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 23, 2013 / Revised: Mar 31, 2014 / Accepted: Apr 26, 2014

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. Graphics processing units (GPUs) are being used for reducing computing time of algorithms. This paper presents an implementation of the G3P-MI algorithm on GPUs for solving multiple instance problems using classification rules. The GPU model proposed is distributable to multiple GPUs, seeking for its scalability across large-scale and high-dimensional data sets. The proposal is compared to the multi-threaded CPU algorithm with SSE parallelism over a series of data sets. Experimental results report that the computation time can be significantly reduced and its scalability improved. Specifically, an speedup of up to 149× 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.

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

1 Introduction

Multiple instance learning (MIL) is a generalization of traditional supervised learning having growing interest [6,12,16,39]. 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. [12], assumes a bag to be positive if it contains at least one positive instance. More recently, other generalized multi-instance models have been formalized [16,39]. 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.

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 [1], content-based image retrieval [25] and image annotation [38], drug activity prediction [35,51], web index page recommendation [52], video concept detection [21,23], semantic video retrieval [7] and predicting student performance [47,49].

Similarly, there are many machine learning methods available to solve these problems, such as multi-instance lazy learning algorithms [43], multi-instance tree learners [8], multi-instance rule inducers [9],

Alberto Cano[1] · Amelia Zafra[1] · Sebastián Ventura[1,2]
1. Department of Computer Science and Numerical Analysis, University of Cordoba, Spain
2. Information Systems Department, Faculty of Computing and Information Technology
King Abdulaziz University, P.O. Box : 80200, 21589 Jeddah (Saudi Arabia)
E-mail: {acano, azafra, sventura}@uco.es

multi-instance bayesian approaches [27], multi-instance kernel methods [22,38], multi-instance ensembles [45,51], and evolutionary algorithms [48,50]. 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 highly convenient to adapt the learning process in order to complete it in a reasonable time. In fact, there are several works that try to optimize the computation time of some algorithms in MIL [4,17,34,37,40].

In this context, graphics processing units (GPUs) have demonstrated efficient performance in traditional rule-based classification [5,18]. Moreover, GPU systems have been widely used in many other evolutionary algorithms and data mining techniques in recent years [15,24,31,33]. 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 [48] algorithm. 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, which allow to extend the application of MIL algorithms across large-scale and high-dimensional data sets.

The proposal is evaluated over multiple 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 149× when compared to the multi-threaded CPU implementation with SSE parallelism. 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 359 million GPops/s. Moreover, it has shown great scalability to two and four GPUs.

This paper is organized as follows. Section 2 defines the multi-instance problem and presents rule-based approaches. Section 3 presents a computational analysis of the multi-instance algorithm. Section 4 presents the GPU implementation. Section 5 describes the experimental study, whose results are discussed in Section 6. Finally, Section 7 presents the conclusions.

2 Multi-instance classification

This section defines the multi-instance 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, multi-instance classification works were based on the standard or Dietterich hypothesis [12]. 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, Weidmann et al. [44] 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.

Regardless of the multi-instance hypothesis, MIL algorithms demand significant resources, taking excessive computing time when data size increases. Their high computational cost prevent their application in large-scale and high-dimensional real world problems within reasonable time. However, the multiple instances learning process using data structures representation with bags and instances is inherently parallel. Therefore, we take advantage of the parallel capabilities of GPUs to speed up the learning process.

2.2 Rule-based models

Rule-based models are white box classification techniques which add comprehensibility to the knowledge discovery process in the form of IF-THEN rules. The comprehensibility of the knowledge discovered has been an area of growing interest and it is considered to be as important as obtaining high accuracy [2, 28].

The evaluation of the 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 [19, 20], and specifically, Genetic Programming [13], have been successfully employed for obtaining classification rules. However, for every generation a population of rules must be evaluated according to a fitness function. Thus, the algorithms perform slowly and their scalability is limited by the dimensionality of the data. The use of GPUs for the evaluation of individuals in evolutionary computation has demonstrated high performance in many studies. These studies include using genetic programming for stock trading [36], classification rules [5, 18], differential evolution [11, 42], image clustering [29], or optimization problems [14]. However, to the best of our knowledge there are no GPU-based implementations of multi-instance classification rules algorithms to date.

G3P-MI [48] is a Grammar-Guided Genetic Programming (G3P) [26] algorithm for multi-instance learning. It is based on the presence-based hypothesis and has demonstrated accurate classification and better performance than many other multi-instance methods. 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 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 Computational analysis of the multi-instance algorithm

G3P-MI 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 method selects the parent candidates from the population, on which the genetic operators (crossover and mutation) are applied, generating new rules (offspring). The evaluation checks the fitness of the offspring. The replacement selects the best rules from the parents and the offspring, leading the population to better fitness landscapes. This process is iteratively repeated along a given number of generations. However, it is well-known and it has been demonstrated in several studies [5,18,33] 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 consists on predicting the class membership of the examples of the data set and to compare 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 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 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 computation. 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 of each rule is an independent computation problem (population parallel approach). Moreover, the coverage of a rule against all the examples is also a paralellizable task (data parallel approach).

3.1 Parallelization on multi-core CPUs and SSE

The parallelization of the evaluation 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. Moreover, current processors include instruction set extensions specifically developed for increasing the performance of parallel workloads. SIMD (Single Instruction Multiple Data) instructions are grouped in a instruction set included by most general-purpose processors and known as Streaming SIMD Extensions (SSE). These SIMD instructions provide a limited form of parallelism working at a small scale on multiple data. Nevertheless, the performance of implementations can be significantly improved as shown in other studies [10]. The evaluation function can take advantage of the SSE instructions to compute multiple instances at once, providing a limited form of data parallelism (data-parallel approach). In particular, there are load/store (*_mm_load_ps*, *_mm_store_ps*) and arithmetic/logical (*_mm_add_ps*, *_mm_mul_ps*, *_mm_and_ps*, *_mm_or_ps*) functions handle four single-precision floating-point values at once, increasing the efficiency of the CPU.

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 having multiple nodes connected through a network. 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 and the network, which eventually reduces the efficiency of the process.

4 Multi-instance rules evaluation on GPUs

The evaluation of the rules is divided into three steps, each implemented in separate GPU kernel. The coverage kernel checks the coverage of the rules over the instances within the bags. The hypothesis kernel identifies bag instances that satisfy the concepts of the rule. Finally, the fitness kernel computes the quality of the rules. The data and computation flow is overviewed in Fig. 1, whereas the GPU evaluation model is shown in Fig. 2. Data memory transactions between CPU and GPU memories are shown dashed and light gray, whereas GPU kernels computation are shown dotted and black gray.

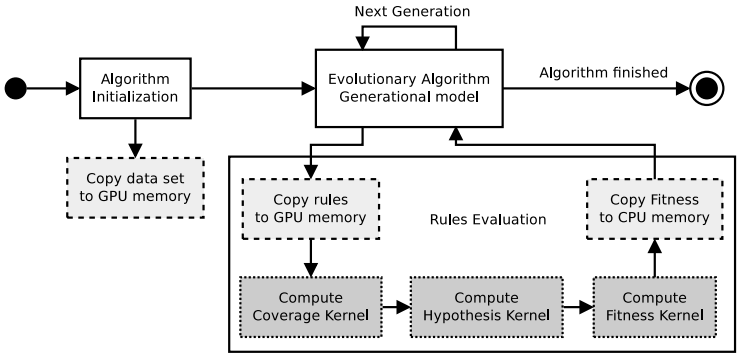

Fig. 1 Data and computation flow overview

The data set is allocated transposed in the GPU global memory to facilitate memory coalescing. 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 in 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 Profiler tool.

4.1 Coverage kernel

The coverage kernel interprets the rules and checks whether the instances of the data set satisfy the conditions of the rules. Example of a rule:

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

where At_1, At_2, and At_3 are attributes of the data set, and V_1, V_2, and V_3 are numeric values. Rules are easily represented in prefix notation and evaluated using stack-based operations due to its ability to distinguish the order of operations without parentheses. The example rule in prefix notation is:

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

The implementation of a prefix interpreter require a stack and every operand/operator perform push/pop operations. Thus, the number of operations 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 stack operations. The conditions are internally represented in prefix notation whereas the rule set is written in postfix. The rule can be rewritten as:

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

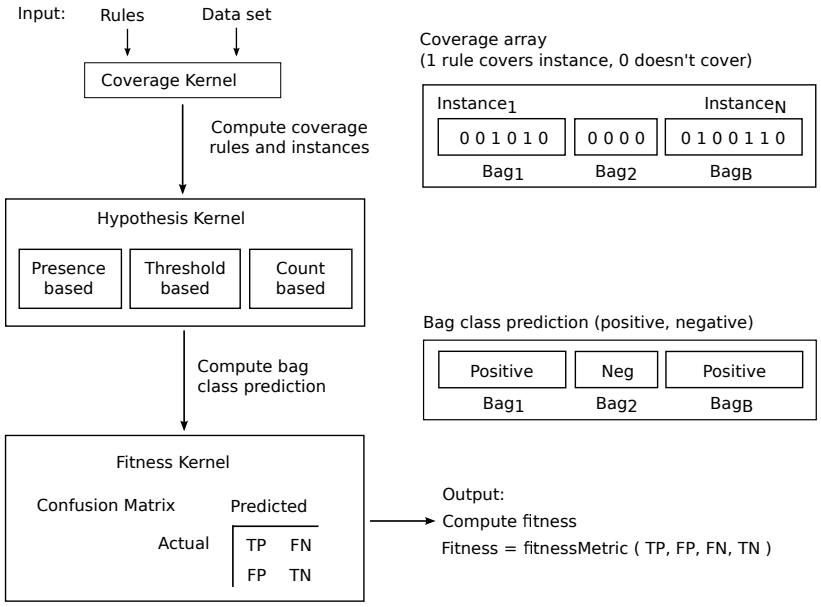

Fig. 2 GPU evaluation model using three kernels

The evaluation of every rule over single instance can be parallelized on the GPU using a two dimensional matrix of threads. The kernel checks the coverage of the rules over the instances of the data set, and stores the results of the coverage matching into an array. 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 since threads are responsible of handling adjacent memory addresses. The array of the coverage result is also coalesced by addressing adjacent memory addresses. Coalescing avoids memory addressing conflicts and permits to improve efficiency of memory transactions.

4.2 Hypothesis kernel

The hypothesis kernel performs the class prediction for the bags, using the coverage results from the previous kernel. Three functions implement the different hypothesis concerning their requirements based on the presence-based, threshold-based, or count-based multiple instance 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 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.

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 latest, and it is not known a priori if a positive instance is going to be found earlier or later. 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 any 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.

4.3 Fitness kernel

The fitness computation kernel evaluates the quality of the rule, i.e., its ability to perform accurate classifications. It computes the confusion matrix values 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 [48] algorithm is to maximize both sensitivity and specificity, computing fitness as their product.

The confusion matrix values result from a reduction operation [46] of the bag class predictions and the actual values. The naive reduction operation is conceived as an iterative and sequential process. However, this operation can be parallelized using a 2-level parallel reduction with shared memory storage, and it is illustrated in Fig. 3.

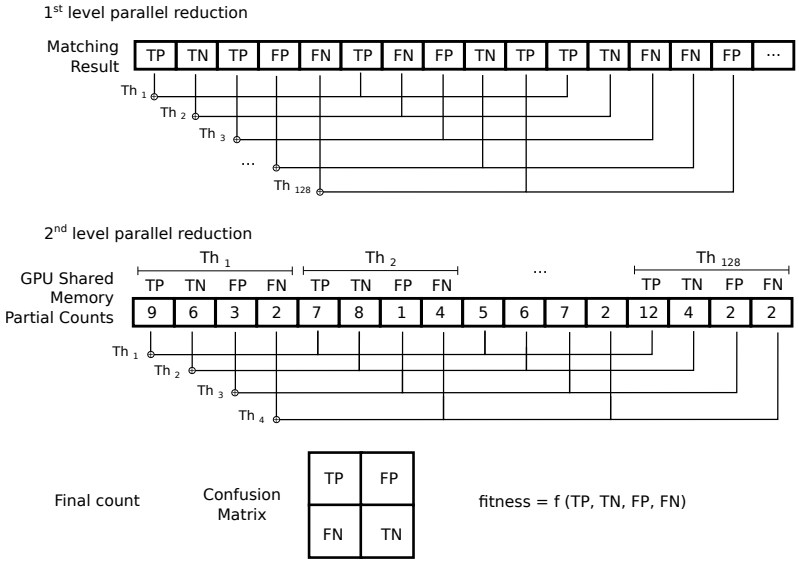

Fig. 3 Fitness computation using 2-level parallel reduction

The first level of the reduction reads the bag class prediction and compares with the actual bag class. The outcome is one of the four possible values of the confusion matrix, which are stored in shared memory. The second level counts the partial results from shared memory and computes the fitness of the rules. To this end, 128 threads are employed. The reason for using 128 threads is that all threads in a thread block must share shared memory, 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 capacities of the GPU, adapting automatically the configuration parameters in order to be capable of achieving maximum performance. The key point of the reduction is that accesses are fully coalesced to avoid memory addressing divergences, and shared memory, which provides fast and low latency access, is efficiently employed. This process is run in parallel for all the rules of the population.

5 Experimental Setup

This section presents the experiments, hardware and setup to evaluate the performance of the GPU model.

5.1 Hardware configuration

The experiments were run on a machine equipped with an Intel Core i7 quad-core processor (i7-3820) running at 3.6 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).

5.2 Configuration settings

The G3P-MI algorithm is implemented in the JCLEC software [41] and its main parameters (collected from the author's proposal) are shown in Table 1.

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

On the other hand, the coverage and hypothesis kernels require to setup the number of threads per block. 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 multiprocessors.

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%

5.3 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 experimental study is provided as additional material at link[1].

5.3.1 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) [3,30–32]. GP interpreters evaluate expression trees, which represent solutions to perform a user-defined task. In this experiment, 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.

5.3.2 Rule-based classification performance

The second experiment evaluates the performance of the proposal across a series of 13 real-world multi-instance data sets. However, these 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. Experiments were run 100 times and the average runtimes are reported.

6 Results

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

6.1 Rules interpreter performance

Table 3 shows the rules interpreter performance regarding the number of rules and the number of instances, 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 with SSE parallelism 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 with SSE parallelism achieves up to 359 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

[1]The GPU kernels code and the 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`

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
100	1,000	4.90×10^6	31	5.10	2.89	1.91	158	961	1,692	2,561
	5,000	2.45×10^7	110	5.54	3.75	2.08	221	4,415	6,534	11,760
	50,000	2.45×10^8	989	13.65	7.23	4.09	247	17,931	33,858	59,834
	100,000	4.90×10^8	1,922	20.84	11.66	5.02	254	23,489	41,987	97,555
250	1,000	1.22×10^7	46	7.84	4.32	2.49	268	1,557	2,826	4,900
	5,000	6.10×10^7	219	7.89	5.32	2.77	278	7,734	11,474	22,027
	50,000	6.10×10^8	2,183	25.03	13.86	9.59	279	24,390	44,061	63,672
	100,000	1.22×10^9	4,518	48.86	24.66	14.55	270	24,991	49,505	83,910
500	1,000	2.41×10^7	87	11.51	6.26	3.85	277	2,091	3,849	6,248
	5,000	1.20×10^8	394	14.64	7.91	4.83	305	8,222	15,214	24,924
	50,000	1.20×10^9	3,685	47.22	25.29	13.09	326	25,491	47,596	91,962
	100,000	2.41×10^9	7,683	90.11	46.99	23.73	313	26,717	51,239	101,465
1,000	1,000	4.85×10^7	160	15.59	8.17	4.10	302	3,114	5,941	11,836
	5,000	2.43×10^8	699	16.13	11.60	7.83	347	15,047	20,923	30,999
	50,000	2.43×10^9	6,919	93.79	47.63	25.31	350	25,881	50,960	95,917
	100,000	4.85×10^9	13,504	179.43	89.38	44.58	359	27,057	54,315	108,908

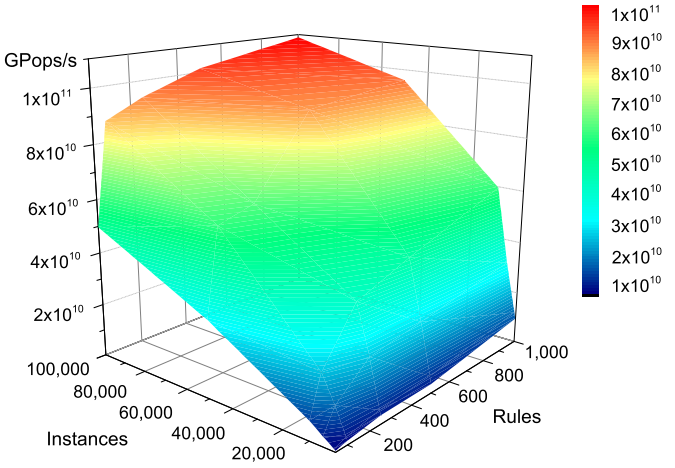

Fig. 4 GPU interpreter performance regarding the number of instances and rules

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 with SSE parallelism. The increasing performance of the interpreter is shown in Fig. 4, 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.

6.2 Rule-based classification performance

Table 4 shows the performance of the G3P-MI algorithm using the Dietterich hypothesis. The table shows the runtime for evaluating the population of rules over multiple 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, results indicate that the higher the dimensionality of the data, the better the speedup achieved. The best performance scenario in which the highest speedup is achieved (131×) corresponds with the *process* data set when using four GPUs, which allows the evaluation time to be reduced from 18 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	360.84	11.15	5.85	3.92	32.37	61.67	92.09
EastWest	25	20	213	2.51	2.25	2.06	1.00	1.11	1.22	2.52
Elephant	231	200	1,391	16.41	1.76	1.36	1.15	9.31	12.10	14.32
Fox	231	200	1,320	16.05	1.90	1.69	1.17	8.46	9.48	13.73
Function	201	5,242	55,536	544.53	15.33	8.62	4.58	35.53	63.15	118.81
Musk1	167	92	476	5.41	1.77	1.01	1.13	3.05	5.36	4.80
Musk2	167	102	6,598	38.25	2.26	1.71	1.21	16.92	22.41	31.68
Mut-atoms	11	188	1,618	15.37	2.17	1.44	1.61	7.09	10.66	9.53
Mut-bonds	17	188	3,995	28.38	2.06	1.91	1.75	13.79	14.83	16.23
Mut-chains	25	188	5,349	36.08	2.55	1.35	1.07	14.15	26.66	33.68
Process	201	11,718	118,417	1,106.58	29.99	16.25	8.43	36.90	68.11	131.24
Suramin	21	11	2,378	17.99	3.53	2.09	1.04	5.10	8.60	17.29
Tiger	231	200	1,220	14.15	2.48	1.67	1.08	5.71	8.45	13.06
Artificial1	100	10	1,000	3.00	2.90	2.15	1.42	1.03	1.40	2.11
Artificial2	100	100	1,000	9.80	3.16	1.62	1.85	3.11	6.04	5.29
Artificial3	100	100	1×10^4	18.48	4.49	2.88	1.84	4.11	6.43	10.07
Artificial4	100	1,000	1×10^4	57.58	4.55	2.52	1.51	12.65	22.82	38.12
Artificial5	100	1,000	1×10^5	152.71	20.42	10.50	6.10	7.48	14.54	25.04
Artificial6	100	1×10^4	1×10^5	475.06	25.07	13.45	6.55	18.95	35.31	72.52
Artificial7	100	1×10^5	1×10^6	4,654.65	213.21	109.38	57.38	21.83	42.55	81.12

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, it is only recommended to use GPUs when the problem size is higher than a certain threshold.

Table 5 shows the performance of the G3P-MI algorithm using the generalized hypothesis. 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 4.2, the presence-based hypothesis allows instances matching to be stopped. 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, having the GPU higher speedups.

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	644.86	15.94	8.47	4.32	40.47	76.11	149.42
EastWest	25	20	213	4.74	1.84	1.18	0.94	2.57	4.02	5.06
Elephant	231	200	1,391	27.51	1.73	1.21	0.77	15.93	22.83	35.83
Fox	231	200	1,320	25.80	1.72	1.23	0.73	14.99	20.98	35.32
Function	201	5,242	55,536	976.79	25.36	14.37	7.84	38.52	67.95	124.59
Musk1	167	92	476	9.70	1.55	1.04	0.72	6.25	9.30	13.40
Musk2	167	102	6,598	121.49	2.55	1.47	1.05	47.72	82.39	115.21
Mut-atoms	11	188	1,618	27.49	1.76	1.18	0.79	15.63	23.24	34.72
Mut-bonds	17	188	3,995	60.61	2.17	1.24	0.79	27.95	48.99	77.04
Mut-chains	25	188	5,349	80.61	2.20	1.38	0.94	36.71	58.23	85.55
Process	201	11,718	118,417	1,964.10	85.03	45.64	24.20	23.10	43.04	81.17
Suramin	21	11	2,378	44.74	2.76	2.07	1.14	16.19	21.58	39.36
Tiger	231	200	1,220	23.24	1.71	1.12	0.87	13.56	20.72	26.60
Artificial1	100	10	1,000	23.43	2.79	1.65	1.48	8.40	14.23	15.78
Artificial2	100	100	1,000	23.88	2.59	1.61	1.44	9.21	14.83	16.58
Artificial3	100	100	1×10^4	165.56	4.52	2.55	1.41	36.61	64.80	117.19
Artificial4	100	1,000	1×10^4	170.54	4.95	2.67	1.31	34.42	63.94	130.09
Artificial5	100	1,000	1×10^5	1,513.44	44.93	24.39	12.63	33.69	62.04	119.87
Artificial6	100	1×10^4	1×10^5	1,422.38	45.97	24.97	12.62	30.94	56.97	112.73
Artificial7	100	1×10^5	1×10^6	14,107.76	454.68	243.40	124.15	31.03	57.96	113.63

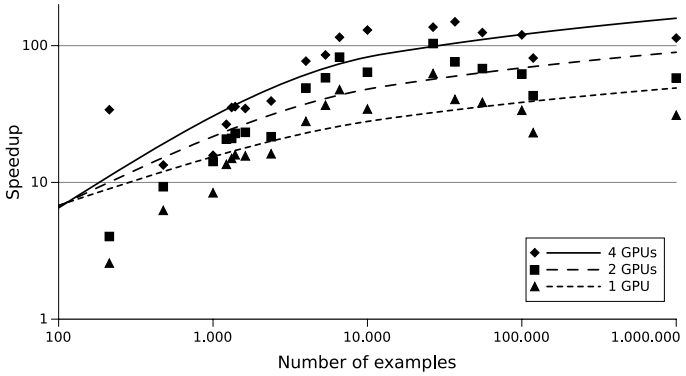

Fig. 5 Analysis of performance scalability using 1, 2, and 4 GPUs

Finally, the scalability of the model performance regarding to the increasing size of the data set is shown in Fig. 5 (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.

7 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 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 SSE implementation over multiple 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 149×. 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, project TIN-2011-22408, and FEDER funds. This research was also supported by the Spanish Ministry of Education under FPU grant AP2010-0042.

References

1. S. Andrews, I. Tsochantaridis, and T. Hofmann. Support vector machines for multiple-instance learning. In *Neural Information Processing System*, pages 561–568, 2002.
2. 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.
3. W. Banzhaf, S. Harding, W. B. Langdon, and G. Wilson. Accelerating Genetic Programming through Graphics Processing Units. In *Genetic Programming Theory and Practice VI*, pages 1–19. 2009.
4. 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.
5. A. Cano, A. Zafra, and S. Ventura. Speeding up the evaluation phase of GP classification algorithms on GPUs. *Soft Computing*, 16:187–202, 2012.
6. S. Chen and L. Jiang. An empirical study on multi-instance learning. *Advances in Information Sciences and Service Sciences*, 4(6):193–202, 2012.
7. 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.
8. Y. Chevaleyre, N. Bredeche, and J. Zucker. Learning rules from multiple instance data: Issues and algorithms. In *9th Inf. Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 455–459, 2002.
9. Y. Chevaleyre and J. Zucker. Solving multiple-instance and multiple-part learning problems with decision trees and decision rules. Application to the mutagenesis problem. volume 2056 of *LNCS*, pages 204–214, 2001.
10. D. Chitty. Fast parallel genetic programming: Multi-core cpu versus many-core gpu. *Soft Computing*, 16(10):1795–1814, 2012.
11. 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.
12. 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.
13. 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.
14. F. Fabris and R. A. Krohling. A co-evolutionary differential evolution algorithm for solving min-max optimization problems implemented on GPU using C-CUDA. *Exp. Sys. with Apps.*, 39(12):10324–10333, 2012.
15. K. L. Fok, T. T. Wong, and M. L. Wong. Evolutionary computing on consumer graphics hardware. *IEEE Intelligent Systems*, 22(2):69–78, 2007.
16. J. Foulds and E. Frank. A review of multi-instance learning assumptions. *Knowledge Engineering Review*, 25(1):1–25, 2010.
17. J. R. Foulds and E. Frank. Speeding up and boosting diverse density learning. In *13th international conference on Discovery science*, pages 102–116, 2010.
18. M. A. Franco, N. Krasnogor, and J. Bacardit. Speeding up the evaluation of evolutionary learning systems using GPGPUs. In *Genetic and Evolutionary Computation Conference*, pages 1039–1046, 2010.
19. A. A. Freitas. *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer-Verlag, 2003.

20. A. A. Freitas. *A Review of Evolutionary Algorithms for Data Mining*, pages 61–93. 2007.
21. S. Gao and Q. Suna. Exploiting generalized discriminative multiple instance learning for multimedia semantic concept detection. *Pattern Recognition*, 41(10):3214–3223, 2008.
22. T. Gartner, P. A. Flach, A. Kowalczyk, and A. J. Smola. Multi-instance kernels. In *19th International Conference on Machine Learning*, pages 179–186, 2002.
23. Z. Gu, T. Mei, J. Tang, X. Wu, and X. Hua. MILC2: A multi-layer multi-instance learning approach to video concept detection. In *14th International Conference of Multimedia Modeling*, pages 24–34, 2008.
24. S. Harding and W. Banzhaf. Fast genetic programming on GPUs. *Lecture Notes in Computer Science*, 4445:90–101, 2007.
25. G. Herman, G. Ye, J. Xu, and B. Zhang. Region-based image categorization with reduced feature set. In *10th IEEE Workshop on Multimedia Signal Processing*, pages 586–591, 2008.
26. 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.
27. 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.
28. 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.
29. D. Konieczny, M. Marcinkowski, and P. . Myszkowski. GPGPU implementation of evolutionary algorithm for images clustering. *Studies in Computational Intelligence*, 457:219–238, 2013.
30. W. B. Langdon. A many threaded cuda interpreter for genetic programming. *Lecture Notes in Computer Science*, 6021:146–158, 2010.
31. W. B. Langdon. Graphics processing units and genetic programming: An overview. *Soft Computing*, 15(8):1657–1669, 2011.
32. 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.
33. 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.
34. 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.
35. O. Maron and T. Lozano-Pérez. A framework for multiple-instance learning. In *Neural Information Processing System*, pages 570–576, 1997.
36. D. McKenney and T. White. Stock trading strategy creation using GP on GPU. *Soft Computing*, 16(2):247–259, 2012.
37. 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.
38. X. Qi and Y. Han. Incorporating multiple svms for automatic image annotation. *Pattern Recognition*, 40(2):728–741, 2007.
39. S. Sabato and N. Tishby. Multi-instance learning with any hypothesis class. *Journal of Machine Learning Research*, 13:2999–3039, 2012.
40. J. Santner, C. Leistner, A. Saffari, T. Pock, and H. Bischof. PROST: Parallel robust online simple tracking. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 723–730, 2010.
41. 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.
42. 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.
43. J. Wang and J.-D. Zucker. Solving the multiple-instance problem: A lazy learning approach. In *17th International Conference on Machine Learning*, pages 1119–1126, 2000.
44. N. Weidmann, E. Frank, and B. Pfahringer. A Two-level Learning Method for Generalized Multi-instance Problems. In *14th European Conference on Machine Learning*, pages 468–479, 2003.
45. I. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. 2nd Edition. Morgan Kaufmann, 2005.
46. X. L. Wu, N. Obeid, and W. M. Hwu. Exploiting more parallelism from applications having generalized reductions on GPU architectures. In *IEEE Computer and Information Technology*, pages 1175–1180, 2010.
47. A. Zafra, C. Romero, and S. Ventura. Multiple instance learning for classifying students in learning management systems. *Expert Systems with Applications*, 38(12):15020–15031, 2011.
48. A. Zafra and S. Ventura. G3P-MI: A genetic programming algorithm for multiple instance learning. *Information Sciences*, 180:4496–4513, 2010.
49. A. Zafra and S. Ventura. Multi-instance genetic programming for predicting student performance in web based educational environments. *Applied Soft Computing*, 12(8):2693–2706, 2012.
50. 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.
51. 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.
52. Z.-H. Zhou, K. Jiang, and M. Li. Multi-instance learning based web mining. *Applied Intelligence*, 22(2):135–147, 2005.

Author Biographies

Alberto Cano was born in Cordoba, Spain, in 1987. He received the Ph.D. degree in Computer Science from the University of Granada in 2014, and the M.Sc. degree in Computer Science from the University of Cordoba in 2010. 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, data mining and its applications.

Amelia Zafra is Associate Professor of Computer Sciences and Artificial Intelligence at the University of Cordoba. 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 is currently an Associate Professor in the Department of Computer Science and Numerical Analysis at the University of Cordoba, where he heads the Knowledge Discovery and Intelligent Systems Research Laboratory. He received his BSc and Ph.D. degrees in sciences from the University of Cordoba, Spain, in 1989 and 1996, respectively. He has published more than 150 papers in journals and scientific conferences, and he has edited three books and several special issues in international journals. He has also been engaged in 11 research projects (being the coordinator of three of them) supported by the Spanish and Andalusian governments and the European Union. His main research interests are in the fields of soft-computing, machine learning, data mining, and their applications. Dr. Ventura is a senior member of the IEEE Computer, the IEEE Computational Intelligence and the IEEE Systems, Man and Cybernetics Societies, as well as the Association of Computing Machinery (ACM).