Information Sciences 288 (2014) 153–173

Contents lists available at ScienceDirect

Information Sciences

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

Evolutionary induction of global model trees with specialized operators and memetic extensions

Marcin Czajkowski, Marek Kretowski *

Faculty of Computer Science, Bialystok University of Technology, Wiejska 45a, 15-351 Bialystok, Poland

ARTICLE INFO

Article history:
Received 18 July 2013
Received in revised form 11 June 2014
Accepted 30 July 2014
Available online 7 August 2014

Keywords:
Evolutionary algorithm
Model tree
Multiple linear regression
Machine learning

ABSTRACT

Metaheuristics, such as evolutionary algorithms (*EA*s), have been successfully applied to the problem of decision tree induction. Recently, an EA was proposed to evolve model trees, which are a particular type of decision tree that is employed to solve regression problems. However, there is a need to specialize the *EA*s in order to exploit the full potential of evolutionary induction. The main contribution of this paper is a set of solutions and techniques that incorporates knowledge about the inducing problem for the global model tree into the evolutionary search. The objective of this paper is to demonstrate that specialized *EA* can find more accurate and less complex solutions to the traditional greedy-induced counterparts and the straightforward application of *EA*.

This paper proposes a novel solution for each step of the evolutionary process and presents a new specialized *EA* for model tree induction called the Global Model Tree (*GMT*). An empirical investigation shows that trees induced by the *GMT* are one order of magnitude less complex than trees induced by popular greedy algorithms, and they are equivalent in terms of predictive accuracy with output models from straightforward implementations of evolutionary induction and state-of-the-art methods.

1. Introduction

The most common predictive tasks in data mining [17] are classification and regression. Decision trees [29,36] are one of the most popular prediction techniques. The success of tree-based approaches can be explained by their ease of application, speed of operation, and effectiveness. Furthermore, the hierarchical tree structure, where appropriate tests from consecutive nodes are sequentially applied, closely resembles a human method of decision making, which makes decision trees natural and easy to understand even for inexperienced analysts. Regression and model trees [22] are variants of decision trees, and they have been designed to approximate real-valued functions instead of being used for classification tasks. The main difference between a regression tree and a model tree is that, in the latter, a constant value in the terminal node is replaced by a regression plane.

Inducing an optimal model tree, as with the problem of learning an optimal decision tree, is known to be NP-complete [24]. Consequently, practical decision-tree learning algorithms are based on heuristics such as greedy algorithms, where locally optimal decisions are made in each tree node. Such algorithms cannot guarantee to return the globally optimal decision tree. The purpose of this paper is to illustrate the application of a specialized evolutionary algorithm (*EA*) [27] to the problem of model tree induction. The objectives are to show that evolutionary induction may result in finding globally

* Corresponding author.

http://dx.doi.org/10.1016/j.ins.2014.07.051
0020-0255/

optimal solutions that are more accurate and less complex than the traditional greedy-induced counterparts and straightforward application of *EA*. This research shows the impact of the application of specialized *EA*s on the tree structure, tests in internal nodes, and models in the leaves. By incorporating the knowledge about global model tree induction, the full potential of *EA*s is exploited. Local optimizations are also proposed for *EA*s problem search, which is known as a memetic algorithm [28,7].

Our previous research showed that global inducers are capable of efficiently evolving accurate and compact univariate regression trees [25], called Global Regression Trees (*GRT*), and model trees with simple linear regression in the leaves [8,10]. In our previous papers, we proposed model trees with multiple linear regression in the leaves [9] and considered how memetic extensions improve the global induction of regression and model trees [11]. This paper reviews and significantly extends our previous work on model trees in almost every step of evolutionary induction. We introduce new specialized operators and local search components that improve pure evolutionary methods and propose a smoothing process to increase the prediction accuracy of the model tree. A new multi-objective optimization strategy (lexicographic analysis) is verified as an alternative fitness function to a weight formula. Additional data sets and new experiments illustrate the advantage of the global search solutions for popular model tree algorithms.

This paper is organized as follows. The following section provides a brief background on model trees, reviews related work, and describes some of the advantages with regard to using EAs for model tree induction. Section 3 describes the approach and demonstrates how each step of the *EA* can be improved. Section 4 presents a validation of the proposed solutions in three sets of experiments. In the last section, the paper is concluded and possible future works are sketched.

The presented experiments demonstrate how each step of the *EA* can be improved.

2. Global vs local induction

Decision trees are often built through a process that is known as a recursive partitioning. The most popular tree-induction is based on the top-down approach [35]. It starts from the root node, where the locally optimal split (test) is searched according to the given optimality measure (e.g., Gini, Twoing, or the entropy rule for classification trees and the least squared or least absolute deviation error criterion for regression trees). Next, the training data is redirected to newly created nodes, and this process is repeated for each node until some stopping-rule is violated. Finally, post-pruning [15] is applied to improve the generalization power of the predictive model. Inducing the decision tree through a greedy strategy is fast and generally efficient in many practical problems, but it usually produces locally optimal solutions.

One of the first and most well-known top-down regression tree solutions is the Classification and Regression Tree (CART) [5]. The method searches for a locally optimal split that minimizes the sum of squared residuals of the model and builds a piecewise constant model with each terminal node fitted by the training sample mean. The following solutions managed to improve the prediction accuracy by replacing single values in the leaves with more advanced models. The M5 system [39] induces a model tree that contains at leaves multiple linear models analogous to piecewise linear functions. The *HTL* [41] is even more advanced and evaluates linear and nonlinear models in terminal nodes.

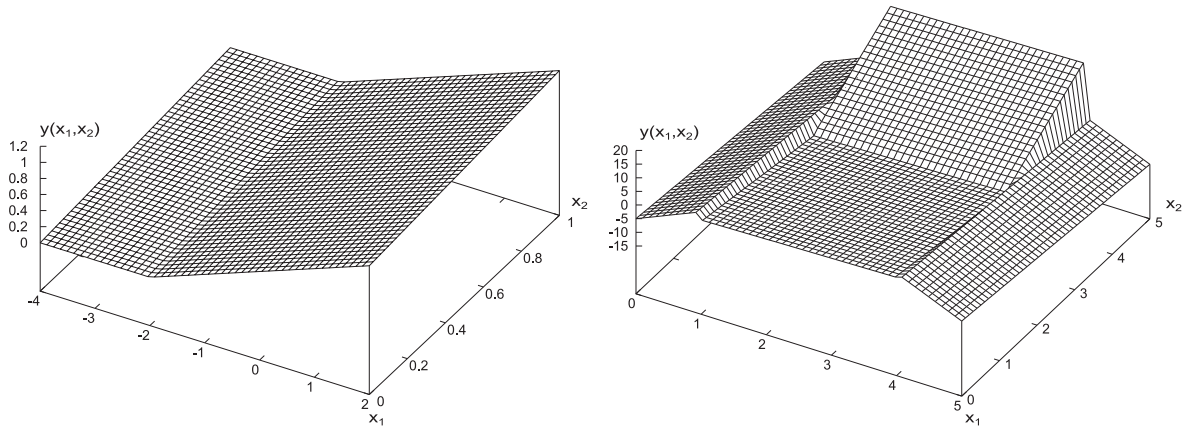
Multiple authors have proposed methods to limit the negative effects of inducing the decision tree with the greedy strategy. In *SECRET* [13], authors suggest that changing a regression problem into a classification one may help in finding more globally optimal partitions. A different solution was proposed in *SMOTI* [26], where regression models exist not only in the leaves but also in the upper parts of the tree. The authors suggested that this technique allows individual predictors to have both global and local effects on the model tree. A more recent innovation for finding optimal splits in nodes was presented in *LLRT* [42]. The *LLRT* solution can do a near-exhaustive evaluation of all possible splits in a node based on the quality of fit of the linear regression models in the resulting branches.

In the literature, there have been some attempts to apply an evolutionary approach for the induction of decision trees, including regression and model trees. For an extensive review, please refer to [3]. In *TARGET* [16], the authors proposed to evolve a *CART*-like regression tree with simple genetic operators. The Bayesian information criterion (*BIC*) [37] was used as a fitness function, which penalizes the tree for over-parameterization. A more advanced system called *E-Motion* was proposed in [2]. The authors evolved univariate trees with linear models in the leaves and optimized their prediction errors and the tree size. *E-Motion* implements standard 1-point crossover and two different mutation strategies (shrinking and expanding) to variate individuals. The *GPMCC* [32] approach proposed to evolve model trees with non-linear models in the leaves. In most of the papers, performing such a global search in the space of candidate solutions successfully competes with popular greedy methods. However, almost all algorithms from [3] apply only the basic variants of *EA*, which do not incorporate knowledge of the decision tree's induction.

In this paper, we would like to fill this gap by proposing specialized operators and memetic extensions for the evolutionary induction of model trees.

To illustrate the simple scenario where evolutionary induced model trees are beneficial, we prepared two artificially generated datasets, with analytically defined decision borders (1) and (2) illustrated in Fig. 1. Both datasets contain an attribute that is linearly dependent with one or two independent attributes.

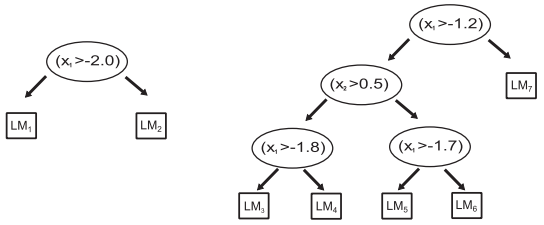
The data set on the left (denoted as *split plane3*) can be perfectly predictable with regression lines on subsets of the data resulting from a single partition at threshold $x_1 = -2$, and it is described by Eq. (1). Most of the popular greedy top-down inducers that minimize the residual sum of squares (like *CART*) or standard deviation (like *M5*) will not find the best



$$y(x_1, x_2) = \begin{cases} 0.2 * x_2 & x_1 < -2 \\ 0.25 * x_1 + 0.2 * x_2 + 0.5 & x_1 \geq -2 \end{cases} \quad (1)$$

$$y(x_1, x_2) = \begin{cases} 10 * x_1 - 1.5 * x_2 - 5 & x_1 < 1 \\ -10 * x_1 - 1.5 * x_2 + 45 & x_1 \geq 4 \\ 0.5 * x_1 - 2.5 * x_2 + 1.5 & x_2 < 3;\ 1 \leq x_1 < 4 \\ 0.5 * x_1 + 10 * x_2 - 35 & x_2 \geq 3;\ 1 \leq x_1 < 4 \end{cases} \quad (2)$$

Fig. 1. Examples of artificial datasets: *split plane3* – left, *armchair3* – right.



EA approach	LM_1: $y(x_1, x_2) = 0.25 * x_1 + 0.2 * x_2 + 0.5$
	LM_2: $y(x_1, x_2) = 0.2 * x_2 + 0.5$
M5 approach	LM_3: $y(x_1, x_2) = 0.1865 * x_2 + 0.0052$
	LM_4: $y(x_1, x_2) = 0.25 * x_1 + 0.2 * x_2 + 0.5$
	LM_5: $y(x_1, x_2) = 0.1936 * x_2 + 0.0079$
	LM_6: $y(x_1, x_2) = 0.25 * x_1 + 0.2 * x_2 + 0.5$
	LM_7: $y(x_1, x_2) = 0.25 * x_1 + 0.2 * x_2 + 0.5$

Fig. 2. Examples of model trees for data set *split plane3* for global approach (left) and greedy M5 algorithm (right) and the corresponding linear models in the leaves.

partitions. In Fig. 2, the output trees for the greedy (*M5*) and evolutionary approach are compared. The evolutionary induced model tree can partition the data at threshold $x_1 = -2.00$ because it can search globally for the best solution. However, the *M5* algorithm is trapped in the local optima and finds the threshold at $x_1 = -1.2$. A non-optimal partition in the root node increases the tree size and has a strong influence on prediction errors. For this dataset, the *CART* also returns non-optimal solutions and splits the root node at $x_1 = -0.44$. The output tree for *CART* is not shown, as the algorithm found an even larger tree than *M5*.

Data set *Armchair3* and its underlying model, Eq. (2) in Fig. 1, is much more complex than dataset *split plane3*. Many traditional approaches will fail to efficiently split the data, as the greedy inducers search only for locally optimal solutions (at the current node). Similar to the previous experiment, evolutionary inducers manage to find the first split at $x_1 = 1.00$ and induce an optimal tree. Trees induced by greedy algorithms (like *M5*) need more than 18 multiple linear regression rules because the split in the root node is incorrect ($x_1 = 3.73$).

3. Evolutionary induction of the global model tree

In this section, we would like to propose the solution called Global Model Tree (*GMT*), which is an evolutionary approach for the global induction of model trees. The *GMT* general structure follows a typical framework for an evolutionary algorithm with an unstructured population and a generational selection. Each step of the *GMT* will be discussed separately: representation,

initialization, fitness function, selection and terminal condition, genetic operators, and smoothing. In each step, knowledge of the model tree induction was incorporated into the evolutionary search. The process diagram of the *GMT* algorithm is illustrated in Fig. 3.

3.1. Representation

The type and the way of representing the individuals may define the type of *EA* used. A genetic algorithm is normally used when solutions are encoded in a fixed-length linear string and only data is encoded. Tree-encoding schemes usually imply genetic programming (*GP*), where the solution encodes data and functions [44].

Decision trees are complicated tree structures in which the number of nodes, the type of tests, and even the number of test outcomes are not known in advance. This is why the second aforementioned approach is more suitable, especially if the entire tree is searched in one EA run. Therefore, in our system, model trees are not encoded in individuals, and they are represented in their actual form as typical univariate trees with multiple linear models in the leaves. An example individual induced in dataset *armchair*3 is illustrated in Fig. 4a and visualized in Fig. 4b. The image on the right visualizes the partition space given by the *GMT*. Each test in a non-terminal node concerns only one attribute (nominal or continuous). In the case of a continuous-valued attribute, typical inequality tests are applied. For a nominal attribute, at least one value is associated with each branch. This means that an inner disjunction is built into the induction algorithm.

In each leaf, a multiple linear model is constructed using the standard regression technique [33] with objects associated with that node. A dependent variable y is explained by the linear combination of multiple independent features $x_1, x_2, \ldots, x_q$:

$$y = \beta_0 + \beta_1 * x_1 + \beta_2 * x_2 + \cdots + \beta_q * x_q, \tag{3}$$

where q is the number of independent variables, $x_{1\ldots q}$ are independent variables, and $\beta_{0\ldots q}$ are fixed coefficients that minimize the sum of the squared residuals of the model. If q is equal to zero, the leaf node will be a regression node with single value equal to β_0.

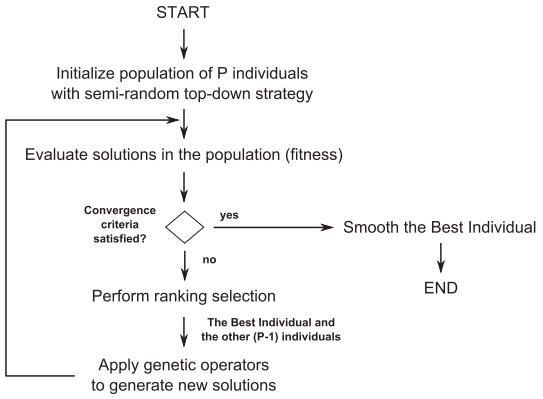


Fig. 3. The GMT process diagram.

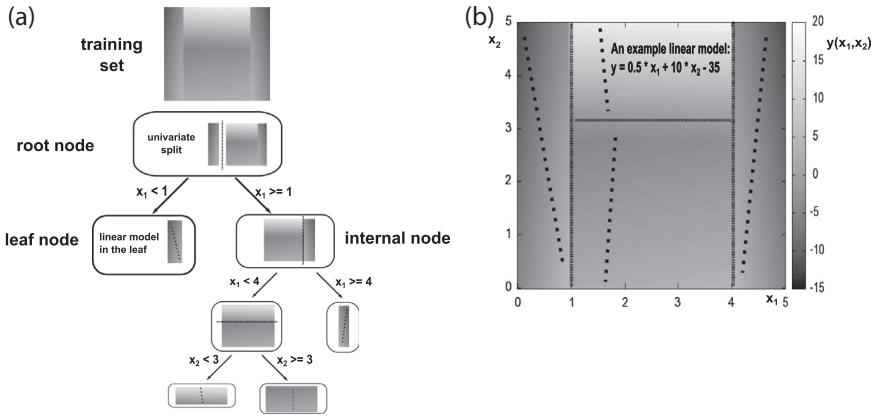


Fig. 4. An example representation of the individual (a) and the input space partition (b) for dataset *armchair3*.

In addition, in every node, information about objects and statistics (error, model size) associated with the node is stored (see Fig. 4a). This enables the algorithm to have a more efficient local structure and tests modifications during applications of genetic operators. When, for example, an internal node of an individual is modified during mutation or crossover, the *GMT* algorithm automatically updates the statistics for its subtree. This way, each node stores actual statistics, and there is no need to examine the entire individual to obtain information about its error or size.

3.2. Initialization

Traditionally, the initial population should be generated randomly to cover the entire range of possible solutions. In addition, the direct application of one of the greedy algorithms can trap the EA in local optima. Therefore, while creating the initial population, we search for a good tradeoff between a high degree of heterogeneity and a relatively low computation time.

Initial individuals are created by applying the classical top-down algorithm, similar to the *M5* approach. At first, we learn a standard regression tree, where each leaf in the tree contains a sample mean of the dependent variable computed on the set of instances that reach that leaf. The recursive partitioning is finished when all the training objects in a node are characterized by similar predicted values or there are less than 5 objects in the node. Next, the multiple linear model is constructed for the instances in each node of the model tree. Instead of using all the attributes, which is time consuming, the model is restricted to the ones that are referenced by tests somewhere in the subtree of this node.

To keep the balance between exploration and exploitation, the initial individual is created based on a chosen subsample of the original training data (10% of data, but not more than 500 examples). To ensure that the subsample contains the objects with the various values of the predicted attribute, the training data is sorted by predicted value, split into a defined number of equal-size folds (default: 10), and then, from these folds, objects are randomly chosen and placed into the subsample. Each individual's non-terminal node test is calculated from a random subset of attributes (default: 50%), and it is chosen by one of the three memetic search strategies, which involves employing the locally optimized tests:

- *Least Squares (LS):* this function reduces the node impurity measured by the sum of the squared residuals.
- *Least Absolute Deviation (LAD):* this function reduces the sum of the absolute deviations. It is more robust and has greater resistance to outlying values than *LS*.
- *Dipolar:* the dipole (a pair of feature vectors) is selected, and then a test is constructed that splits this dipole. The first instance that constitutes the dipole is selected randomly from the set of instances from the current node. The rest of the feature vectors are sorted in decreasing order according to the difference between the dependent variable values and the selected instance. To find a second instance that constitutes the dipole, we applied a mechanism similar to the ranking linear selection [27].

The choice of strategy affects the EA convergence to the global optima. Selecting a greedy optimal strategy such as *LS* or *LAD* may slow down algorithm convergence, as it is more likely for the *EA* to be trapped in local optima. On the other side, seeding the initial population with good solutions can enhance the quality of the search and shorten the execution time. Therefore, we recommend the *mix* strategy for choosing non-terminal nodes in which three proposed search strategies are balanced. Finally, we apply a pessimistic pruning mechanism [15] for individuals. This reduces the tree size of the usually overgrown initial trees.

3.3. Fitness function

The fitness function is one of the most important and sensitive elements in the design of the *EA*. It drives the evolutionary search process by measuring how good a single individual is in terms of meeting the problem objective. A single evaluation measure may degrade the other measures [6]; therefore, multi-objective optimization may present more acceptable overall results. In the context of model trees, a direct minimization of the prediction error measured in the learning set usually leads to the over-fitting problem. In the typical top-down induction of decision trees [36], this problem is partially mitigated by defining a stopping condition and by applying post-pruning [15].

There are three popular multi-objective optimization strategies [18]: the weight formula, lexicographic analysis, and Pareto-dominance. The weight formula transforms a multi-objective problem into a single-objective one by constructing a single formula that contains each objective. The main drawback of this strategy is the need to find adjusted weights for the measures. The lexicographic approach analyzes the objectives values for the individuals one by one based on the priorities. This approach also requires defining thresholds; however, adding up non-commensurable measures, such as tree error and size, is not performed. Pareto-dominance searches not for one best solution, but rather for a group of solutions in such a way, that selecting any one of them in place of another will always sacrifice quality for at least one objective, while improving it for at least one other. Apart from performance issues, the Pareto approach does not consider the fact that in most research problems for decision trees, the error minimization is more important than the size of the tree. In this paper, the first two approaches is applied, leaving Pareto-dominance for future studies.

Algorithm 1. Lexicographic analysis for two individuals: T_1 and T_2.

```
T_1, T_2 ← tested individuals (trees)
RSS_tr ← primary measure threshold
Q_tr ← secondary measure threshold
W_tr ← tertiary measure threshold
repeat
  if |RSS(T_2) − RSS(T_1)| > RSS_tr then
    if RSS(T_2) > RSS(T_1) then
      return T_1
    else
      return T_2
    end if
  end if
  if |Q(T_2) − Q(T_1)| > Q_tr then
    if Q(T_2) > Q(T_1) then
      return T_1
    else
      return T_2
    end if
  end if
  if |W(T_2) − W(T_1)| > W_tr then
    if W(T_2) > W(T_1) then
      return T_1
    else
      return T_2
    end if
  end if
  RSS_tr = RSS_tr/2
  Q_tr = Q_tr/2
  W_tr = W_tr/2
until Individual with higher fitness is not found
```

In previous work, we have tested various weight formulas as fitness functions: *CART*-like formula [25], Akaike's information criterion (*AIC*) [1], and Bayesian information criterion (*BIC*) [37]. Research shows that the *BIC* as a fitness function performs well as a weight formula for evolutionary induced regression trees. This measure of the goodness of fit works also as a penalty for increasing the tree size. The *BIC* is given by

$$Fit_{BIC}(T) = -2 * ln(L(T)) + ln(n) * k(T), \tag{4}$$

where $L(T)$ is the maximum of the likelihood function of the tree T, n is the number of observations in the data, and $k(T)$ is the number of model parameters in the tree. Ln (likelihood) function $L(T)$ is typical for regression models [20] and can be expressed as

$$ln(L(T)) = -0.5n * [ln(2\pi) + ln(RSS(T)/n) + 1], \tag{5}$$

where $RSS(T)$ is the residual sum of squares of the tree T. The term $k(T)$ can also be viewed as a penalty for over-parameterization. In the *GMT*, the value of $k(T)$ depends on the tree size and the number of attributes in the models in the leaves. This penalty term in the *GMT* is parameterized; therefore, there is a possibility to steer the complexity and the performance of the induced trees.

This paper also tests a lexicographic analysis of the *GMT* as a fitness function. Each pair of individuals is evaluated by analyzing, in order of priority, one of three measures: the residual sum of squares $RSS(T)$, the number of nodes $Q(T)$ in the tree, and the attributes $W(T)$ in multiple linear models. We set the first priority to measure the tree error because the research usually seeks for the most accurate trees next to the number of terminal nodes to prevent over-fitting and overgrown trees. The last measure $W(T)$ keeps the models in the leaves as simple as possible and also penalizes for over-parameterization.

The lexicographic analysis of two individuals, T_1 and T_2, is illustrated in Algorithm 1. The algorithm returns the individual with the higher fitness. The analysis starts with the comparison of the tree error for the individuals: $RSS(T_1)$ and $RSS(T_2)$. If the difference between the values is greater than defined by the user threshold denoted as RSS_{tr}, the lexicographic analysis is finished and the individual with the smaller *RSS* is returned. Otherwise, the secondary measure $Q(T)$ with the threshold denoted as Q_{tr} is analyzed and so on. If after the tertiary measure $W(T)$ with threshold W_{tr} the individual with higher fitness is not found, the lexicographic analysis starts from the beginning but with decreased values for the thresholds. The analysis

is finished when one of the measures shows a difference between two individuals that is greater than the given threshold or it turns out that both individuals have the same values for all three measures. In the algorithm, sometimes only the first measure is required to determine which individual has better fitness, while sometimes measures have to be decreased and analyzed more than once.

3.4. Selection and termination condition

Ranking linear selection is applied as a selection mechanism. In each iteration, the single individual with the highest value of fitness function in the current population is copied to the next one *(elitist strategy)*. Evolution terminates when the fitness of the best individual in the population does not improve by at least 0.1% during the fixed number of generations (default: 1000). In case of a slow convergence, the maximum number of generations is also specified (default value: 10,000), as this limits the computation time.

3.5. Genetic operators

To maintain genetic diversity, two specialized genetic meta-operators corresponding to classical mutation and crossover have been proposed. Each evolutionary iteration starts with randomly choosing the operator type, and next, one of its variants. Both operators influence the tree structure, the tests in non-terminal nodes, and the models in the leaves. To the best of the authors' knowledge, all the previous evolutionary induced regression and model trees employ only the basic variants of genetic operators. In this paper, several new and advanced variants of recombination and mutation are proposed that incorporate local search components (memetic algorithms) and knowledge about the global model tree induction problem into the evolutionary search.

After each operation, it is usually necessary to relocate learning vectors between the parts of the tree rooted in the altered node. This can cause prunning of certain parts of the tree that do not contain any learning vectors. In addition, after each operation in the non-terminal node, the models in the corresponding leaves are not recalculated because the linear models can be found by the mutations in the leaves.

3.5.1. Crossover

It begins with selecting positions in two affected individuals. We apply one basic variant of recombination (the first one) and propose four additional ones:

- *exchange subtrees* – subtrees starting in randomly selected nodes are exchanged (this is the most commonly applied variant of crossover).
- *exchange tests* – tests associated with randomly chosen nodes are exchanged (only when non-terminal nodes are chosen and the number of outcomes are equal) – this crossover has lower impact than crossover (i) but greater chance of finding better tests in both individuals.
- *exchange branches* – branches that start from nodes are exchanged in random order (only when non-terminal nodes are chosen and the number of outcomes are equal) - it may be considered redundant, because the same effect can be achieved by combining two (or more) exchanges of the first crossover (i). However, the experiments show that this variant was very useful in escaping the local optima and that it improved the speed of convergence.
- *asymmetric* – the subtree of the first/s individual is replaced by a new one that was duplicated from the second/first individual. The replaced subtree starts in the node denoted as receiver, and the duplicated subtree starts in the node denoted as donor. This is illustrated in Fig. 5. It is preferred that the receiver node has a high error per instance because it is replaced by the donor node, which should have a small value of Mean Absolute Error because it is duplicated. The application of this variant is more likely to improve affected individuals because, with higher probability, the good nodes are duplicated and replace the weak nodes.
- *with best* – subtree that starts in the donor node of the best individual is duplicated on the receiver node of the individual. Only one individual is affected in this recombination. This variant may complement or even replace copying the best individual found so far into the next population (*elitist strategy*).

In the last two variants, a mechanism analogous to the ranking linear selection was applied to decide which node would be affected. Nodes or leaves are selected from the ranked list, which takes into account the absolute error divided by the number of instances in the node.

3.5.2. Mutation

Mutation of the individual starts with randomly choosing the type of node (equal probability of selecting a leaf or internal node). Next, the ranked list of nodes of the selected type for this individual is created. Depending on the type of node, the ranking takes into account.

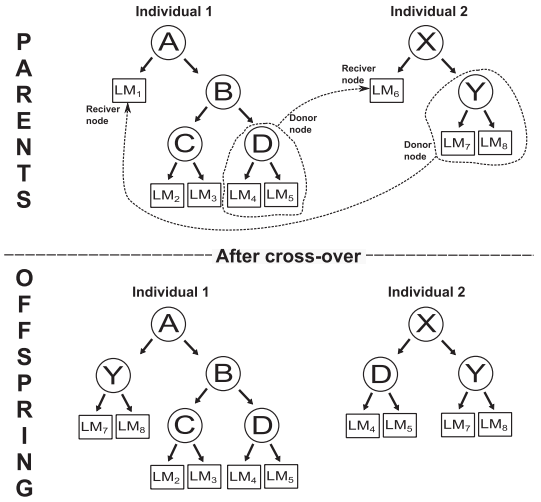


Fig. 5. Crossover between two individuals and the resulting offspring. Each individual has one donor node and one receiver node.

- location (level) of the internal node in the tree – it is evident that modification of the test in the root node affects the entire tree and has a great impact, whereas the mutation of an internal node in the lower parts of the tree has only a local impact. Therefore, internal nodes in the lower parts of the tree are mutated with higher probability.
- Mean Absolute Error – nodes with a higher error per instance are more likely to be mutated.

Finally, a mechanism analogous to the ranking linear selection is applied to decide which node in the individual will be affected.

Straightforward implementation of EA applies only a few basic variants of mutation:

- *prune* – changes an internal node into a leaf – acts like a pruning procedure.
- *random expand* – transforms a leaf into an internal node with a new random test – allows expansion of the tree and searches for more specific regions.
- *nominal dis junction* – re-grouping nominal attribute values by adding/merging branches or moving values between them – modifies tests on nominal attributes in the internal node.
- *new random test* – reinitializes a test in the node using a new random one finds a new test in the internal node.
- *change model* – extends/simplifies/changes the multiple linear model in the leaf by adding/removing/replacing a randomly chosen attribute – allows an optimal set of attributes to be found that will be used to calculate linear regression.

The proposed solution extends this set with specialized variants of mutation:

- *shift threshold* – shifting the splitting threshold at the continuous-valued attribute - allows adjustment of the threshold on the same attribute.
- *new dipolar test* – test in the node is reinitialized by a new dipolar one.
- *dipolar expand* – transforms the leaf into an internal node with a new dipolar test.
- *parent with son* (*branches*) – replaces parent node with random son. It is difficult for the evolution to eliminate the top or middle internal nodes that split only small parts of the data. They have little impact on the prediction and unnecessarily increase the size of the tree. Therefore, this operator can be seen as pruning the middle of the tree.
- *parent with son* (*tests*) – tests between the father and a random son exchanged – gives a chance to affect internal nodes that sub-nodes are not the leaves.
- *recalculate models* – recursively recalculates models in all corresponding leaves. After any mutation or crossover, the corresponding models in the leaves are not recalculated until this variant is selected.
- *new optimal test* – test in the node is reinitialized by the LS or LAD strategy proposed in Section 3.2.
- *optimal expand* – transforms the leaf into an internal node with a new test selected by one of the optimal strategies proposed in Section 3.2.
- *clear model* – deletes from the linear model the least important attribute – helps to decrease the size of the regression model in the leaves.
- *optimal model* – replaces the multiple linear model in the leaf with an optimal simple linear regression model or regression plane.

The last four variants involve local search components that are built into the mutation-like operator. Due to the computational complexity constraints, the memetic extensions optimize tests for a single, randomly chosen attribute.

3.6. Smoothing

In the *M5* algorithm [39], a smoothing process for improving the prediction accuracy of the tree-based models was proposed. When smoothing is enabled, the value of each instance predicted by a model located in the appropriate leaf is modified to reflect the predicted values at the nodes along the path from that leaf to the root. It requires the generation of additional linear models for every internal node of the tree.

In the *GMT*, a form of smoothing that is similar to the one in M5 algorithm is proposed. Smoothing is applied to the best individual at the end of the evolution process. In the first step of smoothing, a value for a test instance according to the model in the appropriate leaf is predicted. Then, this value is smoothed and updated along the path back to the root by the linear models calculated in each of the nodes. Let $Pred(T_i)$ denote the predicted value at the T_i subtree of tree T:

$$Pred(T) = \frac{n_i * Pred(T_i) + k * M(T)}{n_i + k}, \tag{6}$$

where n_i is the number of training instances at T_i, $M(T)$ is the predicted value recalculated from the linear model at T, and k is a smoothing constant (default: 10).

Fig. 6 illustrates the smoothing process for a new test instance. After reaching the appropriate leaf, the predicted value $Pred(T)$ for the tested instance would be equal to the value calculated from the model *LM*4. Next, with the smoothing process enabled, all models on the path from the leaf containing *LM*4 to the root node (*LM*5 and *LM*6) influence the final predicted value $Pred(T)$.

According to [39], smoothing has the greatest effect when models were constructed for a few training instances or when the models along the path predicted instances very differently. However, it should be noted that trees that apply smoothing differ from the classical univariate model trees. Each test instance is predicted not only by a single model at a proper leaf but also by the different linear models generated for each of the internal nodes up to the root node. Smoothing affects the simplicity of the solution, making it more difficult to understand and interpret.

4. Experimental validation

In this section, three sets of experiments are presented. First, we would like to share some details of the *GMT* evaluation. Next, we validate the overall performance of the *GMT* solution with respect to predictive accuracy, build time, and tree and model size. The results are confronted with popular greedy counterparts on a number of large datasets. Finally, we compare the *GMT*, with its baseline denoted as *bGMT* (straightforward application of *EA*), and the solution called *E-Motion* [2], which also applies evolutionary algorithms to model tree induction. Table 1 provides an overview of the performed experiments, the number of datasets, the algorithms, and the tested elements.

4.1. GMT parameters

In all the experiments reported in this paper and in all datasets, we used one default set of attributes in the *GMT*. The only exception is in the first set of experiments, where the evaluation of the *GMT* was performed. In that case, all parameters except the one being tested remained at default.

In all the experiments reported in this paper, the population size was 50. The probability of the mutation of a single node in an individual equals 0.8, and the probability of crossover between two individuals equals 0.2. The probability of the

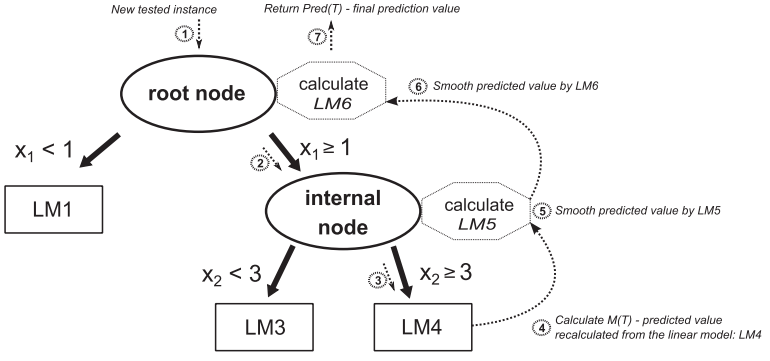


Fig. 6. The smoothing process for the test instances at the leaf with the linear model denoted as *LM*4.

Table 1
An overview of the *GMT* experimental validation.

Settings	Performed experiments		
Type	Evaluation of GMT	GMT vs Greedy	GMT vs EA
No. of datasets	2	26	8
No. of algorithms	–	9	6
Tested elements	Initialization	RMAE	RMSE
	Fitness function	Time	MAE
	Representation	Tree size	Tree size
	Smoothing	Model size	

crossover is small because it has highly destructive power when applied and completely changes the context of modified parts of the trees. On the other hand, it strongly differentiates populations and is crucial for *EA* to successfully escape from local optima. Mutation has a much smaller impact on the individuals, as it modifies only one node in the individual at a time. Therefore, the probability of mutation is set much higher, as it should be applied more often. The verification of these settings was performed on a number of varied datasets, and the results suggest that mutation was in the range of 0.7–0.9 and crossover in the range of 0.1–0.2.

Fig. 7 illustrates the *GMT* results for the *Housing* dataset, which is used to evaluate the *GMT* solution in the first part of the experiments. The illustration on the left shows the performance of the best individual found so far in the *GMT* evolution under different probabilities of mutation and crossover. To keep the image clear and readable, only a few tested settings were enclosed. The illustration on the right of Fig. 7 shows the results on the testing set. We can observe the impact of mutation and crossover and see that the suggested settings strongly improve the speed of the *GMT* convergence.

The *GMT* system applies the *mix* initialization strategy (with pruning), in which the probability of choosing the test search strategies *LS*, *LAD*, and *dipolar* are equal to 0.25, 0.25, and 0.5, respectively. Genetic operators in the *GMT* use all three sets of mutation and crossover variants. Smoothing for the *GMT* is enabled by default, and the smoothing constant k is equal to 10.

The default fitness function for the *GMT* is *BIC*, and the number of model parameters $k(T)$ for tree T equals $2 * (Q(T) + W(T)$, where $Q(T)$ is the number of terminal nodes and $W(T)$ is the sum of the number of attributes in the linear models in the leaves. The threshold values in *Lex* were defined for each measure separately. There is a tolerance of

- 10% of an average value of $RSS(T_1)$ and $RSS(T_2)$ for the RSS_{tr} threshold.
- 40% of the average value of $Q(T_1)$ and $Q(T_2)$ for the Q_{tr} threshold.
- 50% of the average value of $W(T_1)$ and $W(T_2)$ for the *Wtr* threshold.

The proportion between RSS_{tr} and Q_{tr} is similar to the one proposed in the $E - Motion$ solution [2]. Let us investigate a simple example in Table 2 that illustrates the logic behind the choice of such threshold values. We can observe that for the first iteration of Alg. 1, all measures for T_1 and T_2 fall one by one within the tolerance threshold, and there is a need to decrease all

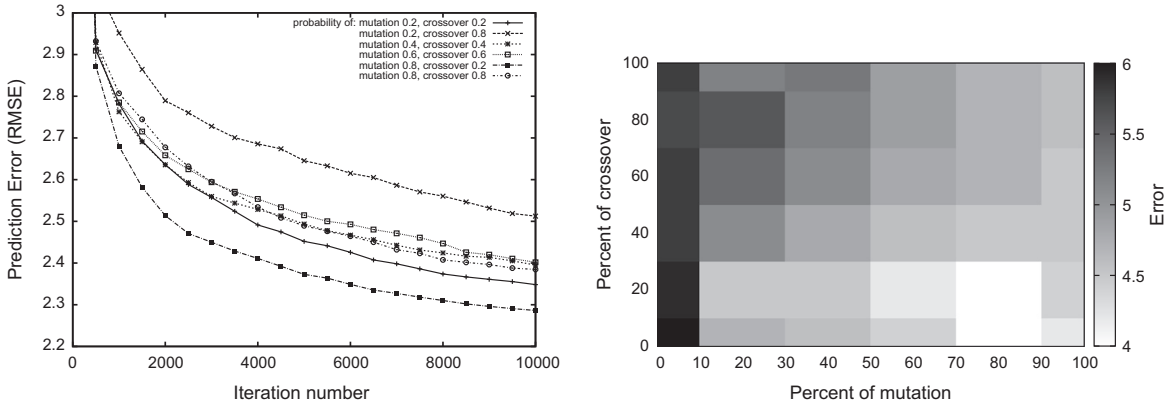


Fig. 7. Results for the *GMT* with different probabilities of mutation and crossover on the *Housing* dataset. The image on the left illustrates the results for the best individual on the training set during the evolution. The image on the right illustrates the results on the testing set.

Table 2
An example of lexicographic analysis with the justification of the measure threshold values illustrated for two individuals: T_1 and T_2.

Measure	T_1	T_2	$T_1 - T_2$	$AVG(T_1 + T_2)$	Threshold (iter. 1)	Threshold (iter. 2)
RSS	3.7	4.0	0.3	3.85	RSS_{tr} = 0.38 (10%)	RSS_{tr} = 0.19 (5%)
Node Count (Q)	10	7	3	8.5	Q_{tr} = 3.40 (40%)	–
Attribute Count (W)	12	8	4	10	W_{tr} = 5.00 (50%)	–

threshold values. In the second iteration, *RSS* between T_1 and T_2 does not fall within the tolerance threshold; therefore, the algorithm returns individual T_1 as having a smaller tree error. In addition, we performed non-exhaustive tests searching for different values of thresholds. The experiments showed that these settings provide a good selection of the best trees.

Setting how often the genetic operator should affect a single individual (or two individuals in the case of crossover) is a very difficult task. Operators are not independent, as they influence each other. Often, the effect of one genetic operator on a particular individual can be achieved by the application of several different variants of mutation and/or crossover. Even if we disable one variant of the genetic operator, it is possible that other operators will take over part of its role. In addition, each dataset may have different optimal settings. The Free Lunch Theorem [43] states that higher performance of *EA* over one, particular problem cause an equally reduced performance over some other problems. Therefore, in the case of *EA*, the key to good results is to provide the *EA* with various tools (specialized operators) and let it find the optimal solution. In the *GMT*, we propose a set of probabilities for selecting one genetic operator that can be applied in single crossover or mutation. Extensive experiments (not included) showed that the application of different probabilities did not significantly change *GMT* performance. The genetic operator settings for *bGMT* and the *GMT*, which were applied in all the experiments in this paper, are illustrated in Table 3.

4.2. Evaluation of the GMT

In the previous section, a set of solutions and techniques that incorporate knowledge about the model tree induction and EAs was proposed. Now, each technique will be verified separately to confirm the approach.

4.2.1. Datasets and settings

The impact of proposed improvements on the *GMT*'s evolution process is presented in two sample datasets: *Abalone* (4177 instances, 7 numeric, and 1 nominal attributes) and *Housing* (506 instances and 13 numeric attributes) from the UCI Machine Learning Repository [4]. Each dataset was divided into a training set (66.6% of observations) and a testing set (33.4%). The datasets greatly differ in the context of finding an optimal or near-optimal solution. The *GMT* algorithm can find good solutions for the *Abalone* dataset with only a few dozen iterations, while for the *Housing* dataset, a few thousand iterations are usually not enough.

In the case of evolutionary algorithms, it is difficult to show the impact of one specific factor on the entire evolutionary process. Each aspect of EA may not be independent and can influence other parameters. To minimize the effect of some suboptimal choices, since factors may interact in a complex way, an average of 20 runs is shown. For each experiment, we used the default *GMT* settings, except for the parameters that were being tested. For example, when the initialization strategies or smoothing were tested, the representation and fitness function remained the same. To improve visualization in Fig. 9, different algorithm names (*GMT BIC* and *Smoothed GMT*) refer to the same *GMT* solution.

4.2.2. Initialization strategies

In Section 3.2, we propose some techniques for initializing the population. Fig. 8 compares the performance of the best individuals in each generation for different strategies. We show the results of the best individual found so far during the evolutionary induction on the training set. The left axis shows the actual tree size of the individual, and the right axis illustrates

Table 3
Probability of selecting a single operator in the *bGMT* and *GMT*.

Genetic operator	*Type*	*Probability in*:	
		bGMT	*GMT*
exchange subtrees	Basic crossover	100	20
exchange tests	Specialized crossover	0	20
exchange branches	Specialized crossover	0	20
asymmetric	Specialized crossover	0	20
with best	Specialized crossover	0	20
prune	Basic mutation	15	10
random expand	Basic mutation	20	5
nominal disjunction	Basic mutation	15	15
new random test	Basic mutation	20	5
change model	Basic mutation	30	15
shift threshold	Specialized mutation	0	5
new dipolar test	Specialized mutation	0	5
dipolar expand	Specialized mutation	0	5
parent with son (branches)	Specialized mutation	0	2.5
parent with son (tests)	Specialized mutation	0	2.5
recalculate models	Specialized mutation	0	2.5
new optimal test	Specialized mutation	0	2.5
optimal expand	Specialized mutation	0	5
clear model	Specialized mutation	0	10
optimal model	Specialized mutation	0	10

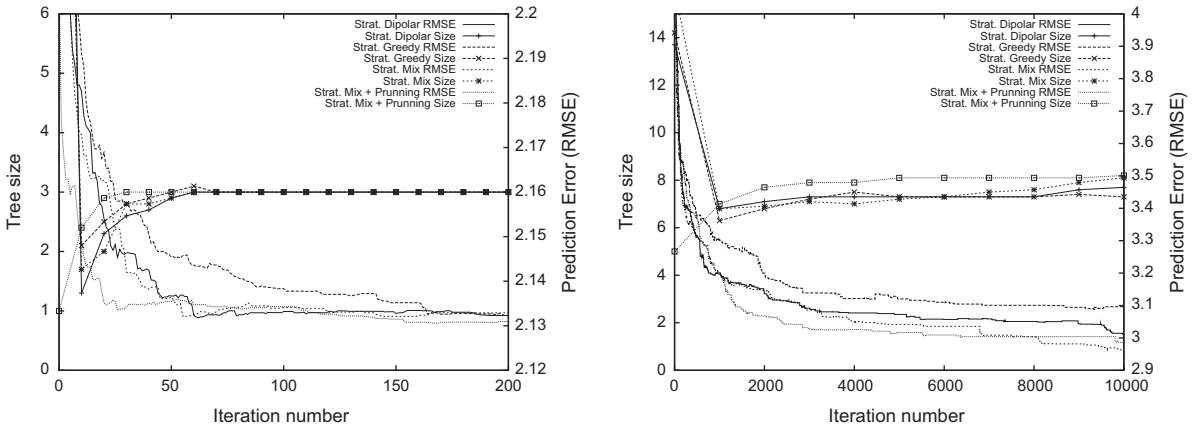


Fig. 8. Influence of the test strategies and initial pruning on the convergence of the best individual to the global optimal for the training sets of the *Abalone* (left) and *Housing* (right) datasets.

its Root Mean Squared Error (*RMSE*). We validate three strategies: *dipolar*, where locally optimized tests use only dipoles; *greedy*, composed of *LS* and *LAD* tests; and *mix*, which applies both strategies (*dipolar* and *greedy*). In addition, we show the results for the *mix* strategy with pruning enabled.

We can observe that the *mix* strategy with pruning doubled the speed of the EA's convergence on the global optima in both datasets. When the *greedy* strategy is used, the *GMT* needs many more iterations to achieve the same results as the proposed strategy.

4.2.3. Variants of the fitness function

In this set of experiments, we compared three fitness functions: improved Akaike information criterion (AIC_c) [23]; Bayesian information criterion (*BIC*) [37]; and lexicographic analysis, which is denoted as *Lex*, as proposed in Section 3.3. In the case of AIC_C, the number of model parameters $k(T)$ for tree T equals $2 * (Q(T) + W(T)$ (the same as for *BIC*).

Fig. 9 presents the *GMT* results for different fitness on the testing sets. We compare the *GMT* prediction error (*RMSE*) and the tree size of each variant: BIC, AIC_c, and *Lex*. There are no significant differences between *BIC* and *Lex* in terms of the *RMSE* and the tree size, and both functions outperform AIC_c. Additional investigation of the AIC_c results showed that the AIC_c error in the training set was the lowest (compared to *BIC* and *Lex*), and this, together with the large size of the induced trees, may suggest over-fitting the *GMT AICc* to the data.

4.2.4. Goodness of representation

Model trees can have various representations. To show the impact of the different *GMT* leaf representations, the *GMT* was tested with:

- multiple linear models in the leaves (denoted as regular *GMT*).
- simple linear models in the leaves (denoted as *GMT SLR*).
- mean value (regression variant) in each leaf (denoted as *GMT REG*).

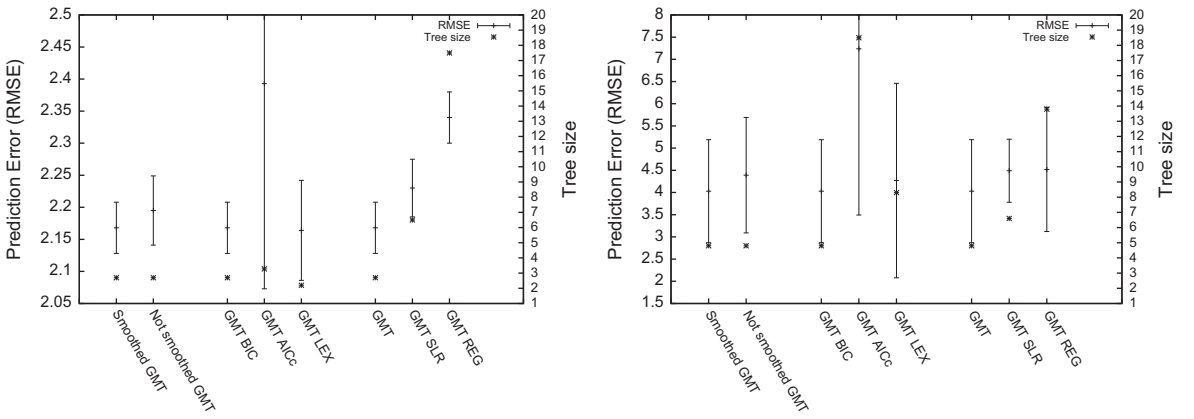


Fig. 9. Results of prediction error and tree size on testing sets *Abalone* (left) and *Housing* (right) for the different variants of the *GMT*. The error bars represent 95% confidence intervals.

We can observe that there exists significant differences between the different types of tree representations. The corrected paired t-test [30], with a significance level of 0.05, showed differences in terms of the tree size between all solutions and in terms of *RMSE* between the proposed *GMT* and *GMT REG* in the *Abalone* dataset. In general, for the same datasets, the *GMT REG* and *GMT SLR* needed much larger trees. We can also observe that for larger trees and more difficult regression problems, the differences between the variants of the *GMT* usually increase.

4.2.5. Smoothing

Application of smoothing was beneficial in both analyzed datasets. Fig. 9 compares the *GMT* with smoothing enabled and disabled. The tree size of both algorithms is always equal, as the smoothing operation is applied after the *EA* algorithm is finished and the best individual has been found. The only difference is the calculation of the prediction value and, as a consequence, the error of the model tree. In future works, running the smoothing inside the *EA* should be considered, as this would indirectly allow the *GMT* to use linear models in the non-terminal node.

4.2.6. Discussion

In this set of experiments, we verified the proposed techniques and analyzed the impact of each factor separately. We should not expect statistically significant differences between algorithms that differ with regard to only one element. The only exception was the differences between *GMT* representations. The strength of the proposed techniques can be truly verified when all factors work together. Therefore, in the next two sets of experiments, we compare the proposed *GMT* solution with the greedy approaches and with evolutionary competitors that apply straightforward *EA* for decision tree induction.

4.3. GMT vs greedy approaches

In order to accurately validate the performance of the *GMT* solution, it has been compared with algorithms from the paper [31] and with a few other greedy *GMT* counterparts. Thanks to Prof. Pfahringer, who provided us with the preprocessed datasets he used in [31], we were able to confront the prediction accuracy, efficiency, and complexity of the output models from the algorithms he tested.

4.3.1. Datasets and setup

The datasets were originally provided by Louis Torgo [40] and the UCI repository, and they were later preprocessed [31] (e.g., the categorical values were replaced with multiple binary indicator attributes and the missing values were inputted using the respective attributes mean value) to ensure that different internal algorithm procedures did not impact the comparison. Each dataset was split into three sets: training, validation (for internal parameter optimization), and testing sets. Table 4 presents the details of each dataset.

All test results reported in the next section correspond to averages of ten runs and were obtained using independent test sets. Relative Mean Absolute Error (*RMAE*) is used as a measure of prediction accuracy to maintain compliance with [31]. When *RMAE* equals 0%, the prediction error also equals zero. When the prediction always returns a global mean, the *RMSE* value equals 100%. This way, *RMAE* can be compared in a meaningful way across different datasets. In the performed experiments, we also report the total run-times and the complexity measure (characterized by the size of the tree and the average model size in the leaves).

4.3.2. Comparison algorithms

The performance of the *GMT* is confronted with several popular systems:

- Random Model Trees (*RMT*) – combination of model trees with random forests [31].
- Optimised Gaussian Process Regression (*GP*) [34] with radial basis function kernels and a conjugate gradient descent solver.

Table 4
Dataset characteristics: name, numeric attributes number (Num), nominal attributes number (Nom), and the number of instances.

Name	Num	Nom	Instances	Name	Num	Nom	Instances
stock	9	0	950	*pol*	48	0	15,000
quake	3	0	2178	*elevators*	18	0	16,599
abalone	7	1	4177	*cal housing*	8	0	20,640
delta ailerons	5	0	7129	*house 16H*	16	0	22,784
bank32nh	32	0	8192	*house 8L*	8	0	22,784
bank8FM	8	0	8192	*2dplanes*	10	0	40,768
cpu act	21	0	8192	*fried*	10	0	40,768
cpu small	12	0	8192	*mv*	7	3	40,768
kin8nm	8	0	8192	*layout*	31	0	66,615
puma32H	32	0	8192	*colorhistogram*	31	0	68,040
puma8NH	8	0	8192	*colormoments*	8	0	68,040
delta elevators	6	0	9517	*cooctexture*	15	0	68,040
ailerons	40	0	13,750	*elnino*	9	0	178,080

- Bagged Additive Groves of Trees (*AG*) [38].
- Linear (Ridge) Regression (*LR*).
- REPTree (*REP*) – popular top-down inducer that builds a regression tree using variance and prunes it using reduced-error pruning (with backfitting).
- *M*5 – state of the art model tree [39], the most adequate greedy counterpart of the *GMT*.
- Boosting M5 (*BO*) – Stochastic Gradient Boosting [19].
- Bagging M5 (*BG*) – ensembles of the *M*5 model tree.

The results of the first three systems were obtained from [31]. Each algorithm had several tuning parameters that were optimized on validation sets. The next five systems were tested using the *Weka* system [21]. For each algorithm, we collected the *RMAE*, the runtime, the size of the tree (in the case of *LR*, it is equal to 1), and the model size in the leaves (in the case of *REP*, it is equal to 1). Parameter tuning on the validation set was not performed for *Weka* algorithms, and each algorithm was run with its default *Weka* settings.

Appropriate parameter settings are important for evolutionary algorithms [14] and have a significant impact on *GMT* induction. However, in this paper, we wanted to show that the *GMT* can obtain good results with the default values of parameters through all datasets. Therefore, we did not improve the *GMT* results by tuning the parameters on the validation sets for each data separately. As with the *Weka* tested algorithms, we used only the training and the testing sets and skipped the validation set.

4.3.3. Performance results

Fig. 10 illustrates the average *RMAE* for all tested algorithms on all datasets, sorted by the *GMT* results. The lower value of *RMAE* indicates a better output model. We can observe that the *GMT* outperforms all single-tree counterparts. The average *RMAE* value in the datasets is smaller in comparison with *REP* and *M*5 by 24% and 9%, respectively. Linear regression was no match for any of the tested algorithms, as it does not work well for non-arbitrary regression problems, particularly when the sample size is large.

When we compare the *GMT* to the methods that built much more complex models, such as Gaussian Process Regression or ensembles of trees (Boosting, Bagging, Random Model Trees, and Additive Groves), it is difficult to point out the best algorithms. However, more detailed comparison reveals that the "small winner" is the *GMT*, as its average *RMAE* was 2% smaller than *BO*, 7% smaller than *RMT* and *BG*, 8% smaller than *AG*, 11% smaller than *GP*, and 42% smaller than *LR*. It should be noted that the proposed solution was also the most stable one, with no occasional catastrophic failures, such as *RMT* and *GP* on *puma32H* or *AG*, *BO*, and *BG* on *colorhistogram* and *layout*. This is surprising because meta-learning methods are usually more stable and easily outperform single-tree solutions in the context of prediction accuracy.

We have also investigated the poor *GMT* result for the *kin8nm* dataset. Although the result was satisfying in comparison with *BG*, *M*5, *REP*, and naturally *LR*, it was behind the algorithms tested in [31], in particular *GP*. However, by changing only one parameter in the *GMT*, which is the terminal condition (we increased the maximum number of *EA* iterations), we managed to achieve scores similar to *RMT* and *BO*. In this particular case, the *GMT* has too slow a convergence, and therefore the *EA* was stopped too early.

4.3.4. Efficiency results

It is known that the evolutionary approach is not the fastest, and the *EA* applied in the *GMT* is not an exception. The results illustrated in Fig. 11 show that the *GMT* is usually one order of magnitude slower than Additive Groves and, for some datasets, as much as two orders of magnitude slower than Gaussian Process Regression, Bagging, or Boosting. However, we must remember that the evolutionary induction process is progressive; therefore, intermediate answers can be harvested at any time, and pre-maturely aborted runs may also yield high-quality results.

The fastest algorithms are the greedy ones: REPTree and linear regression. Surprisingly, *RMT*, which is a combination of model trees with random forests, achieved comparable times to the single-tree greedy *M*5 method. However, we do not know the specifications of the machine for that *RMT* or the rest of the algorithms from [31] were executed; therefore, efficiency comparison results may be biased.

4.3.5. Complexity results

It is expected that a more global approach to decision tree induction may reduce the complexity of the tree, as with the examples from Section 2. Fig. 12 illustrates the size of the *REP*, *M*5, *LR*, and *GMT* trees, and Fig. 13 shows the average model size in the leaves. We do not have any information about the complexity of the models for the algorithms from [31] – that is, *RMT*, *GP* and *AG* – so we cannot show their complexity. However, we can assume that these algorithms are incomparably more complex than single-tree solutions. We also skip results for *Bagging* and *Boosting*, as they are meta-learning methods that use multiple *M*5-type trees.

Fig. 12 shows, on the logarithmic *y*-scale, the number of leaves in the tested algorithms (for linear regression, the size is equal to 1 through all datasets). We can observe enormous differences between locally and globally induced tree sizes. In most of the cases, the *GMT* is significantly smaller than the greedy counterparts, sometimes more than two orders of magnitude smaller. For all datasets, the average tree size for the *M*5 system, which is the most adequate greedy counterpart of the *GMT*, was over five times larger than the *GMT*.

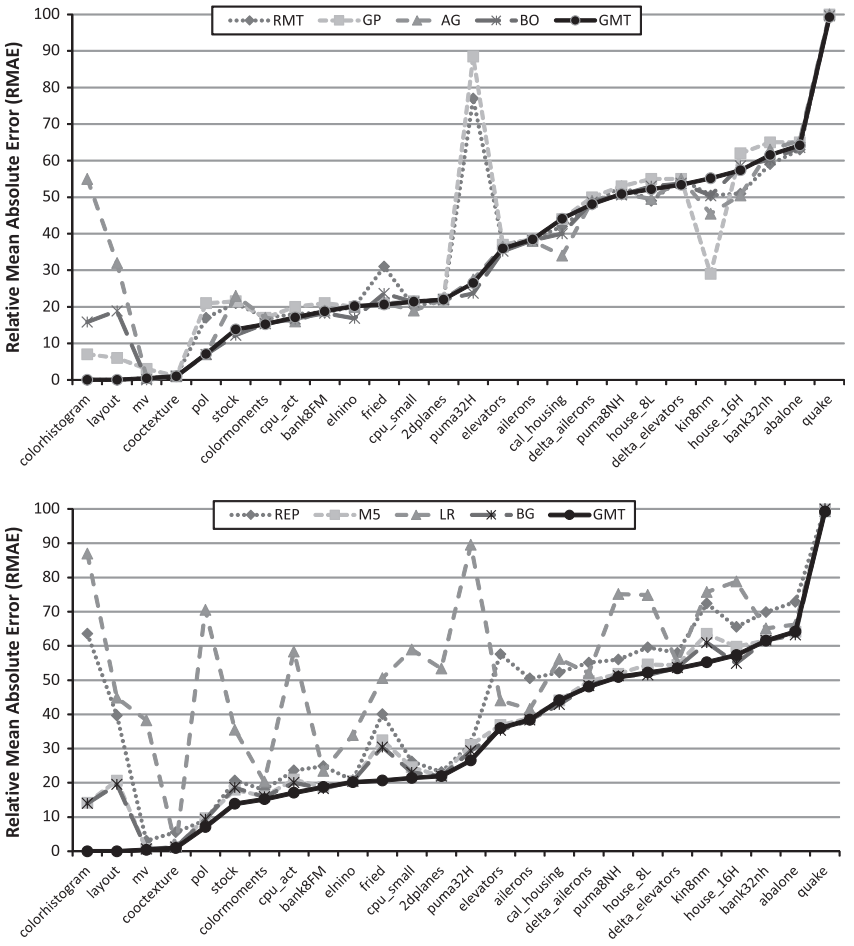


Fig. 10. Relative Mean Absolute Error of the algorithms sorted by the Global Model Tree (*GMT*) results. Top image: Random Model Trees (*RMT*), Gaussian Process (*GP*), Additive Groves (*AG*), Boosting M5 (*BO*), and the *GMT*; bottom image: REPTree (*REP*), *M*5, linear regression (*LR*), Bagging M5 (*BG*), and the *GMT*.

To have a full picture of the complexity results, we need to analyze Fig. 13, which illustrates an average linear model size that is equal to the number of attributes +1 (*REPTree* is a regression tree, so the model size is equal to 1 through all datasets). The simplicity of the linear regression models in the leaves is crucial for the model's understanding and interpretation, as it reveals the relationships between the attributes. We can see that in most of the cases, the *M*5 system has equal or slightly higher complexity than linear regression. The *GMT* managed to significantly decrease the number of attributes in the leaves and built on average models that were more than two times smaller for all datasets.

In this set of experiments, we proved that the *GMT* managed to find more compact prediction models. If we consider tree size and the size of the models in the leaves, then the *GMT* is, on average, over one order of magnitude less complex than the greedy counterparts. Even if for some datasets, such as mv or *cooctexture*, it generates larger trees, the average model in the leaves is much less complex. An opposing example occurred with the *colorhistogram* dataset, where the *GMT* used two more attributes to build a linear model than *M*5. However, the *GMT* induces a tree that has only one leaf (one model), whereas *M*5 needed 395 models, each with around 30 attributes. For the largest analyzed dataset *elnino*, the *GMT* was almost ten times smaller than *M*5 and had two times fewer attributes. Even the *REPTree* algorithm, which built regression trees with only a simple mean value in the leaves, can be considered more complex than the *GMT*. On the largest tested dataset, the *REPTree* had 2396 leaves, and the overall average tree size was 280 (almost 16 times greater than the *GMT*).

4.3.6. Discussion

In this set of experiments, we confronted globally induced model tree *GMT* with popular methods that apply greedy techniques. The purpose of this experiment was to show whether evolutionary induction performs better than greedy induction in terms of prediction accuracy and model complexity. We also wanted to know if a single-tree solution is capable of competing with more complex techniques. We have performed a statistical analysis of the obtained results using the Friedman test and the corresponding Dunn's multiple comparison test (significance level equals 0.05), as recommended by Demsar [12]. The results are illustrated in Table 5 and are denoted as follows: "+" means that the proposed solution is significantly

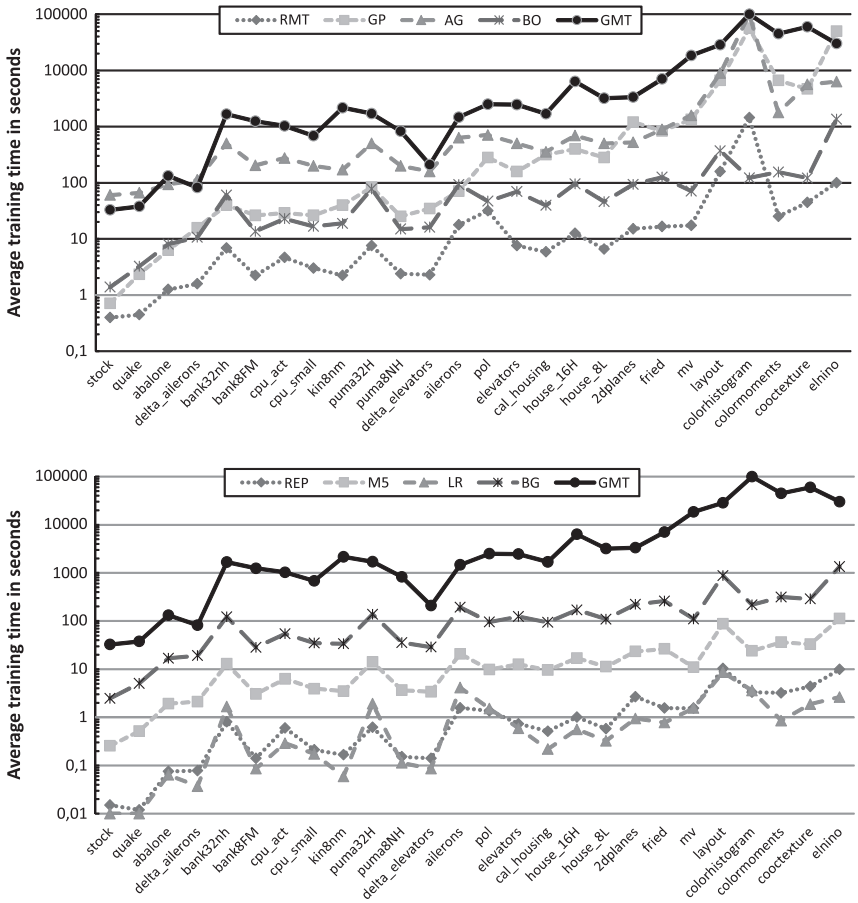


Fig. 11. Average training time in seconds sorted by the number of instances in each dataset. Top image: Random Model Trees (*RMT*), Gaussian Process (*GP*), Additive Groves (*AG*), Boosting M5 (*BO*), and Global Model Tree (*GMT*); bottom image: REPTree (*REP*), *M*5, linear regression (*LR*), Bagging M5 (*BG*), and *GMT*.

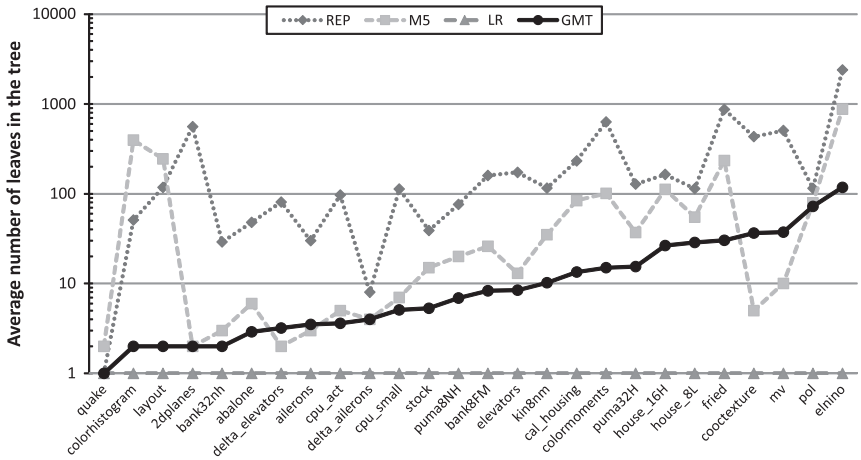


Fig. 12. Average number of leaves in the tree of REPTree (*REP*), *M*5, linear regression (*LR*), and Global Model Tree (*GMT*).

better, "−" that it is significantly worse, "·" that there are no statistical differences between the compared results, and "∗" that there might be a statistical difference. We do not know the complexity of the algorithms analyzed in [31], but we think that they build significantly more complex models than the *GMT*.

The *GMT* solution managed to be significantly better in terms of errors and model complexity than all single-tree models: *bGMT*, *M*5, *REP* and *LR*. Tests were performed on a number of different databases (26 datasets with an average of 16 attributes

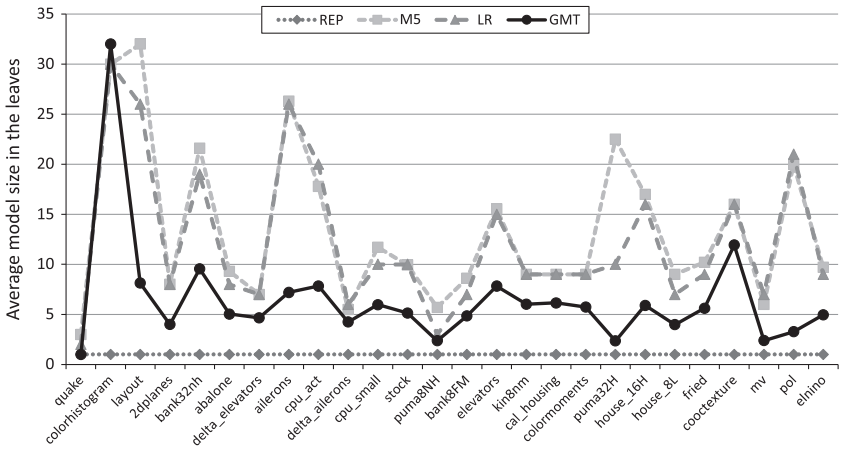


Fig. 13. Average model size in the leaves of the tree of REPTree (*REP*), *M*5, linear regression (*LR*), and Global Model Tree (*GMT*).

Table 5
Statistical differences between *RMAE*, time, and tree and model size between the *GMT* and the tested competitors. *GMT* is significantly better if the sign is +; significantly worse if the sign is −, shows no difference if the sign is ·. The differences are hypothetical if the sign is ∗.

	Algorithm	*RMAE*	Time	Tree Size	Model Size
GMT vs	*RMT*	·	−	∗+	∗+
	GP	+	−	∗+	∗+
	AG	·	−	∗+	∗+
	BG	+	−	+	+
	BO	·	−	∗+	∗+
	REP	+	−	+	−
	M5	+	−	+	+
	LR	+	−	−	+

and almost 30,000 instances each). Surprisingly, the *GMT* solution successfully competes with more advanced, state-of-the-art methods, such as Additive Groves (*AG*), Boosting *M*5 (*BO*), Bagging *M*5 (*BG*), Random Model Trees (*RMT*) or Gaussian Processes Regression (*GP*). Usually, these "black box" algorithms, with their much more complex models, easily outperform less advanced solutions. However, the *GMT* managed to significantly outperform *BG* and *GP* methods and to decrease the average errors calculated on all 26 datasets by 7% when compared to *RMT* and *AG*.

The price that *GMT* must pay for such performance is a much longer time for decision tree induction. This problem can be mitigated by parallelization of *EA*, and it is one of our priorities in future work. However, if an analyst does not need very fast real-time algorithms and can wait a bit longer, our algorithm can provide significantly better results that can offer new insights into underlying processes. To clarify, the execution time of the *GMT* solution on a test instance is very fast.

4.4. GMT vs evolutionary approaches

In the last set of experiments, we compare *GMT* with other evolutionary approaches for model tree induction. To complement our comparison results, we have confronted the *GMT* with the *E-Motion* solution [2] and the *GMT* baseline. A variant of the *GMT*, denoted as *bGMT*, is a straightforward implementation of *EA* to a model tree with a multiple linear regression model in the leaves and with random initialization of the population, basic variants of recombination, and no smoothing.

Table 6
Dataset characteristics: name, numeric attributes number, nominal attributes number, and the number of instances.

Dataset (DT) name	Abbreviation	Numeric	Nominal	Instances
Auto-Mpg	A	4	3	386
Breast tumor	B	1	8	286
Fish catch	F	5	2	158
Machine CPU	M	6	0	209
Quake	Q	3	0	2178
Stock	So	9	0	950
Strike	Sr	5	1	625
Veteran	V	3	4	137

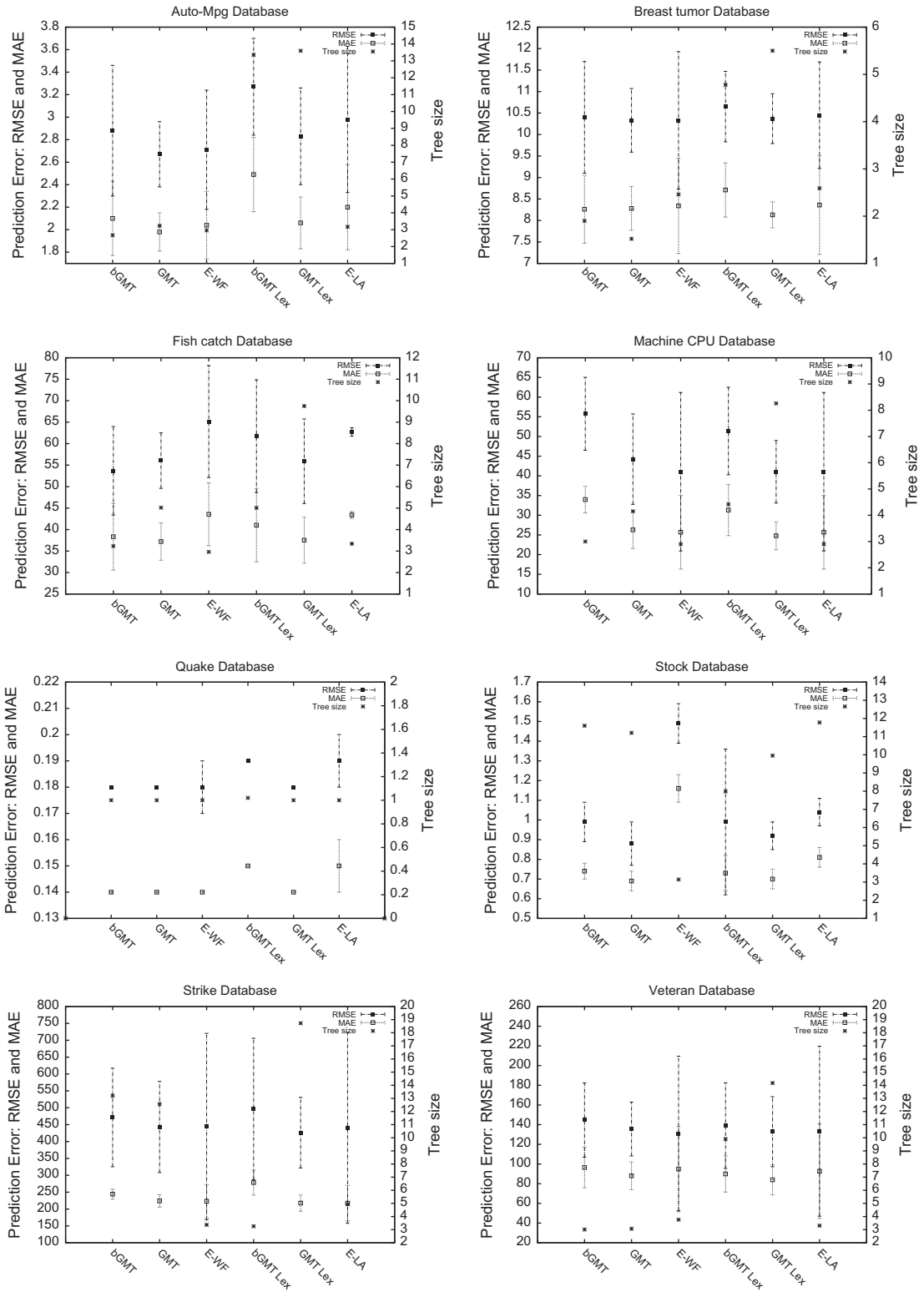


Fig. 14. *RMSE*, *MAE*, and tree size results for the *GMT* and its basic variant *bGMT*, *GMT Lex* and its basic variant *bGMT Lex*, and *E-Motion* with weight (*E-WF*) and lexicographic (*E-LA*) fitness functions. The error bars represent 95% confidence intervals.

Table 7
Statistical differences in prediction errors *RMSE* and *MAE* between algorithms with weight fitness functions (*GMT*, *bGMT*, and *E-WF*) and with lexicographic fitness functions (*GMT Lex*, *bGMT Lex*, and *E-LA*).

Weight fitness	Metric	*bGMT*	*E-WF*
GMT	*RMSE*	· · · M + ·So + Sr + ·	· · · · ·So + · ·
	MAE	· · F + · · So + · ·	· · F + · · So + · ·
Lexicographic fitness	Metric	*bGMT Lex*	*E-LA*
GMT Lex	*RMSE*	A + · · M + · · · ·	· · · · · So + · ·
	MAE	A + B + ·M + · · Sr + ·	· · F + · · So + · ·

4.4.1. Datasets and setup

To compare the *GMT* with *E-Motion*, the methodology in [2] was used, as the source code for the *E-Motion* algorithm was not available. The datasets from [2], which are presented in Table 6, were also tested in this experiment. Ten-fold cross-validation was used, and the results and the average of the ten folds for the 30 executions of each fold are shown. We have calculated the results for the *GMT* and its basic variant *bGMT* to show the benefits of the techniques proposed in this paper. As the *E-Motion* system shows the results for fitness based on lexicographic analysis (*E-LA*) and the weight formula (*E-WF*), to complete the comparison we show our results for the *GMT*, with the lexicographic fitness denoted as *GMT Lex*.

Unfortunately, we cannot compare the execution times of the *E-Motion* algorithm (results are not available) and the size of the linear models in the trees. This is not crucial in the comparison of *EA* solutions, as we can assume that both algorithms are relatively slow and induce comparable and much smaller trees than their greedy counterparts. However, the *E-Motion* algorithm does not evolve models in the leaves but simply builds linear regression models from the available attributes, as with the *M*5 algorithm. In this case, the average model size in the leaves of the tree for *E-Motion* should be similar to the *M*5 solution and therefore, considering the results in Fig. 13 and Table 5, significantly more complex than the ones generated by the *GMT*.

4.4.2. Results and discussion

In the performed experiment, we compared both prediction errors *RMSE* and *MAE* with 95% confidence intervals as well as tree sizes. The results for all tested algorithms are presented for each dataset separately in Fig. 14. To illustrate the statistical differences between the algorithms for each dataset, a corrected paired *t*-test [30] with a significance level of 0.05 and 9 degrees of freedom ($n-1$ degrees of freedom where $n=10$ folds) was applied. These settings are identical to the ones presented in the article that proposed *E-Motion* [2]. We have found a few statistically significant differences between the tested solutions. Table 7 illustrates the results in each dataset for the *EA* algorithms with weight and lexicographic fitness functions. The "·" sign denotes that there is no statistical significance between the algorithms, whereas the abbreviation of the dataset with the "+" sign denotes that the *GMT* or *GMT Lex* results were significantly better (there were no datasets in which the proposed algorithms obtained significantly worse results).

Incorporating the knowledge about model tree induction into the *GMT* and *GMT Lex* strongly improved the baselines algorithms *bGTM* and *bGMT Lex*. Depending on the metric, we managed to significantly reduce the prediction error in 25–50% of the analyzed datasets. Error reduction was also noticeable in the rest of the datasets.

In all tested datasets, the *GMT* and *GMT Lex* successfully competed with the $E-Motion$ solution, and in two out of the eight tested datasets (*Fish catch* and *Stock*) the *GMT* algorithm managed to obtain significantly better results. The performed experiments show that the *GMT* is at least as good if not better than the *E-Motion* algorithm, and thus also indirectly better than *GPMCC* [32] and *TARGET* [16], which were outperformed by *E-Motion*.

In this experiment, we did not focus much on the size differences between the output models, as there is a mix of statistically better and statistically worse results between algorithms, depending on the dataset. The differences were not as large as with the greedy algorithms, where the *GMT* managed to achieve smaller trees by one order of magnitude. However, it is interesting that the *bGMT* results differ from *E-Motion*, despite the fact that both solutions apply a straightforward *EA* approach. The differences in generating the initial population, fitness function, and perhaps most important, the ability of the *GMT* to evolve a linear model in each leaf (*E-Motion* used all available attributes to generate a multivariate linear model) may be the key to understanding these differences.

Comparison between the *GMT* and its baseline *bGMT* showed that there are significant differences in prediction accuracy in favor of the *GMT*. There are no significant differences in tree size; however, it should be noted that the average execution time over all tested datasets was almost 3 times smaller for the *bGMT* compared to the *GMT*, which equals around one minute. This is caused by the proposed specialized operators, particularly ones with additional local searches, which take longer than the basic cross-overs and mutations.

5. Conclusion

Greedy regression and model tree inducers are fast, white box solutions that usually have a slightly lower prediction accuracy when compared to the complex or ensemble-learning techniques. However, when applied to large datasets, they

often lose their important advantage – simplicity – and generate trees with hundreds or even thousands of leaves with regressions models that include dozens of explanatory attributes each. Such large trees are almost impossible to understand and interpret, and therefore ensembles of trees are preferred, but these intentionally sacrifice simplicity to improve predictive accuracy.

In this paper, a solution called the Global Model Tree (*GMT*) is proposed. It is a global approach for regression and model tree induction achieved through the application of *EA*s and the knowledge of the tree induction problem. We have designed a set of techniques that improve the straightforward applications of *EA*s. Although the *GMT* is not as fast as greedy inducers, it manages to remain simple even in large datasets. We have performed extensive experimental evaluations of the *GMT* and show the full potential of a specialized *EA* for model tree induction. An empirical investigation showed that the *GMT* can also successfully compete with ensembles of model trees, *EA* counterparts, and other complex solutions. Therefore, the *GMT* can be applied to a wide variety of problems where there is no space for trade-off between model complexity and prediction accuracy.

We see many promising directions for future research. In particular, we should consider parallelization of the evolutionary algorithm to speed-up its execution time and self-adaptive parameters in *EA* to improve the evolutionary convergence. We also plan to introduce oblique tests in the non-terminal nodes and more advanced models in the leaves. Extending multi-objective optimization to the Pareto-dominance approach is also being considered.

Acknowledgments

The authors thank Bernhard Pfahringer, who provide us with preprocessed datasets. This project was funded by the Polish National Science Center and allocated on the basis of decision 2013/09/N/ST6/04083.

References

[1] H. Akaike, A new look at statistical model identification, IEEE Trans. Auto. Control 19 (1974) 716–723.
[2] R.C. Barros, D.D. Ruiz, M. Basgalupp, Evolutionary model trees for handling continuous classes in machine learning, Inform. Sci. 181 (2011) 954–971.
[3] R.C. Barros, M.P. Basgalupp, A.C. Carvalho, A.A. Freitas, A survey of evolutionary algorithms for decision-tree induction, IEEE Trans. Syst. Man Cybernet. Part C Appl. Rev. 42 (3) (2012) 291–312.
[4] C. Blake, E. Keogh, C. Merz, UCI Repository of Machine Learning Databases, 2009 <http://www.ics.uci.edu/~mlearn/MLRepository.html>.
[5] L. Breiman, J. Friedman, R. Olshen, C. Stone, Classification and Regression Trees, Wadsworth Int. Group, 1984.
[6] C. Burgess, M. Lefley, Can genetic programming improve software effort estimation? A comparative evaluation, Inform. Softw. Technol. 43 (2001) 863–873.
[7] X.S. Chen, Y.S. Ong, M.H. Lim, K.C. Tan, A multi-facet survey on memetic computation, IEEE Trans. Evolution. Comput. 15 (5) (2011) 591–607.
[8] M. Czajkowski, M. Kretowski, Globally induced model trees: an evolutionary approach, in: Proceedings of PPSN XI, LNCS, vol. 6238, 2010, pp. 324–333.
[9] M. Czajkowski, M. Kretowski, An evolutionary algorithm for global induction of regression trees with multivariate linear models, in: Proceedings of ISMIS'11, LNCS, vol. 6804, 2011, pp. 230–239.
[10] M. Czajkowski, M. Kretowski, An evolutionary algorithm for global induction of regression and model trees, Int. J. Data Min., Model. Manage. 5 (3) (2013) 261–276.
[11] M. Czajkowski, M. Kretowski, Does memetic approach improve global induction of regression and model trees? in: Proceedings of ICAISC'12, LNAI, vol. 7269, 2012, pp. 174–181.
[12] J. Demsar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (2006) 1–30.
[13] A. Dobra, J. Gehrke, SECRET: a scalable linear regression tree algorithm, in: Proceedings of KDD'02, 2002.
[14] A.E. Eiben, R. Hinterding, Z. Michalewicz, Parameter control in evolutionary algorithms, IEEE Trans. Evolution. Comput. 3 (2) (1999) 124–141.
[15] F. Esposito, D. Malerba, G. Semeraro, A comparative analysis of methods for pruning decision trees, IEEE Trans. Patt. Anal. Mach. Intell. 19 (5) (1997) 476–491.
[16] G. Fan, J.B. Gray, Regression tree analysis using target, J. Comput. Graph. Statist. 14 (1) (2005) 206–218.
[17] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, R. Uthurusamy (Eds.), Advances in Knowledge Discovery and Data Mining, AAAI Press, 1996.
[18] A. Freitas, A critical review of multi-objective optimization in data mining: a position paper, SIGKDD Explor. Newsl. 6 (2004) 77–86.
[19] J.H. Friedman, Stochastic gradient boosting, Comput. Statist. Data Anal. 38 (1999) 367–378.
[20] P. Gagne, C.M. Dayton, Best regression model using information criteria, J. Mod. Appl. Statist. Meth. 1 (2002) 479–488.
[21] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten, The WEKA data mining software: an update, SIGKDD Explor. 11 (1) (2009).
[22] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning, second ed., Data Mining, Inference and Prediction, Springer, 2009.
[23] C.M. Hurvich, J.S. Simonoff, C.L. Tsai, Smoothing parameter selection in nonparametric regression using an improved Akaike information criterion, J. R. Statist. Soc. 60 (1998) 271–293.
[24] L. Hyafil, R.L. Rivest, Constructing optimal binary decision trees is NP-complete, Inform. Process. Lett. 5 (1) (1976) 15–17.
[25] M. Kretowski, M. Czajkowski, An evolutionary algorithm for global induction of regression trees, in: Proceedings of ICAISC'10, LNAI, vol. 6114, 2010, pp. 157–164.
[26] D. Malerba, F. Esposito, M. Ceci, A. Appice, Top-down induction of model trees with regression and splitting nodes, IEEE Trans. PAMI 26 (5) (2004) 612–625.
[27] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, third ed., Springer, 1996.
[28] P. Moscato, On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms, Caltech Concurrent Computation Program, Report 826, California Institute of Technology, Pasadena, CA, USA, 1989.
[29] S. Murthy, Automatic construction of decision trees from data: a multi-disciplinary survey, Data Min. Knowl. Discov. 2 (1998) 345–389.
[30] C. Nadeau, Y. Bengio, Inference for the generalization error, Mach. Learn. 52 (2003) 239–281.
[31] B. Pfahringer, Semi-random model tree ensembles: an effective and scalable regression method, in: Proceedings of AUS-AI'11, LNAI, vol. 7106, 2011, pp. 231–240.
[32] G. Potgieter, A. Engelbrecht, Evolving model trees for mining data sets with continuous-valued classes, Expert Syst. Appl. 35 (2008) 1513–1532.
[33] W.H. Press, B.P. Flannery, S.A. Teukolsky, W.T. Vetterling, Numerical Recipes in C, Cambridge University Press, 1988.
[34] C.E. Rasmussen, C.K.I. Williams, Gaussian Processes for Machine Learning, MIT Press, 2006.
[35] L. Rokach, O.Z. Maimon, Top-down induction of decision trees classifiers – a survey, IEEE Trans Syst., Man, Cybernet., Part C: Appl. Rev. 35 (4) (2005) 476–487.
[36] L. Rokach, O.Z. Maimon, Data mining with decision trees: theory and application, Mach. Percept. Artif. Intell. 69 (2008).

[37] G. Schwarz, Estimating the dimension of a model, Ann. Statist. 6 (1978) 461–464.
[38] D. Sorokina, R. Caruana, M. Riedewald, Additive groves of regression trees, in: Proceedings of ECML'07, LNCS, vol. 4701, 2007, pp. 323–334.
[39] J. Quinlan, Learning with continuous classes, in: Proceedings of AI'92, World Scientific, 1992, pp. 343–348.
[40] L. Torgo, Regression Data Sets <http://www.liaad.up.pt/ltorgo/Regression/DataSets.html>.
[41] L. Torgo, Functional models for regression tree leaves, in: Proceedings of ICML, Morgan Kaufman, 1997, pp. 385–393.
[42] D. Vogel, O. Asparouhov, T. Scheffer, Scalable look-ahead linear regression trees, in: Proceedings of 13th ACM SIGKDD, ACM Press, New York, 2007, pp. 757–764.
[43] D. Wolpert, W. Macready, No free lunch theorems for optimization, IEEE Trans. Evolut. Comput. 1 (1) (1997) 67–82.
[44] J.R. Woodward, GA or GP? That is not the question, in: Proceedings of IEEE CEC, 2003, pp. 1056–1063.