Global Induction of Decision Trees: From Parallel Implementation to Distributed Evolution

Marek Krętowski and Piotr Popczyński

Faculty of Computer Science, Białystok Technical University
Wiejska 45a, 15-351 Białystok, Poland
mkret@wi.pb.edu.pl

Abstract. In most of data mining systems decision trees are induced in a top-down manner. This greedy method is fast but can fail for certain classification problems. As an alternative a global approach based on evolutionary algorithms (EAs) can be applied. We developed *Global Decision Tree* (GDT) system, which learns a tree structure and tests in one run of the EA. Specialized genetic operators are used, which allow the system to exchange parts of trees, generate new sub-trees, prune existing ones as well as change the node type and the tests. The system is able to induce univariate, oblique and mixed decision trees. In the paper, we investigate how the *GDT* system can profit from a parallelization on a compute cluster. Both parallel implementation and distributed version of the induction are considered and significant speedups are obtained. Preliminary experimental results show that at least for certain problems the distributed version of the *GDT* system is more accurate than its panmictic predecessor.

1 Introduction

Evolutionary algorithms [17] are methods inspired by the process of natural evolution, which are applied to solve many difficult optimization and search problems. Among others they are successfully used in various knowledge discovery systems [7]. However, it is known that the evolutionary approach is not the fastest one and a lot of effort is put into speeding it up. This issue is especially important for future data mining applications, where larger and larger datasets will be processed and analyzed.

Fortunately evolutionary techniques are naturally prone to parallelism and in most of the cases they can be efficiently implemented in distributed environments [1]. Such an implementation can be only aimed at speeding up the calculations without changing the original sequential algorithm (so called *global parallelism*) or can try to extend it by using structured populations. Among the most widely known types of structured EAs are *distributed* (*coarse-grained*) and *cellular* (*fine-grained*) algorithms. In the distributed version a population is partitioned into several independent subpopulations (islands) performing a sparse exchange of individuals, whereas in the cellular algorithm individuals evolve in overlapped small neighborhoods.

In our previous papers, *Global Decision Tree (GDT)* system was introduced for a global induction of decision trees based on evolutionary algorithms. The system was able to generate accurate and simple univariate [12], oblique [13] and mixed [14] trees. In this paper we want to investigate how the global induction of decision trees can profit from a parallelization on a compute cluster. Among many top-down (e.g. [5],[4]) and global (e.g. [10], [8], [19]) evolutionary decision tree inducers, according to our knowledge only two attempts in the framework of the fine-grained approaches were developed: *CGP/SA* - (*Cellular Genetic Programming/Simulated Annealing*) [6] and *GALE* - (*Genetic and Artificial Life Environment*) [16].

The rest of the paper is organized as follows. In the next section the *GDT* system is briefly recalled. In section 3 a parallel implementation of the global induction is presented and in section 4 a distributed version is investigated. Experimental validation of the presented approaches is performed in section 5. The paper is concluded and possible research directions are sketched in the last section.

2 GDT System

In this section, the *GDT* system is briefly presented. The system is able to generate univariate [12], oblique [13] and mixed [14] decision trees. Its general structure follows a typical framework of evolutionary algorithms [17] with an unstructured population and a generational selection.

2.1 Representation, Initialization and Termination Condition

A decision tree is a complicated tree structure, in which the number of nodes, test types and even the number of test outcomes are usually not known in advance for a given learning set. Moreover additional information, e.g. about learning vectors associated with each node, should be accessible during the induction. As a result, decision trees are not specially encoded in individuals and they are represented in their actual form.

Three test types can be used in non-terminal nodes depending on the decision tree type: two types of univariate tests for an univariate tree, only oblique tests for an oblique tree and all previously mentioned for a mixed tree.

In case of univariate tests, a test representation depends on the considered attribute type. For nominal attributes at least one attribute value is associated with each branch starting in the node, which means that an internal disjunction is implemented. For continuous-valued features typical inequality tests with two outcomes are used. In order to speed up the search process only boundary thresholds are considered as potential splits and they are calculated before the EA starts. In an oblique test with binary outcome a splitting hyperplane is represented by a fixed-size table of real values corresponding to a weight vector and a threshold. The inner product is calculated to decide where an example is routed.

Before starting the actual evolution, an initial population is created by applying a simple top-down algorithm based on a dipolar principle [11] to randomly selected sub-samples of the learning set [14].

The evolutionary induction terminates when the fitness of the best individual does not improve during a fixed number of generations (default value is equal to 1000) or the maximum number of generations (default value: 5000) is reached.

2.2 Variation Operators

We use two specialized genetic operators corresponding to the classical mutation and cross-over. Application of any operator can result in a necessity for relocation of the learning examples between tree parts rooted in the modified nodes. Additionally the local maximization of the fitness can be performed by pruning lower parts of the sub-tree on condition that it improves the value of the fitness.

Mutation operator. A mutation-like operator [14] is applied with a given probability to a tree (default value is 0.8) and it guarantees that at least one node of the selected individual is mutated. First, the type of the node (leaf or internal node) is randomly chosen with equal probability and if a mutation of a node of this type is not possible, the other node type is chosen. A ranked list of nodes of the selected type is created and a mechanism analogous to a ranking linear selection [17] is applied to decide which node will be affected.

While concerning internal nodes, the location (the level) of the node in the tree and the quality of the subtree rooted in the considered node are taken into account. Nodes on lower levels of the tree are mutated with higher probability, which promotes local changes. Nodes on the same level are sorted according to the number of misclassified objects by the subtree. As for leaves, the number of misclassified learning examples in leaves is used to put them in order, but homogenous leaves are excluded. As a result, leaves for which classification accuracies are worse are mutated with higher probability.

Modifications performed by a mutation operator depend on the tree type and the node type (i.e. if the considered node is a leaf node or an internal node). For a non-terminal node a few possibilities exist:

- A completely new test (of the approved type) can be found. A pair of objects from different classes (mixed dipole) in this node is randomly chosen and a test which separates them to distinct sub-trees is searched. In case of a univariate tree, such a test can be constructed directly for any feature with different feature values. When an oblique test is considered, the splitting hyperplane is perpendicular to the segment connecting two drawn objects and placed in a halfway position.
- The existing test can be altered by shifting the splitting threshold (continuous-valued feature), by re-grouping feature values (nominal features) or by shifting the hyperplane (oblique test). These modifications can be

purely random or can be guided by the dipolar principles [11] of splitting mixed dipoles and avoiding to split pure ones.
- A test can be replaced by another test or tests can be interchanged.
- One sub-tree can be replaced by another sub-tree from the same node.
- A node can be transformed (pruned) into a leaf.

Modifying a leaf makes sense only if it contains objects from different classes. The leaf is transformed into an internal node and a new test is chosen in the aforementioned way.

Cross-over operator. There are also several variants of exchanging information between individuals. Most of them start with a random selection of one node in each of two affected trees. In the first variant, the subtrees starting in the selected nodes are exchanged. In the second variant, which can be applied only when non-internal nodes are randomly chosen and the numbers of outcomes are equal, only tests associated with the nodes are exchanged. The third variant is also applicable with the same assumptions and branches which start from the selected nodes are exchanged in random order. There is also a variant of crossover inspired by the dipolar principles. In the internal node in the first tree a cut mixed dipole is randomly chosen and for the cross-over the node with the test splitting this dipole is selected in the second tree.

2.3 Fitness Function

A fitness function drives the evolutionary search and is very important and sensitive component of the induction. It is well-known that it is not possible to optimize directly the classification quality on unseen data. Instead, the reclassification quality measured on the learning set can be used, but this may lead to an over-fitting problem. In order to mitigate the problem a complexity term is incorporated into the fitness function. In the *GDT* system the fitness function is maximized and has the following form:

$$Fitness(T) = Q_{Reclass}(T) - \alpha \cdot (Comp(T) - 1.0), \tag{1}$$

where $Q_{Reclass}(T)$ is the reclassification quality of the tree T and α is the relative importance of the classifier complexity (default value is 0.005).

In case of the univariate trees, the tree complexity $Comp(T)$ is defined as the classifier size which is equal to the number of nodes. The penalty associated with the classifier complexity increases proportionally with the tree size and prevents classifier over-specialization. Subtracting 1.0 eliminates the penalty when the tree is composed of only one leaf (in majority voting).

A little bit more elaborated definition of the tree complexity is used when oblique tests are allowed (i.e. in oblique and mixed trees), because it reflects the complexity of tests:

$$Comp(T) = |N_{leaf}(T)| + \sum_{n \in N_{int}(T)} (1 + \beta \cdot (F(n) - 1)), \tag{2}$$

where $N_{leaf}(T)$ and $N_{int}(T)$ are sets of leaves and internal nodes respectively, $F(n)$ is the number of features used in the test associated with the node n and $\beta \in [0,1]$ is the relative importance of the test complexity (default value 0.2). The complexity of the tree is defined as a sum of the complexities of the nodes and it is assumed that for leaves and internal nodes with univariate tests the node complexity is always equal to 1.0. It can be also observed that when $\beta = 1$ the number of features included in the test is used as the node complexity, and if $\beta = 0$, then the node complexity is 1.0.

3 Parallel Implementation of Global Induction

The most costly (time consuming) operation in the typical evolutionary algorithm is evaluation of the fitness of all individuals in the population. As the fitness is calculated independently for every individual this feature gives the most straightforward way of exploiting parallelism in the evolutionary search. The population is evenly distributed to available processors (slaves) and the fitness is calculated in parallel. The remaining parts of the evolution are executed in the master processor.

In evolutionary data mining an individual usually represents a certain type of classifier and its goodness of fit is evaluated by using the learning set. When this set is large, it can be also profitable to divide the learning set into subsets and to distribute them among processors. In each computational node the partial fitness calculations are performed in parallel and finally the overall fitness is summed up in the master processor [15].

In case of the global induction of decision trees none of the aforementioned approaches can be directly applied. In the *GDT* system, information about the learning vectors reaching any node of decision tree is stored in that node. It allows us to apply the genetic operators efficiently and to directly obtain the fitness corresponding to the individual. In fact, the actual fitness calculation is embedded into the post mutation and cross-over processing, when the learning vectors in the affected parts of the tree (or trees) are relocated. This mechanism increases the memory complexity of the induction but significantly reduces its computational complexity. As a consequence, the most time consuming elements of the algorithm are genetic operators and they should be performed in parallel.

The general scheme of a single generation in the parallel implementation of the global induction is presented in Fig. 1. The proposed approach is based on the classical master-slave model. After the selection and the reproduction, subpopulations are sent to the slave processors where individual variation operations are performed. When all individuals in the subpopulation migrated to the given processor are processed, they are sent back to the master processor, where the rest of the algorithm is performed. It should be noticed that also an initial population of the algorithm is created in parallel by slave processors.

Migrating an individual between processors within the framework of the message-passing interface is composed of 3 steeps: packing the individual into a flat message, transferring the message between processors (sending/receiving)

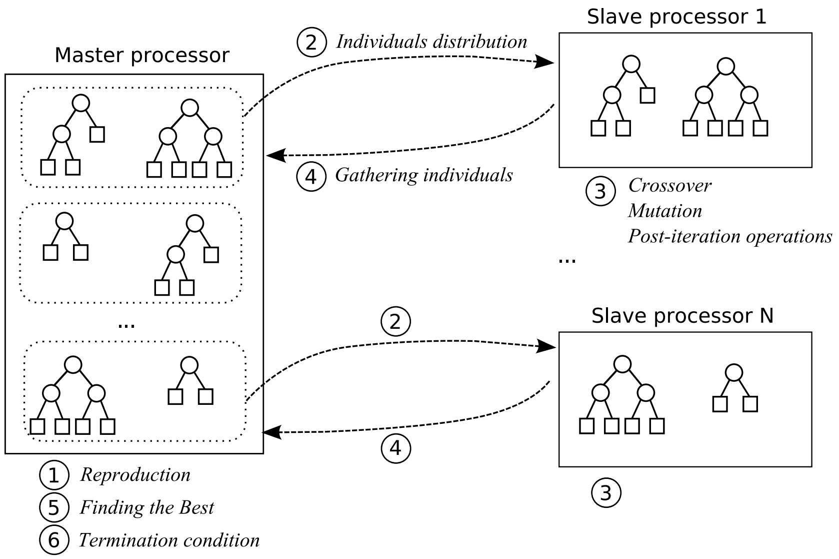

Fig. 1. Parallel implementation of the global decision tree induction based on the master-slave model

and unpacking the corresponding tree. In order to minimize the message size information about learning vectors associated with tree nodes is not stored in the message. Only the number of vectors is included in the message and this speeds up the reconstruction during the message unpacking in the target processor. It should be also noticed that in the master processor, when centralized parts of the algorithm are executed (i.e. the reproduction with elitism and the termination condition verification) packed individuals can be processed on condition that the corresponding values of the fitness are known. This way unnecessary unpacking-packing operations can be eliminated in the master processor. Additionally, a certain number of individuals from the given slave processor survive (or replicate) and in the next iteration they are scheduled to be sent back to that processor. This observation gives another possibility to eliminate unproductive calculation, however the cloned trees should be sent to different processors in order to avoid crossing with identical or very similar individuals.

In spite of all it can be expected that the computational cost associated with scattering and gathering of decision trees is high and the significant improvement in the speedup can be obtained only by reducing migrations. One of the simplest solutions consists in performing the population synchronization not in every iteration. Hence, in these iterations when the synchronization is not performed, subpopulations evolve locally. There are two elements which become different compared to the original algorithm: reproduction and crossover. In every subpopulation the best individual is searched and the reproduction is performed locally. Moreover, individuals can mate only with other individuals located in the same processor. After a few local iterations (default value: 10) a standard iteration with the centralized reproduction is applied. Such a hybrid solution can be treated as a first steep toward fully distributed evolution of decision trees.

4 Distributed Global Evolution of Decision Trees

The proposed solution is based on the classical multi-population (island) model. The general scheme of a distributed induction is presented in Fig. 2. In every dem individuals are evolved locally as in the original sequential version of the *GDT* system. A migration of selected trees is performed periodically (default value: 20 iterations) and selection of trees to send and to discard is done according to their fitness. Islands are organized into a ring and during migrations any dem communicates only with two its neighbors. It sends the same number of individuals to the next island and receives the same number of trees from its predecessor. Such an operation can be straightforwardly and efficiently implemented and the centralized processing is unnecessary. In the simplest case only local best individuals are cloned and sent and the worst individuals are replaced by incoming ones.

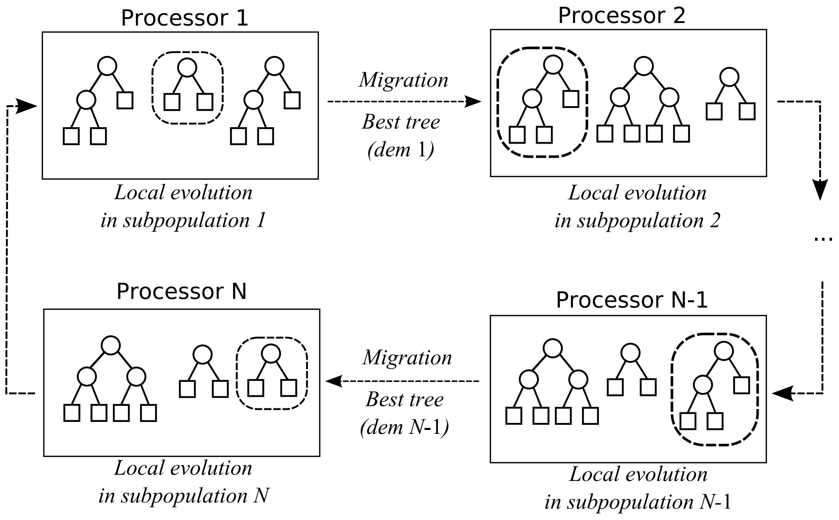

Fig. 2. Distributed evolution based on the island model (ring topology and migration of single individual)

The termination condition of the distributed evolution is verified in a centralized way after migrations. If the overall best value of the fitness does not improve during the fixed number of generations, the algorithm is finished even if the maximum number of generations is not reached. The resulting decision tree of the distributed evolution is chosen according to the fitness function among the best local individuals stored in each island.

5 Experimental Results

In this section the proposed parallel and distributed versions of global decision tree induction are experimentally verified. In order to focus and clarify the presentation only results concerning the mixed decision trees, as the most representative, are included. All presented results were obtained with a default setting of parameters from the sequential version of the *GDT* system.

In the experiments a cluster of sixteen SMP servers running Linux 2.6 and connected by an Infiniband network was used. Each server is equipped with two 64-bit Xeon 3.2GHz CPUs with 2MB L2 cache, 2GB of RAM and an Infniband 10 Gb/s HCA connected to a PCI-Express port. We used the MVAPICH version 0.9.5 [9] MPI implementation.

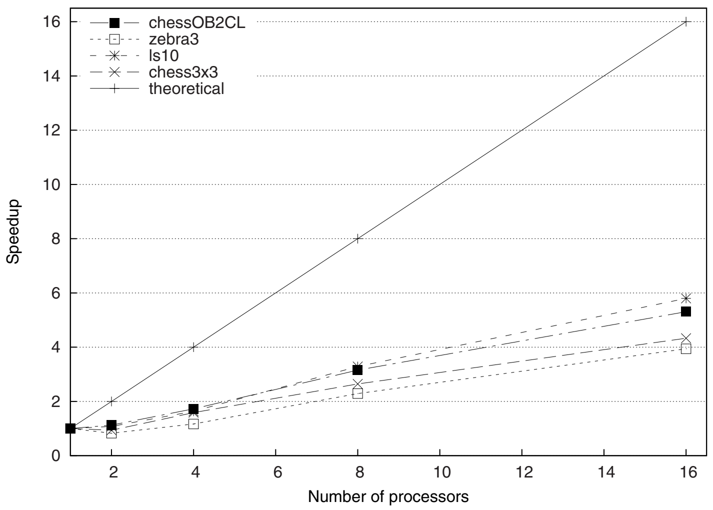

Fig. 3. Efficiency of parallel implementation: speedup measured on exemplar datasets

In Fig. 3 scalability of the parallel implementation measured on four exemplar artificial datasets[1] is presented. It can be observed that the obtained speedup is satisfactory. In order to explain this result detailed time-sharing is necessary and it is presented for one dataset in Fig. 4. It is clearly visible that the problem is caused by the overhead connected with unpacking individuals and redistributing the learning examples to tree nodes at slave processors. The time of applying the variation operators (denoted as *Operation time*) is properly reduced. It can be also noticed that the sending/receiving time (denoted as *MPI time*) is relatively small and it increases with the number of slave processors.

In the next experiment it was verified that the significant increase of speedup can be obtained by reducing migrations. In Fig. 5 execution times measured for

[1] For detailed description of artificial datasets used in the experiments please refer to [14].

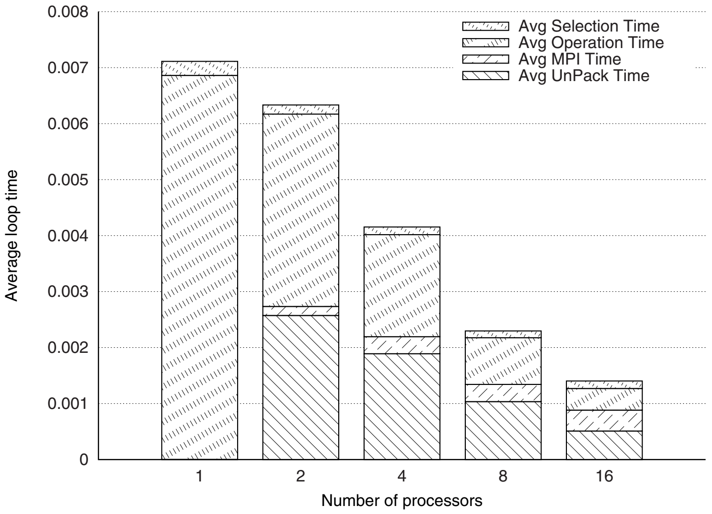

Fig. 4. Efficiency of parallel implementation: detailed time-sharing on *chessOB2CL* dataset

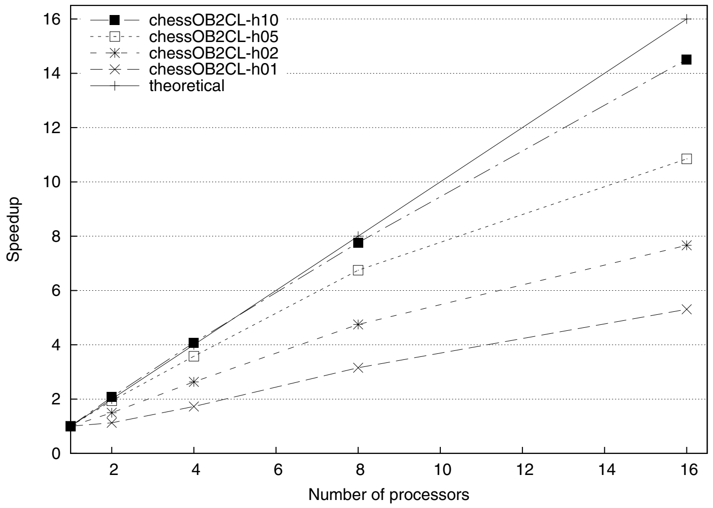

Fig. 5. Speedup of the hybrid solution measured on *chessOB2CL* dataset (with different rate of migrations)

the hybrid implementation with increasing number of local iterations (without centralized reproduction) are presented. It can be observed that sparser migrations result in the important improvement in terms of speedup (almost linear speedup obtained for the run with 10 local iterations after the population synchronization).

Table 1. Results on artificial data

Dataset	**C4.5**		**OC1**		**GDT**		**dGDT**	
	size	quality	size	quality	size	quality	size	quality
chess2x2	1	50	10.1	89.3	4	99.8	4	99.9
chess3x3	9	99.7	21.1	73.7	9	99.3	9	99.6
chessOB2CL	33	95.6	7	77.3	4.1	99.1	4	99.7
chessOB4CL	35	94.6	4.3	49.8	4	98.9	4	99.7
house	21	97.4	8.2	92.8	3.8	96.9	5	99.5
ls10	284	77.3	7.3	95.3	2	97.6	2	98.2
ls2	22	97	2	99.7	2	99.9	2	99.9
zebra1	25	95.3	3	83.5	3	99.6	3	99.9
zebra2	2	59.5	4.8	94.1	4	98.2	4	99.6
zebra3	57	91.2	8.2	24.3	8.4	97	9.3	98.4

Table 2. Results on real datasets

Dataset	**C4.5**		**OC1**		**GDT**		**dGDT**	
	size	quality	size	quality	size	quality	size	quality
balance-scale	57	77.5	5.4	90	2.6	89.5	9.3	94.8
bcw	22.8	94.7	4.7	91.2	2	96.7	2.1	96.7
bupa	44.6	64.7	5.8	65.6	3.5	68.6	32.6	64.1
glass	39	62.5	4.5	55.7	11.6	66.4	31.6	68.5
page-blocks	82.8	97	15.6	96.6	3	94.9	7.3	96.6
pima	40.6	74.6	6.5	69.6	2.2	75.4	13.6	74.4
sat	435	85.5	58.3	78.9	6.4	81.5	14.6	84.8
vehicle	138.6	72.7	21.6	66.4	8.8	65.4	42.3	70.3
waveform	107	73.5	10.5	77.4	4.2	80.5	12.3	81.1
wine	9	85	3.2	87	4.2	91.5	5	88.8

As it can be expected based on the experiments with the hybrid implementation, in case of the distributed version of the *GDT* system almost linear speedup is achieved (15.4 for 16 processors). In the final experiments, an impact of the distributed evolution on accuracy and complexity of decision tree classifiers is verified.

The distributed approach (denoted as *dGDT* in tables) is assessed on 10 artificial and 10 real life datasets and is compared to the well-known top-down univariate ($C4.5$ [20]) and oblique ($OC1$ [18]) decision tree systems. It is also compared to the sequential versions of the global inducer - *GDT*. All prepared artificial datasets comprise training and testing parts. In case of data from *UCI Repository* [2] for which testing data are not provided, a 10-fold stratified cross-validation was employed. The population size for *dGDT* was set to 256 giving 8 sub-populations of 32 individuals each.

In table 1 and table 2 results of experiments with artificial and real life datasets are presented. It can be observed that two global inducers performed better both in terms of accuracy and classifier complexity that its top-down counterparts on artificial datasets. It seems that *dGDT* was even slightly better than the sequential version on these datasets. In case of real life datasets, the situation is more equilibrated. For six datasets *dGDT* was more accurate then GDT, but for three problems the sequential version was better. It can be also noticed that the distributed inducer produced more complex trees. Compared to the top-down systems both global inducers performed well, however for certain problems they were outperformed.

6 Conclusions

In the paper, the parallelization of the global decision tree induction based on evolutionary algorithms is investigated. It was shown that the migration of the individuals is a sensitive point of the process and the cost associated with the migration is relatively high. As a result the most efficient solution is based on the multi-population model. Moreover, it was experimentally shown that at least for certain problems the distributed version of the system is more accurate than its sequential predecessor.

The *GDT* system is still in development and many possible directions of future improvements exist. Currently we are working on more sophisticated island topologies, where random but still decentralized and balanced migrations will be possible. We also plan to implement the global induction in the framework of shared memory multiprocessing (OpenMP), which will reduce the problem of migrations.

Acknowledgments

This work was supported by the grant W/WI/5/08 from Białystok Technical University.

References

1. Alba, E., Tomassini, M.: Parallelism and evolutionary algorithms. IEEE Transactions on Evolutionary Computation 6(5), 443–462 (2002)
2. Blake, C., Keogh, E., Merz, C.: UCI repository of machine learning databases (1998), `http://www.ics.uci.edu/~mlearn/MLRepository.html`
3. Breiman, L., Friedman, J., Olshen, R., Stone, C.: Classification and Regression Trees. Wadsworth Int. Group (1984)
4. Cantu-Paz, E., Kamath, C.: Inducing oblique decision trees with evolutionary algorithms. IEEE Transactions on Evolutionary Computation 7(1), 54–68 (2003)
5. Chai, B., et al.: Piecewise-linear classifiers using binary tree structure and genetic algorithm. Pattern Recognition 29(11), 1905–1917 (1996)
6. Folino, G., Pizzuti, C., Spezzano, G.: Genetic Programming and Simulated Annealing: A Hybrid Method to Evolve Decision Trees. In: Poli, R., Banzhaf, W., Langdon, W.B., Miller, J., Nordin, P., Fogarty, T.C. (eds.) EuroGP 2000. LNCS, vol. 1802, pp. 294–303. Springer, Heidelberg (2000)

7. Freitas, A.: Data Mining and Knowledge Discovery with Evolutionary Algorithms. Springer, Heidelberg (2002)
8. Fu, Z., Golden, B., Lele, S., Raghavan, S., Wasil, E.: A genetic algorithm-based approach for building accurate decision trees. INFORMS Journal on Computing 15(1), 3–22 (2003)
9. Liu, J., Wu, J., Panda, D.K.: High performance RDMA-based MPI implementation over InfiniBand. Int. Journal of Parallel Programming 32(3), 167–198 (2004)
10. Koza, J.: Concept formation and decision tree induction using genetic programming paradigm. In: Schwefel, H.-P., Männer, R. (eds.) PPSN 1990. LNCS, vol. 496, pp. 124–128. Springer, Heidelberg (1991)
11. Krętowski, M.: An Evolutionary Algorithm for Oblique Decision Tree Induction. In: Rutkowski, L., Siekmann, J.H., Tadeusiewicz, R., Zadeh, L.A. (eds.) ICAISC 2004. LNCS (LNAI), vol. 3070, pp. 432–437. Springer, Heidelberg (2004)
12. Krętowski, M., Grześ, M.: Global learning of decision trees by an evolutionary algorithm. In: Information Processing and Security Systems, pp. 401–410. Springer, Heidelberg (2005)
13. Krętowski, M., Grześ, M.: Evolutionary Learning of Linear Trees with Embedded Feature Selection. In: Rutkowski, L., Tadeusiewicz, R., Zadeh, L.A., Żurada, J.M. (eds.) ICAISC 2006. LNCS (LNAI), vol. 4029, pp. 400–409. Springer, Heidelberg (2006)
14. Krętowski, M., Grześ, M.: Evolutionary induction of mixed decision trees. International Journal of Data Warehousing and Mining 3(4), 68–82 (2007)
15. Kwedlo, W., Krętowski, M.: Learning decision rules using a distributed evolutionary algorithm. TASK Quarterly 6(3), 483–492 (2002)
16. Llora, X., Garrell, J.: Evolution of decision trees. In: Proc. of CCAI 2001, pp. 115–122. ACIA Press (2001)
17. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs, 3rd edn. Springer, Heidelberg (1996)
18. Murthy, S., Kasif, S., Salzberg, S.: A system for induction of oblique decision trees. Journal of Artificial Intelligence Research 2, 1–33 (1994)
19. Papagelis, A., Kalles, D.: Breeding decision trees using evolutionary techniques. In: Proc. of ICML 2001, pp. 393–400. Morgan Kaufmann, San Francisco (2001)
20. Quinlan, J.: C4.5: Programs for Machine Learning. Morgan Kaufmann, San Francisco (1993)