ABSTRACT
Software effort prediction has been a challenge for researchers throughout the years. Several approaches for producing predictive models from collected data have been proposed, although none has become standard given the specifics of different software projects. The most commonly employed strategy for estimating software effort, the multivariate linear regression technique has numerous shortcomings though, which motivated the exploration of many machine learning techniques. Among the researched strategies, decision trees and evolutionary algorithms have been increasingly employed for software effort prediction, though independently. In this paper, we propose employing an evolutionary algorithm to generate a decision tree tailored to a software effort data set provided by a large worldwide IT company. Our findings show that evolutionarily-induced decision trees statistically outperform greedily-induced ones, as well as traditional logistic regression. Moreover, an evolutionary algorithm with a bias towards comprehensibility can generate trees which are easier to be interpreted by the project stakeholders, and that is crucial in order to improve the stakeholder’s confidence in the final prediction.

Categories and Subject Descriptors
H.4 [Information Systems Applications]: Miscellaneous; D.2.8 [Software Engineering]: Metrics—complexity measures, performance measures

Keywords
software effort estimation, decision trees, evolutionary algorithms, LEGAL-Tree

1. INTRODUCTION
Software estimation plays an important role within software development projects. For instance, if software development effort is underestimated, the development team will suffer great pressure to finish the product rapidly, and thus the resulting software may not be fully functional or sufficiently tested. The resulting product may contain errors that need to be further corrected within the software life cycle, when the cost of corrective maintenance is higher. Conversely, if effort is overestimated, a great (and potentially unnecessary) deal of resources will be committed to the project. Too high an estimate may prevent the company of securing a contract due to the excessive calculated costs.

The need for accurate software estimates has motivated the investigation of many different strategies for providing predictive models efficiently and effectively. Many researchers addressed the problem by looking for strategies that could be eventually generalized for similar software projects. Specifically regarding software effort, the strategy employed in most of the cases involves the application of regression models for predicting continuous numeric values [10, 17, 20, 22]. These strategies often propose different heuristics in order to provide an accurate regression predictive model. For instance, Barros et al. [5] presented an empirical study on software effort estimation dealing with three distinct approaches: the expert feeling, least-square linear regression and data mining model trees. In this paper, instead of predicting effort as a continuous numeric variable, we propose transforming it to a discrete variable and later applying an algorithm that induces classification models. For that, we propose employing an evolutionary algorithm called LEGAL-Tree [6, 7] that evolves decision-tree models with a bias towards comprehensible trees.

This study was developed inside a worldwide IT Company which intends to improve the quality of its development process through better estimates. We hope the results presented here can serve as basis and guidance for building and deriving predictive models, contributing with the state of the art in the subject. We also hope the issues we have faced will be addressed in similar future projects.

This paper is organized as follows. In Section 2, we review related work that employ evolutionary algorithms for improving software estimation models. Section 3 presents LEGAL-Tree, an evolutionary algorithm for decision-tree induction [6, 7]. Section 4 presents the scenario on which was executed the case study. Section 5 details the experimentation plan we have adopted for guiding our experiments. In Section 6 we present the experimental analysis, in which we compare our approach to logistic regression, as well as to other decision-tree induction algorithms. We conclude the paper with some remarks on
what we have achieved and also some future work suggestion.

2. RELATED WORK

Software metrics estimation is a growing research area. Indeed, the diversity of estimation approaches is very high and increasing [16]. In this paper, we investigate the applicability of an evolutionary algorithm that evolves decision trees in the prediction of software effort in a maintenance project. Our goal is to produce a highly-accurate comprehensible predictive model. Preliminarily, we adopt the distinction suggested by Burgess and Lealay [10] between genetic algorithms (GA) and genetic programming (GP). According to the authors, “GP is an extension of GA, which removes the restriction that the chromosome representing the individual has to be a fixed-length binary string”. In this work, the chromosome is a fully-grown decision tree, which suggests it is best classified as a GP contribution.

We have searched for papers that employ either GA or GP for improving software estimation, and verified which kinds of models are induced, as well as the predictive performance of these models and whether they are comprehensible to the potential stakeholders (project managers).

Among the relevant papers which employ GP to estimate effort (or cost), [10] and [17] propose approaches that produce linear regression models as outputs, whilst the strategy presented in [22] employs GGGP (Grammar-Guided Genetic Programming) for deriving these expressions. Linear equations are fairly comprehensible models [10, 17], especially when they are of reduced size. Notwithstanding, the models generated for predicting effort in [10, 17] are not short of problems, since the improvement detected by the authors was heavily dependent on the metric used for evaluating the results. The GGGP-based expressions [22], on the other hand, presented significant improvement over the traditional linear and logistic regression, but at the expense of comprehensibility, since the strategy produced highly-complex models, which were hard to be understood by stakeholders.

Regarding the papers that employ GA, [14] and [20] apply GA to select and weight attributes for software effort prediction. The first uses Grey Relational Analysis (GRA) as the method to be optimized by GA, and the second makes use of Ridge Regression (RR). Even though both papers report gains with their respective approaches, the authors did not seem to be concerned with the comprehensibility of the induced models. The work of Sheta [23] presents an approach that employs a GA to estimate the COCOMO parameters, and proposes two new extensions of the original COCOMO, whose parameters are also optimized by a GA. The author states that his work presents a significant improvement over the original COCOMO strategy.

The work of Mantere and Alander [18] focuses on the use of evolutionary algorithms in software testing. It is a review of the application of evolutionary computation to software engineering, and the authors make a detailed discussion regarding the use of GA in software testing.

The work that shows some resemblance to the approach we present in this paper is [1]. The authors employ a GA to evolve a set of decision rules for software effort prediction. They compare their findings with the well-known decision-tree system C4.5 [21], reporting an improvement in accuracy. Nevertheless, we propose evolving decision trees and not decision rules. In addition, we encode the individuals as trees, not as a fixed-length binary string. Table 1 summarizes the related work.

To the best of our knowledge, this is the first work to employ an evolutionary algorithm for evolving decision trees tailored to a particular software project data set. Our main contribution is to present an algorithm that can provide highly-accurate comprehensible decision trees for predicting software effort. Also, we show that our approach improves not only the accuracy but also the comprehensibility of the models produced, when compared to traditional decision-tree induction algorithms. Moreover, we show that our results are statistically significant based on a well-known statistical test. We make use of an effort data set from a software project with more than four years of length. Next, we present LEGAL-Tree, which is the evolutionary algorithm we employ for predicting software effort.

3. THE LEGAL-TREE ALGORITHM

LEGicographical Genetic Algorithm for Learning decision Trees (LEGAL-Tree) [6] is an evolutionary algorithm that evolves decision trees in order to avoid the problems inherent to the traditional divide-and-conquer greedy strategy usually employed for inducing decision-trees. The next sections summarize the main steps of the algorithm. For more details, we suggest reading [6,7].

3.1 Solution Representation

While GA applications generally rely on binary strings for representing each individual, LEGAL-Tree adopts the tree representation, because it seems logical that if each individual represents a decision tree, the solution is best represented as a tree. Therefore, each individual is encoded as a set of nodes, with each node being either a non-terminal
or a terminal (leaf) node.

Each non-terminal node contains an attribute, and each leaf node contains a predicted class. A set of edges linking each node with their respective children is also a part of the tree representation. Each link is associated with a set of values of the attribute associated with the node in its origin. Figure 1 shows an example of an individual (decision tree). We can see the case of numeric node-linking by looking at the root node, which contains the numeric attribute X. Nominal node-linking is also presented in Figure 1 by the node which contains nominal attribute Y.

3.2 Generating The Initial Population

Even though most evolutionary algorithms rely on purely random initialization of trees, LEGAL-Tree makes use of decision stumps to populate the first generation of trees.

A decision stump is the simplest case of decision trees.
It consists of a single decision node and two predictive leaves [13]. Such a concept was extended for categorical attributes, where each edge that represents an attribute category (value) will lead to a predictive leaf. A set of decision stumps is generated as follows. LEGAL-Tree divides the training data set into 10 different folds. For each fold, it calculates a different split threshold (for the case of numeric attributes). By dividing the training data set in 10 different pieces, LEGAL-Tree strikes for achieving a certain degree of heterogeneity for the decision stumps involving numeric attributes, because they will be essential in the generation of the initial population. Such process of generating decision stumps is far from being random, since the threshold for numeric attributes is deterministically chosen. For more details on decision stumps generation please refer to [7].

3.3 Lexicographic Multi-objective Fitness Evaluation

LEGAL-Tree follows the assumption that knowledge discovery should be not only accurate but also comprehensible to the user [24], and so it considers not only a quality measure (i.e., accuracy or F-Measure), but also a complexity measure (tree size) for biasing the search space towards simpler (smaller) solutions (decision trees). For such, LEGAL-Tree implements a lexicographic multi-objective fitness evaluation.

The lexicographic fitness evaluation works as follows. Let \( x \) and \( y \) be two decision trees, and let \( a \) and \( b \) be two evaluation measures. In addition, suppose that \( a \) has the highest priority between them and that \( t_a \) and \( t_b \) are tolerance thresholds for \( a \) and \( b \), respectively. The lexicographic approach works according to the following analysis: if \( |a_x - a_y| > t_a \), then it is possible to establish which decision tree is “better” considering only the measure \( a \). Otherwise, the lower-priority measure \( b \) must be evaluated. In this case, if \( |b_x - b_y| > t_b \), then the fittest tree considering \( x \) and \( y \) can be decided only by considering the measure \( b \).

If the difference between values still falls within the assigned threshold \( t_b \), the best value of the higher-priority measure \( a \) is used to determine the fittest tree. For more details on the pros and cons of different multi-objective approaches, we recommend reading [11, 12].

3.4 Genetic Operators

For selecting individuals in the current generation, LEGAL-Tree uses tournament selection, a popular and effective selection method. It also implements the elitism technique, preserving a percentage of the best individuals based on their fitness values. The number of tournament participants and the percentage of elitism are user-defined parameters. At each round of the tournament selection, the participants are compared two by two through the lexicographic fitness, and the individual with the highest number of victories is selected to be part of the reproduction pool. When there is a draw in the number of victories, the individual with the highest absolute value of validation accuracy is selected. This procedure is repeated until the pool is full.

LEGAL-Tree implements the crossover operation as a typical one-point crossover in trees, where two individuals swap randomly selected sub-trees. By exchanging the whole sub-trees from the selected node, and not only specific nodes, it avoids problems like attribute domain irregularities, because each edge refers to attribute characteristics that are represented by a node. It does not prevent, however, redundant rules and inconsistencies. See [7] for details on how LEGAL-Tree addresses such issues.

LEGAL-Tree implements two different strategies for mutation in individuals. The first strategy considers the exchanging of a whole sub-tree, selected randomly from an individual, by a leaf node representing the most frequent class attribute value among the examples covered by that leaf. The second strategy involves the decision stumps that were created during the initial population generation process. This strategy replaces a randomly selected leaf node in an individual by a decision stump previously created.

4. SCENARIO DESCRIPTION

The experiment was conducted within the largest software maintenance project of Hewlett-Packard Enterprise Application Services (HP-EAS) Brazil. The project’s name and all classified information were changed or suppressed in the following description. Project P1 is a continuous maintenance project that inserts, modifies and removes functionalities in a software product previously developed by the company. The software product is divided into versions, where each version corresponds to a single iteration in the maintenance lifecycle. An iteration must deliver a number of labor-hours/month to the client and may have
one or more product releases. Each Service Order (SO) aggregates a set of change requirements for the product. Project P1’s teams are composed by a design team, a client developers team, a server developers team and a testers team. It has three types of requirement documents: (1) Design Requirements, (2) Server Requirements, and (3) Interface/Client Requirements. These documents have well defined structures divided into sections. Typically, such sections are:

- **History of Changes Header:** present in all document types; records all changes in documents;
- **Description:** present in all document types; briefly describes the document;
- **Business Rules:** in Design Requirements document, describes the software functionality; in Server and Interface/Client Requirements, documents business rules pseudo-codes;
- **Screen layouts:** present only in Interface/Client Requirements, contains screen designs and descriptions of their behavior rules.

We observed that project P1 uses a classic waterfall lifecycle. Some tools are used to support the team’s activities during the lifecycle. MS-Project Server is used to schedule the activities planned in the project. IBM-ClearQuest is used to defects control and IBM-RequisitePro manages all the requirements presented in an SO. The effort spent by each team associated to address a task is recorded using a locally-developed software tool. Finally, the estimation strategy of the Company for this particular project is based on the expert judgment only.

### 5. EXPERIMENTAL PLAN

The Company is CMM level 3 certified and has a central data repository for all its projects, as recommended in [15]. This repository is directed to the high-level managers in order to assist their decision-making process, and it is detailed in [8].

The granularity level of data in the organizational repository does not allow its use to estimate work effort for each requirement, because such data are summarized by phase and version, in an approach that disposes of important project details. Consequently, we had to access directly the data sources of the tools adopted by the project. We accessed the data repository from ClearQuest, RequisitePro, and the locally-developed software tool, so as to retrieve the following SO information: (i) adjusted function-points for each SO; (ii) size of a SO in thousands of lines of code; (iii) number of requirement documents for each SO; (iv) whether there were previously performed test-cases; (v) baseline schedule; and (vi) actual effort of each SO in hours.

Considering the information above, we have created a SO dataset consisting of 1522 instances, each instance representing a unique SO of project P1, and 18 attributes. We encountered a couple of problems with this dataset. First, we have identified quality problems with data prior to 2006. That is because until 2006, the metrics collection process was not automated, generating data distortions and many attributes with missing values. Hence, we have deleted all instances that dated of years that preceded 2006. In addition, we have also identified instances with missing values of work effort. These instances, for no apparent reason, did not contain the values of effort for a given SO, even though such a SO was already terminated. Thus, we have deleted these instances, because they can no longer aggregate significance in the building of a prediction model.

After this data cleaning process, we have ended up with 561 instances. Since our target attribute (work effort) is measured in hours, we need to transform its values from numeric to nominal in order to create a classification problem (instead of a regression problem). For that, we divided the data so the class values were equally distributed among the classes. The classes created were: low effort (less than 18 hours), medium effort (between 18 and 67 hours) and high effort (more than 67 hours). The class distribution is 188 instances classified as low, 187 classified as medium and 186 instances classified as high effort.

For assessing the quality of the decision tree generated by LEGAL-Tree, we used 10-fold cross-validation, a widely disseminated approach for validating classification models. In each of the 10 iterations of the cross-validation procedure, the training set is divided into sub-training and validation sets (each one representing 50% of the full training set). Thus, for each iteration we have three subsets: training, validation and test set.

We configured LEGAL-Tree with 3 different measures to be evaluated in the fitness function. From the highest to the lowest priority measure, we considered, respectively, F-Measure of the validation and training sets, and tree size. We set the lexicographic thresholds for these measures as 2%, 2% and 3 nodes, respectively. The parameters used on LEGAL-Tree are presented in Table 2.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial population max depth</td>
<td>3</td>
</tr>
<tr>
<td>Population size</td>
<td>500</td>
</tr>
<tr>
<td>Improvement rate</td>
<td>3%</td>
</tr>
<tr>
<td>Max number of generations</td>
<td>500</td>
</tr>
<tr>
<td>Tournament rate</td>
<td>0.6%</td>
</tr>
<tr>
<td>Elitism rate</td>
<td>5%</td>
</tr>
<tr>
<td>Crossover rate</td>
<td>90%</td>
</tr>
<tr>
<td>Mutation rate</td>
<td>5%</td>
</tr>
</tbody>
</table>

We compared the results provided by LEGAL-Tree to a logistic regression method, as well as to well-known decision-tree induction algorithms, namely J48 and SimpleCART (versions of C4.5 [21] and CART [9] available on the free machine learning tool Weka [24]), and also to BFTree, also available on Weka. All parameter settings used are the algorithms’ default ones. It is important to mention that the 10-fold cross-validation procedure was executed with the same subsets (training, validation and test set) for all algorithms.

For each algorithm’s execution, we measured classification accuracy, F-Measure, recall, precision, and tree size. Due to the fact that GA is a non-deterministic technique, we run LEGAL-Tree 30 times for each one of the 10 training/test set folds generated by the 10-fold cross-validation procedure. After running LEGAL-Tree over the data set, we calculated the average and standard deviation of the 30 executions for
Table 3: Accuracy, F-Measure, Precision, Recall and Tree Size of all decision tree induction algorithms. Average and standard deviations are shown.

<table>
<thead>
<tr>
<th></th>
<th>Logistic</th>
<th>J48</th>
<th>CART</th>
<th>BFTree</th>
<th>LEGAL-Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>0.61</td>
<td>0.62</td>
<td>0.67</td>
<td>0.60</td>
<td>0.64</td>
</tr>
<tr>
<td>F-Measure</td>
<td>0.61</td>
<td>0.61</td>
<td>0.62</td>
<td>0.61</td>
<td>0.63</td>
</tr>
<tr>
<td>Precision</td>
<td>0.63</td>
<td>0.63</td>
<td>0.64</td>
<td>0.61</td>
<td>0.65</td>
</tr>
<tr>
<td>Recall</td>
<td>0.61</td>
<td>0.62</td>
<td>0.68</td>
<td>0.61</td>
<td>0.64</td>
</tr>
<tr>
<td>Tree Size</td>
<td>39.10</td>
<td>160.32</td>
<td>55.40</td>
<td>1133.16</td>
<td>84.80</td>
</tr>
</tbody>
</table>

For assessing the validity and non-randomness of the experimental results, we employed the corrected paired t-test [19] with a significance level of $\alpha = 0.10$ and 9 degrees of freedom ($k - 1$ folds). The null hypothesis is that the algorithms achieve the same performance.

6. RESULTS AND DISCUSSION

Table 3 presents the results for logistic regression and the decision-tree induction algorithms employed in the experiment (J48, Cart, BFTree and LEGAL-Tree). Five different measures were collected from each algorithm, namely accuracy, F-Measure, precision, recall, and tree size (except for logistic regression, which does not generate a decision tree).

Firstly, we can notice that LEGAL-Tree outperforms all other methods regarding the absolute values of every single collected measure. This is a preliminary indication that the evolved decision tree generated by LEGAL-Tree is the most accurate and comprehensible model among the tested algorithms. In order to confirm this assumption, we proceed to analyze the results of the corrected paired t-test [19]. Table 4 depicts these results.

We can observe that LEGAL-Tree outperforms BFTree and SimpleCart with statistical significance, in terms of accuracy, F-Measure and recall. We can also observe that LEGAL-Tree significantly outperforms all other methods (J48, SimpleCart and BFTree) in terms of tree size, i.e., the trees generated by LEGAL-Tree are significantly smaller than the ones generated by the other decision-tree algorithms. This supports the statement that LEGAL-Tree’s fitness function is biased towards more comprehensible decision trees. The fact that LEGAL-Tree’s model is significantly more comprehensible than the other methods is a key finding, bearing in mind the importance of comprehensible predictive models in a wide range of applications, let alone in software effort prediction. By analyzing a comprehensible predictive model, the project manager can make informed decisions on resource allocation for the development of new maintenance SOs.

Even though no significant difference was found between LEGAL-Tree and logistic regression, we could easily recommend the adoption of LEGAL-Tree due to the higher comprehensibility of its model. The reader can refer to Figure 2 to verify how easy is to interpret one of LEGAL-Tree’s generated decision tree. The logistic regression, on the other hand, generated a model with more than 30 distinct coefficients.

7. CONCLUSIONS

In this paper, we proposed the application of an evolutionary algorithm called LEGAL-Tree for evolving decision trees tailored to a worldwide IT Company effort data set. We showed that the global search of an evolutionary algorithm can be more effective than traditional greedy algorithms for decision-tree induction. Furthermore, we argued that with a proper fitness function, we can bias the search space of the evolutionary algorithm in order to look for small-sized comprehensible decision trees.

In an experiment that involved more than 4 years of collected data from an IT Company maintenance project, we showed that an evolutionary-based decision tree can outperform well-known and established approaches for decision-tree induction, as well as traditional logistic regression. This improvement was often statistically
significant according to a widely-employed statistical test. Moreover, the evolutionary-based strategy generated more comprehensible models than all other methods.

We believe our findings indicate the applicability of evolutionary algorithms for decision-tree induction in the context of software effort prediction. In addition, considering the increasing growth in the application of evolutionary-based decision trees [2], we intend to determine its effectiveness in estimating other important software metrics (e.g., software quality), in private and public software development data sets. We also believe an evolutionary algorithm that generates regression/model trees [3,4] could be effectively employed for software metrics prediction.

8. ACKNOWLEDGMENTS

The authors would like to thank Fundaçãão de Amparo à Pesquisa do Estado de São Paulo (FAPESP) for funding this research. In addition, the authors would like to acknowledge HP-EAS Porto Alegre, for providing all data employed in this paper.

9. REFERENCES


