Knowledge-Based Systems 231 (2021) 107419

Contents lists available at ScienceDirect

Knowledge-Based Systems

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

A novel approach based on genetic algorithm to speed up the discovery of classification rules on GPUs

Mohammad Beheshti Roui [a], Mariam Zomorodi [a,b,*], Masoomeh Sarvelayati [a], Moloud Abdar [c], Hamid Noori [a], Paweł Pławiak [b,d], Ryszard Tadeusiewicz [e], Xujuan Zhou [f], Abbas Khosravi [c], Saeid Nahavandi [c,g], U. Rajendra Acharya [h,i,j]

[a] Department of Computer Engineering, Ferdowsi University of Mashhad, Mashhad, Khorasan Razavi, Iran
[b] Department of Computer Science, Faculty of Computer Science and Telecommunications, Cracow University of Technology, Krakow, Poland
[c] Institute for Intelligent Systems Research and Innovation (IISRI), Deakin University, Waurn Ponds, VIC 3216, Australia
[d] Institute of Theoretical and Applied Informatics, Polish Academy of Sciences, Gliwice, Poland
[e] Faculty of Electrical Engineering, Automatics, Computer Science and Biomedical Engineering, AGH University of Science and Technology, Krakow, Poland
[f] School of Management and Enterprise, University of Southern Queensland, Australia
[g] Harvard Paulson School of Engineering and Applied Sciences, Harvard University, Allston, MA 02134, USA
[h] Department of ECE, Ngee Ann Polytechnic, 535 Clementi Road, Singapore 599 489, Singapore
[i] Department of Biomedical Engineering, School of Science and Technology, SUSS University, Singapore
[j] Department of Biomedical Informatics and Medical Engineering, Asia University, Taichung, Taiwan

ARTICLE INFO

Article history:
Received 1 April 2020
Received in revised form 17 August 2021
Accepted 18 August 2021
Available online 21 August 2021

Keywords:
Data mining
Machine learning
Rule discovery
Genetic algorithm
GPU programming
Classification rules

ABSTRACT

This paper proposes a new approach to produce classification rules based on evolutionary computation with novel crossover and mutation operators customized for execution on graphics processing unit (GPU). Also, a novel method is presented to define the fitness function, i.e. the function which measures quantitatively the accuracy of the rule. The proposed fitness function is benefited from parallelism due to the parallel execution of data instances. To this end, two novel concepts; coverage matrix and reduction vectors are used and an altered form of the reduction vector is compared with previous works. Our CUDA program performs operations on coverage matrix and reduction vector in parallel. Also these data structures are used for evaluation of fitness function and calculation of genetic operators in parallel. We proposed a vector called average coverage to handle crossover and mutation properly. Our proposed method obtained a maximum accuracy of 99.74% for Hepatitis C Virus (HCV) dataset, 95.73% for Poker dataset, and 100% for COVID-19 dataset. Our speedup is higher than 20% for HCV and COVID-19, and 50% for Poker, compared to using single core processors.

1. Introduction

Nowadays, machine learning methods including evolutionary algorithms are extensively used in data mining tasks [1]. Data classification is one of the data processing operations which helps to understand the data better and predict the unseen data. The data discrimination can be performed accurately using advanced machine learning and evolutionary algorithms (EAs) [2,3]. Genetic algorithm (GA) is one of the widely used evolutionary algorithms which helps to provide near optimal solutions for problems.

Decision tree based algorithms have been used in rule mining for many years. One of the limitations of the decision tree is that, few leaves in a tree have similar class probabilities. Other classification algorithms such as neural networks do not give the rules in a symbolic and understandable forms [4].

The implementation of GA algorithms for various data mining tasks is simple and it is also compatible with different sizes of data and complex problems. It is a very successful algorithm used to solve nondeterministic polynomial time (NP)-complete problems and it can also be parallelized in many parts. These advantages of GA has made it a popular method in data mining and machine learning [5,6]. On the other hand, the process of rule generation using GA is time consuming in serial processors, which is affected by population size, dataset size, and number of attributes in the dataset. The application of GAs in rule generation

* Corresponding author at: Department of Computer Science, Faculty of Computer Science and Telecommunications, Cracow University of Technology, Krakow, Poland.
E-mail addresses: mbr@mail.um.ac.ir (M. Beheshti Roui), zomorodi@pk.edu.pl (M. Zomorodi), m.sarvelayati@mail.um.ac.ir (M. Sarvelayati), m.abdar1987@gmail.com (M. Abdar), hnoori@um.ac.ir (H. Noori), plawiak@pk.edu.pl (P. Pławiak), rtad@agh.edu.pl (R. Tadeusiewicz), Xujuan.Zhou@usq.edu.au (X. Zhou), abbas.khosravi@deakin.edu.au (A. Khosravi), saeid.nahavandi@deakin.edu.au (S. Nahavandi), aru@np.edu.sg (U.R. Acharya).

https://doi.org/10.1016/j.knosys.2021.107419
0950-7051/

is to implement classification rules by searching the space of possible rules and finding the ones that have an acceptable accuracy on a given dataset.

Classification includes two major phases, training and testing. In the training phase and according to the training dataset, a series of rules (a population in GA) are generated. The testing phase is employed to evaluate each rule (an individual in GA), which has been generated in the training phase. This work used two medical datasets to predict Hepatitis C Virus (HCV) and COVID-19 disease. Evaluating the accuracy of individuals in GA demands access to all data instances to generate the rules [7]. Therefore, serial processing of GA is highly time consuming depending on the number of mentioned factors.

Avoiding serial processing and taking parallel execution is a well-known approach to speed up the execution of a program. The parallel processing demands a capable processor such as multi-core or multi-processor. General-purpose computing on graphics processing units (GPGPU)s as single instruction, multiple data (SIMD) multi-processors, are vastly used for problems that require data parallelism to speed up the operation. In addition, GPUs are able to execute GA based on three mentioned factors at the same time [8].

In this paper, GA is used to discover the classification rules and a novel GA approach is implemented with compute unified architecture (CUDA) as a programming environment on NVIDIA's GPUs.

The main contributions of this work are as follows:

- Designed the serial code to discover the classification rules using GA.
- Implemented a new approach for parallel code of GA for execution on GPU and then we used coverage matrix for rule evaluation which is applied to calculate the fitness value of the rules in parallel.
- Proposed a reduction scheme based on coverage matrix to produce a new data structure called reduction vector. The aim of this operation is to produce a new evaluation metric for rules. Fitness function benefits from reduction vector to measure the confusion matrix parameters in parallel for all rules.
- Defined a new data structure called average coverage for each rule to be used in genetic operators, crossover, mutation, and improve their implementation on GPU.
- Presented a new crossover scheme based on average coverage of the rules.
- Suggested a novel mutation scheme depending on average coverage of the rules.
- Evaluated the speedup on each part of GA and the whole algorithm is performed separately.

The rest of this paper is structured as follows. In Section 2 background of related topics are discussed to facilitate the comprehension of this article. Related works are provided in Section 3. In Section 4, our proposed approach for rule discovery on GPU is presented. The experimental results are described in Section 5. Paper concludes in Section 6 and future works are outlined.

2. Background and problem statement

In this section a summary of background on the use of GA for the classification rule discovery and execution of parallel GA on GPU is presented.

2.0.1. GA for classification

During the first step, a random population of rules is generated. In rule generation, a set of if-then rules are encoded into a chromosome. In the second step, the quality of each individual (rule) is evaluated using the fitness function. Then, GA operators select two parent individuals and combine them into an offspring individual, based on the fitness value. Although selection as a term can be used for both feature and rule selection [9,10], this work is using it as rule selection. The individuals improve to higher fitness value while GA iterates through the following steps [7].

a. Encoding: There are two approaches (Michigan and Pittsburgh) for encoding rules into a GA chromosome. In Michigan approach which we used in this work, each chromosome represents a single rule and in Pittsburgh approach each chromosome represents a set of rules. A rule consists of a conjunction (a logical AND) on the values of n conditions, where n is the number of attributes of each instance (record) in dataset [11]. Fig. 1 illustrates the genotype of an individual in Michigan approach. $Attr_i$, Op_i and Val_i denotes the ith predictor attribute, a comparison operator and the jth value of the domain of Attr, respectively. $Active_i$ is a bit used as a flag to indicate whether the ith condition is active ("1") or inactive ("0") [7]. Michigan encoding is vastly employed in applications with the purpose of rule discovery. Michigan-based discovery of rules can be performed using different methods such as particle swarm optimization (PSO) [12], association rules [13,14] and classification [15].
b. Data classification: in classification process the dataset is divided into training and testing datasets. The data mining discovers rules by accessing the values of the attributes and also the class of each instance in training dataset. Accuracy of extracted rules depends on the number of correct predictions in each class of data within testing dataset [16].
c. Fitness function: It measures the quality of individuals according to confusion matrix. The parameters of this matrix are derived by comparing individuals with every instance of training dataset. Assuming that r is number of individuals (population size), $|inst|$ the number of instances used for training, $|Atrr|$ is number of attributes and i is number of iterations; the time complexity of GA is $O(r \times |inst| \times |Atrr| \times i)$. Hence, with large number of individuals in each generation of GA, data instances, attributes, and iterations of GA, serial processing is highly time consuming [17,18].
d. Selection: Selection can be performed using different methods such as data proportionate and ranking. proportionate selection is usually implemented by a biased roulette-wheel and ranking selection usually starts with sorting the rules in a fitness-based fashion, which can be performed either in increasing or decreasing order. The better the ranking of a rule, the higher its probability of being selected and this work employs a ranking-based selection.
e. Crossover: Crossover or recombination is a GA operator and is employed to combine the genes of two individual parents into an offspring individual. According to the fitness value, the parents are selected and recombined with one of three possible methods that can be a single-point, multi-point or uniform crossover [7].
f. Mutation: Mutation is an operator that performs its operation on a single individual at a time. Unlike crossover, which recombines genetic materials between two or more parents, mutation replaces the value of a gene with a randomly-generated value [7].

2.0.2. CUDA environment

GPUs are known as single instruction multiple data (SIMD) processors. Therefore, data parallelism of programs is crucial for utilization and more efficient processing on GPUs. On the other hand, NVIDIA proposed CUDA architecture to programmers to

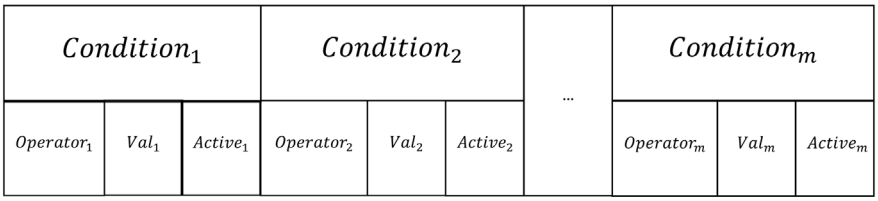

Fig. 1. An individual genotype of Michigan encoding.

exploit the capabilities of this vendor's GPUs for parallel processing. The CUDA programming model provides the execution of thousands of concurrent threads which execute a kernel [19,20].

The number of threads required to execute a program depends on its input size. The kernel execution configuration defines a space of parallel threads which is called grid. Grid includes some blocks that are batches of parallel threads. Each block executes on a streaming multiprocessor (SM) of GPU [19,21].

Although the execution configuration of a kernel affects the performance, but for the sake of simplicity, in this work we used blocks with a constant number of 32 $\times$ 32 threads. Typically, datasets used for classification included less than 32 number of attributes. Therefore, using blocks with 32 $\times$ 32 number of threads is a reasonable configuration for the execution of kernels.

Although memories can cause lack of performance due to loading or storing delays, different types of memories cause various amount of delays [22]. For the sake of simplicity, this work is based on default memory access, which is dynamic random access memory (DRAM) access.

Data parallelism and configuration of kernels are also important. The method of transferring data between central processing unit (CPU) and GPU is another factor used to increase the performance [23]. This work employed multiple techniques to improve the performance of GA execution on GPUs by using special operations on defined data structures.

2.1. Problem statement

The problem that we address in this paper is the implementation of rule discovery using GA on GPU. It is difficult to find an optimal solution in a reasonable time using a big dataset. During the execution of GA, numerous comparisons are required between training instances and conditions of each rule. This requires many memory operations and increases the run time of GA.

In other words, by implementing GA on GPU best rules can be produced for big datasets with many classes and features. The main contribution is the use of special data structures for implementing fitness function and genetic operators. We introduced a new data structure called the average coverage which is used in crossover and mutation operators to rank the rules. The proposed average coverage vector leads to better crossover and mutation. Hence, it increased the accuracy of rules generated in progressive generation of GA.

3. Related work

This section presents the works carried out in the field of classification rule discovery with evolutionary algorithms, machine learning algorithms and parallel execution of GA.

3.1. Classification rule discovery

Evolutionary algorithms have been widely used in rule mining discovery of classification rules. One of the earliest works of using GA for rule mining is proposed in [24]. In this work comprehensible IF-THEN rules were discovered using GA and applied to medical data sets. In their system each individual corresponds to a single rule. Data mining by evolutionary learning (DMEL) used an initial set of first-order rules and then rules of higher order are produced iteratively using GA [4]. In [25] non-random population is initialized, containing all possible encoded operators. Comprehensible and interesting rules in [26] and GA-based approach in [27] are used to calculate the fitness value of individuals. In general, fewer the number of conditions in a rule, more comprehensible it is. An adaptable representation of individuals or chromosomes combined with suitable genetic operators and better fitness function has been used in [28]. Also, GA has been used to improve the classification rules in [29] and the improved rules have been used to diagnose the liver disease.

Developing classification rules for disease prediction using evolutionary algorithms have been investigated in [30] and [29]. In [30], random rules are produced and then using particle swarm optimization (PSO) algorithm, the rules are improved through the steps of the algorithm and then the particles of the last generation are selected as the best rules. In [29], initial rules produced by Boosted C5.0 classification algorithm have been optimized by GA. Both works have been implemented on single-processor platform. Authors have improved this work in [12] by applying modified encoding of rules and fitness function. In addition, different sizes of particles have been taken into account in this work.

Goyal and Soraj [31], have introduced few issues and challenges in the field of classification rule discovery using GA. Initializing the size of first population, size of each chromosome, low quality dataset and different condition types are some of the challenges and issues that affected the process of GA and accuracy of prediction.

In [32], Saif et al. have presented a parallel GA execution method based on empire establishment algorithm. In this method, the population is divided into subpopulations and GA is executed for each subpopulation separately. After dividing the general population and executing GA, a summation of fitness within each subpopulation is calculated and the subpopulation with greater fitness value is replaced with the subpopulation with smaller fitness value.

Al-Maqaleh and Shahbazkia in [33], changed GA in order to perform rule discovery in data mining. The most important modification in their work is on fitness function. The fitness function calculated the fitness using three parameters namely confidence, coverage and complex.

Shobha and Anandhi [34], have proposed an adaptive GA for rule discovery with high accuracy of prediction. Chromosomes of the adaptive GA contained only one type of condition. By using particular dataset, population and just one type of operator for conditions, their accuracy of prediction improved for the presented adaptive GA.

The machine learning algorithms have also been applied widely for the classification of medical datasets [1,35–40]. In [39], an approach based on neural network has been proposed for the medical classification problem. The authors have used integrated Newton–Raphson's Maximum Likelihood and Minimum Redundancy (MLMR) preprocessing model to reduce the classification

time. Two classifiers have been proposed with evolutionary algorithms in [41]. But, an ensemble of classifiers constructed yielded the diagnosis accuracy of 95% for Parkinson's disease.

In addition, [42] and [43] can make more examples of different optimization methods in order to solve rule classification of association problems. In [42], the Reinforcement Learning GA (RL-GA) employed to solve a Capacitated Vehicle Routing Problem (CVRP). The RL-GA designed in a way to adaptively set parameters for a GA. [43] also uses a Whale Optimization (WO) approach to achieve a fine trade-off between intensification and diversification in rule association problems.

3.2. Parallel GA

A parallel genetic programming algorithm which learns rule-based classification is provided in [44]. Each individual consists of a rule based decision list which represents the whole classifier. The rule evaluation phase is parallel using GPU. A review of GA and parallel genetic algorithms (PGA) is provided in [45]. The authors introduced different types of PGA implementation and the parts of GA which can be implemented in parallel. [46], presented an analysis using GA on multi-core CPUs and GPUs. In this work, fitness is implemented on GPU which included two kernels. A work on discovering interesting rules using PGA is presented in [47]. It focused on implementing PGA using multi-core processors and each thread calculated the fitness value of a unique individual. In [48], CUDA is used to implement GA on GPU. In another work by [49], GPU is used to evaluate the association rules and data structures like coverage kernel and reduction kernel applied to several operations on the rules. Coverage kernel checks the coverage of each rule over the dataset instances and reduction kernel computes the reduction operation on the outcome of the coverage kernel. In [50], the population is partitioned into several nodes and they were evolved in parallel using efficient distributed genetic algorithm for classification rule extraction (EDGAR) method. In [51], the population is divided into subpopulations and the fitness is calculated inside each subpopulation using multi-core processors. In this work genetic operators are applied on subpopulations based on the average of their fitness values.

In [52], authors have reviewed different rule discovery techniques based on evolutionary algorithms and presented different models, rule representations, and fitness functions used in the literature.

In [53], Cano et al. have exploited GPUs to solve classification problems using GA. Their work evaluated each part of GA in terms of execution time and then time-consuming parts of GA are executed in parallel using CUDA. The results indicated that the performance of parallel execution improved significantly over serial execution.

Franco et al. [17] have focused on parallel execution of evaluation part of GA. The evaluation part comprised the implementation of two kernels. The first kernel gathered the evaluation parameters and the second kernel used a reduction algorithm to aggregate the gathered parameters.

In [8], Cano et al. have used GPU to implement the evaluation part of GA in order to improve the performance of fitness execution. The execution configuration included three-dimensional thread/blocks. The space of these thread/blocks has been formed by individuals, instances of data and attributes. In [54], Cano et al. have presented a high performance and efficient implementation of Pittsburgh rule-based classifier. The proposed evaluation model is scalable and executed on multiple GPUs and implemented using CUDA. Fitness function evaluated the individuals using two kernels. The first kernel calculated the coverage of each condition on every instance of data. The second kernel exploited the amount of coverage of each chromosome to compute the parameters of confusion matrix. Finally, the fitness value is calculated with the parameters of confusion matrix.

4. The proposed approach

Our proposed approach is based on rule discovery using genetic algorithm (GA) which is implemented on GPU. First we discuss our approach for rule encoding and then a new method is proposed using genetic algorithm and its implementation on GPU. A novel approach for calculating the fitness of each rule is proposed. Also, we customized genetic operators for the purpose of this work. To address parallelism and achieve speedup in the algorithm, GPU utilization is described in each step. Fig. 2 shows the flowchart of the proposed GA-based rule discovery.

Before we proceed with our rule discovery approach on GPU, we introduced few notations which are used in the rest of the paper. These notations are summarized below:

R_c^r: Rule r of class c for the current population. r is used to refer to the rule numbers sequentially.
C: number of classes in the dataset.
v_{yx}: value of feature x in instance number y.
pop_c: Current population of genetic algorithm for class c. $pop_c = \{R_c^1, R_c^2, \ldots, R_c^{|pop_c|}\}$
$|pop_c|$: population size for class c.
$pop\_size = \sum_{(i=1)}^{C} |pop_i|$

Unless specifically stated, the population size is the same for all classes and its number for each class is indicated as SP. In this case, $pop\_size = C \times SP$ which indicates current population of genetic algorithm.

4.1. Rule encoding

Fig. 3 depicts the employed encoding of rules that is based on what demonstrated in Fig. 1. The difference between the encoding of Fig. 1 and Fig. 3 is that Val_i parameter in Fig. 1 is divided into two parameters namely LV and UV; each of them represents the upper and lower value of the original Val_i. Each rule consists of n sections; where n is the number of features in the dataset. Each section is one of the conditions of the rule. The structure of each condition is shown in Fig. 3 which consists of four elements. The meaning of each element is as follows:

RA: indicates the activeness of each condition in the rule.
RC: represents the type of the operator between the rule and its equivalent in the dataset. The operators are applied between the value of the specific condition and its equivalent attribute in the dataset.
LV: means the lower bound value to be compared with the corresponding attribute.
UV: implies the upper bound value to be compared with the corresponding attribute.

In genetic metaphor, rules are chromosomes. So each chromosome is a rule from a given class which is based on the rule encoding of Fig. 3.

4.2. Fitness evaluation

Rule fitness evaluation is performed in three different steps in this work which is one of the most important sections and we intend to take advantage of parallelism. The steps of the evaluation are *coverage*, *reduction*, and *confusion* calculation. Each step of the evaluation is described below:

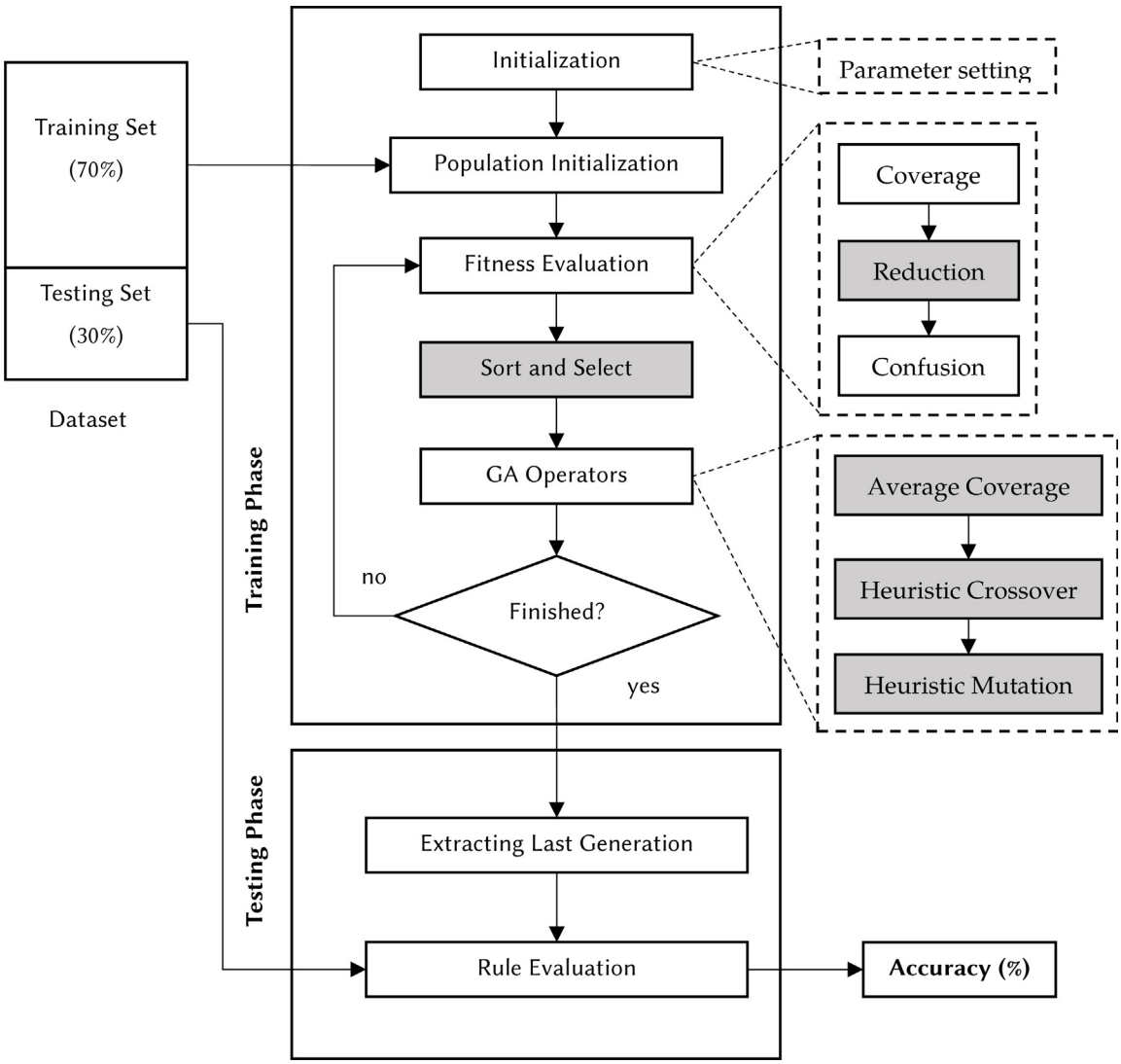

Fig. 2. Detailed representation of the proposed approach.

$Condition_1$				$Condition_2$				...	$Condition_n$			
RA_1	RC_1	LV_1	UV_1	RA_2	RC_2	LV_2	UV_2		RA_n	RC_n	LV_n	UV_n

Fig. 3. The employed Michigan-based encoding of rules.

4.2.1. Coverage

The goal of this step is to find the level of coverage for each condition in the rule according to the corresponding feature in the dataset. Considering the conditions arranged in the GA encoding, the rule is defined as:

If $(Condition_1)$ *and* $(Condition_2)$ *and* ...
...*and*$(Condition_a)$ *then class* $= c$

The comparison of each condition with the respective feature of data instance in dataset should be accomplished for all rules and for the whole population of GA and all generations. So in order to utilize GPU for comparing the conditions, first we put each rule in a matrix called *coverage matrix*, and then the comparison with features of each data instance is performed in parallel for each element of the matrix. This matrix is filled with zeroes and ones, depending on the result of comparison between the rule and dataset. The structure of the coverage matrix is represented in Eq. (1).

$$CM_c^r = \begin{bmatrix} cv_{0,0} & \dots & cv_{0,|attr|} \\ \vdots & \vdots & \vdots \\ cv_{|inst|,0} & \dots & cv_{|inst|,|attr|} \end{bmatrix}_{|inst|\times|attr|} \quad (1)$$

$$| \; cv_{y,x} = \begin{cases} 0: & \text{if}(c_x^r \lfloor \rfloor v_{y,x}) \\ 1: & \text{if}(c_x^r \lceil \rceil v_{y,x}) \end{cases}$$

In this matrix:

r: is the rule number that this matrix belongs to.
c: represents the class number of r.
x: is the feature number in the data set.
y: is the instance number in the data set.
$|inst|$: is number of instances in the data set.

$|attr|$: is number of features in the data set.
cv_{yx}: is '1' if condition x of rule r, matches feature x of data instance number y and otherwise it is zero.
c_x^r: is the value or range of values for condition x of rule r.

The operators $\lfloor\rfloor$ and $\lceil\rceil$ mean mismatch and match conditions of rules with the features in the datasets respectively. Elements of CM_c^r are indicated by their column and row numbers and their value can also be declared as follows:

$CM_c^r[i][j] = 1$; if the value of attribute j of the rule r fits the value of attribute j of the ith instance of data in dataset, otherwise it is '0'. In addition, the coverage of inactive conditions is equal to '1'.

Each thread is responsible for the calculation of a specific $cv_{y,x}$. So this step takes advantage of GPU and comparison operations on the matrix elements are performed in parallel.

Fig. 4 illustrates the execution configuration of the coverage kernel on GPU. The configuration of the coverage kernel contains attributes, data instances and rules as grid dimensions x, y and z, respectively. Each block contains 32×32 threads as shown in Fig. 4.

A specific rule number is accessed by the third dimension through the block number.

In coverage kernel, first each part of the rule is evaluated for feature activeness. If it is '1', then the type of the comparison operator is extracted and then the value of the rule is compared against the values in the dataset. If the result of comparison is true, $cv_{(y,x)}$ becomes '1' and otherwise it is set to '0'.

4.2.2. Reduction

This step aims to give a score to each rule based on the values stored in its coverage matrix and also to execute the rule evaluation process in parallel. To this end, coverage matrix is used and the values in each row of this matrix are summed up in parallel. The summation is performed according to the Eq. (2). This stage of our algorithm is called reduction. It leads to the formation of a vector for each rule in the population called reduction vector.

$$RV_c^r = \begin{bmatrix} rv_0 \\ \vdots \\ rv_{|inst|} \end{bmatrix}_{|inst|\times 1} \quad | \; rv_y = \sum_{x=0}^{|attr|} cv_{y,x} \tag{2}$$

Algorithm 1 *The reduction algorithm applied on each coverage matrix.*

Input: CM_c^r, $|attr|$, $|inst|$, pop_size
Output: RV_c^r

```
function COVERAGE_ REDUCTION(CM_c^r, |attr|, |inst|, pop_size)
    Initialize: RV_c^r
    for i = 0 to |inst| do
        for a = 0 to |attr| do
            rv_i = rv_i + cv_{i,a}
        end for
    end for
return RV_c^r
end function
```

In order to take advantage of parallelism, the summation is performed in the following steps:

1. Each of two consecutive elements in each row are added and the result is stored in the first element. This process is performed for all pairs in parallel.
2. This same process is repeated for the elements that have the results of the previous stage.
3. Operation continues until the summation of all elements is done. Finally, the summation is copied to the first element of the reduction vector. As the elements of the coverage matrix are zero or one, it is clear that elements of the reduction vector follow the following formula:

$$0 \le rv_i \le |attr| \tag{3}$$

The total number of reduction stages is $log_2^{|attr|}$. As stated earlier $|attr|$ is the number of attributes in the data set. This also indicates that the speedup obtained in this stage is in the order of $O(log_2^{|attr|})$, while sequential code executes this part in $O(|list| \times |attr|)$ which is a significant improvement.

$$\#\ of\ reduction\ stages\ =\ log_2^{|attr|} \tag{4}$$

Algorithm 1 presents the reduction step using sequential code and Listing 1 shows the same part using parallel programming in CUDA. The reduction used in this work is also known as "pre-fix scan" or "pre-fix sum" and it is available in one of NVIDIA documents [55] as "Example 3. The Up-Sweep (Reduce)". Although NVIDIA did not share any CUDA code for this type of pre-fix sum, We provided the code in Listing 1 and it is based on Algorithm 2 that is presented in [55]. According to Algorithm 2, the inner loop as a "for all" statement must be coded as a "for" with an "if" statement inside in order to check the index of the iterator and prevent the index to exceed over the number of attributes. Therefore, an "if statement is needed inside the inner for loop of the Listing 1 and this will cause an inevitable divergent execution on GPU.

Algorithm 2 *The up-sweep algorithm presented by NVIDIA.*

```
for i = 0 to log_2^{a-1} do
    for all k = 0 to a − 1 by 2^{i+1} in parallel do
        x[k + 2^{i+1} − 1] = x[k + 2^i − 1] + x[k + 2^i + 1 − 1]
    end for
end for
```

As an example, the reduction step for an 8-element reduction vector is represented in Fig. 5. The numbers stored in each element of the reduction vector shows how fit the rule is with respect to the instances of the dataset. Having the maximum value equal to $|attr|$ in each row, implies that the rule completely covers all instances in the dataset.

4.2.3. Confusion

This final step at fitness evaluation is responsible to produce the value of fitness function, based on the parameters in the confusion matrix.

Confusion matrix construction

We have four parameters in the confusion matrix and its calculation is based on the reduction matrix obtained in the previous stage.

First we calculate the following parameters for each data instance using Eqs. (5) to (8) which are based on the values obtained for the reduction vector elements:

$$m_{(tp,y)} = \begin{cases} 1: & \text{if}(((rv_y \in RV_c^r) = |attr|) \,\&\, (c = class\ of\ y)) \\ 0: & otherwise \end{cases} \tag{5}$$

$$m_{(fp,y)} = \begin{cases} 1: & \text{if}(((rv_y \in RV_c^r) = |attr|) \,\&\, (c \neq class\ of\ y)) \\ 0: & otherwise \end{cases} \tag{6}$$

$$m_{(tn,y)} = \begin{cases} 1: & \text{if}(((rv_y \in RV_c^r) \neq |attr|) \,\&\, (c \neq class\ of\ y)) \\ 0: & otherwise \end{cases} \tag{7}$$

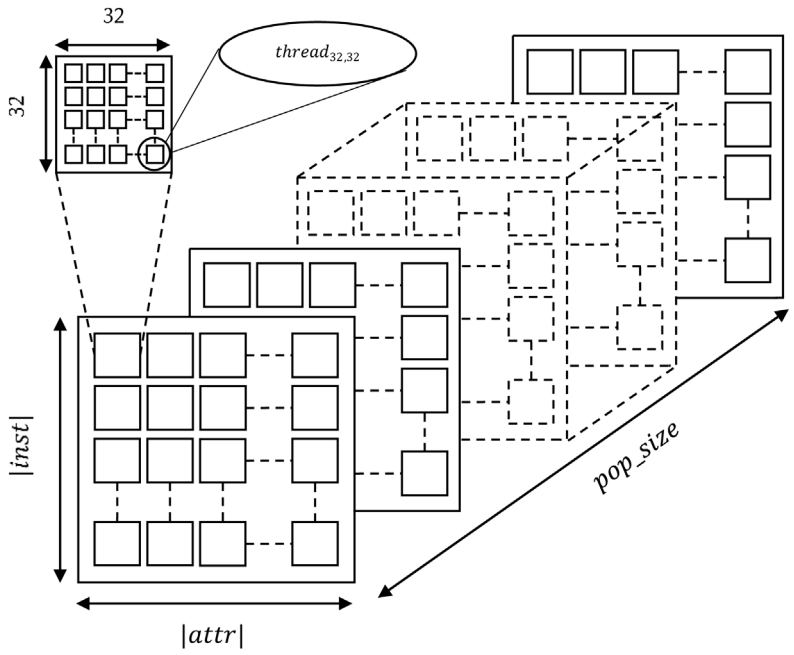

Fig. 4. Configuration of coverage kernel.

Listing 1: Parallel implementation of Reduction Vector (RV) GPU using CUDA

```
__global__ void GPU_RV(int *CM, int Attr, int Inst, int PopSize)
{
    int Row = (blockIdx.x*blockDim.x) + threadIdx.x;
    int Col = (blockIdx.y*blockDim.y) + threadIdx.y;

    if ((Row < Inst*PopSize) && (Col < Attr))
    {
        for (int a = 0;a <= log2f(Attr);a++)
        {
            if ((2 * (int)powf(2, a)*Col) + (int)powf(2, a) < Attr)
            {
                CM[Row*Attr + (2 * (int)powf(2, a)*Col)] += CM[Row*Attr + (2 * (int)powf(2, a)*Col) + (int)powf(2, a)];
            }
            __syncthreads();
        }
    }
}
```

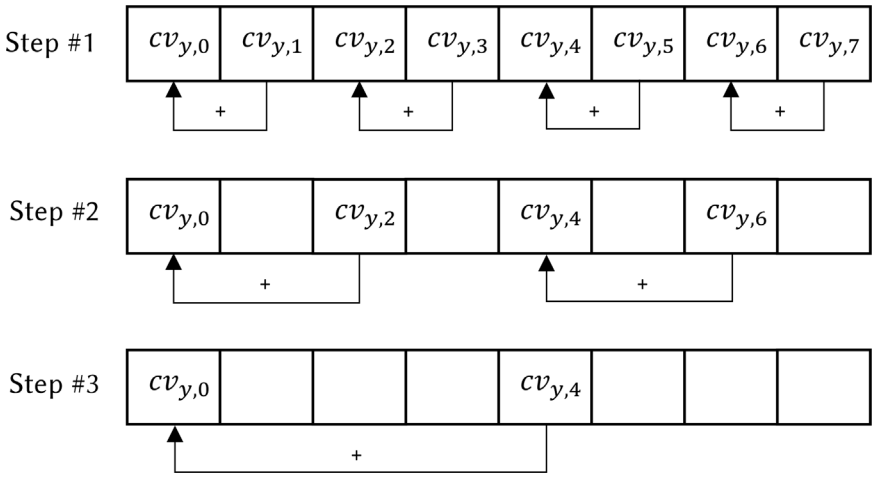

Fig. 5. An example of reduction on an instance of *coverage matrix* of chromosome r.

$$m_{(fn,y)} = \begin{cases} 1: & \text{if}(((rv_y \in RV_c^r) \neq |attr|) \,\&\, (c = class\ of\ y)) \\ 0: & otherwise \end{cases} \tag{8}$$

The above parameters are calculated for all elements of the reduction matrix and then the parameters of the confusion matrix are obtained using Eqs. (9) to (12). According to fitness calculation in [7], Eqs. (13) to (15) are used to measure the fitness value of each rule.

$$TP = \sum_y m_{(tp,y)} \tag{9}$$

$$FP = \sum_y m_{(fp,y)} \tag{10}$$

$$TN = \sum_y m_{(tn,y)} \tag{11}$$

$$FN = \sum_y m_{(fn,y)} \tag{12}$$

$$Precision = \frac{TP}{TP + FP} \tag{13}$$

$$True\ Positive\ Rate = \frac{TP}{TP + FN} \tag{14}$$

$$Fitness = Precision \times True\ Positive\ Rate \tag{15}$$

** TP: true positive, FP: false positive, TN: false negative, FN: false negative.

To prevent race condition between threads which want to increase the parameter of confusion matrix simultaneously, increase of these parameters is performed using atomic operation in CUDA.

Using TP, FP, TN, and FN, we obtained the accuracy, precision, and other relevant factors of the rules.

4.3. Selection

The selection phase of our GA approach sorts rules according to their fitness and then the first 50 of most qualified rules are selected for crossover. Mutation operator uses this sorted list and chooses the low fitness rules and provides them the opportunity for reproduction.

4.4. Genetic operators

We propose two novel approaches for genetic operators including mutation and crossover, which are customized for our problem. First, we introduced a concept named average coverage as follows:

4.4.1. Average coverage

It is defined as a vector for each rule in the algorithm. Each element of this vector shows the average coverage for the respective condition in the rule. Its definition is as Eq. (16).

$$AC_r = \begin{bmatrix} ac_0 & ac_1 & \dots & ac_{|attr|} \end{bmatrix}_{1\times|attr|} \mid ac_x = \frac{\sum_{n=0}^{|inst|_c} cv_{n,x}}{|inst|_c} \tag{16}$$

where $cv_{y,x}$ is the element of coverage matrix of Eq. (1) and $|inst|_c$ is number of instances in class c.

The **average coverage** vector (AC) is used in genetic operators and it leads to better choices for the next generation rules.

Table 1
Experimental setup of this work.

Component	1st system	2nd system
CPU	Intel Core i7-4790K	Intel Core i7-6700K
Main Memory	16 GB	16 GB
PCIE Version	3.0/2.0 x16	3.0/2.0 x16
GPU	Geforce GTX 970	NVIDIA Geforce GTX 1070
GPU Architecture Series	Maxwell	Pascal
GPU DRAM capacity	4 GB	8 GB
Number of CUDA cores	1664	1920
Number of SMs	13	15
Operating System	Windows 10	Ubuntu Desktop 18.1
CUDA Toolkit Version	8.0	10.1

4.4.2. Proposed crossover

Using the above idea of having the average coverage of all conditions in a rule, the crossover between two rules with fitness higher than the average, is defined as follows:

Rule crossover: For rules r and r' if r has higher fitness than r' then the condition x' of r' is replaced with its peer condition x of r, if and only if the average coverage of x is higher than the average coverage of x'. Fig. 6 shows the crossover operation between two chromosomes, where $\uparrow\downarrow$ means crossover operator between two corresponding conditions in the rule.

4.4.3. Proposed mutation

Mutation is performed on the rules with lower fitness in order to improve their chances. Therefore, the idea of mutation in this work is to replace the conditions of lower coverage with some random conditions for the lower half of the rules.

Hence, for each c_r rule, the AC_r vector is calculated and elements with the value below 1 are replaced with another random condition. In order to increase the chances for the rules with lower fitness to be improved, the mutation operator is applied on the lower fitness half of the population.

Fig. 7 shows an example of mutation operation applied on a chromosome. We notice that mutation is performed on the first and last genes because their ac parameter is less than 1. In the next section, we showed the results based on our proposed approach for generating discovery rules on GPU.

5. Experimental results

In this section our results obtained using three datasets are presented. Two of them, Hepatitis C Virus (HCV) [56] and Poker [57] are acquired from the University of California, Irvine (UCI) machine learning repository and the third one COVID-19 is available on Kaggle [58]. We preprocessed this dataset and the preprocessed version along with the preprocessing method is also available on Kaggle [59]. In addition, the implemented code of the proposed PGA is provided in [60].

5.1. Experimental setup

We run our algorithms on two different systems. Hardware and software specifications of these systems used during our experiments are presented in Table 1. In CUDA environment, CPU and GPU are called "host" and "device", respectively. Parallel portions of an application are executed on the device (GPU) as kernels while serial parts of the code are executed on the host (CPU). According to Table 1, there are one host (CPU) and one device (GPU) for each system. As mentioned earlier, the serial and parallel execution are performed using CPU and GPU, respectively. Hence, the speedup results are reported individually based on the system executed the proposed PGA.

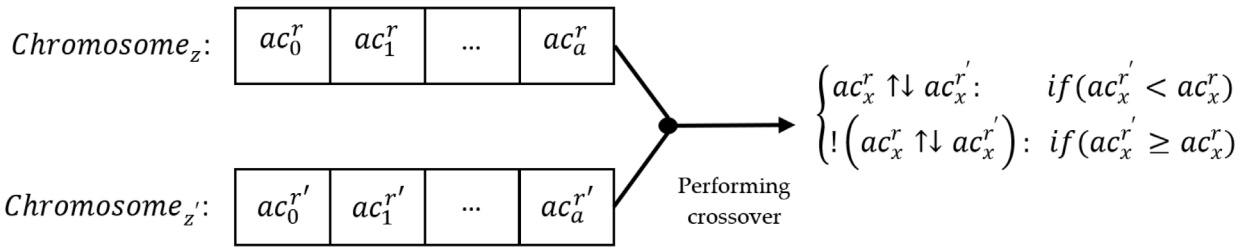

Fig. 6. Illustration of crossover of two chromosomes r and r'.

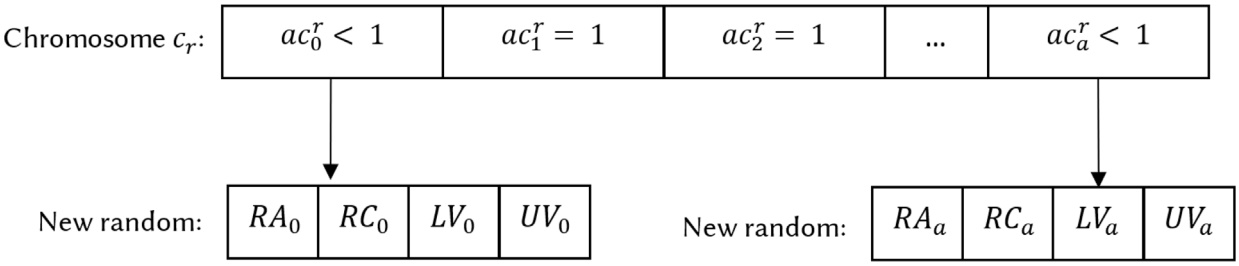

Fig. 7. An example of performing mutation on chromosome z.

Table 2
Details of datasets used.

Dataset	*Classes*	*|attr|*	*|inst|*
HCV	4	28	1385
Poker	10	10	1025010
COVID-19	3	22	300

5.2. Results

Our primary goal is to test our algorithm on big datasets. Hence, we used Hepatitis C Virus (HCV), and COVID-19 medical datasets. The third non-medical dataset is Poker hand dataset. HCV and Poker datasets are obtained from UCI machine learning repository. The characteristics of these three datasets are presented in Table 2.

Table 3 shows the best chromosomes of each class achieved via the proposed PGA. As an example for COVID-19 dataset in Table 3, people with the age greater than 37 along with several specific symptoms are classified into class of 'death state'. The mentioned symptoms are rhinorrhea, respiratory symptoms, headache, weakness and chest pain. As an example of interpretation of rules in Table 3, although the best chromosomes for each class of COVID-19 dataset are indicating many attributes with the value equal to '0', the condition for such attributes does not mean a don't-care term. Following the previous example of COVID-19, attributes with the index greater than '4' represent a symptom. For instance, a_4 represents "fever" symptom and the patient can either have or not have fever and that means a_4 can be equal to '1' or '0', respectively. However, in some classes, a symptom can be reported as 'seen' in a proportion of patients, while there exist some cases not having such symptom. Following to such cases with the probability of having or not having a symptom, the value of the attributes should be greater or equal to '0'. Therefore, having a condition with a value that is greater or equal to '0' defines a "don't-care" term on the related attribute.

We tested our method on HCV medical dataset with 28 attributes and 414 instances, categorized as test-set, based on the 70–30 principle. The mentioned 414 instances are 30% of the total instances inside HCV data and they are randomly selected, whereas the rest of the instances with the number of 971 are selected as training set. It is a new dataset with many records, and hence makes it a good choice to test the classification performance of our algorithm on GPU. Also, we have used another new COVID-19 dataset. The original version of the employed COVID-19 dataset of this work was provided in Kaggle and had a lot of issues such as having many missing values and merged attributes. Attributes such as "symptoms" included any symptom that has been seen in a patient. For example, a patient with "fever", "cough" and "fatigue" had one attribute (column) as "fever-cough-fatigue". Such merging of attributes made the dataset impossible to train. Therefore, we preprocessed and modified the original dataset and separated all reported symptoms into different attributes. To see the effect of using GPU on larger datasets, we used Poker dataset consisting of 1025010 instances and 10 classes. We observed that our algorithm performed well on all datasets with best accuracy of 99.74%, 100%, and 95.73% using HCV, COVID-19, and Poker datasets respectively.

We tested our algorithm on different population sizes and iterations. The training set included 70% of the instances and the next 30% are chosen as the test set (We changed the default numbers in Poker dataset). Fig. 8 illustrate the accuracy obtained for different population sizes with different number of iterations of GA for each dataset using 1st system. This figure presents the accuracy for various rules of each class, iterations and datasets. In addition, Table 4 provides more details about Fig. 8. The results illustrated in Fig. 8 shows a spike on SP = 4 for COVID-19. Table 4 also shows that the proposed PGA has higher accuracy using 10000 iterations for SP = 4 on COVID-19 compared to subsequent number of SPs (SP = 8, SP = 16). Such event might occur based on two subjects knows as *(1) the quantity of the instances inside the dataset* and *(2) the quality of the randomness caused by heuristic mutation*. The mentioned subjects are explained as follows:

1. As an example, for deceased patients there are only 4 instances that two of them are selected to train and the rest are used to test the generated rules. Therefore, any miss-predictions may lead to a serious loss of accuracy.
2. Mutation is a random process responsible for some changes in prediction by random selection of conditions. In this particular case, a set of random conditions over 10000 iterations may have caused that predictions be true with a high probability.

Table 3
Best Chromosomes of each class in employed datasets.

Dataset	Class	Best Chromosome
HCV	*No Fibrosis*	$(a_2 \geq 22)$, $(a_3 \geq 1)$, $(a_6 \geq 1)$, $(a_7 \geq 1)$, $(a_8 \geq 1)$, $(a_9 \geq 1)$, $(a_{13} \geq 93013)$, $(a_{16} \geq 39)$, $(a_{13} \geq 93013)$, $(a_{18} \geq 39)$, $(a_{19} \geq 29)$, $(a_{20} \geq 29)$, $(a_{21} \geq 11)$, $(a_{22} \geq 53)$, $(a_{13} \geq 93013)$, $(a_{23} \geq 254)$, $(a_{25} \geq 5)$, $(a_{27} \geq 3)$,
	Few Septa	$(a_1 \geq 1)$, $(a_4 \geq 1)$, $(a_6 \geq 1)$, $(a_7 \geq 1)$, $(a_9 \geq 1)$, $(a_{12} \geq 10)$, $(a_{13} \geq 93131)$, $(a_{15} \geq 39)$, $(a_{17} \geq 39)$, $(a_{18} \geq 39)$, $(a_{19} \geq 12)$, $(a_{23} \geq 577)$, $(a_{26} \geq 7141)$, $(a_{27} \geq 3)$
	Many Septa	$(a_0 \geq 32)$, $(a_2 \geq 22)$, $(a_4 \geq 1)$, $(a_5 \geq 1)$, $(a_6 \geq 1)$, $(a_7 \geq 1)$, $(a_{10} \geq 3019)$, $(a_{13} \geq 95026)$, $(a_{14} \leq 127)$, $(a_{17} \geq 39)$, $(a_{22} \geq 197)$
	Cirrhosis	$(a_2 \geq 22)$, $(a_3 \geq 1)$, $(a_4 \geq 1)$, $(a_5 \geq 1)$, $(a_8 \geq 1)$, $(a_9 \geq 1)$, $(a_{11} \geq 3816957)$, $(a_{12} \geq 10)$, $(a_{14} \geq 39)$, $(a_{15} \geq 39)$, $(a_{16} \geq 39)$, $(a_{18} \geq 39)$, $(a_{21} \geq 7)$, $(a_{22} \geq 46)$, $(a_{24} \geq 3537)$, $(a_{25} \geq 11754)$, $(a_{25} \geq 5)$
Poker	*Nothing in hand*	$(a_0 \geq 1)$, $(a_1 \geq 1)$, $(a_2 \geq 1)$, $(a_3 \geq 1)$, $(a_5 \geq 1)$, $(a_7 \geq 12)$
	One pair	$(a_0 \geq 1)$, $(a_2 \geq 1)$, $(a_3 \geq 1)$, $(a_4 \geq 1)$, $(1 \leq a_5 \leq 12)$, $(a_6 \leq 1)$, $(a_7 \geq 1)$, $(a_8 \geq 1)$
	Two pairs	$(a_0 \geq 2)$, $(a_2 \geq 1)$, $(a_3 \leq 12)$, $(a_4 \geq 1)$, $(2 \leq a_5 \leq 11)$, $(a_6 \leq 1)$, $(a_7 \leq 12)$, $(a_8 \geq 2)$, $(a_9 \geq 4)$
	Three of a kind	$(a_0 \geq 1)$, $(a_1 \leq 10)$, $(a_3 \leq 6)$, $(2 \leq a_5 \leq 10)$, $(a_7 \leq 3)$, $(a_8 \leq 1)$, $(a_9 \geq 4)$
	Three of a kind	$(a_0 \geq 1)$, $(a_1 \leq 10)$, $(a_3 \leq 6)$, $(2 \leq a_5 \leq 10)$, $(a_7 \leq 3)$, $(a_8 \leq 1)$, $(a_9 \geq 4)$
	Straight	$(a_1 \leq 12)$, $(a_5 \geq 9)$, $(a_6 \geq 1)$, $(a_7 \geq 3)$, $(a_9 \leq 12)$
	Flush	$(a_1 \geq 1)$, $(a_3 \geq 1)$, $(a_4 \geq 1)$, $(a_5 \geq 1)$, $(a_7 \geq 1)$, $(a_8 \geq 1)$, $(a_9 \geq 1)$
	Full house	$(a_0 \geq 1)$, $(a_1 \geq 1)$, $(a_3 \geq 1)$, $(a_5 \geq 1)$, $(a_8 \geq 1)$, $(a_9 \geq 1)$
	Four of a kind	$(a_0 \geq 1)$, $(a_1 \geq 1)$, $(a_3 \geq 1)$, $(a_4 \geq 1)$, $(a_5 \geq 1)$, $(a_9 \geq 1)$
	Straight flush	$(a_0 \geq 1)$, $(a_2 \geq 1)$, $(a_3 \geq 1)$, $(a_4 \geq 1)$, $(a_5 \geq 1)$, $(a_7 \geq 1)$, $(a_8 \geq 1)$, $(a_9 \geq 1)$
	Royal flush	$(a_0 \geq 1)$, $(a_1 \geq 1)$, $(a_2 \geq 1)$, $(a_3 \geq 1)$, $(a_4 \geq 1)$, $(2 \leq a_9 \leq 12)$
COVID-19	*Infected*	$(a_0 \leq 76)$, $(a_2 \geq 0)$, $(0 \leq a_3 \leq 11)$, $(a_4 \geq 0)$, $(a_{12} = 0)$, $(a_{14} = 0)$, $(a_{15} \geq 0)$, $(a_{16} \geq 0)$, $(a_{17} \geq 0)$, $(a_{18} \geq 0)$, $(a_{21} = 0)$
	Discharged or Recovered	$(8 \leq a_0 \leq 73)$, $(a_1 \geq 0)$, $(2 \leq a_2 \leq 3)$, $(a_3 \geq 1)$, $(a_4 \geq 0)$, $(a_6 = 0)$, $(a_{10} \geq 0)$, $(a_{11} \geq 0)$, $(a_{13} = 0)$, $(a_{17} \geq 0)$
	Deceased	$(a_0 \geq 37)$, $(a_6 = 0)$, $(a_9 = 0)$, $(a_{11} \geq 0)$, $(a_{12} = 0)$, $(a_{14} = 0)$, $(a_{15} = 0)$, $(a_{16} \geq 0)$, $(a_{17} \geq 0)$, $(a_{18} \geq 0)$, $(a_{19} = 0)$, $(a_{20} = 0)$, $(a_{21} \geq 0)$

Table 4
Average accuracy on both datasets and for different SP and iteration numbers using 1st system.

Data set	SP	Number of iterations					
		100	500	1000	3000	5000	10000
HCV	4	0.23	2.31	12.69	16.22	47.29	67.41
	8	53.71	18.15	42.05	72.58	80.96	87.56
	16	61.38	72.21	91.81	90.36	94.84	97.87
	32	86.64	89.89	97.10	98.78	99.74	99.51
Poker	4	79.39	80.04	80.31	89.26	76.65	80.96
	8	88.64	88.04	90.87	89.99	82.45	83.99
	16	91.48	91.87	90.27	90.87	92.22	95.73
	32	91.57	89.55	91.22	90.34	89.86	91.05
COVID-19	4	43.65	33.33	70.24	48.81	33.33	94.67
	8	65.87	66.67	60.71	61.11	83.33	83.33
	16	82.94	83.33	94.44	61.11	83.33	83.33
	32	66.27	77.78	99.60	94.44	83.33	100.00

Table 5
Average accuracy on both datasets and for different SP and different iteration numbers using 2nd system.

Data set	SP	Number of iterations					
		100	500	1000	3000	5000	10000
HCV	4	11.00	11.44	21.70	51.13	51.40	50.17
	8	14.63	23.78	43.09	83.83	81.40	83.75
	16	40.52	67.54	71.17	87.77	95.08	93.87
	32	81.23	93.00	87.80	95.07	90.34	99.29
Poker	4	68.80	83.28	86.98	84.17	83.33	83.33
	8	88.69	81.35	85.58	88.33	85.83	87.78
	16	95.71	94.25	90.12	90.38	91.67	94.17
	32	91.30	88.60	91.75	89.60	91.01	94.17
COVID-19	4	27.78	60.32	56.74	55.55	66.67	60.32
	8	59.13	81.35	66.67	61.11	83.33	83.33
	16	72.22	83.33	83.33	83.33	83.33	88.89
	32	88.89	83.33	83.33	94.84	99.60	99.69

Similar to Fig. 8, Fig. 9 depicts the accuracy of the proposed PGA using 2nd system. Both Fig. 8 and Fig. 9 show the accuracy of the best discovered rules, number of rules of each class (SP) and number of iterations. In general, the results of both figures show that the accuracy improved with the increase in the number of iterations and rules of each class (SP). Furthermore, Table 5 provides more details about the illustrated results in Fig. 9.

In this work, the speedup of each part of the PGA is defined based on Eq. (17). In (17), X represents an individual step of the PGA that can be coverage, reduction or confusion.

$$Speedup_X = \frac{Serial\ Execution\ Time_X}{Parallel\ Execution\ Time_X} \tag{17}$$

Table 6 provides the results of speedup for different parts of the algorithm using 1st system. This figure shows that the speedup obtained by using the GPU of the 1st system is higher in the reduction stage than in the other stages of algorithm. According to the reduction scheme, all reduction operations in all rows of the reduction vector are executed in parallel. Each operation is a summation which is performed in the order of $log_2^{|attr|}$.

Fig. 10 illustrates the average speedup of each kernel of the PGA based on the different number of SPs for each dataset using 1st system. The results show almost the same speedup for different number of SPs for HCV and Poker. The reason for such event is that HCV and Poker both contain large amount of data. According to the large amount of data within these datasets, the program

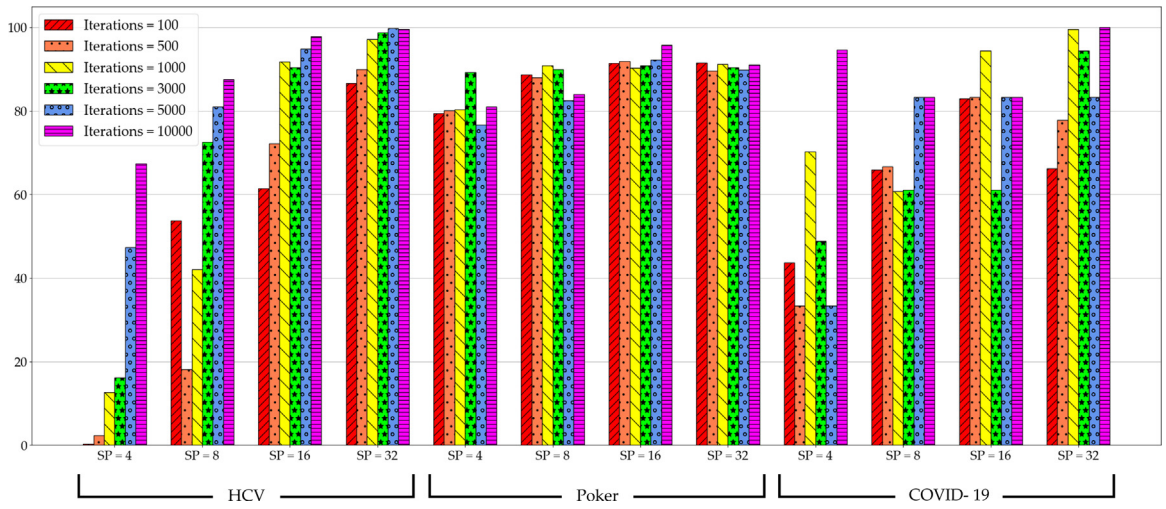

Fig. 8. Accuracy of discovered rules for each dataset, based on the population size and number of iterations.

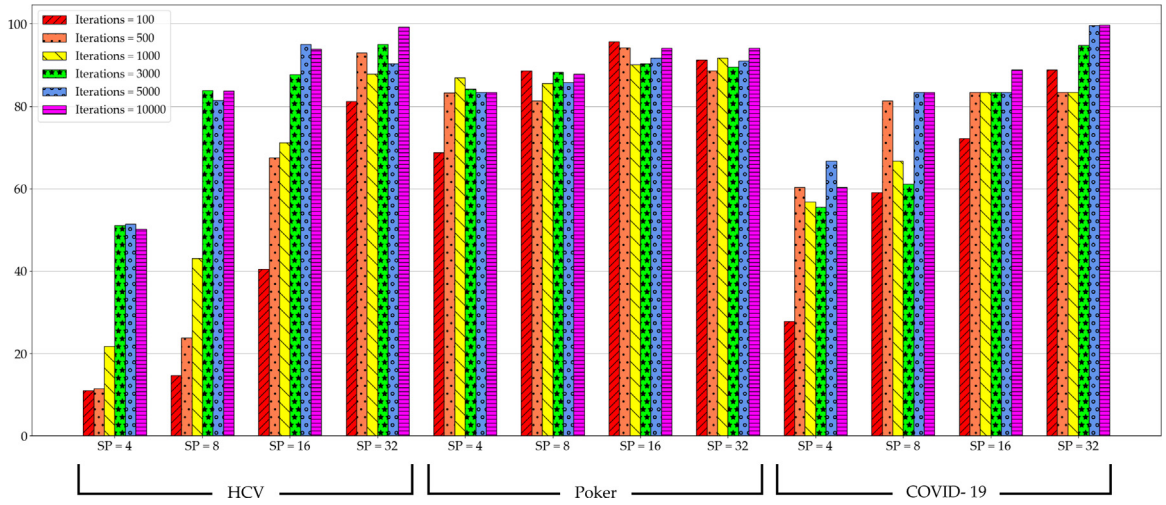

Fig. 9. Accuracy of discovered rules for each dataset, based on the population size and number of iterations.

becomes architectural dependent. To prove the architectural dependency of our PGA, NVIDIA GTX 1070 with more architectural capabilities is employed to test the speedup of the PGA. Therefore, the primary reason of employing the 2nd system is to show that the speedup of the proposed PGA is scalable beyond the number of SP equal to 4.

Table 7 provides the results of speedup versus evaluation part, population size, dataset and number of iterations using 2nd system. Similar to Fig. 10, Fig. 11 illustrates the average speedup of each evaluation part using 2nd system and based on population size, dataset and number of iterations using. Fig. 11 proves that the proposed PGA is scalable for the number of SPs greater than 4. According to Fig. 10 and Fig. 11, it can be noted from the results that the speedup rises with the growth of population size, number of attributes and number of data instances. For instance, in Poker dataset, speedup of reduction section is higher than the speedup of reduction for HCV dataset. In addition, the results indicate higher speedup due to increase in the number of rules in each class (SP).

According to Section 2.0.1, time complexity of *coverage* is a member of $O(r \times |inst| \times |Atrr| \times i)$ in case of serial execution. Hence, the ideal speedup of *coverage* kernel is expected to be equal to $(r \times |inst| \times |Atrr|)$, because each thread is assigned to an individual element in Fig. 4.

According to Section 4.2.2, time complexity of *reduction* is $O(|inst| \times |Atrr|)$ in a serial execution. Therefore, the ideal speedup of *reduction* is equal to $\frac{|inst| \times |Atrr|}{log_2^{|Attr|}}$. In addition, there is no specific expected speedup for *confusion* in parallel execution based on the atomic transactions needed to be performed in this kernel. However, both Fig. 10 and Fig. 11 show that all achieved speedups for each kernel is lower than its related ideal speedup.

Such outcome is caused by the type of the kernels. A kernel is memory-bound if it spends most of its time on executing memory instructions. A kernel is compute-bound if most of the operations are ALU instructions [61].

Although *reduction* is the kernel with higher computational instructions performing addition operations, it is a memory bound kernel based on the studies presented in [62]. Therefore, *coverage* and *confusion* are also memory bound kernels, since both of them have less computational operations compared to *reduction*.

In a memory bound kernel, performance is saturated after a certain size of input data due to bandwidth and limitations of other architectural memory resources. Since all of the kernels in the proposed approach are memory bound, increasing the size of the input data may not have any noticeable effect on speedup. Fig. 10 and Fig. 11 also show that increasing the number of SPs for all of the datasets has the least effect on the speedup of *coverage* and *confusion* kernels using both systems. However, there is a

Table 6
Speedup (%) on both datasets for SP and different iteration numbers using 1st System.

Data set	SP	PGA step	Number of iterations					
			100	500	1000	3000	5000	10000
HCV	4	CM	6.54	5.81	6.22	4.90	5.12	4.69
		RV	17.70	21.71	19.51	19.09	19.21	18.86
		Conf.	0.15	0.35	0.34	0.36	0.32	0.31
	8	CM	5.59	6.40	4.99	5.03	4.70	4.35
		RV	20.74	19.03	19.74	19.02	19.08	19.61
		Conf.	0.40	0.34	0.48	0.49	0.62	0.60
	16	CM	6.42	5.33	6.62	5.91	4.30	5.76
		RV	20.91	20.41	19.40	21.28	20.99	20.84
		Conf.	1.13	0.64	0.75	0.74	0.79	0.79
	32	CM	5.84	6.75	6.36	6.04	6.29	6.81
		RV	20.6	22.07	20.41	22.74	22.11	23.06
		Conf.	1.25	1.09	1.08	0.93	1.08	1.02
Poker	4	CM	5.38	4.92	4.98	5.21	4.20	4.76
		RV	49.76	50.21	50.96	51.28	50.95	50.61
		Conf.	0.99	0.80	0.83	0.81	0.80	0.82
	8	CM	3.52	3.38	5.08	4.31	3.38	4.77
		RV	49.97	51.03	51.18	51.74	55.37	52.45
		Conf.	1.00	0.88	0.86	0.82	0.86	0.82
	16	CM	4.88	4.73	5.10	4.31	4.62	4.19
		RV	52.37	57.15	52.15	53.16	53.21	48.19
		Conf.	1.10	0.99	0.88	0.90	0.89	0.79
	32	CM	3.86	5.15	3.98	4.15	3.95	4.89
		RV	51.35	51.80	53.78	50.22	48.16	56.30
		Conf.	1.19	1.07	1.09	0.95	0.95	1.12
COVID-19	4	CM	0.14	0.88	1.14	1.00	1.17	1.00
		RV	6.82	8.25	8.38	8.13	8.63	10.00
		Conf.	0.06	0.00	0.00	0.00	0.20	0.00
	8	CM	1.45	4.00	2.67	2.33	1.88	1.50
		RV	9.27	9.57	14.00	12.91	15.11	13.80
		Conf.	0.00	0.10	0.33	0.17	0.17	0.17
	16	CM	3.08	2.64	3.30	2.33	3.00	2.55
		RV	17.00	20.21	16.76	17.38	17.56	18.13
		Conf.	0.11	0.25	0.33	0.43	0.29	0.29
	32	CM	1.91	4.21	4.00	4.14	3.93	4.29
		RV	22.00	19.96	21.58	21.35	22.12	19.61
		Conf.	0.10	0.50	0.40	0.40	0.67	0.56

CM = Coverage Matrix; RV = Reduction Vector; Conf. = Confusion Matrix.

Table 7
Speedup (%) on both datasets for SP and different iteration numbers using 2nd System.

Data set	SP	PGA step	Number of iterations					
			100	500	1000	3000	5000	10000
HCV	4	CM	7	10.5	6.58	8.67	8.44	8.11
		RV	33.15	32.46	34.67	31.77	31.62	34.33
		Conf.	2.33	0.8	0.67	0.5	0.67	0.67
	8	CM	9.71	9.65	9.81	9.33	8.69	9
		RV	38.5	37.73	39.33	37.5	37.45	37.45
		Conf.	0.71	0.5	0.5	0.88	0.86	0.88
	16	CM	9.97	12.08	12.25	11.7	11.82	12.1
		RV	35.89	40.29	38.53	40.41	40.22	41.23
		Conf.	2	1.33	1.78	1.4	1.27	1.3
	32	CM	11.25	12.87	10.65	10.51	10.83	10.29
		RV	44.85	47.1	43.46	43.46	43.99	43.97
		Conf.	1.93	2.17	1.71	1.86	1.67	1.67
Poker	4	CM	9.13	9.19	9.01	8.32	9.42	8.85
		RV	103.13	107.75	106.3	107.09	107.66	111.51
		Conf.	1.38	1.16	1.21	1.19	1.13	1.19
	8	CM	8.44	8.35	7.79	7.99	7.68	8.37
		RV	108.68	109.11	110.45	111.95	110.17	112.16
		Conf.	1.51	1.23	1.21	1.22	1.2	1.24
	16	CM	9.03	9.06	8.61	8.52	9.17	8.83
		RV	110.39	112.12	111.06	111	114.56	113.81
		Conf.	1.84	1.75	1.61	1.56	1.65	1.66
	32	CM	10.28	10.39	10.86	10.42	10.64	10.27
		RV	137.55	137.29	145.24	143.15	142.32	137.18
		Conf.	2.82	2.5	2.48	2.5	2.45	2.37
COVID-19	4	CM	2	1.33	1.5	2	2	1.75
		RV	12.5	17.75	13.6	17	13.4	13.6
		Conf.	0	0.33	0	0.25	0	0.25
	8	CM	6.5	3.4	3	3.75	3.2	2.8
		RV	18.86	22.67	22.67	26.8	26.8	22.33
		Conf.	0.25	0.25	0.2	0.25	0.25	0.33
	16	CM	7.2	4.71	4.43	4.29	3.75	4.43
		RV	36.5	34.25	34.13	33.88	38.57	33.63
		Conf.	0.75	0.8	0.4	0.4	0.4	0.4
	32	CM	4.13	8	7	6.2	7.63	7
		RV	42.1	41.92	44.92	44.83	41.38	41.38
		Conf.	0.2	0.8	1.4	1	0.67	0.67

CM = Coverage Matrix; RV = Reduction Vector; Conf. = Confusion Matrix.

slight increase on speedup of *reduction* on HCV and Poker using 2nd system in Fig. 11 compared to the speedup gained using 1st system in Fig. 10. In addition, the speedup of *reduction* applied on COVID-19 using both systems have an expected behavior since the COVID-19 dataset is a small dataset and it causes the *reduction* not to exceed limits of the architectural resources.

Table 8 shows a comparison between the accuracy of our work and other related state of the art. The proposed PGA of our work has 14.04% higher accuracy on HCV dataset compared to [63] that employs Decision Tree as its method. In addition, for Poker Hand dataset, our PGA provides 44.67%, 22.17% and 1.73% of higher accuracy compared to PANFIS [64],Bottom-up Pittsburgh [65] and ANN [66], respectively.

Table 8 also shows the accuracy result compared with MoMAC algorithm, which is a more up-date work. Since MoMAC was applied to different datasets than those are used in this work, Breast Cancer dataset [67] is employed in order to make the comparison between the proposed PGA and MoMAC possible. The results show that the proposed approach in this work has 3.32% higher accuracy compared to MoMAC employing Breast Cancer dataset.

Table 8
Accuracy comparison between the proposed PGA and other states of the art.

Method	Dataset	Accuracy (%)
Decision Tree [63]	HCV	85.7
PANFIS without active learning [64]	Poker Hand	51.06
Bottom-up Pittsburgh with entropy [65]	Poker Hand	73.56
Artificial Neural Network (ANN) [66]	Poker Hand	94.00
MoMAC [68]	Breast Cancer	92
Proposed PGA	HCV	**99.74**
Proposed PGA	Poker Hand	**95.73**
Proposed PGA	Breast Cancer	**95.32**

5.3. K-fold validation

All of the accuracy results provided in Tables 4 and 5 are based on the accuracy achieved from 10 executions in average using 70/30 validation method. K-Fold validation is also used to confirm the results obtained from 70/30. In this case, K is considered as equal to 10 that is known as 10-Fold validation. Table 9 shows the accuracy results of the performed 10-Fold validation using 1st system.

As a statistic analysis of the proposed GA, standard deviations of the 10 folds for each individual execution are provided in Table 10. Each cell in Table 10 represents the standard deviation of the accuracy results obtained from the execution of 10 folds

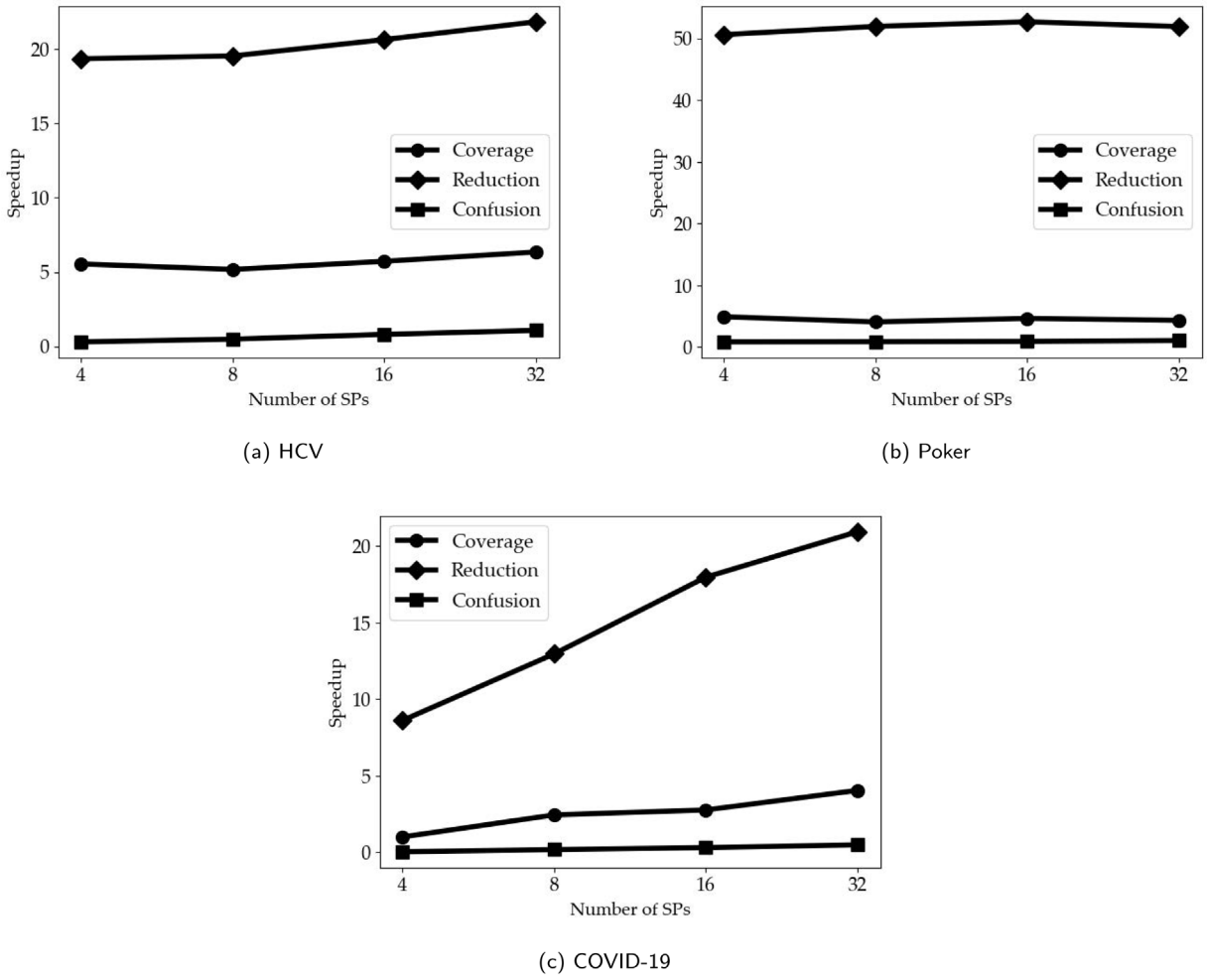

Fig. 10. The speedup of each kernel in average, based on the different number of SPs for each dataset using 1st system.

Table 9
Average accuracy for different SPs and number of iterations using 1st system, based on 10-Fold validation.

Data set	SP	Number of iterations					
		100	500	1000	3000	5000	10000
HCV	4	2.315	10.21	8.11	35.40	45.89	57.91
	8	11.23	29.91	44.06	69.44	78.78	88.04
	16	44.01	75.88	84.03	91.62	92.07	92.46
	32	81.65	94.05	97.32	97.85	98.25	99.11
Poker	4	78.09	75.23	75.56	81.50	78.78	79.26
	8	84.38	80.19	81.25	83.02	79.21	83.02
	16	88.81	85.84	88.76	87.76	89.08	89.56
	32	92.07	90.68	93.07	92.87	93.72	94.65
COVID-19	4	65.68	68.77	59.14	69.51	69.88	73.09
	8	62.09	66.29	69.75	73.33	73.33	76.66
	16	83.33	90.00	86.66	90.00	86.66	86.66
	32	86.66	90.00	89.88	93.33	96.67	99.96

Table 10
Standard deviations of the accuracy based on different SP sizes, different number of iterations and 10-Fold validation using 1st system.

Data set	SP	Number of iterations					
		100	500	1000	3000	5000	10000
HCV	4	3.79	11.77	4.67	9.06	11.92	14.92
	8	10.73	8.26	11.93	12.62	8.63	7.25
	16	17.10	11.69	7.64	5.30	5.15	6.23
	32	7.27	4.38	1.94	3.72	1.45	1.10
Poker	4	8.31	8.57	5.55	10.85	6.20	6.25
	8	4.38	6.98	4.32	7.53	8.17	6.95
	16	6.42	6.65	5.26	5.53	4.77	4.32
	32	4.22	4.46	3.80	4.16	3.77	1.18
COVID-19	4	10.16	22.72	19.31	23.59	17.98	13.48
	8	10.27	15.19	17.77	13.34	13.34	15.28
	16	16.67	15.28	16.33	15.28	16.33	16.33
	32	16.33	15.28	15.47	13.34	10.00	0.11

based on the number of SPs and iterations. The results show there is less variance between the accuracy of each fold while using more iterations and larger sizes of SPs. Therefore, the results in Table 10 proves that having more iteration and larger sizes of the SPs leads to higher and more stable accuracy results.

5.4. Improved kernel utilization

According to [69], the parallel reduction kernel proposed in this work has the potential to execute in GPU having more utilization. The proposed reduction kernel provided in Listing 1 can be optimized using shared memory and removing divergent

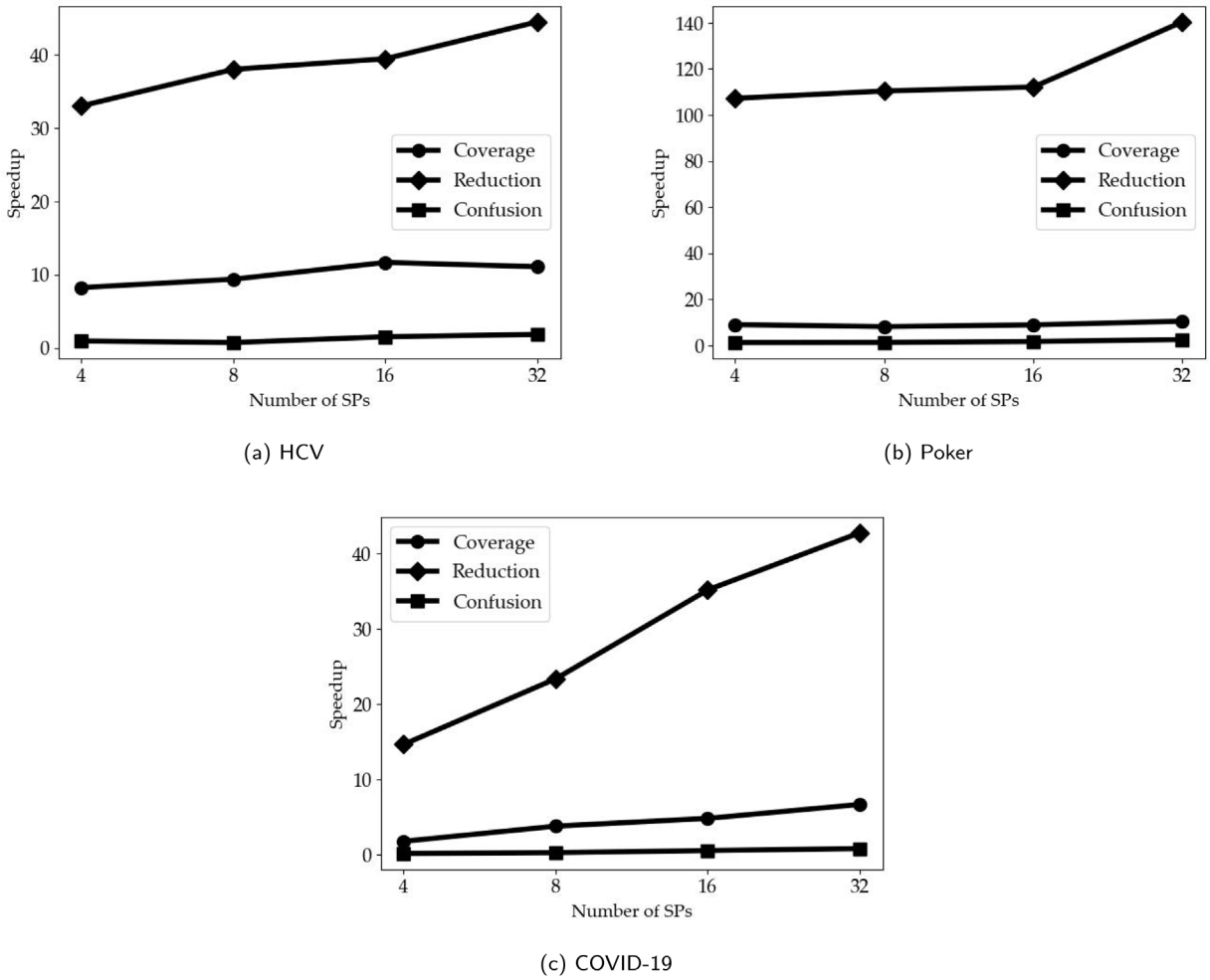

Fig. 11. The speedup of each kernel in average, based on the different number of SPs for each dataset using 2nd system.

branches from the code [69]. Listing 2 shows another reduction kernel considering non-divergent branches along with using shared memories for each block executing in GPU.

The results of the improved reduction kernel using 1st system are provided in Table 11. According to Table 11, up to 3.95 speedup can be obtained by executing the improved reduction kernel. Since the number of iterations does not affect the speedup of the execution and it only influences the accuracy, Table 11 does not contain the results for different number of iterations. Table 11 shows the speedup obtained by employing the improved reduction kernel does not depend on the size of SPs. Instead of Lines 6 to 16 in Listing 1, Lines 10 to 17 of Listing 2 calculate each reduction step using the defined shared memory (sdata) and non-divergent branches by modifying the *"for"* loop and *"if"* in lines 10 and 12, respectively. The added statements are responsible for calculating each reduction step and also comparing the thread indices with the number of attributes. Therefore, the speedup obtained by the new code depends on the number of attributes and this is why the results vary by changing datasets not changing the size of SP.

Table 11
Speedup of executing the improved reduction kernel (Listing 2) compared to the basic reduction kernel (Listing 1) using 1st system.

Dataset	Size of SP			
	4	8	16	32
HCV	3.89	3.92	3.90	3.95
Poker	2.04	2.05	2.05	2.05
COVID-19	2.38	2.55	2.70	2.86

6. Conclusion and future works

In this work we proposed a new efficient approach for discovering classification rules on GPU based on genetic algorithm. Our results are promising in terms of accuracy and speed up. We obtained accuracy up to 99.74% and 95.73% for HCV and poker datasets respectively. We obtained maximum speedup of 23.06%, 22.12%, and 57.15% for HCV, COVID-19, and Poker data respectively using our proposed algorithm.

It can be noted from our results (Table 4) that, the accuracy increase is proportional to population increase for most of the cases in both (HCV and Poker) databases. The reduction step has benefited most in speed due to the parallel execution. Since the calculation of the addition operations in the reduction vector is performed using parallel threads, we can observe a lot of speed up in this part. The search space of possible rules is huge and hence GA is an effective method for rule discovery. The problem with sequential GA is that all operations in each GA iteration have to be performed, one by one, for all training instances in the dataset. This process can be done in parallel using GPU and so number of instances have little impact on the run time of GA.

The parallel execution has reduced the number of steps and thus increased the speed of calculations. The main disadvantage of this work, is that it is likely that the algorithm may fall into local optima using GA producing rules. Selecting good measures in

Listing 2: Parallel implementation of the improved Reduction Vector (RV) using CUDA

```
template <int BLOCK_SIZE> __global__ void GPU_CoverageReduction_NoDivegence(int *g_iRV, int *g_oRV)
{
    int Row = (blockIdx.x*blockDim.x) + threadIdx.x;
    int Col = (blockIdx.y*blockDim.y) + threadIdx.y;

    __shared__ int sdata[BLOCK_SIZE][BLOCK_SIZE];
    sdata[threadIdx.x][threadIdx.y] = g_iRV[(Row * BLOCK_SIZE) + Col];
    __syncthreads();

    for (int a = BLOCK_SIZE / 2;a > 0;a >>= 1)
    {
        if (threadIdx.x < a)
        {
            sdata[threadIdx.x][threadIdx.x] += sdata[threadIdx.x][threadIdx.x + a];
        }
        __syncthreads();
    }
    if (Col == 0)
    {
        g_oRV[Row] = sdata[threadIdx.x][0];
    }
}
```

the fitness function is an important factor to avoid this problem. In future, we propose to use other GPU platforms and choose the best performing architecture. Also, using powerful GPUs, we intend to use genetic algorithm in combination with deep learning to find optimal values for deep network parameters and perform classification task using them. It is worth to compare the speedup between using GPU to produce rules by an evolutionary algorithm like the one in this work and use GPU to run a deep network with parameters set by an evolutionary algorithm.

CRediT authorship contribution statement

Mohammad Beheshti Roui: Conceptualization, Methodology, Software, Writing. **Mariam Zomorodi:** Conceptualization, Methodology, Supervision, Writing - original draft. **Masoomeh Sarvelayati:** Methodology, Writing, Software. **Moloud Abdar:** Data curation, Investigation, Writing - review & editing. **Hamid Noori:** Validation. **Paweł Pławiak:** Writing - review & editing, Investigation. **Ryszard Tadeusiewicz:** Supervision, Writing - review & editing. **Xujuan Zhou:** Writing - review & editing. **Abbas Khosravi:** Supervision. **Saeid Nahavandi:** Supervision. **U. Rajendra Acharya:** Supervision, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] M. Abdar, W. Książek, U.R. Acharya, R.S. Tan, V. Makarenkov, P. Pławiak, A new machine learning technique for an accurate diagnosis of coronary artery disease, Comput. Methods Programs Biomed. 179 (2019) 104992, http://dx.doi.org/10.1016/j.cmpb.2019.104992, https://doi.org/10.1016/j.cmpb.2019.104992.

[2] F. Pourpanah, R. Wang, X. Wang, Y. Shi, D. Yazdani, mBSO: A multi-population brain storm optimization for multimodal dynamic optimization problems, in: 2019 IEEE Symposium Series on Computational Intelligence, SSCI, IEEE, 2019, pp. 673–679.

[3] F. Pourpanah, R. Wang, C.P. Lim, D. Yazdani, A review of the family of artificial fish swarm algorithms: Recent advances and applications, 2020, arXiv preprint arXiv:2011.05700.

[4] W.H. Au, C.C. Chan, X. Yao, A novel evolutionary data mining algorithm with applications to churn prediction, IEEE Trans. Evol. Comput. 7 (6) (2003) 532–544, http://dx.doi.org/10.1109/TEVC.2003.819264, URL: https://ieeexplore.ieee.org/abstract/document/1255389/.

[5] P. Pławiak, M. Abdar, U. Rajendra Acharya, Application of new deep genetic cascade ensemble of SVM classifiers to predict the Australian credit scoring, Appl. Soft Comput. 84 (2019) 105740, http://dx.doi.org/10.1016/j.asoc.2019.105740, URL: https://doi.org/10.1016/j.asoc.2019.105740.

[6] P. Pławiak, M. Abdar, J. Pławiak, V. Makarenkov, U.R. Acharya, DGHNL: A new deep genetic hierarchical network of learners for prediction of credit scoring, Inform. Sci. 516 (2020) 401–418, http://dx.doi.org/10.1016/j.ins.2019.12.045, URL: https://www.sciencedirect.com/science/article/pii/S0020025519311569.

[7] A.A. Freitas, Data Mining and Knowledge Discovery with Evolutionary Algorithms, Springer Science & Business Media, 2013.

[8] A. Cano, A. Zafra, S. Ventura, Speeding up the evaluation phase of GP classification algorithms on GPUs, Soft Comput. 16 (2) (2012) 187–202, http://dx.doi.org/10.1007/s00500-011-0713-4, URL: https://www.researchgate.net/publication/216316010.

[9] F. Pourpanah, Y. Shi, C.P. Lim, Q. Hao, C.J. Tan, Feature selection based on brain storm optimization for data classification, Appl. Soft Comput. 80 (2019) 761–775.

[10] J. Gholami, F. Pourpanah, X. Wang, Feature selection based on improved binary global harmony search for data classification, Appl. Soft Comput. 93 (2020) 106402.

[11] M. Muntean, C. Rotar, I. Ileană, H. Vălean, Learning classification rules with genetic algorithm, 2010, pp. 213–216, http://dx.doi.org/10.1109/ICCOMM.2010.5509117, URL: https://ieeexplore.ieee.org/abstract/document/5509117/.

[12] M. Zomorodi-moghadam, M. Abdar, Z. Davarzani, X. Zhou, P. Pławiak, U.R. Acharya, Hybrid particle swarm optimization for rule discovery in the diagnosis of coronary artery disease, Expert Syst. (2019) http://dx.doi.org/10.1111/exsy.12485.

[13] S. Dehuri, A. Jagadev, A. Ghosh, R. Mall, Multi-objective genetic algorithm for association rule miningusing a homogeneous dedicated cluster of workstations, 2006.

[14] P. Kumar, A.K. Singh, Efficient generation of association rules from numeric data using genetic algorithm for smart cities, in: Security in Smart Cities: Models, Applications, and Challenges, Springer, 2019, pp. 323–343.

[15] M.V. Fidelis H S Lopes A A Freitas, Discovering comprehensible classification rules with a genetic algorithm, Ieeexplore.Ieee.Org (2000) 805–810, http://dx.doi.org/10.1109/CEC.2000.870381, https://doi.org/10.1109/CEC.2000.870381.

[16] B. M. Al-Maqaleh, H. Shahbazkia, A genetic algorithm for discovering classification rules in data mining, Int. J. Comput. Appl. 41 (18) (2012) 40–44, http://dx.doi.org/10.5120/5644-8072.

[17] M.A. Franco, N. Krasnogor, J. Bacardit, Speeding up the evaluation of evolutionary learning systems using GPGPUs, in: Proceedings of the 12th Annual Genetic and Evolutionary Computation Conference, GECCO '10, 2010, pp. 103–110, http://dx.doi.org/10.1145/1830483.1830672, URL: https://www.researchgate.net/publication/220742154.

[18] C. Lemnaru, M. Cuibus, A. Bona, A. Alic, R. Potolea, A distributed methodology for imbalanced classification problems, in: Proceedings - 2012 11th International Symposium on Parallel and Distributed Computing, ISPDC 2012, 2012, pp. 164–171, http://dx.doi.org/10.1109/ISPDC.2012.30, URL: https://ieeexplore.ieee.org/abstract/document/6341508/.

[19] NVIDIA, Cuda C Programming Guide, Technical Report September, 2015, pp. 1–261, URL: www.nvidia.com.

[20] W. Hwu, Programming massively parallel processors, 2017, Edition, D Kirk - Special and undefined 2009. http://dx.doi.org/10.1016/c2015-0-02431-5. URL: http://www.academia.edu/download/30841796/10.1.1.188.691.pdf{#}page=108.
[21] W.M.W. Hwu, GPU Computing Gems Jade Edition, 2012, http://dx.doi.org/10.1016/C2010-0-68654-8, URL: https://books.google.com/books?hl=en{&}lr={&}id=LsNVFUnzcVMC{&}oi=fnd{&}pg=PP1{&}dq=gpu-computing-gems-jade-edition{&}ots=v-ld0QuImB{&}sig=Uzx{_}Kr8r72lyP1DEO9z182TPlll.
[22] W.M.W. Hwu, GPU Computing Gems Emerald Edition, 2011, http://dx.doi.org/10.1016/C2010-0-65709-9.
[23] M. Harris, How to overlap data transfers in CUDA C/C++, Nvidia (2015) 1, URL: http://devblogs.nvidia.com/parallelforall/how-overlap-data-transfers-cuda-cc/.
[24] M.V. Fidelis, H.S. Lopes, A.A. Freitas, Discovering comprehensible classification rules with a genetic algorithm, in: Proceedings of the 2000 Congress on Evolutionary Computation, CEC 2000, vol. 1, IEEE, 2000, pp. 805–810, http://dx.doi.org/10.1109/CEC.2000.870381, https://doi.org/10.1109/CEC.2000.870381.
[25] K.K. Gündoğan, B. Alataş, A. Karci, Mining classification rules by using genetic algorithms with non-random initial population and uniform operator, Turk. J. Electr. Eng. Comput. Sci. 12 (1) (2004) 43–52, URL: https://journals.tubitak.gov.tr/elektrik/abstract.htm?id=6712.
[26] J. Vashishtha, D. Kumar, S. Ratnoo, K. Kundu, Mining comprehensible and interesting rules: A genetic algorithm approach, Int. J. Comput. Appl. 31 (1) (2011) 39–47, http://dx.doi.org/10.5120/3792-5221, URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.259.4592{&}rep=rep1{&}type=pdf.
[27] X.J. Shi, H. Lei, A genetic algorithm-based approach for classification rule discovery, in: Proceedings of the International Conference on Information Management, Innovation Management and Industrial Engineering, ICIII 2008, vol. 1, 2008, pp. 175–178, http://dx.doi.org/10.1109/ICIII.2008.289, URL: https://ieeexplore.ieee.org/abstract/document/4737521/.
[28] T. Shobha, R.J. Anandhi, Classification rule discovery using variant genetic algorithm, in: 2nd International Conference on Circuits, Controls, and Communications, CCUBE 2017 - Proceedings, 2018, pp. 222–225, http://dx.doi.org/10.1109/CCUBE.2017.8394151, URL: https://ieeexplore.ieee.org/abstract/document/8394151/.
[29] M. Hassoon, M.S. Kouhi, M. Zomorodi-Moghadam, M. Abdar, Rule optimization of boosted C5.0 classification using genetic algorithm for liver disease prediction, in: 2017 International Conference on Computer and Applications, ICCA 2017, 2017, pp. 299–305, http://dx.doi.org/10.1109/COMAPP.2017.8079783, URL: https://ieeexplore.ieee.org/abstract/document/8079783/.
[30] A.H. Alkeshuosh, M.Z. Moghadam, I. Al Mansoori, M. Abdar, Using PSO algorithm for producing best rules in diagnosis of heart disease, in: 2017 International Conference on Computer and Applications, ICCA, IEEE, 2017, pp. 306–311.
[31] P. Goyal, Genetic algorithms for classification rule discovery : Issues and challenges, Int. J. Adv. Res. Comput. Commun. Eng. 5 (6) (2016) 514–518, http://dx.doi.org/10.17148/IJARCCE.2016.56110.
[32] S.M. Saif, H. Shah-Hosseini, M.R. Feizi, Empire establishment algorithm, in: ICCTD 2009 - 2009 International Conference on Computer Technology and Development, vol. 1, 2009, pp. 295–299, http://dx.doi.org/10.1109/ICCTD.2009.71, URL: https://ieeexplore.ieee.org/abstract/document/5359676/.
[33] B. M. Al-Maqaleh, H. Shahbazkia, A genetic algorithm for discovering classification rules in data mining, Int. J. Comput. Appl. 41 (18) (2012) 40–44, http://dx.doi.org/10.5120/5644-8072, URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.259.1812{&}rep=rep1{&}type=pdf.
[34] T. Shobha, R.J. Anandhi, Adaptive strategy operators based GA for rule discovery, Int. J. Inf. Technol. (2019) http://dx.doi.org/10.1007/s41870-019-00303-z.
[35] M. Abdar, V.N. Wijayaningrum, S. Hussain, R. Alizadehsani, P. Plawiak, U.R. Acharya, V. Makarenkov, IAPSO-AIRS: A novel improved machine learning-based system for wart disease treatment, J. Med. Syst. 43 (7) (2019) http://dx.doi.org/10.1007/s10916-019-1343-0, URL: https://www.researchgate.net/publication/333023902.
[36] R. Alizadehsani, M. Abdar, M. Roshanzamir, A. Khosravi, P.M. Kebria, F. Khozeimeh, S. Nahavandi, N. Sarrafzadegan, U.R. Acharya, Machine learning-based coronary artery disease diagnosis: A comprehensive review, Comput. Biol. Med. 111 (2019) http://dx.doi.org/10.1016/j.compbiomed.2019.103346, https://doi.org/10.1016/j.compbiomed.2019.103346.
[37] Z. Sherkatghanad, M. Akhondzadeh, S. Salari, M. Zomorodi-Moghadam, M. Abdar, U.R. Acharya, R. Khosrowabadi, V. Salari, Automated detection of autism spectrum disorder using a convolutional neural network, Front. Neurosci. 13 (2020) http://dx.doi.org/10.3389/fnins.2019.01325, URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6971220/.
[38] E. Nasarian, M. Abdar, M.A. Fahami, R. Alizadehsani, S. Hussain, M.E. Basiri, M. Zomorodi-Moghadam, X. Zhou, P. Pławiak, U.R. Acharya, R.-S. Tan, N. Sarrafzadegan, Association between work-related features and coronary artery disease: a heterogeneous hybrid feature selection integrated with balancing approach, Pattern Recognit. Lett. (2020) http://dx.doi.org/10.1016/j.patrec.2020.02.010, URL: https://www.sciencedirect.com/science/article/pii/S0167865520300507.
[39] J.A. ALzubi, B. Bharathikannan, S. Tanwar, R. Manikandan, A. Khanna, C. Thaventhiran, Boosted neural network ensemble classification for lung cancer disease diagnosis, Appl. Soft Comput. 80 (2019) 579–591.
[40] M. Abdar, U.R. Acharya, N. Sarrafzadegan, V. Makarenkov, NE-Nu-SVC: A new nested ensemble clinical decision support system for effective diagnosis of coronary artery disease, IEEE Access 7 (2019) 167605–167620.
[41] M.A. Lones, S.L. Smith, J.E. Alty, S.E. Lacy, K.L. Possin, D.R. Jamieson, A.M. Tyrrell, Evolving classifiers to recognize the movement characteristics of Parkinson's disease patients, IEEE Trans. Evol. Comput. 18 (4) (2014) 559–576, http://dx.doi.org/10.1109/TEVC.2013.2281532, URL: http://ieeexplore.ieee.org.
[42] J. Quevedo, M. Abdelatti, F. Imani, M. Sodhi, Using reinforcement learning for tuning genetic algorithms, 2021.
[43] K. Heraguemi, H. Kadri, A. Zabi, Whale optimization algorithm for solving association rule mining issue, Int. J. Comput. Digit. Syst. 10 (1) (2021) 333–342.
[44] A. Cano, A. Zafra, S. Ventura, A parallel genetic programming algorithm for classification, in: Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 6678 LNAI, 2011, pp. 172–181, http://dx.doi.org/10.1007/978-3-642-21219-2_23, URL: https://www.researchgate.net/publication/221053488.
[45] F.M. Johar, F.A. Azmin, M.K. Suaidi, A.S. Shibghatullah, B.H. Ahmad, S.N. Salleh, M.Z.A. Abd Aziz, M.M. Shukor, A review of genetic algorithms and parallel genetic algorithms on graphics processing unit (GPU), in: Proceedings - 2013 IEEE International Conference on Control System, Computing and Engineering, ICCSCE 2013, 2013, pp. 264–269, http://dx.doi.org/10.1109/ICCSCE.2013.6719971, URL: https://www.researchgate.net/publication/279205302.
[46] L. Zheng, Y. Lu, M. Ding, Y. Shen, M. Guoz, S. Guo, Architecture-based performance evaluation of genetic algorithms on multi/many-core systems, in: Proc. - 14th IEEE Int. Conf. on Computational Science and Engineering, CSE 2011 and 11th Int. Symp. on Pervasive Systems, Algorithms, and Networks, I-SPA 2011 and 10th IEEE Int. Conf. on IUCC 2011, 2011, pp. 321–334, http://dx.doi.org/10.1109/CSE.2011.65, URL: https://ieeexplore.ieee.org/abstract/document/6062894/.
[47] S.R. Dash, S. Dehuri, S. Rayaguru, Discovering Interesting Rules from Biological Data using Parallel Genetic Algorithm, 2013, pp. 631–636, http://dx.doi.org/10.1109/IAdCC.2013.6514300.
[48] P. Pospichal, J. Jaros, J. Schwarz, Parallel Genetic Algorithm on the CUDA Architecture, 2010, pp. 442–451, http://dx.doi.org/10.1007/978-3-642-12239-2_46.
[49] A. Cano, J.M. Luna, S. Ventura, High performance evaluation of evolutionary-mined association rules on GPUs, J. Supercomput. 66 (3) (2013) 1438–1461, http://dx.doi.org/10.1007/s11227-013-0937-4.
[50] M. Rodríguez, D.M. Escalante, A. Peregrín, Efficient distributed genetic algorithm for rule extraction, Appl. Soft Comput. 11 (1) (2011) 733–743, http://dx.doi.org/10.1016/j.asoc.2009.12.035.
[51] P. Sharma, S. Saroj, Discovery of classification rules using distributed genetic algorithm, Procedia Comput. Sci. 46 (2015) 276–284, http://dx.doi.org/10.1016/j.procs.2015.02.021, www.sciencedirect.com.
[52] M.J. del Jesus, J.A. Gamez, P. Gonzalez, J.M. Puerta, On the discovery of association rules by means of evolutionary algorithms, Wiley Interdiscip. Rev.: Data Min. Knowl. Discov. 1 (5) (2011) 397–415.
[53] A. Cano, A. Zafra, S. Ventura, Solving classification problems using genetic programming algorithms on GPUs, in: Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 6077 LNAI, (PART 2) 2010, pp. 17–26, http://dx.doi.org/10.1007/978-3-642-13803-4_3.
[54] A. Cano, A. Zafra, S. Ventura, Parallel evaluation of pittsburgh rule-based classifiers on GPUs, Neurocomputing 126 (2014) 45–57, http://dx.doi.org/10.1016/j.neucom.2013.01.049.
[55] M. Harris, S. Sengupta, J.D. Owens, Parallel prefix sum (scan) with CUDA, GPU Gems 3 (39) (2007) 851–876.
[56] D. Dua, C. Graff, UCI Machine Learning Repository, University of California, Irvine, School of Information and Computer Sciences, 2019, URL: http://archive.ics.uci.edu/ml/datasets/HCV+data.
[57] D. Dua, C. Graff, UCI Machine Learning Repository, University of California, Irvine, School of Information and Computer Sciences, 2017, URL: https://archive.ics.uci.edu/ml/datasets/Poker+Hand.
[58] SRK, Novel Corona Virus 2019 Dataset, 2020, https://www.kaggle.com/sudalairajkumar/novel-corona-virus-2019-dataset. (Accessed 20 March 2020).
[59] M. Beheshti Roui, M. Sarvelayati, Novel corona virus 2019 dataset, 2020, https://www.kaggle.com/msarvelayati/modified-covid19-open-line-list. (Accessed 20 March 2020).
[60] M. Beheshti Roui, M. Sarvelayati, GitHub - 2b2ak/CUDA_PGA: Article: A Novel Approach based on Genetic Algorithm to Speed up the Discovery of Classification Rules on GPUs, URL: https://github.com/2b2ak/CUDA{_}PGA.

[61] P. Micikevicius, Analysis-driven optimization, in: GPU Technology Conference, 2010, pp. 1–55.
[62] B. Li, Modeling and Runtime Systems for Coordinated Power-Performance Management (Ph.D. thesis), Virginia Tech, 2019.
[63] S. Hashem, G. Esmat, W. Elakel, S. Habashy, S. Abdel Raouf, S. Darweesh, M. Soliman, M. Elhefnawi, M. El-Adawy, M. ElHefnawi, Accurate prediction of advanced liver fibrosis using the decision tree learning algorithm in chronic hepatitis C Egyptian patients, Gastroenterol. Res. Pract. 2016 (2016).
[64] C. Za'in, M. Pratama, E. Pardede, Evolving large-scale data stream analytics based on scalable PANFIS, Knowl.-Based Syst. 166 (2019) 186–197.
[65] P. Sharma, S. Ratnoo, Bottom-up pittsburgh approach for discovery of classification rules, in: 2014 International Conference on Contemporary Computing and Informatics, IC3I, IEEE, 2014, pp. 31–37.
[66] S. Jabin, Poker hand classification, in: 2016 International Conference on Computing, Communication and Automation, ICCCA, IEEE, 2016, pp. 269–273.
[67] D. Dua, C. Graff, UCI Machine Learning Repository, University of California, Irvine, School of Information and Computer Sciences, 2017, URL: https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original).
[68] D. Bui-Thi, P. Meysman, K. Laukens, MoMAC: Multi-objective optimization to combine multiple association rules into an interpretable classification, Appl. Intell. (2021) 1–13.
[69] M. Harris, et al., Optimizing parallel reduction in CUDA, Nvidia Dev. Technol. 2 (4) (2007) 1–39.