[3] J. Alcalá-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. García, L. Sánchez, F. Herrera, KEEL data-mining software tool: data set repository, integration of algorithms and experimental analysis framework, analysis framework, Journal of Multiple-Valued Logic and Soft Computing 17 (2011) 255–287.
[4] J. Alcalá-Fdez, L. Sánchez, S. García, M. del Jesus, S. Ventura, J. Garrell, J. Otero, C. Romero, J. Bacardit, V. Rivas, J. Fernández, F. Herrera, KEEL: a software tool to assess evolutionary algorithms for data mining problems, Soft Computing 13 (2009) 307–318.
[5] J. Alonso, L. Magdalena, Hilk++: an interpretability-guided fuzzy modeling methodology for learning readable and comprehensible fuzzy rule-based classifiers, Soft Computing 15 (2011) 1959–1980.
[6] S. Alonso, E. Herrera-Viedma, F. Chiclana, F. Herrera, A web based consensus support system for group decision making problems and incomplete preferences, Information Sciences 180 (2010) 4477–4495.
[7] T. Amin, I. Chikalov, M. Moshkov, B. Zielosko, Dynamic programming approach to optimization of approximate decision rules, Information Sciences 221 (2013) 403–418.
[8] R. Axelrod, The Complexity of Cooperation: Agent-based Models of Competition and Collaboration, Princeton University Press, 1997.
[9] J. Bacardit, J. Garrell, Bloat control and generalization pressure using the minimum description length principle for a Pittsburgh approach learning classifier system, in: T. Kovacs, X. Llor, K. Takadama, P. Lanzi, W. Stolzmann, S. Wilson (Eds.), Learning Classifier Systems, Lecture Notes in Computer Science, vol. 4399, Springer, 2007, pp. 59–79.
[10] J. Bacardit, N. Krasnogor, Performance and efficiency of memetic Pittsburgh learning classifier systems, Evolutionary Computation 17 (2009) 307–342.
[11] F.J. Berlanga, A.J.R. Rivas, M.J. del Jesús, F. Herrera, GP-COACH: genetic programming-based learning of compact and accurate fuzzy rule-based classification systems for high-dimensional problems, Information Sciences 180 (2010) 1183–1200.
[12] E. Bernadó-Mansilla, J.M. Garrell-Guiu, Accuracy-based learning classifier systems: models, analysis and applications to classification tasks, Evolutionary Computation 11 (2003) 209–238.
[13] U. Bhanja, S. Mahapatra, R. Roy, An evolutionary programming algorithm for survivable routing and wavelength assignment in transparent optical networks, Information Sciences 222 (2013) 634–647.
[14] L. Bull, E. Bernadó-Mansilla, J.H. Holmes (Eds.), Learning Classifier Systems in Data Mining, Studies in Computational Intelligence, vol. 125, Springer, 2008.
[15] M.V. Butz, T. Kovacs, P.L. Lanzi, S.W. Wilson, Toward a theory of generalization and learning in XCS, IEEE Transactions on Evolutionary Computation 8 (2004) 28–46.
[16] C. Campbell, Kernel methods: a survey of current techniques, Neurocomputing 48 (2002) 63–84.
[17] A. Cano, A. Zafra, S. Ventura, An ep algorithm for learning highly interpretable classifiers, in: Proceedings of the 11th International Conference on Intelligent Systems Design and Applications (ISDA), 2011, pp. 325–330.
[18] J. Cano, F. Herrera, M. Lozano, Evolutionary stratified training set selection for extracting classification rules with trade off precision-interpretability, Data and Knowledge Engineering 60 (2007) 90–108.
[19] D. Carvalho, A. Freitas, A hybrid decision tree/genetic algorithm method for data mining, Information Sciences 163 (2004) 13–35.
[20] W.J. Choi, T.S. Choi, Genetic programming-based feature transform and classification for the automatic detection of pulmonary nodules on computed tomography images, Information Sciences 212 (2012) 57–78.
[21] M. Cintra, M. Monard, H. Camargo, On rule learning methods: a comparative analysis of classic and fuzzy approaches, Studies in Fuzziness and Soft Computing 291 (2013) 89–104.
[22] W. Cohen, Fast effective rule induction, in: Proceedings of the 12th International Conference on Machine Learning, 1995, pp. 1–10.
[23] J. Demšar, Statistical comparisons of classifiers over multiple data sets, Machine Learning Research 7 (2006) 1–30.
[24] O. Dunn, Multiple comparisons among means, Journal of the American Statistical Association 56 (1961) 52–64.
[25] P. Espejo, S. Ventura, F. Herrera, A survey on the application of genetic programming to classification, IEEE Transactions on Systems, Man, and Cybernetics, Part C 40 (2010) 121–144.
[26] D. Fisch, B. Kühbeck, B. Sick, S.J. Ovaska, So near and yet so far: new insight into properties of some well-known classifier paradigms, Information Sciences 180 (2010) 3381–3401.
[27] S. García, A. Fernández, J. Luengo, F. Herrera, A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability, Soft Computing 13 (2009) 959–977.
[28] S. García, A. Fernández, J. Luengo, F. Herrera, Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: experimental analysis of power, Information Sciences 180 (2010) 2044–2064.
[29] S. García, D. Molina, M. Lozano, F. Herrera, A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study, Journal of Heuristics 15 (2009) 617–644.
[30] A. González, R. Perez, Selection of relevant features in a fuzzy genetic learning algorithm, IEEE Transactions on Systems and Man and and Cybernetics and Part B: Cybernetics 31 (2001) 417–425.
[31] D.P. Greene, S.F. Smith, Competition-based induction of decision models from examples, Machine Learning 13 (1994) 229–257.
[32] S. Guan, F. Zhu, An incremental approach to genetic-algorithms-based classification, IEEE Transactions on Systems and Man and and Cybernetics and Part B 35 (2005) 227–239.
[33] J.E. Hopcroft, R. Motwani, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, Chapter 4: Context-Free Grammars, Addison-Wesley, 2006.
[34] J. Huysmans, K. Dejaeger, C. Mues, J. Vanthienen, B. Baesens, An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models, Decision Support Systems 51 (2011) 141–154.
[35] H. Ishibuchi, Y. Kaisho, Y. Nojima, Complexity, interpretability and explanation capability of fuzzy rule-based classifiers, in: Proceedings of the IEEE International Conference on Fuzzy Systems, 2009, pp. 1730–1735.
[36] U. Johansson, C. Sönstrd, T. Löfström, H. Boström, Obtaining accurate and comprehensible classifiers using oracle coaching, Intelligent Data Analysis 16 (2012) 247–263.
[37] S.A. Kazarlis, S.E. Papadakis, J.B. Theocharis, V. Petridis, Microgenetic algorithms as generalized hill-climbing operators for GA optimization, IEEE Transactions on Evolutionary Computation 5 (2001) 204–217.
[38] R. Kohavi, A study of cross-validation and bootstrap for accuracy estimation and model selection, in: Proceedings of the 14th International Joint Conference on Artificial Intelligence, vol. 2, 1995, pp. 1137–1143.
[39] T. Kovacs, Genetics-based machine learning, in: G. Rozenberg, T. Bäck, J. Kok (Eds.), Handbook of Natural Computing: Theory, Experiments, and Applications, Springer-Verlag, 2011.
[40] P.L. Lanzi, Learning classifier systems: then and now, Evolutionary Intelligence 1 (2008) 63–82.
[41] M. Li, Z.W. Z, A hybrid coevolutionary algorithm for designing fuzzy classifiers, Information Sciences 179 (2009) 1970–1983.
[42] D. Martens, B. Baesens, T.V. Gestel, J. Vanthienen, Comprehensible credit scoring models using rule extraction from support vector machines, European Journal of Operational Research 183 (2007) 1466–1476.
[43] D.D. Nauck, Measuring interpretability in rule-based classification systems, in: Proceedings of IEEE International Conference on Fuzzy Systems, 2002, pp. 196–201.
[44] C. Nguyen, W. Pedrycz, T. Duong, T. Tran, A genetic design of linguistic terms for fuzzy rule based classifiers, International Journal of Approximate Reasoning 54 (2013) 1–21.
[45] A. Palacios, L. Sánchez, I. Couso, Extending a simple genetic cooperative-competitive learning fuzzy classifier to low quality datasets, Evolutionary Intelligence 2 (2009) 73–84.
[46] M. Paliwal, U. Kumar, Neural networks and statistical techniques: a review of applications, Expert Systems with Applications 36 (2009) 2–17.

[47] R. Parpinelli, H. Lopes, A. Freitas, Data mining with an ant colony optimization algorithm, IEEE Transactions on Evolutionary Computation 6 (2002) 321–332.
[48] J. Quinlan, C4.5: Programs for Machine Learning, 1993.
[49] D. Richards, Two decades of ripple down rules research, Knowledge Engineering Review 24 (2009) 159–184.
[50] J. Rissanen, Minimum description length principle, in: Encyclopedia of Machine Learning, 2010, pp. 666–668.
[51] R.L. Rivest, Learning decision lists, Machine Learning 2 (1987) 229–246.
[52] L. Sánchez, I. Couso, J. Corrales, Combining gp operators with sa search to evolve fuzzy rule based classifiers, Information Sciences 136 (2001) 175–192.
[53] D. Stavrakoudis, G. Galidaki, I. Gitas, J. Theocharis, Reducing the complexity of genetic fuzzy classifiers in highly-dimensional classification problems, International Journal of Computational Intelligence Systems 5 (2012) 254–275.
[54] K. Tan, Q. Yu, J. Ang, A coevolutionary algorithm for rules discovery in data mining, International Journal of Systems Science 37 (2006) 835–864.
[55] S. Tsumoto, Mining diagnostic rules from clinical databases using rough sets and medical diagnostic model, Information Sciences 162 (2004) 65–80.
[56] S. Ventura, C. Romero, A. Zafra, J. Delgado, C. Hervás, JCLEC: a Java framework for evolutionary computation, Soft Computing 12 (2007) 381–392.
[57] W. Verbeke, D. Martens, C. Mues, B. Baesens, Building comprehensible customer churn prediction models with advanced rule induction techniques, Expert Systems with Applications 38 (2011) 2354–2364.
[58] T. Wiens, B. Dale, M. Boyce, G. Kershaw, Three way k-fold cross-validation of resource selection functions, Ecological Modelling 212 (2008) 244–255.
[59] F. Wilcoxon, Individual comparisons by ranking methods, Biometrics Bulletin 1 (1945) 80–83.
[60] I.H. Witten, E. Frank, Data Mining: Practical Machine Learning Tools and Techniques, Morgan Kaufmann Publishers, San Francisco, CA, USA, 2005.
[61] M.L. Wong, K.S. Leung, Data Mining Using Grammar Based Genetic Programming and Applications, Kluwer Academic Publisher, 2000.
[62] N. Xie, Y. Liu, Review of decision trees, in: Proceedings of the 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT), vol. 5, 2010, pp. 105–109.
[63] J. Yang, H. Xu, P. Jia, Effective search for pittsburgh learning classifier systems via estimation of distribution algorithms, Information Sciences 198 (2012) 100–117.
[64] X. Yao, Y. Liu, G. Lin, Evolutionary programming made faster, IEEE Transactions on Evolutionary Computation 3 (1999) 82–102.
[65] X. Yu, M. Gen, Introduction to Evolutionary Algorithms, Springer, 2010.
[66] A. Zafra, S. Ventura, G3P-MI: a genetic programming algorithm for multiple instance learning, Information Sciences 180 (2010) 4496–4513.

TITLE:

Weighted Data Gravitation Classification for Standard and Imbalanced Data

AUTHORS:

A. Cano, A. Zafra, and S. Ventura

IEEE Transactions on Cybernetics, *Volume 43, Issue 6, pp. 1672-1687, 2013*

RANKING:

Impact factor (JCR 2012): 3.236

Knowledge area:

Computer Science, Cybernetics: 1/21

Computer Science, Artificial Intelligence: 12/114

Automation and Control Systems: 4/58

DOI: 10.1109/TSMCB.2012.2227470

Weighted Data Gravitation Classification for Standard and Imbalanced Data

Alberto Cano, *Member, IEEE*, Amelia Zafra, *Member, IEEE*, and Sebastián Ventura, *Senior Member, IEEE*

***Abstract*—Gravitation is a fundamental interaction whose concept and effects applied to data classification become a novel data classification technique. The simple principle of data gravitation classification (DGC) is to classify data samples by comparing the gravitation between different classes. However, the calculation of gravitation is not a trivial problem due to the different relevance of data attributes for distance computation, the presence of noisy or irrelevant attributes, and the class imbalance problem. This paper presents a gravitation-based classification algorithm which improves previous gravitation models and overcomes some of their issues. The proposed algorithm, called DGC+, employs a matrix of weights to describe the importance of each attribute in the classification of each class, which is used to weight the distance between data samples. It improves the classification performance by considering both global and local data information, especially in decision boundaries. The proposal is evaluated and compared to other well-known instance-based classification techniques, on 35 standard and 44 imbalanced data sets. The results obtained from these experiments show the great performance of the proposed gravitation model, and they are validated using several nonparametric statistical tests.**

***Index Terms*—Classification, covariance matrix adaptation evolution strategy (CMA-ES), data gravitation, evolutionary strategies, imbalanced data.**

I. Introduction

SUPERVISED learning is one of the most fundamental tasks in machine learning. A supervised learning algorithm analyzes a set of training examples and produces an inferred function to predict the correct output for any other examples. Classification is a common task in supervised machine learning which aims at predicting the correct class for a given example. Classification has been successfully implemented using many different paradigms and techniques, such as artificial neural networks [1], support vector machines (SVMs) [2], instance-based learning methods [3], or nature-inspired techniques such as genetic programming [4].

The nearest neighbor (NN) algorithm [5] is an instance-based method which might be the simplest classification algorithm. Its classification principle is to classify a new sample with the class of the closest training sample. The extended version of NN to k neighbors (KNN) and their derivatives are indeed one of the most influential data mining techniques, and they have been shown to perform well in many domains [6]. However, the main problem with these methods is that they severely deteriorate with noisy data or high dimensionality: their performance becomes very slow, and their accuracy tends to deteriorate as the dimensionality increases, especially when classes are nonseparable or they overlap [7].

In recent years, new instance-based methods based on data gravitation classification (DGC) have been proposed to solve the aforementioned problems of the NN classifiers [8]–[10]. DGC models are inspired by Newton's law of universal gravitation and simulate the accumulative attractive force between data samples to perform the classification. These gravitation-based classification methods extend the NN concept to the law of gravitation among the objects in the physical world. The basic principle of DGC is to classify data samples by comparing the data gravitation among the training samples for the different data classes, whereas KNNs vote for the k training samples that are the closest in the feature space.

This paper presents a DGC algorithm (DGC+) that compares the gravitational field for the different data classes to predict the class with the highest magnitude. The proposal improves previous data gravitation algorithms by learning the optimal weights of the attributes for each class and solves some of their issues such as nominal attributes handling, imbalanced data performance, and noisy data filtering. The weights of the attributes in the classification of each class are learned by means of the covariance matrix adaptation evolution strategy (CMA-ES) [11] algorithm, which is a well-known, robust, and scalable global stochastic optimizer for difficult nonlinear and nonconvex continuous domain objective functions [12]. The proposal improves accuracy results by considering both global and local data information, especially in decision boundaries.

The experiments have been carried out on 35 standard and 44 imbalanced data sets collected from the KEEL [13] and UCI [14] repositories. The algorithms compared for both standard and imbalanced classifications have been selected from the KEEL [15] and WEKA [16] software tools. The experiments consider different problem domains, number of instances, attributes, and classes. The algorithms from the experiments include some of the most relevant instance-based and imbalanced classification techniques presented up to now. The results reported show the competitive performance of the proposal, obtaining significantly better results in terms of predictive accuracy, Cohen's kappa rate [17], [18], and area under the curve (AUC) [19], [20]. The experimental study includes a statistical

Manuscript received May 24, 2012; revised September 5, 2012; accepted October 25, 2012. Date of publication January 4, 2013; date of current version November 18, 2013. This work was supported by the Regional Government of Andalusia and the Ministry of Science and Technology, projects P08-TIC-3720 and TIN-2011-22408, FEDER funds, and Ministry of Education FPU grant AP2010-0042. This paper was recommended by Editor J. Basak.

The authors are with the Department of Computer Science and Numerical Analysis, University of Cordoba, 14071 Cordoba, Spain (e-mail: acano@uco.es; azafra@uco.es; sventura@uco.es).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMCB.2012.2227470

analysis based on the Bonferroni–Dunn [21] and Wilcoxon [22] nonparametric tests [23], [24] in order to evaluate whether there are significant differences in the results of the algorithms.

The remainder of this paper is organized as follows. The next section presents some definitions and briefly reviews related background. Section III describes the proposed algorithm. Section IV describes the experimental study, whose results are discussed in Section V. Finally, Section VI presents some concluding remarks.

II. Background

This section briefly reviews the most important developments related to distance and instance weighting in NN classifiers and overviews data gravitation models.

A. NN Models

The following proposals focus on improvements of the NN classifier to consider distance weighting and training set transformation because these ideas have been considered when designing the proposed algorithm.

NN classifiers simply perform a class voting among the k closest training instances. Thus, the real problem is to decide which are the k closest instances based on a distance criterion and how they compete for class voting.

Dudani introduced the distance-weighted NN rule (DW-KNN) [25] based on the idea that evidences nearby sample observations are stronger. More specifically, he proposed to weight the evidence of a neighbor close to an unclassified example more heavily than the evidence of another neighbor which is at a greater distance from the unclassified example. Thus, the k closest neighbors $p_1, p_2, \ldots, p_k$ are ordered so that p_1 is the nearest and p_k is the farthest from the unclassified example. The corresponding distances of these neighbors from the unclassified example are $d_1, d_2, \ldots, d_k$. The voting weight for a neighbor p_j is defined as

$$w_j = \left\{ \begin{array}{ll} \frac{d_k - d_j}{d_k - d_1} & \text{if} \quad d_k \neq d_1 \\ 1 & \text{if} \quad d_k = d_1 \end{array} \right\}. \tag{1}$$

Having computed the weights, the DW-KNN assigns the unclassified example to the class for which the weights of the representatives among the k NNs sum to the greatest value.

Gao and Wang applied this idea and proposed a center-based NN classifier (CNN) [26] which uses the distance between train instances and centers of their class as a reference of how far the train instance is from the unclassified example. CNN considers the center-based line passing through an example with known label and the center of the data class. However, this method performs bad classification when the center of the data classes is overlapped.

Wang *et al.* proposed an adaptive k NN algorithm (KNN-A) [27] which involves both a locally adaptive distance measure for identifying the NNs to an unknown example and a weighting scheme that assigns a weight to each NN based on its statistical confidence.

Paredes and Vidal [28], [29] proposed a class-dependent weighted dissimilarity measure in vector spaces to improve the performance of the NN classifier. Under their proposed framework, the accuracy is improved by using a dissimilarity measure such that distances between points belonging to the same class are small while interclass distances are large.

Prototype selection [30] and prototype generation [31] are also commonly employed together with NN classifiers to reduce the data size and to improve the accuracy.

Paredes and Vidal extended their NN model together with a prototype reduction algorithm [32], which simultaneously trains both a reduced set of prototypes and a suitable local metric for them. Starting with an initial selection of a small number of prototypes, their proposal iteratively adjusts both the position (features) of these prototypes and the corresponding local-metric weights.

Zhou and Chen [33] proposed a cam weighted distance (CamNN) to ameliorate the curse of dimensionality which optimizes the distance measure based on the analysis of inter-prototype relationships.

Triguero *et al.* [34] applied differential evolution to the prototype selection problem as a position adjusting of prototypes. Their experimental results advocate for the great performance of SSMA [35] + SFLSDE [36], which results in a combination of a prototype selection stage with an optimization of the position of prototypes, prior to the NN classification. Similarly, Jahromi *et al.* [37] improve the classification rate of the NN rule by adjusting the weights of the training instances, thus reducing the size of the training set.

As observed, these proposals weight the class voting of the k NNs based on the relative distances between data samples and/or perform data reduction via prototype selection or generation. The NN classification process is transparent and easy to understand, implement, and debug. However, most of these methods are very sensitive to irrelevant, redundant, or noisy features because all features contribute to the similarity/distance function and thus to the classification. This problem is enhanced as the dimensionality of the data increases.

B. Data Gravitation Models

The first machine learning algorithm inspired by the physical gravitation was proposed in 1977 by Wright [38] for performing cluster analysis on Euclidean data. More recently, Endo and Iwata [39] proposed in 2005 a dynamic clustering algorithm based on universal gravitation which takes advantage of both global and local information of the data. Although both proposals are focused on clustering, they are inspired by the gravitation principles.

Regarding gravitational classification models, Wang and Chen [9] presented in 2005 an improvement of the NN classifier using simulated gravitational collapse. They simulated the physical process of gravitational collapse to trim the boundaries of the distribution of each class, since the performance of the NN classifier drops significantly with the increase of the overlapping of the distribution of different classes. Their algorithm generates prototypes for the NN classifier by a migration of the original samples.

There are more contributions that apply gravitation theory to classification, such as Yang *et al.* [40] for abnormal network intrusion detection or Zong-Chang [10] who proposed in 2007

a vector gravitational force model derived from the vector geometric analysis of the linear classifier.

The most recent and complete work related to DGC was presented by Peng *et al.* [8] in 2009. They employ weights to describe the importance of the attributes in the distance calculation of the gravitation model. The weights are optimized by a random iterative algorithm named tentative random feature selection (TRFS). Artificial data particles are created from the training data to represent set of samples for the different classes. Gravitation from a particle to the different classes is calculated using the mass of the data particles (number of instances represented) and the distance to its centroid, which represents the mass center of the data particle.

The algorithm creates data particles using the maximum distance principle. However, this method reduces the accuracy, especially in the area away from the data particle centroid and along the border between classes. The gravitation function performs well on standard data, but it suffers severely from imbalanced data, which is one of the major drawbacks concluded by their authors.

III. Algorithm Description

This section presents the proposed algorithm, whose main characteristics are depicted in the following sections. First, the main concepts and principles of DGC are presented. Second, the definition of distance and the procedure to calculate the optimal weights of the attributes for the different classes are presented. Third, the adaptation of the gravitation function to consider the imbalance data problem is discussed. Finally, the main steps conducted by the evolutionary algorithm to obtain the classifier are summarized.

A. DGC Principles

Newton published in 1687 the law of universal gravitation, which states that every point mass in the universe attracts every other point mass with a force that is directly proportional to the product of their masses and inversely proportional to the square of the distance between them. General relativity generalizes Newton's law of universal gravitation, providing a unified description of gravity as a geometric property of space-time. The curvature of space-time is usually represented using embedding diagrams [41]. These diagrams show the gravity well, which represents the gravitational field surrounding an object in space. Each point in space will have a value of the gravitational field proportional to the function

$$\vec{g} = -G\frac{M}{r^2}\vec{u}_r \tag{2}$$

where G is the gravitational constant, M is the mass of the object, and r is the distance to the object.

The principle of DGC is to assign to an instance the class with the highest gravitational field magnitude. Therefore, given a sample $\vec{x}$, the algorithm calculates the gravitation to $\vec{x}$ using the training samples for the different classes and predicts the class with the highest gravitation

$$Class(\vec{x}) = \arg\max g(\vec{x}, k)\ \forall k \in C \tag{3}$$

where $g(\vec{x}, k)$ is the gravitation exerted to a data sample $\vec{x}$ by the instances from the class k and C is the set of classes. There are significant differences between this proposal, NN, and previous gravitation methods, especially concerning how gravitation is calculated considering the distances among the samples. These differences are detailed in the next sections.

B. Definition of Distance

Gravitation is inversely proportional to the square of the distance. Newtonian gravitation usually considers the case of Euclidean geometry. The Euclidean distance between two objects $\vec{x}_1$ and $\vec{x}_2$ is defined as

$$d(\vec{x}_1, \vec{x}_2) = \sqrt{\sum_{i=1}^{f} (x_{1i} - x_{2i})^2} \tag{4}$$

where the number of attributes (features or dimensions) is f. Normalization is an essential preprocessing step in data mining [42], especially concerning instance-based methods [43]. All numerical attributes have to be scaled to [0, 1] for an equitable calculation of distances using the min–max normalization [44]. Otherwise, distances from attributes with larger domains would disrupt others from smaller domains. In order to find a reasonable distance between a pair of samples with nominal attributes, the overlap metric is used for the labels. This metric is defined as

$$\delta(x_{1i}, x_{2i}) = \left\{ \begin{array}{lll} 0 & \text{if} & x_{1i} = x_{2i} \\ 1 & \text{if} & x_{1i} \neq x_{2i} \end{array} \right\}. \tag{5}$$

Although the definition of distance from (4) considers all the attributes equally relevant, feature selection has been shown to improve the performance of classification algorithms [45], [46]. Feature selection selects a subset of relevant features and removes most irrelevant and redundant features from the data. Embedded feature selection methods [47] realize that the learning part and the feature selection part cannot be separated.

Peng *et al.* [8] proved that weighting attributes performed well in calculating the gravitation using the relevant attributes. Weighting attributes within the learning process is a way of embedded feature selection by giving more or less importance to the different attributes. However, similar to Paredes and Vidal [28], [29], we consider that the weight of each attribute must be different for each class since the distribution of the samples is not usually similar. Thus, instead of using an attribute weight vector of length f, the proposed algorithm employs a class-dependent attribute-class weight matrix $W[f, c]$, where f is the number of attributes and c is the number of classes.

$$W = \begin{bmatrix} w_{1,1} & \dots & w_{1,c} \\ \dots & \dots & \dots \\ w_{f,1} & \dots & w_{f,c} \end{bmatrix}. \tag{6}$$

Therefore, the distance function from (4) is rewritten to take into account the weights of the attributes for each class

$$d(\vec{x}_1, \vec{x}_2, k) = \sqrt{\sum_{i=1}^{f} w_{i,k} \cdot (x_{1i} - x_{2i})^2} \tag{7}$$

where $w_{i,k}$ is the weight of the attribute i for the class k.

The optimization of the W attribute-class weight matrix is a real optimization problem whose dimension $f \cdot c$ depends on the number of attributes and the number of classes. Accordingly, the CMA-ES is one of the most powerful evolutionary algorithms for difficult nonlinear nonconvex real-valued single-objective optimization [11]. The main advantages of CMA-ES lie in its invariance properties, which are achieved by carefully designed variation and selection operators, and in its efficient self-adaptation of the mutation distribution. CMA-ES does not require a tedious parameter tuning for its application since a strategy for finding good parameters is considered as part of the algorithm design and not as part of its application. Therefore, the proposal employs CMA-ES to optimize the real values of the attribute-class weight matrix instead of the TRFS algorithm proposed by Peng *et al.* [8] to optimize their attribute weight vector.

C. Definition of Gravitation

Gravitation depends on the mass and the distances among the objects. Peng *et al.* [8] perform data transformation by creating artificial data particles which replace the original training samples. Gravitation to a data class is calculated using the superposition principle, i.e., the sum of gravitation to the samples belonging to the data class. Therefore, their approach may fail because of the imbalance data since gravitation from a certain class is extremely strong or extremely weak, misclassifying minority class examples. The problem of learning from imbalanced data is a relatively new challenge that has attracted growing attention from both academia and industry [48]–[50]. This problem is concerned with the performance of learning algorithms in the presence of underrepresented data and severe class distribution skews.

This problem would be enhanced in our proposal since it considers all of the training instances. Gravitation of minority classes would be eclipsed by those of the majority classes. Therefore, the gravitation of a class is weighted by its number of instances and the total number of instances. In this way, a balance between the gravitation of majority and minority classes is achieved. Finally, the gravitation of a sample $\vec{x}$ for a class k is defined as

$$g(\vec{x}, k) = \left(1 - \frac{N_k - 1}{N}\right) \cdot \sum_{i=1}^{N} \frac{1}{d(\vec{x}_i, \vec{x}, k)^2} | \vec{x}_i \, \epsilon \, k \quad (8)$$

where N_k is the number of instances of the class k and N is the total number of instances.

Peng *et al.*'s proposal calculates the centroid of data particles, which is the center of masses, for distance computation. However, the weakness of the data class centroid is the loss of information about the shape of the instance cloud (local information), which does not happen when using all of the samples. The aforementioned problem is especially noticeable when the shape of the instance cloud is irregular; hence, the path of the border between the classes cannot be properly fitted. Instead, our proposal deals with all of the training samples. The advantage of using all of the samples to calculate gravitation is that it provides accurate classification in local classification, where there are many closer train instances which provide high gravitation nearby. It also provides good generalization where there are no closer training instances (global information) since gravitation is inversely proportional to the square of the distance, disregarding confidence from those faraway.

D. Evolutionary Algorithm

The main steps conducted by the algorithm to obtain the classifier, the optimal attribute-class weights, and the predictions for a data set are now summarized.

First, all attributes from the data set are scaled and normalized to the range [0, 1] using the min–max normalization. The problem of absolute distance calculation is that the outcome depends on the range and the values of each attribute domain. Thus, distances from attributes with larger domains would disrupt others from smaller domains. Therefore, scaling is used to avoid this problem and to relativize the distance calculation from the different attributes. For every attribute in the data set, we find the maximum value $v_{\max}$ and the minimum value $v_{\min}$. For every instance within the data set and attribute, the normalized attribute value v_{norm} for an original attribute value v is scaled as follows:

$$v_{\text{norm}} = \frac{v - v_{\min}}{v_{\max} - v_{\min}}. \quad (9)$$

Second, we must find the optimal values for the attribute-class weight matrix W in order to discriminate irrelevant or noisy attributes and to enhance truly relevant attributes for each class. Therefore, we encode a population of real array individuals to represent the weights of the attributes for each class. The initial values are set to 0.5, i.e., all attributes are initially considered equally relevant. CMA-ES is run to optimize these values according to the fitness function, which evaluates the accuracy of the candidate gravitation classifiers using the current attribute-class matrix values and the train data. The configuration settings and parameters of CMA-ES are described in Section IV-C. When CMA-ES meets any of the stop criteria, the best attribute-class matrix from the CMA-ES population is employed to build the final classifier.

Finally, the class predictions are obtained from submitting the instances from the data set to the final classifier. The class predictions are compared to the actual class of the instances, obtaining the values of the confusion matrix. The confusion matrix is used to compute the quality performance metrics described in Section IV-B.

IV. Experimental Study

This section describes the details of the experiments performed in order to evaluate the capabilities of the proposal and compares it to other classification methods. The experimental study is divided in two sets of experiments. On the one hand, the proposal is evaluated and compared to seven other instance-based classification techniques over 35 standard classification data sets. On the other hand, the proposal is evaluated over 44

imbalanced data sets and compared to eight other techniques specifically designed for imbalanced classification.[1]

First, the application domains are presented, followed by a description of the performance measures to evaluate. Then, the algorithms used for the comparison and the description of the experimental methodology are presented. Finally, the statistical tests used for validating the results are described.

A. Problem Domains

The data sets used in the experiments have been collected from the KEEL [13] and UCI [14] machine learning repository Web sites, and they are very varied in their degree of complexity, number of classes, number of attributes, number of instances, and imbalance ratio (the ratio of the size of the majority class to the size of the minority class). There are 35 standard and 44 imbalanced data sets. The number of classes ranges up to 11, the number of attributes ranges from 2 to 60, the number of instances ranges from 148 to 12 690, and the imbalance ratio is up to 129.44. Detailed information about the data sets and their number of instances, classes, attributes, and imbalance ratio are shown in the Web site.[1]

The standard data sets are partitioned using the stratified tenfold cross-validation procedure [51], [52]. The imbalanced data sets are partitioned using the stratified fivefold cross-validation procedure to ensure the presence of minority class instances in the test sets. All experiments are repeated with ten different seeds for stochastic methods.

B. Performance Measures

There are many performance measures for classification algorithms depending on the classification subfield problem [53], [54]. Different measures allow us to observe different behaviors, which increases the strength of the empirical study in such a way that more complete conclusions can be obtained from different (not opposite yet complementary) deductions [55], [56]. These measures are based on the values of the confusion matrix, where each column of the matrix represents the count of instances in a predicted class, while each row represents the number of instances in an actual class.

The standard performance measure for classification is the accuracy rate, which is the number of successful predictions relative to the total number of classifications. However, the accuracy rate may be misleading when data classes are strongly imbalanced since the all-positive or all-negative classifier may achieve a very good classification rate. Real-world problems frequently deal with imbalanced data. Therefore, the evaluation of the model should be done by means of other criteria rather than accuracy.

Cohen's kappa rate [17], [18] is an alternative measure to accuracy since it compensates for random hits. The kappa measure evaluates the merit of the classifier, i.e., the actual hits that can be attributed to the classifier and not by mere chance. Cohen's kappa statistic ranges from -1 (total disagreement) to 0 (random classification) to 1 (total agreement). It is calculated by means of the confusion matrix as follows:

$$Kappa = \frac{N\sum_{i=1}^{k} x_{ii} - \sum_{i=1}^{k} x_{i.}x_{.i}}{N^2 - \sum_{i=1}^{k} x_{i.}x_{.i}} \tag{10}$$

where x_{ii} is the count of cases in the main diagonal of the confusion matrix, N is the number of examples, and $x_{.i}$ and $x_{i.}$ are the column and row total counts, respectively. Cohen's kappa rate also penalizes all-positive or all-negative predictions, especially in imbalanced data problems. Kappa is very useful for multiclass problems, measuring a classifier's accuracy while compensating for random successes.

The area under the receiver operating characteristic (ROC) curve (AUC) [19], [20] is a commonly used evaluation measure for imbalanced classification. The ROC curve presents the tradeoff between the true positive rate and the false positive rate [57]. The classifier generally misclassifies more negative examples as positive examples as it captures more true positive examples. AUC is computed by means of the confusion matrix values: true positives (T_P), false positives (F_P), true negatives (T_N), and false negatives (F_N), and it is calculated by relating the true positive and false positive ratio

$$AUC = \frac{1+TP_{\text{rate}}-FP_{\text{rate}}}{2} = \frac{1+\frac{T_P}{T_P+F_N}-\frac{F_P}{F_P+T_N}}{2}. \tag{11}$$

C. Comparison of the Algorithms and Experimental Settings

This section describes the algorithms used in the experimental study and their parameter settings, which are obtained from the KEEL [15] and WEKA [16] software tools. Several instance-based methods, including the most recent DGC method, have been selected and compared to determine whether the proposal is competitive in different domains with the other approaches. Algorithms are compared on equal terms and without specific settings for each data problem. The parameters used for the experimental study in all classification methods are the optimal values from the tenfold cross-validation, and they are now detailed.

1) Classification Methods for Standard Data Sets: Table I summarizes the experimental settings for the standard classification algorithms.

1) DGC [8] employs weights to describe the importance of the attributes in the DGC model. It transforms the data set using data particles, which are constructed using the nearest instances to reduce the complexity of the data set. Gravitation from one particle to the different classes is calculated using the mass of the data particles (number of instances represented) and the distance to its centroid, which represents the mass center of the data particle.
2) KNN [5] classifies an instance with the class with the higher value of the number of neighbors to the instance that belongs to such class. Vicinity is defined as the

[1]The data set description together with their partitions, the algorithms as stand-alone runnable files, and the experimental settings are fully described and publicly available to facilitate the replicability of the experiments and future comparisons with previous or new proposals in the Web site http://www.uco.es/grupos/kdis/kdiswiki/DGC.

TABLE I
EXPERIMENTAL SETTINGS FOR STANDARD CLASSIFICATION

Algorithm	Parameter	Value
DGC	Population size	100
	Number of generations	200
	Distance to centroid threshold	0.2
KNN	Number of neighbours	3
	Distance	Euclidean
KNN-A	Number of neighbours	3
	Distance	Euclidean
DW-KNN	Number of neighbours	3
	Distance	Euclidean
CamNN	Number of neighbours	3
CNN	No parameters	
SSMA-SFLSDE	Population size	30
	Number of evaluations	10000
	Cross prob per bit	0.5
	Mutation prob per bit	0.001
	Number of Neighbours	1
	Distance	Euclidean
	Auxiliar population size	50
	Maximum of iterations	500
	Iterations SFGSS	8
	Iterations SFHC	20
	Fl	0.1
	Fu	0.9
	Tau1	0.1
	Tau2	0.1
	Tau3	0.03
	Tau4	0.07
	Strategy	RandToBest

k instances with lower distances to the instance being classified.

3) Adaptive KNN (KNN-A) [27] is a classifier in which distances are weighted by the distance from the training instance to its nearest instance belonging to a different class. Thus, instances near to the decision boundaries become more relevant.
4) DW-KNN [25] weights the vote of the k NNs regarding their relative distances to the uncategorized example.
5) Cam weighted distance (CamNN) [33] addresses the curse of dimensionality optimizing a distance measure based on the analysis of relations among data samples.
6) Center NN (CNN) [26] classifier is an enhanced 1-NN classifier which uses the distance between train instances and centers of their class as a reference of how far the train instance is from another instance. This algorithm is parameter-free.
7) SSMA-SFLSDE [34] combines a prototype selection stage with an optimization of the position of prototypes, prior to the NN classification.

2) Classification Methods for Imbalanced Data Sets: There are two common approaches to deal with imbalanced data: cost-sensitive learning and resampling methods.

Cost-sensitive learning modifies the algorithm by introducing numeric costs to any formula used to estimate the error of the model. A model builder will take into account the different costs of the outcome classes and will build a model that does not readily dismiss the very much underrepresented outcome.

Resampling is a data preprocessing step, and it aims to redress the balance. Thus, the base classifier used by the model builder does not need to be modified. Resampling methods are divided in two categories: undersampling (remove over-represented class instances) and oversampling (generate under-represented class instances). The resampling methods used to balance data class distribution are the following.

TABLE II
EXPERIMENTAL SETTINGS FOR IMBALANCED CLASSIFICATION

Algorithm	Parameter	Value
ADAC2	Pruned	True
	Confidence	0.25
	Instances per leaf	2
	Number of classifiers	10
	Train method	Cost-sensitive
	Cost set-up	Adaptive
NN-CS	Hidden layers	2
	Hidden nodes	15
	Transfer function	Htan
	Eta	0.15
	Alpha	0.1
	Lambda	0.0
	Cycles	10000
CSVM-CS	Kernel	Polynomial
	C	100.0
	eps	0.001
	degree	1
	vgamma	0.01
	coef0	0.0
	shrinking	True
C4.5-CS	Pruned	True
	Confidence	0.25
	Instances per leaf	2
	Minimum expected cost	True
C4.5-RUS	Pruned	True
	Confidence	0.25
	Instances per leaf	2
C4.5-SMOTE	Number of neighbours	5
	Distance	HVDM [65]
	Interpolation	Standard
	Pruned	True
	Confidence	0.25
	Instances per leaf	2
C4.5-SMOTE-TL	Number of neighbours	5
	Distance	HVDM [65]
	Pruned	True
	Confidence	0.25
	Instances per leaf	2

1) Random undersampling (RUS) [58] randomly removes a subset of the overrepresented class to approach the same number as the underrepresented class.
2) Synthetic minority oversampling technique (SMOTE) [59] generates underrepresented class instances from other instances in the original data set by selecting k NNs and using them to perform arithmetical operations to generate new instances.
3) Tomek links (TLs) [60] obtain the instance set belonging to spaces near the decision boundaries. SMOTE is usually employed together with TL (SMOTE-TL) [58] for oversampling with SMOTE and then for removing instances near boundaries using TL.

These resampling techniques are combined with a classifier to produce accurate imbalanced classification results. Next, some commonly used imbalanced data classifiers, whose parameters for the experimental study are summarized in Table II, are now detailed.

1) ADAC2 [61] is a boosting algorithm that produces an ensemble of decision trees (C4.5) from a set of given examples. Each instance has an associated cost depending

on the class. It tries to solve the imbalance problem by increasing the weight of the instances from the minority class in each iteration of the boosting algorithm. It can be executed using the cost-sensitive C4.5 or a resampling procedure after the weight update. C2 modification refers to the way in which the costs are introduced in the boosting procedure. The costs are automatically selected by considering the imbalance ratio.

2) NN-CS [62] is a neural network presented by Zhou and Liu, in which they study the effect of sampling and threshold-moving methods that have been shown to be effective in addressing the class imbalance problem, applied to cost-sensitive neural networks.
3) CSVM-CS [63] builds an SVM model with the training data, taking into account the cost associated to the class distribution, and then classifies all test data by means of the trained SVM.
4) C4.5-CS [64] is the C4.5 decision tree algorithm that weights the instances of the input data to consider the associated costs from imbalanced classification.
5) C4.5-RUS preprocesses the data set by using RUS to delete overrepresented class instances randomly. Then, C4.5 decision tree is applied over the transformed data.
6) C4.5-SMOTE preprocesses the data set by using SMOTE to generate underrepresented class instances. Then, C4.5 decision tree is applied over the transformed data.
7) C4.5-SMOTE-TL preprocesses the data set by using SMOTE to generate underrepresented class instances and then removes instances near the boundaries using TL. Finally, C4.5 decision tree is applied over the transformed data.

On the other hand, the proposed DGC+ has been implemented in the JCLEC software [66], it is available as a WEKA module, and its parameters are the ones from the CMA-ES algorithm. CMA-ES does not require a laborious parameter tuning since finding good parameters is considered as part of the algorithm design and not as part of its application. For the application of CMA-ES, the optimal values recommended by Hansen [11] are employed, and it is only necessary to detail some parameter information. CMA-ES employs a chromosome whose dimension (C_L) is the number of classes times the number of attributes. Each gene represents the weight of an attribute for a certain class. The initial solution is set to 0.5 for every gene within the chromosome, and the initial standard deviation is set to 0.3. CMA-ES is run to optimize the attribute-class weight matrix which obtains the highest accuracy. The termination criterion is a drop of the difference of the fitness function values below $1E-13$, and the maximum number of iterations is set to 500. The population size self-adapts to the problem dimension, and it is set to $4 * \log C_L^2$. In CMA-ES, the population size can be freely chosen because the learning rates prevent degeneration even for small population sizes. Small population sizes usually lead to faster convergence, while large population sizes help in avoiding local optima. The number of restarts is two, and the population size is not increased.

D. Statistical Analysis

In order to analyze the results from the experiments, some nonparametric statistical tests are used to validate the results [23], [67]. To evaluate whether there are significant differences in the results of the algorithms, the Iman and Davenport test is performed. This useful nonparametric test, recommended by Demsar [68], is applied to rank the K algorithms over the D data sets according to the F-distribution. If the Iman and Davenport test indicates that the results are significantly different, the Bonferroni–Dunn post hoc test [21] is used to find the significant differences occurring between algorithms in multiple comparisons. It assumes that the performance of two classifiers is significantly different if the corresponding average ranks differ by at least a critical difference value. Finally, the Wilcoxon rank-sum test [22] is used to perform multiple pairwise comparisons among the algorithms [69]. The Wilcoxon rank-sum test statistic is the sum of the ranks for observations from one of the samples.

V. Results

This section presents and discusses the experimental results from the different experimental studies. First, the accuracy and Cohen's kappa rate for standard data sets are compared for the different methods, and the results are validated using nonparametric statistical tests. Second, the AUC and Cohen's kappa rate for imbalanced data sets are compared, and the results are validated. Third, the convergence process of the proposal and the output weights are analyzed. Finally, a discussion is carried out to analyze the reasons for the results obtained.

A. Standard Data Sets

1) Accuracy: Table III shows the average accuracy results from the tenfold cross-validation test for the different data sets (rows) and methods (columns). The proposed gravitation algorithm outperforms the other methods in 16 of the 35 data sets and obtains competitive accuracy results in the other data sets. Next, some interesting comments to highlight the accuracy results for some methods and data sets are discussed.

Hayes-Roth is an artificial data set originally created to test the behavior of prototype-based classifiers. It contains an attribute which was generated at random, adding noise to the data. Therefore, NN methods sensitive to noisy data obtain the worst accuracy since they are not able to filter the noisy attribute. They are KNN, KNN-A, CamNN, and CNN, with the original KNN being the worst with only an accuracy of 0.2500. On the other hand, the DW-KNN is capable of avoiding noisy data, as well as the prototypes generated by SSMA+SFLSDE and DGC data particles. These proposals overcome successfully the noise and perform much better than the other algorithms, proving the good performance of attribute weighting and prototype generation. Finally, DGC+ achieves the highest accuracy for this data set with the best noisy attribute filtering.

The Iman and Davenport test establishes an F-distribution value $= 2.0482$ for a significance level of alpha $= 0.05$. The Iman and Davenport statistic (distributed according to the

TABLE III
ACCURACY RESULTS FOR STANDARD DATA SETS

Data set	DGC+	DGC	KNN	KNN-A	DW-KNN	Cam-NN	CNN	SSMA+SFLSDE
Appendicitis	0.8409	0.8713	0.8427	**0.8882**	0.8227	0.8491	0.7736	0.8509
Australian	0.8374	0.8496	0.8478	0.8391	0.8319	0.8522	0.8174	**0.8594**
Balance	0.8966	**0.8999**	0.8337	0.8943	0.8560	0.8670	0.7855	0.8896
Banana	0.8952	0.8931	0.8864	0.8940	0.8836	0.8858	0.5725	**0.8992**
Bupa	**0.6744**	0.6527	0.6066	0.6257	0.6376	0.5962	0.6316	0.6426
Car	**0.9523**	0.9126	0.9231	0.8797	0.8148	0.9045	0.8773	0.9097
Contraceptive	0.4945	0.4954	0.4495	0.4827	0.4488	0.4719	0.4203	**0.4969**
Dermatology	0.9544	0.9170	**0.9690**	0.9521	0.9633	0.8966	0.9578	0.9410
Ecoli	0.8217	0.7672	0.8067	0.8247	**0.8248**	0.7824	0.6789	0.8009
Flare	0.6655	0.6733	0.6229	0.4634	**0.7148**	0.6041	0.6492	0.7102
German	**0.7322**	0.7020	0.6960	0.7270	0.7210	0.7120	0.7080	0.7100
Glass	0.7036	0.6893	0.7011	0.6498	**0.7218**	0.6102	0.7094	0.7176
Haberman	0.7171	**0.7277**	0.7058	0.6927	0.6863	0.6963	0.6865	0.6994
Hayes-Roth	**0.8400**	0.7738	0.2500	0.3750	0.7063	0.4750	0.4625	0.7562
Heart	**0.8452**	0.8119	0.7741	0.8111	0.7630	0.7704	0.7667	0.8296
Hepatitis	0.8628	0.8343	0.8251	0.8508	0.8108	**0.8651**	0.8233	0.7617
Ionosphere	0.9311	0.6724	0.8518	**0.9372**	0.8747	0.7379	0.8917	0.9088
Iris	**0.9533**	**0.9533**	0.9400	**0.9533**	0.9400	0.9467	0.9267	**0.9533**
Lymphography	**0.8140**	0.8033	0.7739	0.8085	0.7855	0.7530	0.7599	0.7950
Monk-2	**0.9995**	0.9982	0.9629	0.7058	0.8437	0.8494	0.7490	0.9681
New-thyroid	**0.9786**	0.8684	0.9537	0.9721	0.9677	0.8693	0.8753	0.9677
Nursery	**0.9696**	0.9378	0.9254	0.8610	0.8131	0.8637	0.7875	0.8489
Page-blocks	0.9508	0.9268	0.9591	**0.9629**	0.9624	0.8942	0.9543	0.9530
Phoneme	0.8718	0.8471	0.8849	0.8901	**0.8999**	0.8679	0.8830	0.8533
Pima	0.7451	0.6662	0.7319	**0.7476**	0.7163	0.7280	0.6994	0.7463
Saheart	**0.7118**	0.7105	0.6818	0.6927	0.6752	0.7099	0.6407	0.6860
Sonar	0.8487	0.7694	0.8307	0.8798	0.8648	0.7743	**0.8940**	0.8079
Tae	**0.6715**	0.6709	0.4113	0.4375	0.5704	0.5112	0.3983	0.5371
Thyroid	**0.9704**	0.9256	0.9389	0.9396	0.9368	0.9300	0.9250	0.9463
Tic-tac-toe	**0.8549**	0.6906	0.7756	0.7672	0.6848	0.7629	0.7317	0.7348
Vehicle	0.7116	0.6572	0.7175	0.6879	0.7258	0.6170	**0.7423**	0.6456
Vowel	0.9824	0.9788	0.9778	0.9687	0.9909	0.8879	**0.9929**	0.8980
Wine	**0.9731**	0.9706	0.9549	0.9663	0.9438	0.9497	0.9663	0.9438
Yeast	**0.5926**	0.5151	0.5317	0.5539	0.5418	0.4515	0.4670	0.5708
Zoo	**0.9553**	0.9348	0.9281	0.9447	0.9447	0.8919	0.9281	0.9100
Avg. Values	**0.8349**	0.7991	0.7849	0.7865	0.7969	0.7667	0.7581	0.8043
Avg. Ranks	**2.3857**	4.3857	4.8286	3.8429	4.7714	5.8000	5.9143	4.0714

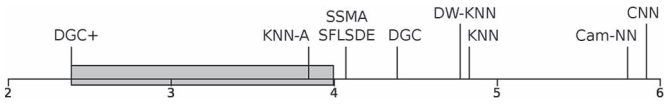


Fig. 1. Bonferroni–Dunn test for accuracy and standard data.

TABLE IV
WILCOXON TEST FOR ACCURACY AND STANDARD DATA

DCG+ vs	R^+	R^-	p-value
DGC	535.0	60.0	1.126E-5
KNN	592.0	38.0	3.840E-7
KNN-A	494.0	101.0	4.648E-4
DW-KNN	536.5	93.5	1.338E-4
Cam-NN	616.0	14.0	6.402E-9
CNN	583.0	47.0	1.253E-6
SSMA+SFLSDE	528.0	67.0	2.344E-5

F-distribution with 7 and 238 degrees of freedom) is 9.2342 for accuracy. Thus, the test rejects the null hypothesis, and therefore, it can be said that there are statistically significant differences between the accuracy results of the algorithms.

Fig. 1 shows the application of the Bonferroni–Dunn test to the accuracy rate with alpha $= 0.05$, whose critical difference is 1.5751. This graph represents a bar chart, whose values are proportional to the mean rank obtained from each algorithm. The critical difference value is represented as a thicker horizontal line, and those values that exceed this line are algorithms with significantly different results than the control algorithm, which is the proposed DGC+. Therefore, the algorithms right beyond the critical difference from the proposal value are significantly worse. Observing this figure, all the other methods but KNN-A perform significantly worse than DGC+. DGC+ successfully overcomes the other gravitation method (DGC), which obtains the fourth best ranking and third best average results.

Table IV shows the results of the Wilcoxon rank-sum test for accuracy to compute multiple pairwise comparisons among the proposal and the other methods. KNN-A is the best among the other methods, but DGC+ outperforms it, and there are significant differences between the two algorithms with a confidence level higher than 99%.

2) Cohen's Kappa Rate: Table V shows the average kappa results from the tenfold cross-validation test for the different data sets (rows) and methods (columns). DGC+ outperforms the other methods in 15 of the 35 data sets and obtains competitive kappa results in the other data sets.

In the previous section, it was mentioned how noise-sensitive NN methods performed badly over the Hayes-Roth data set. The kappa metric is able to detect these errors and penalizes this behavior with a significantly lower kappa value. Interestingly, the kappa rate of KNN over Hayes-Roth is negative, i.e., the predictions are completely wrong because of the noise, even performing worse than doing random predictions.

TABLE V
COHEN'S KAPPA RATE RESULTS FOR STANDARD DATA

Data set	DGC+	DGC	KNN	KNN-A	DW-KNN	Cam-NN	CNN	SSMA+SFLSDE
Appendicitis	0.4549	0.4710	0.4565	**0.5880**	0.4650	0.3808	0.2773	0.4573
Australian	0.6724	0.6976	0.6917	0.6719	0.6607	0.7035	0.6313	**0.7146**
Balance	0.8083	**0.8144**	0.7004	0.8042	0.7403	0.7559	0.6271	0.7953
Banana	0.7873	0.7825	0.7699	0.7850	0.7642	0.7675	0.1367	**0.7955**
Bupa	**0.3076**	0.2220	0.1944	0.2021	0.2645	0.1024	0.2571	0.2731
Car	**0.8981**	0.8025	0.8264	0.7009	0.5255	0.7935	0.7457	0.8023
Contraceptive	0.1994	0.1952	0.1358	0.1730	0.1517	0.1396	0.1079	**0.2112**
Dermatology	0.9425	0.8939	**0.9608**	0.9394	0.9538	0.8689	0.9468	0.9257
Ecoli	0.7498	0.6585	0.7300	0.7539	**0.7579**	0.6882	0.5620	0.7245
Flare	0.5662	0.5738	0.5101	0.2757	**0.6314**	0.4753	0.5518	0.6261
German	0.2812	0.0092	0.2194	0.2539	**0.3015**	0.1651	0.2880	0.2560
Glass	0.5834	0.5548	0.5887	0.5224	**0.6194**	0.4229	0.6037	0.6069
Haberman	0.0558	0.0315	0.1362	0.0752	0.1374	0.1417	**0.1832**	0.0935
Hayes-Roth	**0.7468**	0.6294	-0.2326	0.0441	0.5217	0.1835	0.1721	0.6146
Heart	**0.6826**	0.6094	0.5420	0.6168	0.5192	0.5420	0.5291	0.6501
Hepatitis	**0.5512**	0.2000	0.4683	0.4069	0.4084	0.3615	0.4672	0.3451
Ionosphere	0.8487	0.1142	0.6494	**0.8595**	0.7083	0.5145	0.7526	0.7986
Iris	**0.9300**	**0.9300**	0.9100	**0.9300**	0.9100	0.9200	0.8900	**0.9300**
Lymphography	**0.6289**	0.6026	0.5507	0.6162	0.5765	0.4993	0.5333	0.5974
Monk-2	**0.9991**	0.9963	0.9254	0.3987	0.6818	0.6965	0.4968	0.9357
New-thyroid	**0.9544**	0.6566	0.8957	0.9362	0.9282	0.7570	0.7073	0.9310
Nursery	**0.9551**	0.9083	0.8907	0.7963	0.7279	0.8001	0.6860	0.7786
Page-blocks	0.7152	0.4457	0.7655	**0.7890**	0.7887	0.5792	0.7445	0.7246
Phoneme	0.6898	0.5889	0.7172	0.7296	**0.7539**	0.6764	0.7144	0.6379
Pima	0.4063	0.0702	0.3892	**0.4259**	0.3576	0.3838	0.3191	0.4196
Saheart	0.3081	0.2708	0.2646	0.2929	0.2510	**0.3335**	0.2056	0.2687
Sonar	0.6943	0.5187	0.6554	0.7549	0.7248	0.5364	**0.7861**	0.6100
Tae	**0.5062**	0.5049	0.1171	0.1559	0.3563	0.2650	0.0946	0.3017
Thyroid	**0.7709**	-0.0002	0.4012	0.3593	0.4235	0.1049	0.3996	0.4608
Tic-tac-toe	**0.6607**	0.1472	0.4149	0.3998	0.1153	0.4243	0.4054	0.3941
Vehicle	0.6152	0.5437	0.6233	0.5844	0.6342	0.4905	**0.6563**	0.5273
Vowel	0.9807	0.9767	0.9756	0.9656	0.9900	0.8767	**0.9922**	0.8878
Wine	**0.9590**	0.9552	0.9318	0.9491	0.9152	0.9228	0.9491	0.9145
Yeast	**0.4695**	0.3309	0.3942	0.4225	0.4070	0.2413	0.3176	0.4430
Zoo	**0.9426**	0.9127	0.9041	0.9254	0.9255	0.8553	0.9043	0.8870
Avg. Values	**0.6664**	0.5320	0.5735	0.5744	0.5885	0.5248	0.5326	0.6097
Avg. Ranks	**2.5571**	4.9857	4.7714	4.1429	4.2143	5.9000	5.3571	4.0714

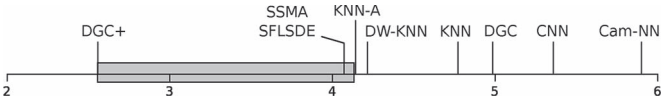


Fig. 2. Bonferroni–Dunn test for kappa rate and standard data.

Another interesting kappa result is found with the thyroid data set. The thyroid accuracy results from Table III do not suggest a noticeable difference in the accuracy performance of the different methods, all around 0.93. However, the kappa results do indicate considerable differences, i.e., DGC+ obtains 0.7709, whereas DGC interestingly obtains -0.0002.

The Iman and Davenport test establishes an F-distribution value $= 2.0482$ for a significance level of alpha $= 0.05$. The Iman and Davenport statistic (distributed according to the F-distribution with 7 and 238 degrees of freedom) is 7.0013 for kappa rate. Thus, the test rejects the null hypothesis, and therefore, it can be said that there are statistically significant differences between the kappa results of the algorithms.

Fig. 2 shows the application of the Bonferroni–Dunn test to kappa for alpha $= 0.05$, whose critical difference is 1.5751. Similar to the case with the accuracy rate, the algorithms right beyond the critical difference from the proposal's value are significantly worse. Observing this figure, all the other methods but SSMA+SFLSDE perform significantly worse than our proposal. Interestingly, the ranking of DGC is relegated to the sixth position.

TABLE VI
WILCOXON TEST FOR KAPPA AND STANDARD DATA

DGC+ vs	R^+	R^-	p-value
DGC	560.0	35.0	5.016E-7
KNN	553.0	77.0	3.154E-5
KNN-A	470.5	124.5	0.002405
DW-KNN	487.0	143.0	0.004030
Cam-NN	606.0	24.0	4.436E-8
CNN	552.5	77.5	3.308E-5
SSMA+SFLSDE	513.0	82.0	9.722E-5

Table VI shows the results of the Wilcoxon rank-sum test for kappa to compute multiple pairwise comparisons among the proposal and the other methods. Even though SSMA+SFLSDE is the algorithm with the second highest ranking in the multiple-algorithm comparison, DW-KNN is the best algorithm from the pairwise comparison versus DGC+. The p value for DGC+ versus DW-KNN is 0.004, which is lower than 0.01. Thus, our DGC+ proposal outperforms it and the others with a confidence level higher than 99% for pairwise comparisons.

B. Imbalanced Data Sets

1) AUC: Table VII shows the average AUC results from the cross-validation test for the different imbalanced data sets (rows) and methods (columns). DGC+ obtains the best results in 14 of the 44 data sets, and CSVM-CS achieves the best results in 12 of the 44 data sets. However, their

TABLE VII
AUC RESULTS FOR IMBALANCED DATA SETS

Algorithm	DGC+	DGC	ADAC2	NN	CSVM	C4.5	C4.5	C4.5	C4.5
Re-sampling / CS	None	None	CS	CS	CS	CS	RUS	SMT	SMT-TL
abalone19	0.6609	0.5000	0.5085	0.4949	**0.7615**	0.5701	0.6248	0.5719	0.5663
abalone9-18	0.8124	0.5000	0.7172	0.6490	**0.8740**	0.6655	0.6892	0.7892	0.7022
ecoli-0-1-3-7_vs_2-6	0.8427	0.7200	0.8154	0.7236	**0.8500**	0.8281	0.7900	0.7318	0.8172
ecoli-0_vs_1	0.9799	0.9642	0.9692	0.9796	0.9671	**0.9832**	0.9796	**0.9832**	0.9761
ecoli1	0.8804	0.7632	0.8763	0.8489	0.9062	**0.9114**	0.8852	0.8926	0.8801
ecoli2	**0.9378**	0.7919	0.8845	0.6880	0.5000	0.8905	0.8693	0.8906	0.8979
ecoli3	**0.8713**	0.6246	0.8478	0.8263	0.7925	0.8326	0.8453	0.8645	0.8502
ecoli4	0.9092	0.5300	0.9280	0.8702	**0.9529**	0.8636	0.8613	0.9513	0.8449
glass-0-1-2-3_vs_4-5-6	0.9215	0.7893	0.9033	**0.9308**	0.8445	0.8777	0.8811	0.9023	0.9102
glass-0-1-6_vs_2	0.6350	0.5000	0.5400	0.4038	0.5000	0.6155	0.6500	**0.7869**	0.6895
glass-0-1-6_vs_5	0.8363	0.5300	0.8800	0.8771	0.5000	**0.9886**	0.9429	0.9157	0.9243
glass0	**0.8650**	0.8553	0.8101	0.6792	0.5074	0.8212	0.8206	0.7754	0.8039
glass1	0.7496	0.7002	**0.7866**	0.6132	0.6264	0.7160	0.7153	0.7434	0.7711
glass2	0.7134	0.5000	0.7099	0.4396	0.5953	0.6416	0.6663	0.7112	**0.7172**
glass4	0.8869	0.5667	0.8706	0.8732	0.9126	0.8431	0.8347	0.8452	**0.9151**
glass5	0.8093	0.5000	0.9732	0.8805	0.9732	0.9427	0.9366	**0.9756**	0.8634
glass6	**0.9091**	0.8300	0.8923	0.9041	0.8725	0.8896	0.8824	0.8923	0.8965
habermanImb	0.6213	0.5062	0.5604	0.6245	0.5382	0.5752	0.6423	**0.6539**	0.6203
iris0	**1.0000**	0.9980	0.9900	**1.0000**	**1.0000**	0.9900	0.9900	0.9900	0.9900
new-thyroid1	**0.9939**	0.8023	0.9464	0.8679	0.9687	0.9746	0.9159	0.9492	0.9718
new-thyroid2	**0.9916**	0.8309	0.9575	0.8829	0.9829	0.9802	0.9071	0.9579	0.9440
page-blocks-1-3_vs_4	0.9395	0.6200	**0.9978**	0.9251	0.8566	0.9789	0.9436	0.9888	0.9774
page-blocks0	0.9412	0.6701	0.8816	0.7299	0.9254	**0.9458**	0.9448	0.9395	0.9388
pimaImb	**0.7394**	0.5274	0.7114	0.7175	0.7289	0.7125	0.7235	0.7134	0.6948
segment0	0.9927	0.9754	0.9826	0.5000	**0.9965**	0.9919	0.9788	0.9914	0.9914
shuttle-c0-vs-c4	0.9975	0.9845	0.9997	0.9070	**1.0000**	0.9997	**1.0000**	0.9994	0.9997
shuttle-c2-vs-c4	0.9743	0.7000	0.9500	0.7585	**1.0000**	**1.0000**	0.9840	0.9588	**1.0000**
vehicle0	0.9460	0.5533	0.9438	0.6801	**0.9493**	0.9289	0.9357	0.9274	0.9281
vehicle1	**0.7585**	0.5243	0.7531	0.6389	0.7546	0.7013	0.6820	0.6814	0.7422
vehicle2	0.9487	0.6782	**0.9729**	0.5945	0.9571	0.9434	0.9326	0.9556	0.9507
vehicle3	0.7516	0.5301	0.7345	0.6385	**0.7904**	0.7283	0.7005	0.7112	0.7463
vowel0	**0.9879**	0.7711	0.9706	0.6817	0.8461	0.9422	0.9438	0.9722	0.9850
wisconsinImb	0.9668	0.8986	0.9653	0.9582	**0.9719**	0.9636	0.9522	0.9579	0.9627
yeast-0-5-6-7-9_vs_4	0.7857	0.5000	0.7610	0.6894	0.5000	0.7243	0.7858	**0.8308**	0.7548
yeast-1-2-8-9_vs_7	0.6416	0.5000	0.6376	0.4307	0.5000	**0.6769**	0.6050	0.5966	0.5755
yeast-1-4-5-8_vs_7	0.5806	0.5000	0.5426	0.4893	0.5000	0.5540	0.5739	**0.5913**	0.5270
yeast-1_vs_7	**0.7594**	0.5000	0.7049	0.5771	0.5000	0.6139	0.6603	0.6871	0.6994
yeast-2_vs_4	0.9103	0.5120	0.9172	0.7330	0.5000	0.8866	0.8856	0.8810	**0.9309**
yeast-2_vs_8	0.7706	0.6650	0.6218	0.6588	0.7664	0.8652	0.7347	0.8415	**0.8686**
yeast1	**0.7183**	0.5116	0.6604	0.5864	0.6749	0.6779	0.7161	0.7112	0.6809
yeast3	**0.9202**	0.5533	0.9108	0.7643	0.8951	0.9117	0.9073	0.9119	0.9148
yeast4	**0.8252**	0.5000	0.7204	0.6554	0.8155	0.7222	0.8194	0.7420	0.7838
yeast5	0.9498	0.5236	0.8833	0.6139	0.9656	0.9330	0.9375	0.9517	**0.9712**
yeast6	0.8530	0.5057	0.7163	0.5891	**0.8758**	0.8082	0.8113	0.8309	0.8462
Avg. Values	**0.8520**	0.6479	0.8251	0.7176	0.7885	0.8321	0.8293	0.8443	0.8414
Avg. Ranks	**2.7273**	8.2841	4.8182	7.1023	4.7159	4.4432	4.8864	4.0114	4.0114

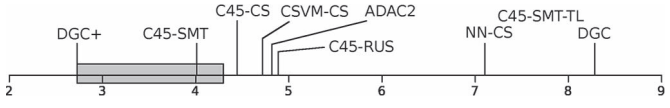


Fig. 3. Bonferroni–Dunn test for AUC and imbalanced data.

average results and rankings are significantly different. On the other hand, DGC stands out for achieving the worst AUC results.

The Iman and Davenport test establishes an F-distribution value $= 1.9653$ for a significance level of alpha $= 0.05$. The Iman and Davenport statistic (distributed according to the F-distribution with 8 and 344 degrees of freedom) is 26.2834 for AUC. Thus, the test rejects the null hypothesis, and therefore, it can be said that there are statistically significant differences between the AUC results of the algorithms.

Fig. 3 shows the application of the Bonferroni–Dunn test to AUC for alpha $= 0.05$, whose critical difference is 1.5905. All the other methods but C4.5-SMT and C4.5-SMT-TL show to perform statistically worse than DGC+.

Table VIII shows the results of the Wilcoxon rank-sum test for AUC. C4.5-SMOTE is the best from the other methods, but our gravitational proposal outperforms it, and the test reports a p value $= 0.0558$. On the other hand, DGC is the worst from among the other methods.

2) Cohen's Kappa Rate: Table IX shows the average kappa results from the cross-validation test for the different imbalanced data sets (rows) and methods (columns). DGC+ obtains the best results in 11 of the 44 data sets, and CSVM-CS achieves the best results in 12 of the 44 data sets. However, their average results and rankings are significantly different. On the other hand, NN-CS and DGC stand out for achieving the worst kappa results over imbalanced data.

The Iman and Davenport test establishes an F-distribution value $= 1.9653$ for a significance level of alpha $= 0.05$. The Iman and Davenport statistic (distributed according to the

TABLE VIII
WILCOXON TEST FOR AUC AND IMBALANCED DATA

Algorithm	DGC+	DGC	ADAC2	NN	CSVM	C4.5	C4.5	C4.5	C4.5
Re-sampling / CS	None	None	CS	CS	CS	CS	RUS	SMT	SMT-TL
abalone19	0.0185	0.0000	0.0052	-0.0010	0.0339	0.0370	0.0092	**0.0465**	0.0375
abalone9-18	0.3652	0.0000	0.3284	0.0934	**0.4341**	0.2958	0.1167	0.3554	0.2325
ecoli-0-1-3-7_vs_2-6	0.6221	0.4781	0.3632	0.0442	**0.7317**	0.5114	0.1509	0.2839	0.2665
ecoli-0_vs_1	0.9652	0.9436	0.9305	0.9598	0.9479	**0.9695**	0.9598	**0.9695**	0.9504
ecoli1	0.6783	0.5979	0.6815	0.5837	0.6926	**0.7577**	0.6843	0.7082	0.6881
ecoli2	**0.8505**	0.6718	0.6581	0.1979	0.0000	0.6803	0.6862	0.6874	0.6871
ecoli3	0.5613	0.3425	0.4968	0.3087	0.3783	0.5136	0.4597	**0.5685**	0.5466
ecoli4	**0.7221**	0.0925	0.7065	0.3152	0.6951	0.5525	0.3312	0.6905	0.5230
glass-0-1-2-3_vs_4-5-6	**0.8141**	0.6612	0.7976	0.7944	0.7441	0.7295	0.7373	0.7958	0.7858
glass-0-1-6_vs_2	0.2049	0.0000	0.0868	-0.1112	0.0000	0.1914	0.1170	**0.3853**	0.2581
glass-0-1-6_vs_5	0.5764	0.0662	0.6259	0.2666	0.0000	**0.8289**	0.4545	0.5164	0.6081
glass0	0.7037	**0.7248**	0.5812	0.2941	0.0181	0.5942	0.5942	0.5113	0.5431
glass1	0.4774	0.4501	**0.5323**	0.1956	0.2139	0.4131	0.4181	0.4468	0.5059
glass2	0.3039	0.0000	**0.3251**	-0.0507	0.0303	0.2171	0.0904	0.2380	0.3188
glass4	**0.6921**	0.1928	0.4202	0.2814	0.5595	0.3244	0.3912	0.4019	0.5872
glass5	0.5256	0.0000	0.7205	0.2263	0.7181	**0.8394**	0.4092	0.6200	0.3958
glass6	**0.8257**	0.7664	0.7412	0.6731	0.7704	0.7154	0.5907	0.7406	0.7972
habermanImb	0.1977	0.0182	0.0946	0.2399	0.0840	0.1110	0.2614	**0.2627**	0.1787
iris0	**1.0000**	0.9969	0.9846	**1.0000**	**1.0000**	0.9846	0.9846	0.9846	0.9846
new-thyroid1	**0.9648**	0.7069	0.8351	0.5898	0.9477	0.9191	0.6985	0.8504	0.9061
new-thyroid2	0.9648	0.7527	0.8923	0.5488	**0.9658**	0.9496	0.7129	0.8337	0.7686
page-blocks-1-3_vs_4	0.8725	0.3544	**0.9645**	0.5537	0.7707	0.9578	0.5443	0.8431	0.7248
page-blocks0	0.7481	0.4696	0.4374	0.2473	0.6402	**0.8254**	0.7302	0.7382	0.6814
pimaImb	0.4504	0.0682	0.3878	0.3943	**0.4551**	0.3976	0.4266	0.4064	0.3486
segment0	0.9827	0.9703	0.9662	0.0000	**0.9843**	0.9788	0.8953	0.9670	0.9670
shuttle-c0-vs-c4	0.9965	0.9830	0.9958	0.3890	**1.0000**	0.9958	**1.0000**	0.9916	0.9958
shuttle-c2-vs-c4	0.8803	0.4000	0.9297	0.1170	**1.0000**	**1.0000**	0.8575	0.6780	**1.0000**
vehicle0	0.8442	0.1530	0.8493	0.2384	**0.8896**	0.8419	0.8092	0.8186	0.7773
vehicle1	0.4281	0.0698	0.4241	0.2159	**0.4350**	0.3580	0.3019	0.3248	0.3978
vehicle2	0.8671	0.4478	**0.9358**	0.1318	0.8943	0.8700	0.8369	0.8838	0.8626
vehicle3	0.3901	0.0868	0.3920	0.2020	**0.4816**	0.3990	0.3229	0.3496	0.3909
vowel0	**0.9768**	0.6730	0.9475	0.1377	0.7627	0.8170	0.7252	0.8708	0.9006
wisconsinImb	0.9230	0.8281	0.9233	0.9164	**0.9392**	0.9114	0.8918	0.9104	0.9113
yeast-0-5-6-7-9_vs_4	0.4028	0.0000	0.3442	0.1385	0.0000	0.3655	0.3684	**0.4471**	0.3591
yeast-1-2-8-9_vs_7	0.0905	0.0000	0.1427	-0.0143	0.0000	**0.1821**	0.0417	0.0843	0.0632
yeast-1-4-5-8_vs_7	0.0600	0.0000	0.0604	-0.0017	0.0000	**0.0809**	0.0481	0.0636	0.0184
yeast-1_vs_7	**0.3169**	0.0000	0.1875	0.0332	0.0000	0.2081	0.1335	0.2388	0.2077
yeast-2_vs_4	**0.7310**	0.0343	0.6773	0.1744	0.0000	0.6510	0.6118	0.6115	0.7178
yeast-2_vs_8	0.4780	0.4607	0.2435	0.0633	0.5404	**0.6652**	0.1453	0.4415	0.4932
yeast1	0.3888	0.0327	0.2443	0.1280	0.3078	0.2834	**0.3989**	0.3849	0.2998
yeast3	0.6824	0.1659	0.6176	0.2064	0.6094	**0.7110**	0.6047	0.6999	0.6763
yeast4	0.2670	0.0000	0.2333	0.0404	0.2363	0.2632	0.1735	0.2865	**0.2872**
yeast5	0.6577	0.0825	**0.7537**	0.0278	0.4475	0.6901	0.4168	0.6504	0.6137
yeast6	**0.4042**	0.0196	0.2124	0.0133	0.2446	0.3669	0.1692	0.3423	0.3663
Avg. Values	**0.6108**	0.3355	0.5609	0.2683	0.4910	0.5899	0.4844	0.5666	0.5598
Avg. Ranks	**2.6818**	7.5341	4.3864	7.7841	4.5682	3.6818	5.9432	4.0000	4.4205

TABLE IX
COHEN'S KAPPA RATE RESULTS FOR IMBALANCED DATA SETS

DGC+ vs	R^+	R^-	p-value
DGC	990.0	0.0	1.14E-13
ADAC2	809.5	180.5	1.325E-4
NN-CS	915.0	31.0	5.40E-10
CSVM-CS	694.5	251.5	0.006719
C4.5-CS	768.0	222.0	0.001082
C4.5-RUS	834.0	156.0	3.128E-5
C4.5-SMT	659.0	331.0	0.055820
C4.5-SMT-TL	676.0	314.0	0.034240

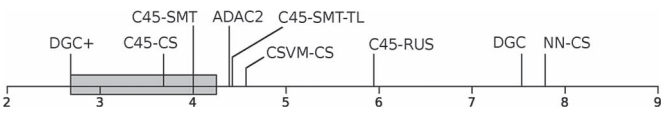


Fig. 4. Bonferroni–Dunn test for kappa and imbalanced data.

F-distribution with 8 and 344 degrees of freedom) is 28.8119 for kappa. Thus, the test rejects the null hypothesis, and therefore, it can be said that there are statistically significant differences between the kappa results of the algorithms.

TABLE X
WILCOXON TEST FOR KAPPA AND IMBALANCED DATA

DGC+ vs	R^+	R^-	p-value
DGC	984.0	6.0	1.59E-12
ADAC2	751.0	239.0	0.002298
NN-CS	941.0	5.0	2.27E-12
CSVM-CS	740.0	206.0	9.228E-4
C4.5-CS	649.0	341.0	0.073020
C4.5-RUS	964.0	26.0	1.22E-10
C4.5-SMT	771.0	219.0	9.418E-4
C4.5-SMT-TL	826.0	164.0	5.104E-5

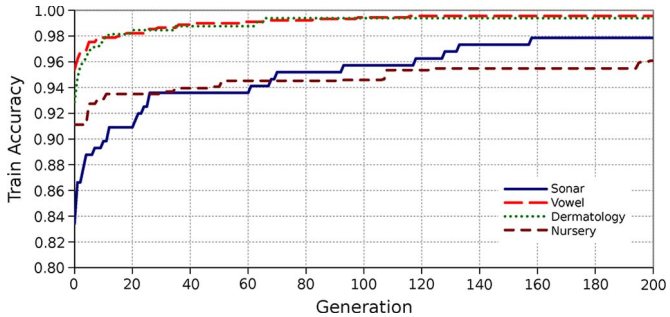


Fig. 5. Convergence rate.

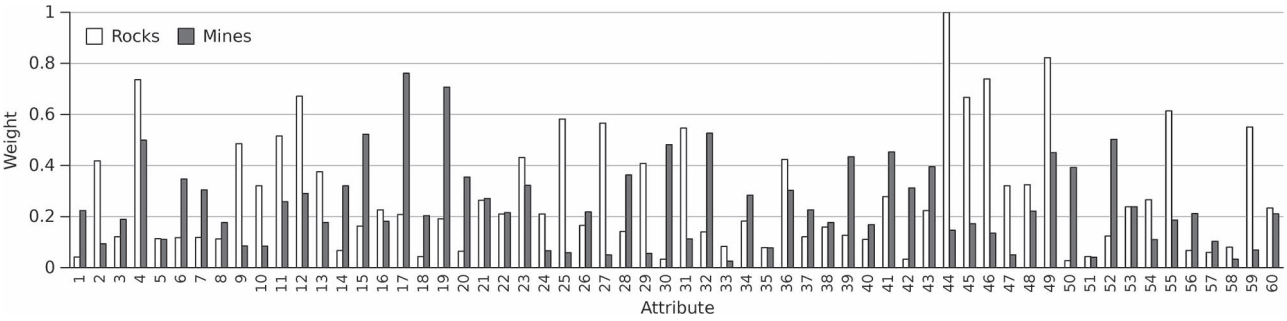


Fig. 6. Sonar weights.

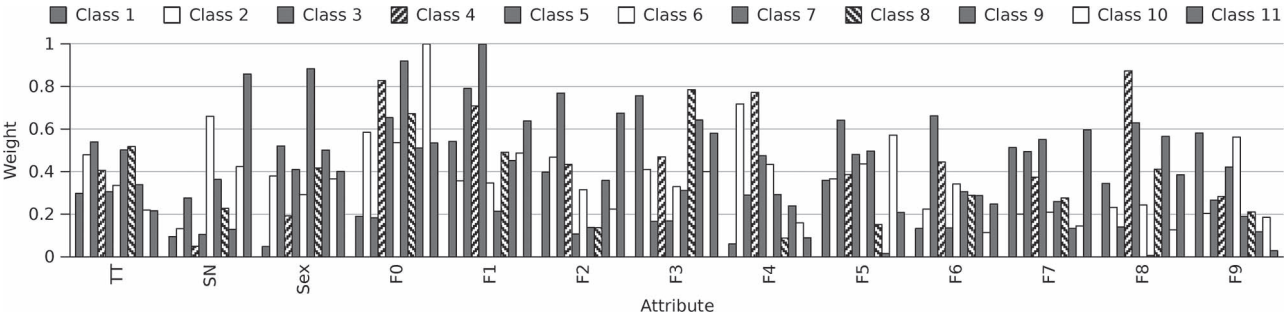


Fig. 7. Vowel weights.

Fig. 4 shows the application of the Bonferroni–Dunn test to kappa for alpha $= 0.05$, whose critical difference is 1.5905. C4.5-SMT is competitive with DGC+, together with C4.5-CS, but C4.5-SMT-TL and the other methods perform statistically worse than DGC+ regarding the Cohen's kappa rate.

Table X shows the results of the Wilcoxon rank-sum test for kappa. C4.5-CS is the best compared to the other methods, but our gravitational proposal outperforms it, and the test reports a p value $= 0.0730$. On the other hand, NN-CS and DGC are the worst from among the other methods.

C. Convergence Analysis

As mentioned in the description of the algorithm, the proposal assigns a weight to each attribute and class, and these weights allow us to recognize relevant attributes and thus improve the classification performance by giving less importance to those attributes that introduce noisy information into the learning process. This section presents the convergence rate of the best fitness values over the data sets sonar (2 classes and 60 attributes), vowel (11 classes and 13 attributes), nursery (5 classes and 8 attributes), and dermatology (6 classes and 34 attributes). These data sets have been selected because of their high dimensionality regarding the product of the number of classes and the number of attributes, which represents the genotype length of the CMA-ES algorithm, whose lengths are 120, 143, 40, and 204, respectively. Fig. 5 shows the best fitness values along the generations. The results at generation 0 indicate the initial accuracy when all of the attributes are considered with the same weight (0.5). The CMA-ES algorithm iterates, finding better weights for the different classes and attributes over the generations. When CMA-ES meets any of the stop criteria, the best solution from the CMA-ES population is selected, and its attribute-class matrix is the one shown in the previous section. The step-size control from CMA-ES effectively prevents premature convergence yet allows fast convergence to an optimum.

D. Attribute-Class Weighting Outputs

In this section, we evaluate and visualize the allocation of the weights assigned to the attributes over the data sets analyzed in the previous section. The weights assigned are shown in Figs. 6–9, and they are valued within the range [0, 1], where 0 stands for unconsidered attributes and 1 for very important ones. The most relevant attributes detected in sonar are the frequency bands 4, 44, 46, and 49 for the rock class and 17 and 19 for the mine class, whereas bands 5, 33, 51, 57, and 58 are not considered for any class. Regarding vowel data set, attributes F0 and F1 are the most relevant to classify class 10 and class 5, respectively. On the other hand, F5 is the less relevant attribute for class 9. The weights from the nursery data set indicate that the social behavior is not considered (but used for predicting the class spec_prior), whereas the health attribute is the most relevant (but for priority class). Finally, the high deviation among the weight bars of the dermatology data set indicate that the algorithm has been able to discriminate many attributes, weighted lower than 0.2, and few attributes have been considered truly relevant, weighted higher than 0.5.

Finally, it can be concluded that having different weights for each class is a convenient idea as seen in the examples (since an attribute is not equally relevant for all of the classes), which differs with conventional feature selection algorithm behaviors.

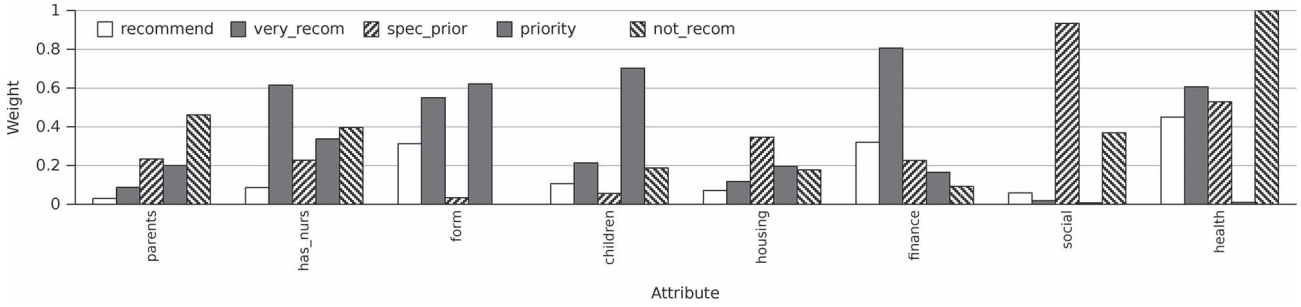


Fig. 8. Nursery weights.

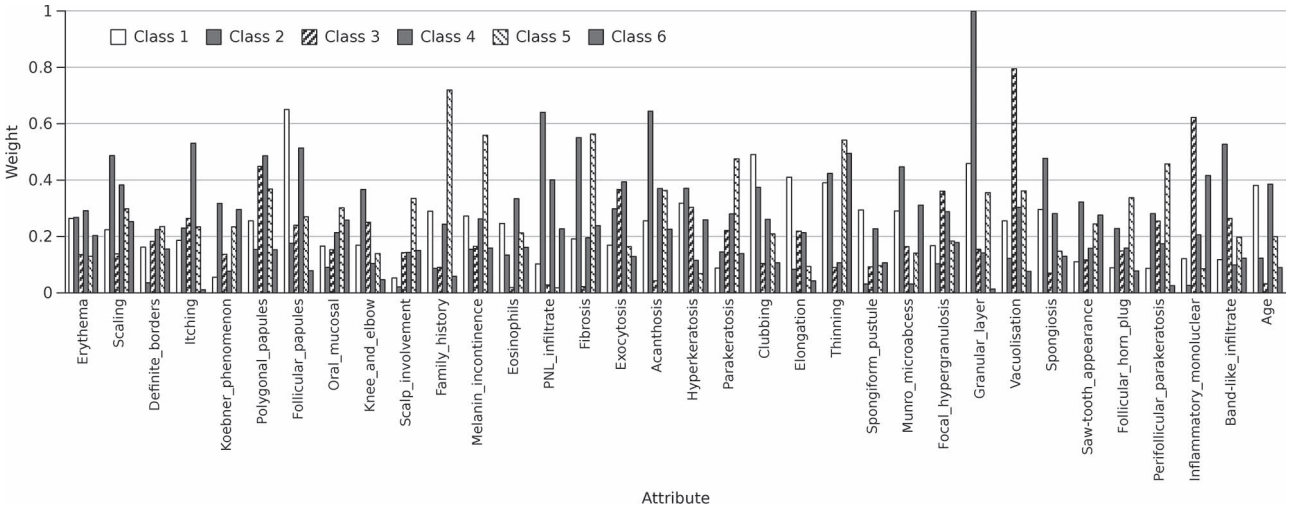


Fig. 9. Dermatology weights.

TABLE XI
TIME PERFORMANCE: DGC+ VS DGC

Data set	Train Instances	DGC Particles	Evaluation Time (ms) DGC+	DGC
Banana	4770	16	8.1707	0.0783
Ionosphere	316	308	5.1187	4.8494
Iris	135	50	0.3381	0.1433
Nursery	11664	9141	107.3627	61.6712
Page-blocks	4924	328	31.0672	1.7721
Sonar	187	187	8.0320	8.0131
Thyroid	6480	1179	87.0068	12.2841
Vowel	891	614	7.3176	4.5143

E. Time Performance: DGC+ vs DGC

Even though the experimental results advocate for the higher accuracy of the proposed DGC+ over the previous gravitation method DGC, it is interesting to analyze the time performance of both methods, considering the data reduction of DGC by means of the data particle creation, which transforms the train data into a subset of prototypes to reduce the complexity of the data set. Table XI shows the number of train instances, the number of artificial data particles created by DGC, and the evaluation time of a candidate classifier for both DGC+ and DGC over a subset of representative data sets. Evaluation times for all data sets are shown in the Web site provided in [1].

The computational complexity of the gravitation calculation is similar for both methods. However, in original DGC theory, the data particle is employed to obtain higher classification speed but at the cost of the accuracy, whereas DGC+ uses all train instances for the calculation of the gravitation. Therefore, the higher speed of the evaluation of a DGC classifier will depend on the size of the reduction of the data, which, in turn, depends on the original data distribution. Thus, while some data sets can be reduced to many fewer data particles, such as banana, page-blocks, or thyroid, others cannot be simplified. In this way, it would be recommended to apply a data reduction preprocessing prior to DGC+ classification when the computation time due to the large number of instances exceeds the target time of the researcher.

F. Discussion

This section analyzes the results from the exhaustive experimental study with respect to data problem statistics which can be helpful in showing the proposal's performance, advantages, and disadvantages under specific conditions. The proposed DGC+ is generally better than other methods and achieves the best accuracy, Cohen's kappa rate, AUC, and average and ranking results for the algorithms and data sets used in the experimental study.

On the one hand, regarding standard data performance, the 35 data sets comprise an average number of 12.48 attributes, 1495.65 examples, and 3.74 classes. DGC+ obtains the first and second best results in 22 of the 35 data sets, whose data subset comprises an average number of 10.54 attributes, 1564.04 examples, and 3.22 classes. DGC+ worst results are among the fifth and sixth in 6 of the 35 data sets, whose data subset comprises an average number of 8 attributes, 2032 examples, and 3.33 classes.

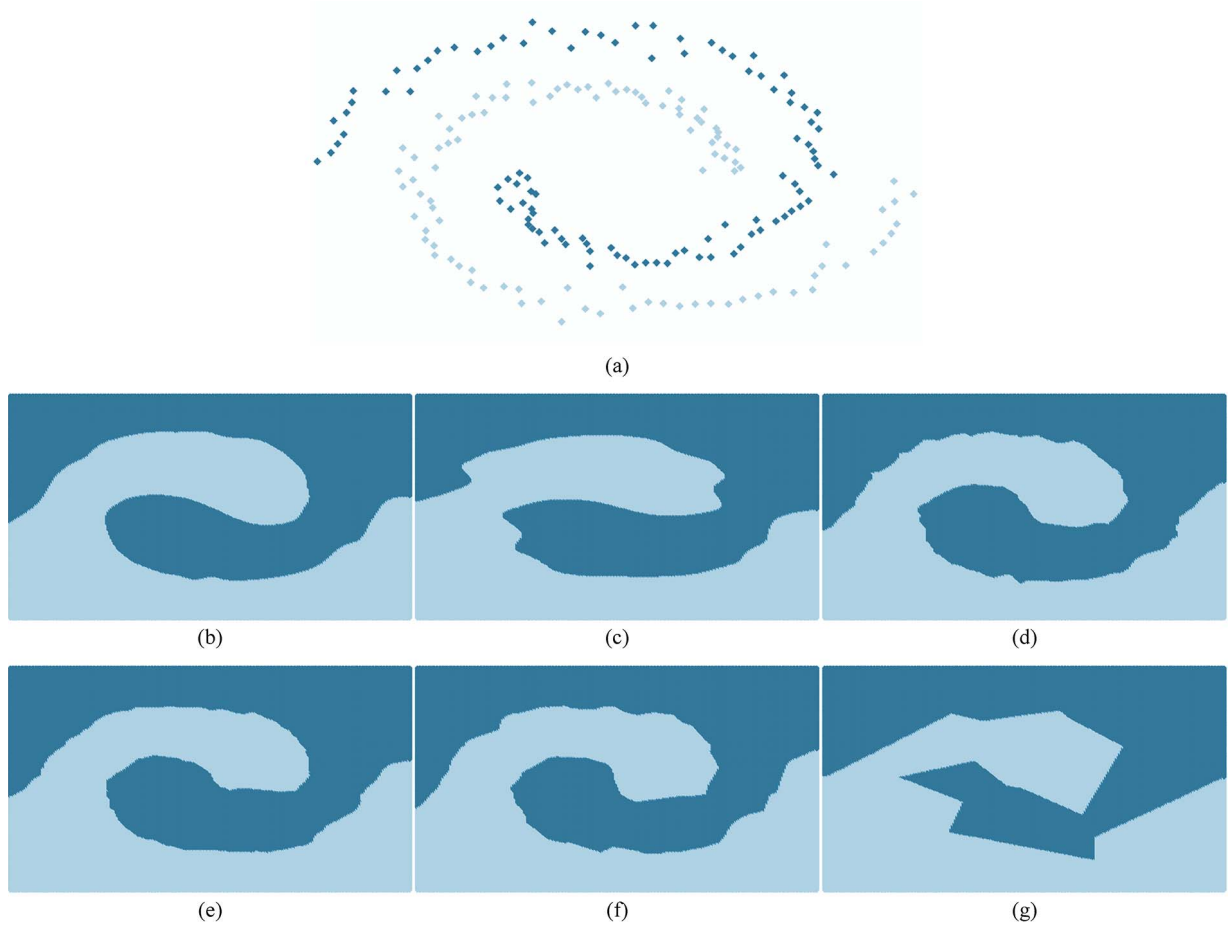


Fig. 10. Two spiral data set predictions. (a) Data set. (b) DGC+. (c) DGC. (d) KNN. (e) KNN-A. (f) DW-KNN. (g) SSMA+SFLSDE.

On the other hand, regarding imbalance data performance, the 44 data sets comprise an average number of 9.13 attributes, 831.5 examples, and an imbalance ratio of 14.55. DGC+ obtains the first and second best results in 26 of the 44 data sets, whose data subset comprises an average number of 9.42 attributes, 869.76 examples, and an imbalance ratio of 15.78. DGC+ worst results are among the sixth, seventh, and eighth in 4 of the 44 data sets, whose data subset comprises an average number of 9.25 attributes, 674.75 examples, and an imbalance ratio of 17.98.

These results do not indicate solid evidences to conclude that the proposal will perform badly in certain classification domains. However, the lack of minority class examples in extremely high imbalance data classification might result to a condition where the minority class examples do not sufficiently represent other minority class examples. On the other hand, attribute-class weight learning has demonstrated to overcome noisy and irrelevant data. Thus, the proposal will perform accurate classification in these domains, better than those from other methods without distance weighting or feature selection.

Finally, it is interesting to show the smoothness of the classification border between data class predictions. Fig. 10 compares the classification performance of the best ranked classification methods used in the experimental study for standard data sets, over the two spiral data sets with Gaussian noise. The proposed DGC+ obtains accurate and smooth border classification. DGC is noticed to suffer from data particle prototype generation, achieving a worse local smoothness. KNN methods provide less smooth predictions than DGC+ by means of sawtooth predictions. SSMA+SFLSDE classifies according to the coarse prototypes generated.

VI. Conclusion

In this paper, a DGC algorithm called DGC+ has been presented. The proposal includes attribute-class weight learning for distance weighting to improve classification results. The weights were optimized by means of the CMA-ES algorithm, which showed to perform an effective learning rate of optimal weights for the different attributes and classes, ignoring noisy attributes and enhancing relevant ones. The effects of gravitation around the instances allowed an accurate classification considering both local and global data information, providing smooth classification and good generalization. Gravitation was successfully adapted to deal with imbalanced data problems. The proposal achieved better classification accuracy, Cohen's kappa rate, and AUC results than other well-known instance-based and imbalanced classification methods. The results were validated using multiple and pairwise nonparametric statistical tests, whose reports support the statistically significant better performance of the proposal.

REFERENCES

[1] M. Paliwal and U. A. Kumar, "Neural networks and statistical techniques: A review of applications," *Expert Syst. Appl.*, vol. 36, no. 1, pp. 2–17, Jun. 2009.

[2] A. Widodo and B. S. Yang, "Support vector machine in machine condition monitoring and fault diagnosis," *Mech. Syst. Signal Process.*, vol. 21, no. 6, pp. 2560–2574, Aug. 2007.

[3] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-based learning algorithms," *Mach. Learn.*, vol. 6, no. 1, pp. 37–66, Jan. 1991.

[4] P. G. Espejo, S. Ventura, and F. Herrera, "A survey on the application of genetic programming to classification," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 40, no. 2, pp. 121–144, Mar. 2010.

[5] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theory*, vol. IT-13, no. 1, pp. 21–27, Jan. 1967.

[6] I. Kononenko and M. Kukar, *Machine Learning and Data Mining: Introduction to Principles and Algorithms*. Cambridge, U.K.: Horwood Publ., 2007.

[7] B. Li, Y. W. Chen, and Y. Q. Chen, "The nearest neighbor algorithm of local probability centers," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 38, no. 1, pp. 141–154, Feb. 2008.

[8] L. Peng, B. Peng, Y. Chen, and A. Abraham, "Data gravitation based classification," *Inf. Sci.*, vol. 179, no. 6, pp. 809–819, Mar. 2009.

[9] C. Wang and Y. Q. Chen, "Improving nearest neighbor classification with simulated gravitational collapse," in *Proc. ICNC*, 2005, vol. 3612, pp. 845–854.

[10] Y. Zong-Chang, "A vector gravitational force model for classification," *Pattern Anal. Appl.*, vol. 11, no. 2, pp. 169–177, May 2008.

[11] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evol. Comput.*, vol. 9, no. 2, pp. 159–195, Jun. 2001.

[12] N. Hansen, "The CMA evolution strategy: A comparing review," in *Towards a New Evolutionary Computation. Advances on Estimation of Distribution Algorithms*, J. A. Lozano, P. Larranaga, I. Inza, and E. Bengoetxea, Eds. New York: Springer-Verlag, 2006, pp. 75–102.

[13] J. Alcalá-Fdez, A. Fernandez, J. Luengo, J. Derrac, S. García, L. Sánchez, and F. Herrera, "KEEL data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework," *J. Multiple-Valued Logic Soft Comput.*, vol. 17, pp. 255–287, 2011.

[14] A. Frank and A. Asuncion, UCI machine learning repository, Univ. California, School Inf. Comput. Sci., Irvine, CA. [Online]. Available: http://archive.ics.uci.edu/ml/citation_policy.html

[15] J. Alcalá-Fdez, L. Sánchez, S. García, M. del Jesus, S. Ventura, J. Garrell, J. Otero, C. Romero, J. Bacardit, V. Rivas, J. Fernández, and F. Herrera, "KEEL: A software tool to assess evolutionary algorithms for data mining problems," *Soft Comput.—Fusion Found., Methodol. Appl.*, vol. 13, no. 3, pp. 307–318, Oct. 2009.

[16] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemannr, and I. H. Witten, "The WEKA data mining software: An update," *SIGKDD Explorat.*, vol. 11, no. 1, pp. 10–18, Jun. 2009.

[17] A. Ben-David, "Comparison of classification accuracy using Cohen's weighted kappa," *Expert Syst. Appl.*, vol. 34, no. 2, pp. 825–832, Feb. 2008.

[18] A. Ben-David, "About the relationship between ROC curves and Cohen's kappa," *Eng. Appl. Artif. Intell.*, vol. 21, no. 6, pp. 874–882, Sep. 2008.

[19] A. P. Bradley, "The use of the area under the ROC curve in the evaluation of machine learning algorithms," *Pattern Recognit.*, vol. 30, no. 7, pp. 1145–1159, Jul. 1997.

[20] J. Huang and C. X. Ling, "Using AUC and accuracy in evaluating learning algorithms," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 3, pp. 299–310, Mar. 2005.

[21] O. J. Dunn, "Multiple comparisons among means," *J. Amer. Stat. Assoc.*, vol. 56, no. 293, pp. 52–64, Mar. 1961.

[22] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometr. Bull.*, vol. 1, no. 6, pp. 80–83, Dec. 1945.

[23] S. García, A. Fernández, J. Luengo, and F. Herrera, "Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power," *Inf. Sci.*, vol. 180, no. 10, pp. 2044–2064, May 2010.

[24] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*. London, U.K.: Chapman & Hall, 2007.

[25] S. A. Dudani, "The distance-weighted k-nearest-neighbor rule," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. SMC-6, no. 4, pp. 325–327, Apr. 1976.

[26] Q. Gao and Z. Wang, "Center-based nearest neighbor classifier," *Pattern Recognit.*, vol. 40, no. 1, pp. 346–349, Jan. 2007.

[27] J. Wang, P. Neskovic, and L. N. Cooper, "Improving nearest neighbor rule with a simple adaptive distance measure," *Pattern Recognit. Lett.*, vol. 28, no. 2, pp. 207–213, Jan. 2007.

[28] R. Paredes and E. Vidal, "A class-dependent weighted dissimilarity measure for nearest neighbor classification problems," *Pattern Recognit. Lett.*, vol. 21, no. 12, pp. 1027–1036, Dec. 2000.

[29] R. Paredes and E. Vidal, "Learning weighted metrics to minimize nearest-neighbor classification error," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 7, pp. 1100–1110, Jul. 2006.

[30] S. Garcia, J. Derrac, J. R. Cano, and F. Herrera, "Prototype selection for nearest neighbor classification: Taxonomy and empirical study," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 3, pp. 417–435, Mar. 2012.

[31] I. Triguero, J. Derrac, S. Garcia, and F. Herrera, "A taxonomy and experimental study on prototype generation for nearest neighbor classification," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 42, no. 1, pp. 86–100, Jan. 2012.

[32] R. Paredes and E. Vidal, "Learning prototypes and distances: A prototype reduction technique based on nearest neighbor error minimization," *Pattern Recognit.*, vol. 39, no. 2, pp. 180–188, Feb. 2006.

[33] C. Zhou and Y. Chen, "Improving nearest neighbor classification with cam weighted distance," *Pattern Recognit.*, vol. 39, no. 4, pp. 635–645, Apr. 2006.

[34] I. Triguero, S. García, and F. Herrera, "Differential evolution for optimizing the positioning of prototypes in nearest neighbor classification," *Pattern Recognit.*, vol. 44, no. 4, pp. 901–916, Apr. 2011.

[35] S. García, J. R. Cano, and F. Herrera, "A memetic algorithm for evolutionary prototype selection: A scaling up approach," *Pattern Recognit.*, vol. 41, no. 8, pp. 2693–2709, Aug. 2008.

[36] F. Neri and V. Tirronen, "Scale factor local search in differential evolution," *Memetic Comput.*, vol. 1, no. 2, pp. 153–171, Jun. 2009.

[37] M. Z. Jahromi, E. Parvinnia, and R. John, "A method of learning weighted similarity function to improve the performance of nearest neighbor," *Inf. Sci.*, vol. 179, no. 17, pp. 2964–2973, Aug. 2009.

[38] W. E. Wright, "Gravitational clustering," *Pattern Recognit.*, vol. 9, no. 3, pp. 151–166, Oct. 1977.

[39] Y. Endo and H. Iwata, "Dynamic clustering based on universal gravitation model," *Model. Decisions Artif. Intell.*, vol. 3558, pp. 183–193, 2005.

[40] B. Yang, L. Peng, Y. Chen, H. Liu, and R. Yuan, "A DGC-based data classification method used for abnormal network intrusion detection," in *Proc. ICONIP*, 2006, vol. 4234, pp. 209–216.

[41] J. Giblin, D. Marolf, and R. Garvey, "Spacetime embedding diagrams for spherically symmetric black holes," *Gen. Relativ. Gravit.*, vol. 36, no. 1, pp. 83–99, Jan. 2004.

[42] L. Al Shalabi, Z. Shaaban, and B. Kasasbeh, "Data mining: A preprocessing engine," *J. Comput. Sci.*, vol. 2, no. 9, pp. 735–739, Sep. 2006.

[43] N. K. Visalakshi and K. Thangavel, "Impact of normalization in distributed k-means clustering," *Int. J. Soft Comput.*, vol. 4, no. 4, pp. 168–172, 2009.

[44] J. Han, *Data Mining: Concepts and Techniques*. San Francisco, CA: Morgan Kaufmann, 2005.

[45] G. V. Lashkia and L. Anthony, "Relevant, irredundant feature selection and noisy example elimination," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 34, no. 2, pp. 888–897, Apr. 2004.

[46] B. Apolloni, S. Bassis, and A. Brega, "Feature selection via Boolean independent component analysis," *Inf. Sci.*, vol. 179, no. 22, pp. 3815–3831, Nov. 2009.

[47] T. N. Lal, O. Chapelle, J. Western, and A. Elisseeff, "Embedded methods," *Stud. Fuzziness Soft Comput.*, vol. 207, no. 1, pp. 137–165, Jan. 2006.

[48] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 9, pp. 1263–1284, Sep. 2009.

[49] H. Kaizhu, Y. Haiqin, K. Irwinng, and M. R. Lyu, "Imbalanced learning with a biased minimax probability machine," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 36, no. 4, pp. 913–923, Aug. 2006.

[50] R. Akbani, S. Kwek, and N. Japkowicz, "Applying support vector machines to imbalanced datasets," in *Proc. 15th ECML*, 2004, pp. 39–50.

[51] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proc. 14th Int. Joint Conf. Artif. Intell.*, San Francisco, CA, 1995, vol. 2, pp. 1137–1143.

[52] T. Wiens, B. Dale, M. Boyce, and G. Kershaw, "Three way k-fold cross-validation of resource selection functions," *Ecolog. Model.*, vol. 212, no. 3/4, pp. 244–255, Apr. 2008.

[53] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Inf. Process. Manag.*, vol. 45, no. 4, pp. 427–437, Jul. 2009.

[54] Q. Gu, L. Zhu, and Z. Cai, "Evaluation measures of the classification performance of imbalanced data sets," *Commun. Comput. Inf. Sci.*, vol. 51, no. 1, pp. 461–471, Oct. 2009.

[55] A. Fernández, S. García, J. Luengo, E. Bernado-Mansilla, and F. Herrera, "Genetics-based machine learning for rule induction: State of the art, taxonomy, and comparative study," *IEEE Trans. Evol. Comput.*, vol. 14, no. 6, pp. 913–941, Dec. 2010.
[56] M. Galar, A. Fernández, E. Barrenechea, H. Bustince, and F. Herrera, "An overview of ensemble methods for binary classifiers in multi-class problems: Experimental study on one-vs-one and one-vs-all schemes," *Pattern Recognit.*, vol. 44, no. 8, pp. 1761–1776, Aug. 2011.
[57] A. Fernández, M. J. del Jesus, and F. Herrera, "On the 2-tuples based genetic tuning performance for fuzzy rule based classification systems in imbalanced data-sets," *Inf. Sci.*, vol. 180, no. 8, pp. 1268–1291, Apr. 2010.
[58] G. Batista, R. Prati, and M. Monard, "A study of the behavior of several methods for balancing machine learning training data," *SIGKDD Explorat.*, vol. 6, no. 1, pp. 20–29, Jun. 2004.
[59] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *Artif. Intell. Res.*, vol. 16, no. 1, pp. 321–357, Jun. 2002.
[60] I. Tomek, "Two modifications of CNN," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 6, no. 11, pp. 769–772, Nov. 1976.
[61] Y. Sun, M. Kamel, A. Wong, and Y. Wang, "Cost-sensitive boosting for classification of imbalanced data," *Pattern Recognit.*, vol. 40, no. 12, pp. 3358–3378, Dec. 2007.
[62] Z. H. Zhou and X. Y. Liu, "Training cost-sensitive neural networks with methods addressing the class imbalance problem," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 1, pp. 63–77, Jan. 2006.
[63] Y. Tang, Y.-Q. Zhang, and N. Chawla, "SVMS modeling for highly imbalanced classification," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 39, no. 1, pp. 281–288, Feb. 2009.
[64] K. M. Ting, "An instance-weighting method to induce cost-sensitive trees," *IEEE Trans. Knowl. Data Eng.*, vol. 14, no. 3, pp. 659–665, May/Jun. 2002.
[65] D. R. Wilson and T. R. Martinez, "Reduction techniques for instance-based learning algorithms," *Mach. Learn.*, vol. 38, no. 3, pp. 257–286, Mar. 2000.
[66] S. Ventura, C. Romero, A. Zafra, J. A. Delgado, and C. Hervás, "JCLEC: A java framework for evolutionary computation," *Soft Comput.—Fusion Found., Methodol. Appl.*, vol. 12, no. 4, pp. 381–392, Oct. 2007.
[67] S. García, D. Molina, M. Lozano, and F. Herrera, "A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: A case study," *J. Heurist.*, vol. 15, no. 6, pp. 617–644, Dec. 2009.
[68] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, no. 1, pp. 1–30, Jan. 2006.
[69] S. García and F. Herrera, "An extension on statistical comparisons of classifiers over multiple data sets for all pairwise comparisons," *J. Mach. Learn. Res.*, vol. 9, no. 12, pp. 2677–2694, Dec. 2008.

Alberto Cano (M'10) was born in Cordoba, Spain, in 1987. He received the M.Sc. degree in computer science from the Department of Computer Science and Numerical Analysis, University of Cordoba, Cordoba, where he is currently working toward the Ph.D. degree.

His research is performed as a member of the Knowledge Discovery and Intelligent Systems Research Laboratory and is focused on general purpose GPU systems, parallel computing, soft computing, machine learning, and data mining and its applications.

Amelia Zafra (M'08) was born in Pamplona, Spain, in 1982. She received the B.Sc. and Ph.D. degrees from the University of Granada, Granada, Spain, in 2005 and 2009, respectively.

She is an Associate Professor of computer science and artificial intelligence with the University of Cordoba, Cordoba, Spain. Her teaching is devoted to artificial intelligence, bioinformatics in computer science, and data mining. Her research is performed as a member of the Knowledge Discovery and Intelligent Systems Research Laboratory and is focused on soft computing, machine learning, and data mining and its applications.

Sebastián Ventura (M'07–SM'09) was born in Cordoba, Spain, in 1966. He received the B.Sc. and Ph.D. degrees from the University of Cordoba, Cordoba, Spain, in 1989 and 1996, respectively.

He is an Associate Professor of computer science and artificial intelligence with the University of Cordoba, Cordoba, Spain, where he heads the Knowledge Discovery and Intelligent Systems Research Laboratory. He is the author or coauthor of more than 90 international publications. He has also been engaged in 11 research projects (being the coordinator of two of them) supported by the Spanish and Andalusian Governments and the European Union, concerning several aspects of the area of evolutionary computation, machine learning, and data mining and its applications.

TITLE:

A Classification Module for Genetic Programming Algorithms in JCLEC

AUTHORS:

A. Cano, J. M. Luna, A. Zafra, and S. Ventura

Journal of Machine Learning Research, *Submitted, 2013*

RANKING:

Impact factor (JCR 2012): 3.420

Knowledge area:

Computer Science, Artificial Intelligence: 10/114

Automation and Control Systems: 2/58

Journal of Machine Learning Research 01 (2013) 1-4 Submitted 01/13; Published 01/13

A Classification Module for Genetic Programming Algorithms in JCLEC

Alberto Cano ACANO@UCO.ES
José María Luna JMLUNA@UCO.ES
Amelia Zafra AZAFRA@UCO.ES
Sebastián Ventura SVENTURA@UCO.ES
Department of Computer Science and Numerical Analysis
University of Córdoba, Spain

Editor: Editor name

Abstract

JCLEC-Classification is a usable and extensible open source library for genetic programming classification algorithms. It houses implementations of rule-based methods for classification based on genetic programming, supporting multiple model representations and providing to users the tools to implement any classifier easily. The software is written in Java and it is available from `http://jclec.sourceforge.net/classification` under the GPL license.

Keywords: Classification, Evolutionary Algorithms, Genetic Programming, JCLEC

1. Introduction

In the last decade, the increasing interest in storing information has led to its automatic processing, discovering knowledge that is potentially useful. Data mining involves the use of data analysis tools to discover this knowledge previously unknown, valid patterns, and close relationships in databases. One of the most used data mining tasks is classification, which learns from a set of training examples to produce predictions about future examples.

The classification models are being applied to enormous databases in areas such as bioinformatics, marketing, banks or web mining. Existing classification libraries provide algorithms following many different methodologies. However, it is difficult to find a library that contains GP (Genetic Programming) algorithms, an important evolutionary computation paradigm. The conceptual difficulty of GP makes it difficult to implement algorithms following this paradigm despite its algorithms perform well as it is proved by many researchers (Espejo et al., 2010).

GP is an efficient and flexible heuristic technique that uses complex representations such as trees. This technique provides comprehensible models, which are useful in different application domains. For instance, it is applied to supervised learning tasks like regression, classification and unsupervised learning tasks like clustering and association. In classification tasks, the application of GP is an important issue since it may offer results that are comprehensible to humans. Additionally, it offers interesting advantages such as flexibility, and the possiblity of using different kinds of representations, e.g., decision trees, rule-based

systems, discriminant functions, etc. An extension of GP is grammar-guided genetic programming (G3P), which makes the knowledge extracted more expressive and flexible by means of a context-free grammar (McKay et al., 2010).

This paper presents an open source software for researchers and end-users to develop classification algorithms based on GP and G3P models. It is an intuitive and usable tool which extends the JCLEC evolutionary computation library (Ventura et al., 2007). The software presented includes some GP and G3P proposals described in literature, and provides the necessary classes and methods to develop any kind of evolutionary algorithms for solving classification problems easily.

This paper is organized as follows. Firstly, Section 2 provides a description of the module, its structure and the way to use it. Finally, the documentation and the requirements of this module are outlined in Section 3.

2. Description of the module

The classification module is presented in this section, describing the library structure and its main characteristics.

2.1 Structure of the module

The *net.sf.jclec.problem.classification.base* package roots the hierarchical structure of the classification module, and provides the abstract classes with the properties and methods that any classification algorithm must contain, e.g., *ClassificationAlgorithm*, *ClassificationReporter*, *Rule* and *RuleBase*. A new algorithm included in the module should inherit from these classes regardless the classification model. In this context, we focus on rule-based classifiers which comprise one or more classification rules, each of them being a knowledge representation model consisting of an antecedent and a consequent. The antecedent of each classification rule is made up of a series of conditions to be met by an instance to consider that it belongs to the class specified by the consequent.

Based on whether an algorithm uses a GP or G3P encoding, JCLEC-Classification makes a differentiation between expression-tree and syntax-tree respectively. In such a way, each GP classification individual is represented by means of the *ExprTreeRuleIndividual* class, which represents an individual, comprising all the features required to do it: the genotype, the phenotype and the fitness function value. The nodes and functions in GP trees are defined by the *ExprTreeSpecies* class. Similarly to GP individuals, the *SyntaxTreeRuleIndividual* class specifies all the features required to represent a G3P individual, while the *SyntaxTreeSpecies* allows us to define the terminal and nonterminal symbols of the grammar used to generate individuals. Furthermore, the module allows to encode multiple syntax and expression trees for Pittsburgh style encodings or Multi Expression Programming by means of the *MultiExprTree* and *MultiSyntaxTree* classes.

In order to represent the phenotype of a rule-base individual, crisp and fuzzy rules are generated by using the *CrispRule* and *FuzzyRule* classes, respectively. These classes provide the antecedent of the rule in an expression-tree shape and the consequent assigned to this antecedent. In addition, methods to classify a whole dataset or a particular instance are provided in these classes. These methods compute whether the antecedent of a rule satisfies an instance, returning the consequent of the rule, otherwise the instance is not covered by

the antecedent and therefore no predictions can be made. Besides those packages that represent the main characteristics of any individual, the *net.sf.jclec.problem.classification.listener* package to make reports for the train and test classification processes is provided. This package contains the *RuleBaseReporter* class with methods to make reports specifying the classifier features such as the rule base, the number of rules, the average number of conditions, the percentage of correct predictions, the percentage of correct predictions per class, the geometric mean, the kappa rate and the confusion matrix.

Finally, it is noteworthy that several utility classes, which make it easy to load data from KEEL [1] and ARFF [2] formatted files, are provided by a *dataset* package. Three different attribute types may be represented by this package, integer, continuous and categorical, and a number of characteristics from the dataset are given, comprising type of attributes, number of classes, number of instances, etc.

The module houses three G3P classification algorithms (De Falco et al., 2001; Bojarczuk et al., 2004; Tan et al., 2002), which can guide developers to write new algorithms.

2.2 Usage of the module

Including new classification algorithms in this module is very simple. We focus on the algorithm described by Bojarczuk et al. (Bojarczuk et al., 2004). This algorithm, which is provided in the module (see the *net.sf.jclec.problem.classification.algorithm.bojarczuk* package), is constructed with only three additional classes. One of them, the *BojarczukAlgorithm* class is inherited from the *ClassificationAlgorithm* class and provides the own features of this algorithm.

Another class required to be implemented is the evaluator, which computes the fitness of the individuals. This class, named *BojarczukEvaluator* in this algorithm, inherits from the JCLEC core *AbstractParallelEvaluator* class or from the *AbstractEvaluator* class, depending on whether the individuals are evaluated in a sequential or parallel way.

Finally, a class to define the grammar to be followed in the individual generation stage is implemented. This class, named *BojarczukSyntaxTreeSpecies* in this example, inherits from the class *SyntaxTreeSpecies* since G3P individuals are defined in this algorithm.

Only defining these three classes, the complete classification algorithm is represented. Due to the core of this module is JCLEC, before an algorithm is ready to run, it is necessary to carry out a set-up process by using a configuration file as shown in Figure 1. This configuration file and the steps required to execute the algorithm are described in the JCLEC website. In this file we specify those parameters required such as the algorithm to be run, the parent selector, the genetic operators, the evaluator, etc. All the required parameters are provided by JCLEC, existing a numerous variety of them as it is described in the JCLEC specification (Ventura et al., 2007).

3. Documentation and Requirements

The JCLEC-Classification online documentation [3] describes the software packages, presents a user oriented usage example, as well as developer information to include new algorithms,

1. http://www.keel.es
2. http://www.cs.waikato.ac.nz/ml/weka/arff.html
3. http://jclec.sourceforge.net/data/JCLEC-classification.pdf

Figure 1: Sample configuration file

```
<experiment>
   <process algorithm-type="net.sf.jclec.problem.classification.algorithm.bojarczuk.BojarczukAlgorithm">
      <rand-gen-factory seed="123456789" type="net.sf.jclec.util.random.RanecuFactory"/>
      <population-size>100</population-size>
      <max-of-generations>100</max-of-generations>
      <max-deriv-size>20</max-deriv-size>
      <dataset type="net.sf.jclec.problem.util.dataset.ArffDataSet">
         <train-data>data/iris/iris-10-1tra.arff</train-data>
          <test-data>data/iris/iris-10-1tst.arff</test-data>
         <attribute-class-name>Class</attribute-class-name>
      </dataset>
      <recombination-prob>0.8</recombination-prob>
      <copy-prob>0.01</copy-prob>
      <listener type="net.sf.jclec.problem.classification.listener.RuleBaseReporter">
         <report-dir-name>reports/reportFreitas</report-dir-name>
         <global-report-name>summaryFreitas</global-report-name>
         <report-frequency>10</report-frequency>
      </listener>
   </process>
</experiment>
```

API reference and running tests. JCLEC requires Java 1.6, Apache commons logging 1.1, Apache commons collections 3.2, Apache commons configuration 1.5, Apache commons lang 2.4, and JUnit 4.5 (for running tests).

Acknowledgments

This work has been financed in part by the TIN2008-06681-C06-03 project of the Spanish Inter-Ministerial Commission of Science and Technology (CICYT), the P08-TIC-3720 project of the Andalusian Science and Technology Department, and FEDER funds.

References

C. C. Bojarczuk, H. S. Lopes, A. A. Freitas, and E. L. Michalkiewicz. A constrained-syntax genetic programming system for discovering classification rules: application to medical data sets. *Artificial Intelligence in Medicine*, 30(1):27–48, 2004.

I. De Falco, A. Della Cioppa, and E. Tarantino. Discovering interesting classification rules with genetic programming. *Applied Soft Computing*, 1(4):257–269, 2001.

P. G. Espejo, S. Ventura, and F. Herrera. A Survey on the Application of Genetic Programming to Classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 40(2):121–144, 2010.

R. McKay, N. Hoai, P. Whigham, Y. Shan, and M. ONeill. Grammar-based Genetic Programming: a Survey. *Genetic Programming and Evolvable Machines*, 11:365–396, 2010. ISSN 1389-2576.

K. C. Tan, A. Tay, T. H. Lee, and C. M. Heng. Mining multiple comprehensible classification rules using genetic programming. In *Proceedings of the Evolutionary Computation on 2002. CEC '02*, volume 2, pages 1302–1307, 2002.

S. Ventura, C. Romero, A. Zafra, J.A. Delgado, and C. Hervás. JCLEC: a Java Framework for Evolutionary Computation. *Soft Computing*, 12:381–392, 2007.

Other publications related to the Ph.D. dissertation

Software results and other applications

The software developed for classification models was made publicly available under the JCLEC software [54] and produced another journal publication which presents a classification module for JCLEC [89]. It is available through the JCLEC website at http://jclec.sf.net/classification

Finally, we applied the developed models to other data mining tasks, heuristics, and real-world applications, farther than the initial objectives aimed in this dissertation.

The GPU parallelization methodology was also applied on an Ant Programming algorithm [90], the association rule mining problem [91], and the numeric data discretization problem [92], all achieving high performance and good scalability to increasing size of the data. This wide research over multiple domains demonstrates the broad scope of GPU computing application to data mining problems and its excellent performance.

The ICRM model was also applied on real-world data to predict student's failure in high school students from Zacatecas, Mexico [79]. The highly comprehensible rule-based classifiers it created allowed for understanding of reasons of student's failure. This model helped teachers to support their decisions to improve teaching and preventing school failure.

Performance on imbalanced data was also improved by proposing a new algorithm capable of discretizing imbalanced data appropriately, named ur-CAIM [93]. This model demonstrated to discretize accurately both balanced and imbalanced data while most of the recently proposed state-of-the-art discretization methods failed significantly on imbalanced data.

TITLE:

Parallel Multi-Objective Ant Programming for Classification Using GPUs

AUTHORS:

A. Cano, J. L. Olmo, and S. Ventura

Journal of Parallel and Distributed Computing, *Volume 73, Issue 6, pp. 713-728, 2013*

RANKING:

Impact factor (JCR 2011): 0.859

Knowledge area:

Computer Science, Theory and Methods: 40/100

DOI: 10.1016/J.JPDC.2013.01.017

Abstract:

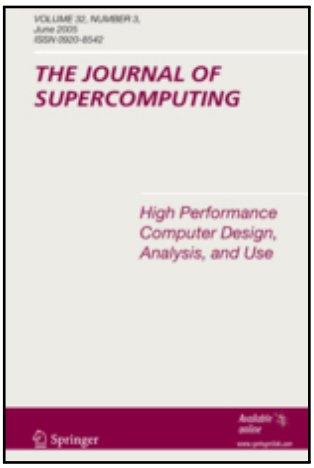
Classification using Ant Programming is a challenging data mining task which demands a great deal of computational resources when handling data sets of high dimensionality. This paper presents a new parallelization approach of an existing multi-objective Ant Programming model for classification, using GPUs and the NVIDIA CUDA programming model. The computational costs of the different steps of the algorithm are evaluated and it is discussed how best to parallelize them. The features of both the CPU parallel and GPU versions of the algorithm are presented. An experimental study is carried out to evaluate the performance and efficiency of the interpreter of the rules, and reports the execution times and speedups regarding variable population size, complexity of the rules mined and dimensionality of the data sets. Experiments measure the original single-threaded and the new multi-threaded CPU and GPU times with different number of GPU devices. The results are reported in terms of the number of Giga GP operations per second of the interpreter (up to 10 billion GPops/s) and the speedup achieved (up to 834x vs CPU, 212x vs 4–threaded CPU). The proposed GPU model is demonstrated to scale efficiently to larger datasets and to multiple GPU devices, which allows of expanding its applicability to significantly more complicated data sets, previously unmanageable by the original algorithm in reasonable time.

TITLE:

High Performance Evaluation of Evolutionary-Mined Association Rules on GPUs

AUTHORS:

A. Cano, J. M. Luna, and S. Ventura



Journal of Supercomputing, *Volume 66, Issue 3, pp. 1438-1461, 2013*

RANKING:

Impact factor (JCR 2012): 0.917

Knowledge area:

Computer Science, Hardware and Architecture: 28/50

Computer Science, Theory and Methods: 39/100

Engineering, Electrical and Electronic: 143/242

DOI: 10.1007/s11227-013-0937-4

Abstract:

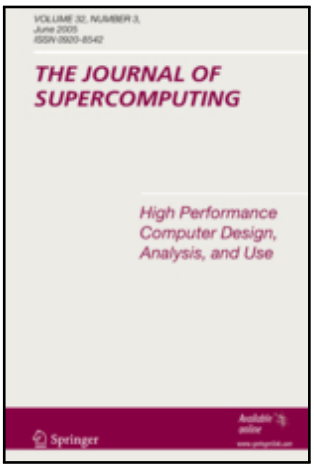
Association rule mining is a well-known data mining task, but it requires much computational time and memory when mining large scale data sets of high dimensionality. This is mainly due to the evaluation process, where the antecedent and consequent in each rule mined are evaluated for each record. This paper presents a novel methodology for evaluating association rules on graphics processing units (GPUs). The evaluation model may be applied to any association rule mining algorithm. The use of GPUs and the compute unified device architecture (CUDA) programming model enables the rules mined to be evaluated in a massively parallel way, thus reducing the computational time required. This proposal takes advantage of concurrent kernels execution and asynchronous data transfers, which improves the efficiency of the model. In an experimental study, we evaluate interpreter performance and compare the execution time of the proposed model with regard to single-threaded, multi-threaded, and graphics processing unit implementation. The results obtained show an interpreter performance above 67 billion giga operations per second, and speed-up by a factor of up to 454 over the single-threaded CPU model, when using two NVIDIA 480 GTX GPUs. The evaluation model demonstrates its efficiency and scalability according to the problem complexity, number of instances, rules, and GPU devices.

TITLE:

Scalable CAIM Discretization on Multiple GPUs Using Concurrent Kernels

AUTHORS:

A. Cano, S. Ventura, and K. J. Cios



Journal of Supercomputing, *Submitted, 2013*

RANKING:

Impact factor (JCR 2012): 0.917

Knowledge area:

Computer Science, Hardware and Architecture: 28/50

Computer Science, Theory and Methods: 39/100

Engineering, Electrical and Electronic: 143/242

Abstract:

CAIM (Class-Attribute Interdependence Maximization) is one of the state-of-the-art algorithms for discretizing data for which classes are known. However, it may take a long time when run on high-dimensional large-scale data, with large number of attributes and/or instances. This paper presents a solution to this problem by introducing a GPU-based implementation of the CAIM algorithm that significantly speeds up the discretization process on big complex data sets. The GPU-based implementation is scalable to multiple GPU devices and enables the use of concurrent kernels execution capabilities of modern GPUs. The CAIM GPU-based model is evaluated and compared with the original CAIM using single and multi-threaded parallel configurations on 40 data sets with different characteristics. The results show great speedup, up to 139 times faster using 4 GPUs, which makes discretization of big data efficient and manageable. For example, discretization time of one big data set is reduced from 2 hours to less than 2 minutes.

TITLE:

ur-CAIM: Improved CAIM Discretization for Unbalanced and Balanced Data

AUTHORS:

A. Cano, D. T. Nguyen, S. Ventura, and K. J. Cios

IEEE Transactions on Knowledge and Data Engineering, *Submitted, 2013*

RANKING:

Impact factor (JCR 2012): 1.892

Knowledge area:

Computer Science, Artificial Intelligence: 30/114

Computer Science, Information Systems: 22/132

Engineering, Electrical and Electronic: 55/242

Abstract:

Supervised discretization is one of basic data preprocessing techniques used in data mining. CAIM (Class-Attribute Interdependence Maximization) has been state of the art algorithm for almost a decade for discretization of data for which the classes are known. However, new arising challenges such as the presence of unbalanced data sets, call for new algorithms capable of handling them, in addition to balanced data. This paper presents a new discretization algorithm, ur-CAIM, which improves on the CAIM algorithm in three important ways. First, it generates more flexible discretization schemes while keeping low number of intervals. Second, the quality of the intervals is improved based on data classes distribution, which leads to better classification performance on balanced and, importantly, unbalanced data. Third, the runtime of the algorithm is lower than CAIM's. The ur-CAIM was compared with 11 well-known discretization methods on 28 balanced, and 70 unbalanced data sets. The results show that it performs well on both types of data, which is its significant advantage over other supervised discretization algorithms.

TITLE:

Multi-Objective Genetic Programming for Feature Extraction and Data Visualization

AUTHORS:

A. Cano, S. Ventura, and K. J. Cios

IEEE Transactions on Evolutionary Computation, *Submitted, 2013*

RANKING:

Impact factor (JCR 2012): 4.81

Knowledge area:

Computer Science, Artificial Intelligence: 3/114

Computer Science, Theory and Methods: 1/100

Abstract:

Feature extraction transforms high-dimensional data into a new subspace of fewer dimensions. Traditional algorithms do not consider the multi-objective nature of this task. Data transformations should improve the classification performance on the new subspace, as well as data visualization, which has attracted increasing attention in recent years. Moreover, new challenges arising in data mining, such as the need to deal with unbalanced data sets call for new algorithms capable of handling this type of data, in addition to balanced data. This paper presents a Pareto-based multi-objective genetic programming algorithm for feature extraction and data visualization. The algorithm is designed to obtain data transformations which optimize the classification and visualization performance both on balanced and unbalanced data. Six different classification and visualization measures are identified as objectives to optimize by the multi-objective algorithm. The algorithm is evaluated and compared to 10 well-known feature extraction techniques, and to the performance over the original high-dimensional data. Experimental results on 20 balanced and 20 unbalanced data sets show that it performs very well on both types of data that is its significant advantage over existing feature extraction algorithms.

TITLE:

Predicting student failure at school using genetic programming and different data mining approaches with high dimensional and imbalanced data

AUTHORS:

C. Márquez-Vera, A. Cano, C. Romero, and S. Ventura

Applied Intelligence, *Volume 38, Issue 3, pp. 315-330, 2013*

RANKING:

Impact factor (JCR 2012): 1.853

Knowledge area:

Computer Science, Artificial Intelligence: 32/114

DOI: 10.1007/s10489-012-0374-8

Abstract:

Predicting student failure at school has become a difficult challenge due to both the high number of factors that can affect the low performance of students and the imbalanced nature of these types of datasets. In this paper, a genetic programming algorithm and different data mining approaches are proposed for solving these problems using real data about 670 high school students from Zacatecas, Mexico. Firstly, we select the best attributes in order to resolve the problem of high dimensionality. Then, rebalancing of data and cost sensitive classification have been applied in order to resolve the problem of classifying imbalanced data. We also propose to use a genetic programming model versus different white box techniques in order to obtain both more comprehensible and accuracy classification rules. The outcomes of each approach are shown and compared in order to select the best to improve classification accuracy, specifically with regard to which students might fail.

Publications in conferences

- A. Cano, A. Zafra, E.L. Gibaja, and S. Ventura. *A Grammar-Guided Genetic Programming Algorithm for Multi-Label Classification*. In Proceedings of the 16th European Conference on Genetic Programming, EuroGP'13, Lecture Notes in Computer Science, vol 7831, pages 217-228, 2013.

- J.L. Olmo, A. Cano, J.R. Romero, and S. Ventura. *Binary and Multiclass Imbalanced Classification Using Multi-Objective Ant Programming*. In Proceedings of the 12th International Conference on Intelligent Systems Design and Applications, ISDA'12, pages 70-76, 2012.

- A. Cano, A. Zafra, and S. Ventura. *An EP algorithm for learning highly interpretable classifiers*. In Proceedings of the 11th International Conference on Intelligent Systems Design and Applications, ISDA'11, pages 325-330, 2011.

- A. Cano, A. Zafra, and S. Ventura. *A parallel genetic programming algorithm for classification*. In Proceedings of the 6th International Conference on Hybrid Artificial Intelligent Systems (HAIS). Lecture Notes in Computer Science, 6678 LNAI(PART 1):172-181, 2011.

- A. Cano, A. Zafra, and S. Ventura. *Solving classification problems using genetic programming algorithms on GPUs*. In Proceedings of the 5th International Conference on Hybrid Artificial Intelligent Systems (HAIS). Lecture Notes in Computer Science, 6077 LNAI(PART 2):17-26, 2010

- A. Cano, A. Zafra and S. Ventura. *Parallel Data Mining Algorithms on GPUs*. XIX Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB), pages 1603-1606, 2013.

- A. Cano, J.L. Olmo, and S. Ventura. *Programación Automática con Colonias de Hormigas Multi-Objetivo en GPUs*. XIX Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB), pages 288-297, 2013.

- A. Cano, J.M. Luna, A. Zafra, and S. Ventura. *Modelo gravitacional para clasificación*. VIII Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB), pages 63-70, 2012.
- A. Cano, A. Zafra, and S. Ventura. *Speeding up evolutionary learning algorithms using GPUs*. In ESTYLF 2010 XV Congreso Español sobre Tecnologías y Lógica Fuzzy, pages 229-234, 2010.