Accelerating GPU-based Evolutionary Induction of Decision Trees - Fitness Evaluation Reuse

Krzysztof Jurczuk(✉), Marcin Czajkowski, and Marek Kretowski

Faculty of Computer Science, Bialystok University of Technology, Wiejska 45a, 15-351 Bialystok, Poland
{k.jurczuk,m.czajkowski,m.kretowski}@pb.edu.pl

Abstract. The rapid development of new technologies and parallel frameworks is a chance to overcome barriers of slow evolutionary induction of decision trees (DTs). This global approach, that searches for the tree structure and tests simultaneously, is an emerging alternative to greedy top-down solutions. However, in order to be efficiently applied to big data mining, both technological and algorithmic possibilities need to be fully exploited. This paper shows how by reusing information from previously evaluated individuals, we can accelerate GPU-based evolutionary induction of DTs on large-scale datasets even further. Noting that some of the trees or their parts may reappear during the evolutionary search, we have created a so-called repository of trees (split between GPU and CPU). Experimental evaluation is carried out on the existing Global Decision Tree system where the fitness calculations are delegated to the GPU, while the core evolution is run sequentially on the CPU. Results demonstrate that reusing information about trees from the repository (classification errors, objects' locations, etc.) can accelerate the original GPU-based solution. It is especially visible on large-scale data where the cost of the trees evaluation exceeds the cost of storing and exploring the repository.

Keywords: Evolutionary algorithms · Decision trees · Big data mining · Graphics processing unit (GPU) · CUDA

1 Introduction

Decision trees (DTs) [9] are one of the most useful supervised learning methods for classification. During over 50 years of their applications [12], they were mainly induced using greedy heuristics like a top-down approach [16]. Involving evolutionary algorithms (EAs) into the trees induction [2] was a breath of fresh air. However, evolutionary tree induction is much more computationally demanding, and there were raised many questions of the time efficiency of such an approach. Nevertheless, evolutionary induced DTs have managed to gain in popularity for less demanding data mining problems for which the generation time of the prediction model was not crucial. Their main advantage is the global approach in which a tree structure, tests in internal nodes and predictions in leaves are searched simultaneously [10]. As a result, the generated trees are significantly simpler and at least as accurate as the greedy alternatives that are based on the classical divide and conquer schema.

Rapid development of new computational technologies and parallel frameworks is a chance to conquer the barriers concerning slow evolutionary DT induction [7]. To make it work, proposed new solutions need to be efficient for large-scale data and run on relatively cheap and generally available hardware/software. For these reasons, we have concentrated on using graphics processing units (GPU)s to parallelize and speed up the induction.

This paper focuses on accelerating a GPU-based evolutionary induction of classification trees [8] for the large-scale data. In the approach, the main evolutionary loop (selection, genetic operators, etc.) is performed sequentially on a CPU, while the most time-consuming operations like fitness calculation are delegated to a GPU. Noting that some of the trees or their parts may reappear during the evolutionary search, we examine if it is worth to store and reuse this information. In this paper, we introduce a concept of a repository of previously evaluated DTs in order to limit fitness recalculation of the new ones founded by EA. The search of the same (or similar) individuals is performed fully on the GPU where they are stored as a part of the repository. The second part of the repository, which gathers the corresponding fitness results, is located on the CPU in order to limit CPU/GPU memory transfers. The reuse strategy is not new in EA; however, it was studied in the context of improving the genetic diversity [3] (e.g., by chromosome revisiting) and it was not applied to DTs induction.

The next section provides a brief background on DTs and the Global Decision Tree (GDT) system which is used to test the proposed solution. Section 3 presents our approach, and Sect. 4 shows its experimental validation. In the last section, the paper is concluded and possible future work is outlined.

2 Evolutionary Induction of Decision Trees

Decision trees [9] (DTs) have a knowledge representation structure that is built of nodes and branches, where each internal node holds a test on one or more attributes; each branch represents the outcome of a test; and each leaf (terminal node) holds a prediction. Such a hierarchical tree structure, where appropriate tests from consecutive nodes are sequentially applied, closely resembles the human way of making decisions. The success of tree-based approaches can be explained by their ease of application, fast operation, and effectiveness. Nevertheless, the execution times and resource utilization still require improvement to meet ever-growing computational demands.

Evolutionary algorithms [14] (EAs) belong to a family of meta-heuristic methods. They represent techniques for solving a wide variety of difficult optimization problems. The framework of EA is inspired by biological mechanisms of evolution. The algorithm operates on individuals that compose a current population. Each individual represents a candidate solution to the target problem.

Individuals are assessed using a fitness function that measures their performance. Next, individuals with higher fitness usually have a higher probability of being selected for reproduction. Genetic operators such as mutation and crossover influence individuals, thereby producing new offspring(s). This guided random search (the offspring usually inherits some traits from its ancestors) is stopped when some convergence criteria is satisfied.

Typically, DTs are induced by a top-down approach [16] that realises the classical divide and conquer schema. The main consequences of locally optimal choices made in each tree node during the induction are overgrown and often less stable DT classifiers. Emerging alternatives to the top-down solutions include primarily EAs. Their global approach limits the negative effects of locally optimal decisions but it is much more computationally demanding [2,6].

2.1 Parallelization of DTs Induction

Fortunately, EAs are naturally prone to parallelism and the artificial evolution can be implemented in various ways [4]. There are at least three main strategies that have been studied to parallelize EAs: master-slave model, island model and cellular model. Since EAs work on a set of independent solutions, it is relatively easy to distribute the computational load among multiple processors through a data or/and population decomposition approach. Recent research on the parallelization of various evolutionary computation methods has seemed to focus on GPUs as the implementation platform [10,18]. The popularity of GPUs results from their general availability, relatively low cost and high computational power.

In the context of parallelization of DTs induction, not enough has been done and the GPGPU topic has hardly been studied in the literature at all. There are several GPU-based parallelizations but they consider either greedy inducers [11] or random forests [13]. As for evolutionary induced DTs, we found only a few papers that cover parallel extensions of the Global Decision Tree (GDT) data mining system: using MPI/OpenMP [5], GPU [8] and Spark [15]. One of the possible reasons why this topic has not yet been adequately explored by other systems is that the straightforward EA parallelization of DT induction may be insufficient. In order to achieve high speedup and exploit the full potential of e.g., GPU parallelization, there is a strong need to incorporate knowledge about DT specificity and its evolutionary induction.

2.2 Global Decision Tree System

This paper uses the GDT system [10] which supports various decision trees, and it has been already studied in terms of parallelism. Another benefit of the GDT system is that its scheme follows a typical EA framework [14] with an unstructured, fixed size population and a generational selection. In this study, we have deliberately limited the GDT description to a univariate binary classification tree version to facilitate understanding and to eliminate less important details.

Individuals in the population are not specially encoded and are represented and processed in their actual form as univariate classification trees. This is the most popular and basic variant of DTs that uses typical inequality tests with two outcomes in the internal nodes. Initialization is performed with a simplified top-down manner, which is applied to randomly selected small sub-samples of the learning data. Tests in the internal nodes are created by randomly selecting two objects from different classes (mixed dipole) that are located in the considered node [10]. Then, an effective test that separates these two objects into subtrees is randomly created, considering attributes only with different feature values.

In order to preserve population diversity, the GDT system applies two specialized genetic meta-operators corresponding to the classical mutation and crossover. For both operators framework provides several variants [10] that influence the tree structure and the splitting tests in the internal nodes, e.g.: *(i)* replace subtree/branch/node/test between two affected individuals; *(ii)* prune the internal node into the leaf; *(iii)* modify the test in internal nodes (shift threshold); *(iv)* replace existing test with a new one created on a randomly chosen dipole.

Each time, the selection of the variant and affected node (or nodes) are random. The probability distributions are different for nodes and for variants. For nodes, the location in the tree is taken into account (modification of nodes from the upper levels results in more global changes) and the quality of the subtree (less accurate nodes should be modified more frequently).

Direct minimization of the prediction error measured on the learning dataset usually leads to the over-fitting problem, especially in the case of DTs. In typical top-down induction, this is partially mitigated by defining a stopping condition and post-pruning. In the case of an evolutionary approach, the multi-objective function is required to minimize the prediction error and the tree complexity at the same time. As, in this paper, only univariate classification trees are considered, the simple weighted form of the fitness function is used:

$$Fitness(T) = Accuracy(T) - \alpha * (Size(T) - 1.0), \tag{1}$$

where $Accuracy(T)$ represents the classification quality of the tree T estimated on the learning set, $Size(T)$ is the number of nodes in T and α is a user-defined parameter that reflects the relative importance of the complexity term.

The selection mechanism is based on the linear ranking selection [14]. Additionally, in each iteration one individual with the highest fitness in the current population is copied to the next one (elitist strategy). The evolution ends when the fitness of the best individual in the population does not improve during the fixed number of generations or the maximum number of generations is reached.

3 Repository-Supported GPU-based Approach

The general flowchart of the proposed solution is illustrated in Fig. 1. The following operations: initialization of the population as well as selection of the individuals/trees remain unchanged compared to the original GDT system. They are

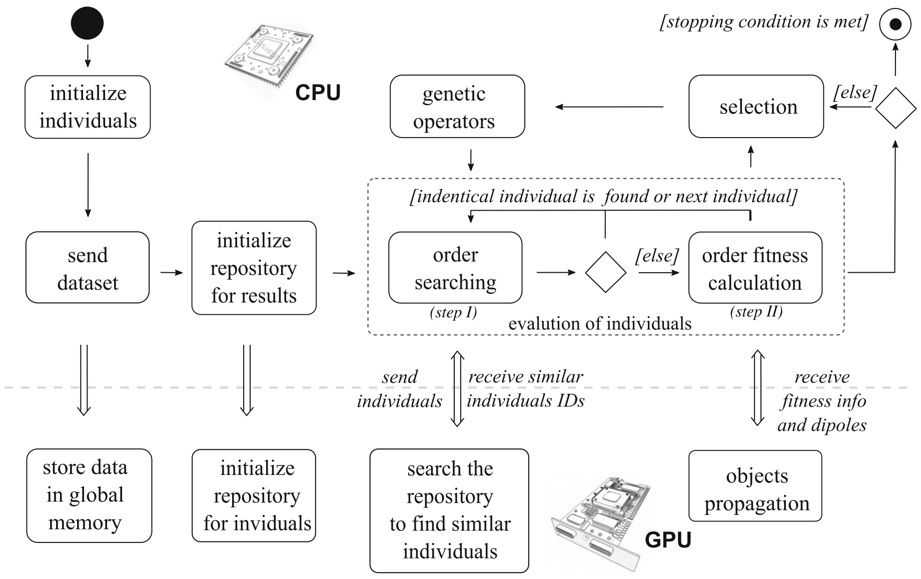

[stopping condition is met]
initialize individuals
CPU
genetic operators
selection
[else]
[indentical individual is found or next individual]
send dataset
initialize repository for results
order searching
(step I)
[else]
order fitness calculation
(step II)
evalution of individuals
send individuals
receive similar individuals IDs
receive fitness info and dipoles
store data in global memory
initialize repository for inviduals
search the repository to find similar individuals
GPU
objects propagation

Fig. 1. Flowchart of a GPU-based approach supported by the repository of individuals.

run in a sequential manner on a CPU since they are relatively fast. In addition, the initial population is created only once on a small fraction of the dataset. The GPU is called after each successful application of the crossover/mutation operator, when there is a need to evaluate the individuals. It is the most time-consuming operation because all dataset objects have to be propagated from the tree root to one of the leaves. Both the dataset size and individual size influence the computational requirements. This part of the algorithm is isolated and performed in parallel on a GPU. This way, the parallelization does not affect the behavior of the original EA.

In the initialization phase (see Fig. 1), the whole dataset is sent to the GPU. This CPU to GPU transfer is done only once and the data is saved in the allocated space in the global memory. This way, all objects of the dataset are accessible for all threads at any time. In addition, the repository of the individual is created. A fixed amount of memory is allocated both on the GPU and CPU. The size of the repository is set before the evolution. Initially, the repository is empty. On the GPU, the repository is devoted to store the structure of DTs (tree nodes, branches, tests, etc.). The part of repository located on the CPU is responsible for keeping fitness results and dipoles of the trees stored on the GPU. The splitting of the repository location between CPU and GPU allows us to avoid unnecessary CPU/GPU memory transfers.

In the evolutionary loop, each time, when the genetic operator is successfully applied, the GPU is asked to help the CPU. First, the modified individual/tree is sent to the GPU. Then, the GPU searches in the repository for a similar individual (see Fig. 1, step I). If such an individual is found, the GPU returns

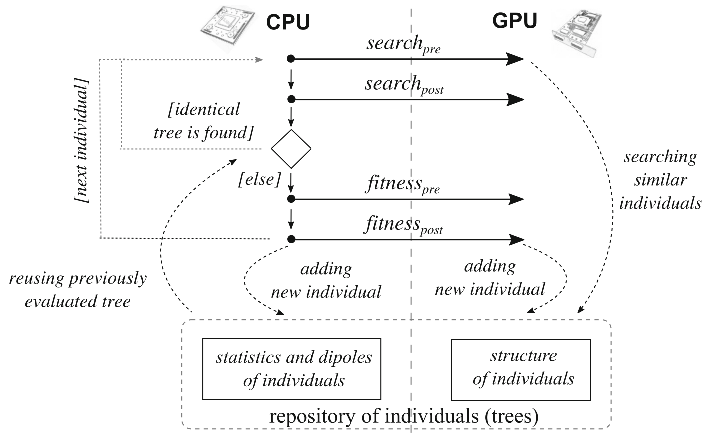

CPU
GPU
$search_{pre}$
$search_{post}$
[identical tree is found]
[else]
$fitness_{pre}$
$fitness_{post}$
[next individual]
searching similar individuals
reusing previously evaluated tree
adding new individual
adding new individual
statistics and dipoles of individuals
structure of individuals
repository of individuals (trees)

Fig. 2. GPU kernels arrangement during the evaluation of individuals.

its identifier in the repository and the level of similarity. This information is used by the CPU to get previously calculated (individual's) statistics from its part of the repository. If the same individual is found, then the CPU has only to copy previously calculated statistics and dipoles from the repository to the currently evaluated tree. If a similar individual is found, then information from the matched tree part is used, while for the remaining part of the tree, the GPU is called to calculate the missing information. In the current version of the algorithm, when the trees aren't the same, two types of similarity are considered. The first case concerns the situation when the tree root and its left subtree are the same as in the tree from the repository. The second case is analogical but it refers to the right subtree of the root.

To find a similar individual, two kernel functions are called (see Fig. 2). The first kernel ($search_{pre}$) is used to compare the evaluated indvidual with all individuals from the repository. A two-level decomposition is applied. Each GPU block processes one individual from the repository. Threads inside blocks are responsible for comparing various nodes inside a tree. On the GPU, trees are represented as one-dimensional arrays where the position of the left and right child of the i-th node equals $(2 * i + 1)$ and $(2 * i + 2)$, respectively [8]. Thanks to this, the comparison of the trees consists in checking corresponding array elements. Additionally, for two types of similar trees, one-dimensional arrays (so-called maps) are prepared to know which tree nodes have to be checked. Finally, this kernel provides the level of similarity for each tree in the repository.

The second kernel ($search_{post}$) goes through the results from the first kernel and provides the identifiers of similar individuals in the repository and their level of similarity. If no individual is found, -1 identifier is returned. The second kernel uses only one block to synchronize the results merging effectively. Each GPU thread is responsible for a bunch of individuals from the repository.

In case of not finding similar individuals in the repository, the GPU is called to calculate the required information as well as to search dipoles (see Fig. 1, part II) [8]. Two kernel functions are used (see Fig. 2). The first kernel ($fitness_{pre}$) is called to propagate all objects in the training dataset from the tree root to appropriate leaves. It uses the data decomposition strategy. At first, the whole dataset is spread into smaller parts over GPU blocks. Next, in each block, the assigned objects are further spread over the threads. Each GPU block makes a copy of the evaluated individual that is loaded into the shared memory. This way, the threads process the same individual in parallel but handle different chunks of the dataset. At the end of the kernel, in each tree leaf, the number of objects of each class that reach that particular leaf is stored.

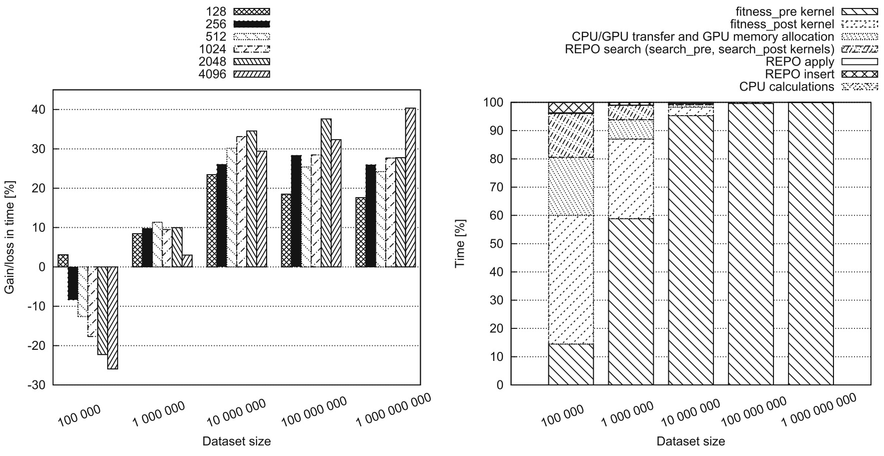
The second kernel function ($fitness_{post}$) merges information from multiple copies of the individual allocated in each GPU block. This operation sums up the counters from copies of the individual, and the total number of objects of each class in each tree leaf is obtained. Finally, reclassification errors in each leaf are calculated. Then, all gathered information is propagated from the leaves towards the root. The obtained tree statistics (like coverage, errors) as well as dipoles are sent back to the CPU in order to finish the individual evaluation.

Each time when the GPU calculates new fitness results, the repository is updated. On the CPU, the obtained tree statistics and the dipoles are added to the repository. As regards the GPU, the corresponding tree structures are stored there. In the current implementation, when a new tree is added to the repository, it replaces the oldest one if the repository is full.

4 Experimental Validation

Experimental validation was performed on an artificially generated dataset called *chess3x3*. This dataset represents a classification problem with two classes, two real-values attributes and objects arranged on a 3×3 chessboard [10]. We used the synthetic dataset to scale it freely, unlike real-life datasets. All presented results correspond to averages of 5–10 runs and were obtained with a default set of parameters from the original GDT system. As we are focused in this paper only on size and time performance, results for the classification accuracy are not included. However, for the tested dataset variants, the GDT system managed to induce trees with optimal structures and accuracy over 99%.

Experiments were performed on a server equipped with 2 eight-core processors Intel Xeon E5-2620 v4 (20 MB Cache, 2.10 GHz), 256 GB RAM and running 64-bit Ubuntu Linux 16.04.02 LTS. We tested NVIDIA Tesla P100 GPU card (3 584 CUDA cores and 12 GB of memory). The original GDT system was implemented in C++ and compiled with the use of gcc version 5.4.2. The GPU-based parallelization was implemented in CUDA-C [17] and compiled by nvcc CUDA 8.0 [1] (single-precision arithmetic was applied).



(a) (b)

Fig. 3. Performance of the accelerated approach: (a) influence of the repository size (from 128 to 4096 trees) on the reduction of induction time, (b) detailed time-sharing information (mean time as a percentage) for repository size of 512 individuals.

4.1 Results

Figure 3(a) shows how much the reusing mechanism accelerates the evolutionary induction. The influence of both the dataset size and the repository size is presented. It is clearly visible that the gain in time is more prominent when the dataset size increases. For more than 10 millions of objects, the repository-supported version is about 30% faster than the original GPU-acceleration. An optimal number of trees stored in the repository grows when larger data is processed (for 100 000 objects 128 trees, for 1 000 000 objects 512 trees, etc.)

For smaller datasets (e.g., 100 000 objects), the induction time is even increased if too many individuals are held in the repository. When the number of trees exceeds 256, the loss in time is observed. It is caused by a repository overhead, which is illustrated in Fig. 3(b). Searching in the repository is mainly blamed for time efficiency drop ($REPO_{search}$ is the time that the algorithm spent on searching in the repository). The time spent on other repository operations, like inserting new individuals ($REPO_{insert}$) and reusing stored data ($REPO_{apply}$), is negligible and does not depend on the repository size. For the smallest dataset, the repository search overhead is not compensated by the time profit resulting from the reusage. When the number of objects increases, then the evaluation of individuals is more time demanding and the repository overhead becomes negligible.

In Table 1, we present mean execution times of all tested implementations. For the GPU-based version with the reusing mechanism, the optimal repository size is considered. The results show that the gain in time is more important when large-scale data is processed. For 1 billion of objects, the proposed solution

Table 1. Mean execution times of the repository-supported (GPU with REPO), GPU, OpenMP and sequential versions of the algorithm (in seconds, for larger datasets also time in hours is included).

Dataset size	GPU with REPO		GPU		OpenMP	Sequential
100 000	21.1		21.8		100.2	685
1 000 000	55.3		62.7		3 605.7	23 536
10 000 000	449.9		666.7		47 600.4	324 000
100 000 000	5 428.4	(1.5 h)	7 471.3	(2 h)	weeks	months
1 000 000 000	59 404.2	(16.5 h)	84 245.2	(23.5 h)	months	over a year

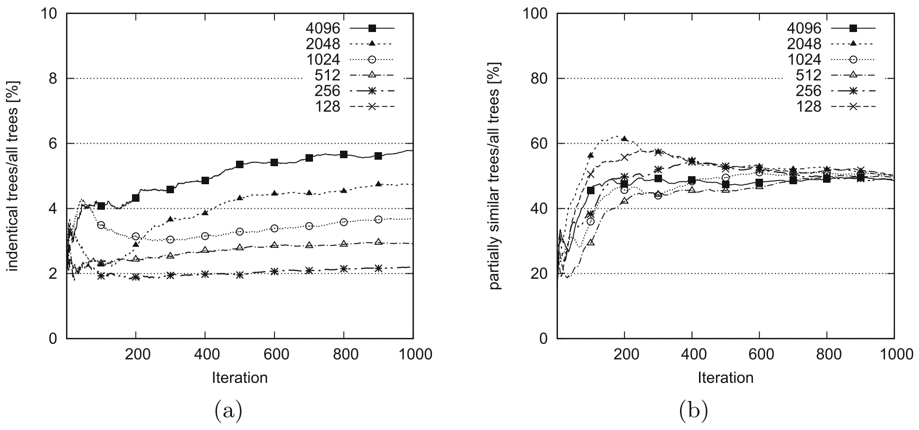


Fig. 4. Searching success ratio for different repository size (from 128 to 4096 individuals): (a) identical tree, (b) similar trees (root and its left or right child).

decreases the induction time by ≈7 h (it is 40% faster). It was estimated that the sequential GDT system would need over a year to calculate such a dataset and the OpenMP parallelization [5] would decrease this time to a few months.

We have also examined the searching success ratio (how frequently the search in the repository ends with a success). Figure 4(a) shows that an identical tree is found in a few percent of cases (from 2% to 6%). Moreover, we observe that this ratio grows with the larger repository. However, we should remind that the repository size influences the search overhead which in the case of smaller datasets could be significant (as it is shown in Fig. 3). As regards partial similarity, the search ends with success much more often (in about 50% of trials) and it is not dependent on the repository size. The results suggest that similar trees contribute more to obtain an acceleration.

5 Conclusion

In this paper, we extend the GPU-based approach for evolutionary induction of decision trees. The concept of fitness evaluation reusage is proposed. A so-called repository of individuals is used to store previously considered trees. When a new individual is evaluated, first, a similar tree to reuse is searched in the repository. If it is found, the results are read from the repository. Otherwise, the GPU has to perform required calculations. The results show that the proposed strategy is able to speed up the induction even further, especially on large-scale data.

This research is a first step in building a so-called 'multi-tree' strategy. It assumes that similar individuals are corepresented by partially sharing fragments/structures in memory. Such a strategy may allow us to observe the evolution dynamics in detail, e.g., to follow diversity at each level of a decision tree. At the same time, it can also speed up further the evolutionary induction of decision trees. Other future works include the concept of searching even deeper resemblance in the repository as well as a multi-GPU approach.

Acknowledgments. This work was supported by the grant S/WI/2/18 from Bialystok University of Technology founded by Polish Ministry of Science and Higher Education.

References

1. NVIDIA Developer Zone - CUDA Toolkit Documentation (2019). https://docs.nvidia.com/cuda/cuda-c-programming-guide/
2. Barros, R.C., Basgalupp, M.P., De Carvalho, A.C., Freitas, A.A.: A survey of evolutionary algorithms for decision-tree induction. IEEE Trans. SMC, Part C **42**(3), 291–312 (2012)
3. Charalampakis, A.E.: Registrar: a complete-memory operator to enhance performance of genetic algorithms. J. Glob. Optim. **54**(3), 449–483 (2012)
4. Chitty, D.M.: Fast parallel genetic programming: multi-core CPU versus many-core GPU. Soft Comput. **16**(10), 1795–1814 (2012)
5. Czajkowski, M., Jurczuk, K., Kretowski, M.: A parallel approach for evolutionary induced decision trees. MPI+OpenMP implementation. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2015, Part I. LNCS (LNAI), vol. 9119, pp. 340–349. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19324-3_31
6. Czajkowski, M., Kretowski, M.: Evolutionary induction of global model trees with specialized operators and memetic extensions. Inf. Sci. **288**, 153–173 (2014)
7. Franco, M.A., Bacardit, J.: Large-scale experimental evaluation of GPU strategies for evolutionary machine learning. Inf. Sci. **330**(C), 385–402 (2016)
8. Jurczuk, K., Czajkowski, M., Kretowski, M.: Evolutionary induction of a decision tree for large-scale data: a GPU-based approach. Soft Comput. **21**(24), 7363–7379 (2017)
9. Kotsiantis, S.B.: Decision trees: a recent overview. Artif. Intell. Rev. **39**(4), 261–283 (2013)
10. Kretowski, M.: Evolutionary Decision Trees in Large-Scale Data Mining. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21851-5

11. Lo, W.T., Chang, Y.S., Sheu, R.K., Chiu, C.C., Yuan, S.M.: CUDT: A CUDA based decision tree algorithm. Sci. World J. (2014)
12. Loh, W.Y.: Fifty years of classification and regression trees. Int. Stat. Rev. **82**(3), 329–348 (2014)
13. Marron, D., Bifet, A., Morales, G.D.F.: Random forests of very fast decision trees on GPU for mining evolving big data streams. In: Proceedings of the Twenty-First European Conference on Artificial Intelligence, ECAI 2014, pp. 615–620 (2014)
14. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs, 3rd edn. Springer, Heidelberg (1996). https://doi.org/10.1007/978-3-662-03315-9
15. Reska, D., Jurczuk, K., Kretowski, M.: Evolutionary induction of classification trees on spark. In: Rutkowski, L., Scherer, R., Korytkowski, M., Pedrycz, W., Tadeusiewicz, R., Zurada, J.M. (eds.) ICAISC 2018, Part I. LNCS (LNAI), vol. 10841, pp. 514–523. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91253-0_48
16. Rokach, L., Maimon, O.: Top-down induction of decision trees classifiers - a survey. IEEE Trans. Syst. Man Cybern. Part C (Appl. Rev.) **35**(4), 476–487 (2005)
17. Storti, D., Yurtoglu, M.: CUDA for Engineers : An Introduction to High-Performance Parallel Computing. Addison-Wesley, New York (2016)
18. Tsutsui, S., Collet, P. (eds.): Massively Parallel Evolutionary Computation on GPGPUs. Natural Computing Series. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37959-8