Improving GP Classification Performance by Injection of Decision Trees

Rikard König, Ulf Johansson, Tuve Löfström and Lars Niklasson

Abstract—**This paper presents a novel hybrid method combining genetic programming and decision tree learning. The method starts by estimating a benchmark level of reasonable accuracy, based on decision tree performance on bootstrap samples of the training set. Next, a normal GP evolution is started with the aim of producing an accurate GP. At even intervals, the best GP in the population is evaluated against the accuracy benchmark. If the GP has higher accuracy than the benchmark, the evolution continues normally until the maximum number of generations is reached. If the accuracy is lower than the benchmark, two things happen. First, the fitness function is modified to allow larger GPs, able to represent more complex models. Secondly, a decision tree with increased size and trained on a bootstrap of the training data is injected into the population. The experiments show that the hybrid solution of injecting decision trees into a GP population gives synergetic effects producing results that are better than using either technique separately. The results, from 18 UCI data sets, show that the proposed method clearly outperforms normal GP, and is significantly better than the standard decision tree algorithm.**

I. Introduction

Knowledge Discovery in Databases is an interactive, iterative procedure that attempts to extract implicit, previously unknown useful knowledge from data [1]. Often Knowledge Discovery boils down to classification, i.e., the task of training some sort of model capable of assigning a class from a predefined set of labels to unlabeled instances. The classification task is characterized by well-defined classes, and a training set consisting of pre-classified examples [2]. Accuracy on the unlabeled data is usually the main goal when training a model, but most decision makers would require at least a basic understanding of a predictive model to use it for decision support [3], [4], [5]. Comprehensibility is normally achieved by using high-level knowledge representations. A popular one, in the context of data mining, is a set of IF-THEN prediction rules [6]. It is, however, also important to realize that even if a model is transparent, it is still not comprehensible if it is larger than a human decision maker can grasp. Hence, creating reasonable sized models is also an important goal when comprehensibility is required.

This work was supported by the Information Fusion Research Program (www.infofusion.se) at the University of Skövde, Sweden, in partnership with the Swedish Knowledge Foundation under grant 2003/0104.

R. König, U. Johansson and T. Löfström are with the School of Business and Informatics, University of Borås, Allégatan 1, 501 90 Borås, Sweden (e-mail: rikard.konig@hb.se, ulf.johansson@hb.se, tuve.lofstrom@hb.se)

L. Niklasson is with the School of Humanities and Informatics, University of Skövde, Box 408, 541 28 Skövde, Sweden (e-mail: lars.niklasson@his.se)

In the data mining community, decision tree algorithms are very popular since they are relatively fast to train and produce transparent models. Greedy top-down construction is the most commonly used method for tree induction. Even if greedy splitting heuristics are efficient and adequate for most applications, they are essentially suboptimal [7]. More specifically, decision tree algorithms are suboptimal since they optimize each split locally without considering the global model. Furthermore, since finding the smallest decision tree consistent with a specific training set is a NP-complete problem [8], machine learning algorithms for constructing decision trees tend to be non-backtracking and greedy in nature [9]. Hence, due to the local non backtracking search, decision trees may get stuck in local minima.

An alternative to the greedy search is to globally optimize the model using some evolutionary technique, e.g., Genetic Programming (GP) [10]. GP has, however, a well known weakness called *bloating*, i.e., the size of evolved trees may grow out of control. Bloating is a serious problem since large programs are computationally expensive to evolve, will be hard to interpret, and tend to exhibit poor generalization [11]. It should be noted, on the other hand, that a certain growth may be necessary to find a more complex solution. Normally, bloating is handled by incorporating some kind of size related punishment in the fitness function, see e.g. [12],[13]. A down side to this approach is that the size punishment will enforce an upper bound for the tree size, even if no exact size limit is set. Hence, different settings for the size punishment is often used to evolve smaller or larger trees, see e.g. [12], [13].

Theoretically, every point in the search space has a nonzero probability of being sampled; but for most problems of interests the search space is so large that it is impractical to wait long enough for guaranteed global optimums [15]. Larger trees with more nodes naturally lead to larger search spaces since the attributes and possible splits can be combined in more ways. Therefore a higher size punishment could also be used to restrict the size of the evolved tree and thereby keep the search space within practical limits.

Since smaller rules are more comprehensible, the choice of length punishment becomes a crucial design choice. On one hand, a size punishment favoring smaller trees may result in a sub-optimal accuracy since larger complex rules may not be evolved. On the other hand, if larger trees are favored, it may result in an unnecessary complex solution or a suboptimal solution due to the very large search space. Based on the discussion above, this paper presents a hybrid

method combining decision trees and GP. The method not only automatically adjusts the size punishment to an appropriate level, but also guides the GP process towards good solutions in the case of an impractically large search space.

II. Background

This section will first present the main decision tree algorithms, and some approaches aimed at overcoming the problems caused by the greedy search. After that, classifier systems based on genetic programming will be discussed, followed by a section describing hybrid methods.

A. Decision Trees

A decision tree algorithm typically optimizes some information theoretic measure, like information gain, on a training set. The generation of the tree is done recursively by splitting the data set on the independent variables. Each possible split is evaluated by calculating the *purity gain* it would result in if it was used to divide the data set D into the new subsets $S=\{D_1, D_2,\ldots,D_n\}$. The purity gain is the difference in purity between the original data set and the subsets as defined in equation 1 below, where $P(D_i)$ is the proportion of D that is placed in D_i. The split resulting in the highest purity gain is selected, and the procedure is then repeated recursively for each subset in this split.

$$gain(D,S) = purity(D) \sum_{i=1}^{s} P(D_i) * purity(D_i) \qquad (1)$$

There are several different decision tree algorithms, two of the more well-known are C4.5 [16] and CART [17]. Slightly different purity functions are used, C4.5 optimizes entropy E, (equation 2) while CART optimizes the *gini index* (GDI in equation 3.) In the equations below, C is the possible classes, p is the estimated class probability and t is the current tree node.

$$E(S) = \sum_{i=1}^{c} p_i \log\left(\frac{1}{p_i}\right) \qquad (2)$$

$$GDI = 1 - \sum_{i=1}^{c} p_i^2(t) \qquad (3)$$

Compared to optimizing GDI, entropy tends to lead to smaller and purer nodes, which is favorable for problems with a clear underlying relationship, but inferior when the data contain a lot of noise or are missing a real relationship [18].

When no splits improving purity can be found, the tree needs to be *pruned* to remove overly specific nodes to improve the generalization ability of the tree. Pruning is typically performed by choosing the best sub-tree based on the error rate on an unseen validation data set.

1) Improving suboptimal trees

Many researchers have tried to improve decision tree performance by considering several sequential splits instead of only the next. However, most studies have shown that this approach generally fails to improve the performance, and that it even may be harmful, see e.g. [7], [19].

There have also been several attempts to improve suboptimal decision trees using a second stage where the tree is modified using another search technique. An example of this approach is [20], where a tree is first created using fuzzy logic search. In the second stage, the terminal nodes in the tree are adjusted to be optimized on the whole training set. Other examples are [21] where a suboptimal decision tree is optimized using dynamic programming and [22] where multi-linear programming is applied in a similar way. It should be noted, however, that even if these algorithms improve the suboptimal decision trees, they are not truly using global search, since they are dependent of the initial structure of the tree.

2) Genetic Programming for Classification

Normally a full atomic representation is used for GP classification. An atomic representation uses atoms in internal and leaf nodes [9]. Each internal atom represents a test consisting of an *attribute,* an *operator* and a *value,* where the *operator* is a Boolean function. Leaf nodes contain atoms representing a class of the predicted attribute.

GP classification representations can be divided into the Michigan and Pittsburgh approaches [6]. In the Michigan approach, each individual encodes a single prediction rule, whereas in the Pittsburgh approach each individual encodes a set of prediction rules.

Another important issue to consider is that conventional GP is based on the basic assumption of *closure* [10]. To achieve closure, the output of any node in a GP tree must be able to handle all possible parent nodes. This typically becomes a problem when a dataset contains both categorical and continuous attributes since they need to be handled by different functions.

One way to handle this is to use *constrained syntactic structures,* see [23], where a set of rules defines allowed sub nodes for each non terminal function. These rules are then enforced when creating new trees and during crossover and mutation. Another slightly more flexible solution is *strongly typed GP* [24], which instead defines the allowed data types for each argument of each non-terminal function and the returned types of all nodes.

B. GP classifier systems

GP classification has been used successfully in many applications for a survey see [25]. A common setup is to use the Pittsburg approach with a full atomic representation. Most often the Boolean functions <, = are used as operators while class assignment of the target attribute is used as terminals. A typical fitness function minimizes both some error metric and the size of the tree, thus maximizing accuracy and comprehensibility, see equation 4

$$Fitness = w_1 ErrorRate + w_2 Simp \qquad (4)$$

where w_1 and w_2 are two user-defined weights and *Simp* is a measure of the rule simplicity (comprehensibility). Normally

Simp is inversely proportional to the number of conditions in the rule, i.e., the shorter the rule, the more comprehensible it is [6]. Variations of the fitness function that follow this general form can be found in, for instance, [12],[26],[27],[28].

C. Extensions of standard GP-classification

Since the search space for classification tasks tends to become extremely large [9], several methods for improving classification performance of GP systems have been suggested.

In [29], a GP-system that uses stepwise adaptation of weights (SAW) is presented. The basic idea is to dynamically assign higher weights for errors on "hard" instances, in an online fashion. However, the experiments show that the suggested SAWing approach fails to consistently reach a higher performance, compared to standard GP classification.

In another study [9], machine learning techniques are used to refine the GP search space. One method applies *K-means clustering* to reduce the huge number of possible atoms. Another approach uses C4.5's entropy gain criterion (calculated on the whole data set) to create *n* new partitions that are used instead of the normal atomic representation. Both methods show promising results compared to normal GP and C4.5. A problem for both techniques is, however, that their performances are clearly dependent on certain non-trivial parameter values, i.e., the number of clusters used in K-means and the number of partitions.

A classifier system based on the Michigan representation is used in [30], where several classification rules are evolved. Each rule consists of enumerations of attribute value pairs, combined with only two Boolean functions, *AND* and *NOT*. If more than one rule covers an instance, the best rule will be used for classification. GPC, as the method is called, also uses a covering algorithm to calculate the overlap of different rules and penalize redundant rules. Experiments on nine data sets show that the algorithm performs comparably to other classifiers, despite the simple representation.

D. Hybrid Systems

The flexibility of GP makes it easy to combine it with other techniques in hybrid systems. Hence, numerous variations of GP classification systems exist, including decision trees, artificial neural networks, fuzzy logic and support vector machines. The following section, will only present some typical hybrid methods for creating decision trees, since decision trees is the targeted technique in this study.

In [31], decision trees are used to create the initial GP population. More specifically, each individual in the population is created using C4.5. To achieve some diversity, each C4.5 tree is created from a random subset of the original training set. In the reported experimentation, the suggested method performs remarkably well, compared to the other techniques. This is very surprising since CART reaches the same training accuracy but has almost 20% lower test accuracy. Since both techniques use almost identical representations, and the results are averaged over five folds, more similar accuracies should be expected. On one fold, both CART and the hybrid method achieve 100% accuracy on training data, but on the test data CART only reaches 69% compared to 90% for the hybrid method. Furthermore, the benefit of using C4.5 as creation method is hard to evaluate based on this study alone, simply because the results for comparable standalone runs of normal GP and C4.5 are missing, i.e., there is no way of knowing if GP increased the accuracy of the original rules.

A similar method also utilizing C4.5 to create the initial population was presented in [32]. The aim of this study was to produce short and comprehensible rules. This was motivated by the statement that C4.5 produced much shorter and comprehensible rules but could not be trained incrementally. The suggested method created the initial population in the same way as the previous study but used C4.5 settings that created smaller trees. Experiments were performed on a digit recognition data set, where the proposed method produced smaller and slightly more accurate trees, compared to a normal GP implementation. However, when the number of generations increased, the GP trees achieved the same accuracy as the hybrid method. Furthermore, a survey of 13 articles performed in [25], showed that GP produced more comprehensible rules in 66% of 60 analyzed occasions.

Genetic algorithms (GA) can be used in a similar way to GP, even if the representation of arbitrary sized rules is not as straightforward. In [33], a GA and decision tree hybrid is proposed to handle small disjuncts in classification problems better. The motivation is that since the general belief is that large disjuncts capture generalization better, most rule induction algorithms have a bias favoring the discovery of large disjuncts. Although each disjunct only covers a few examples, a set of disjuncts can cover a large part of the data set. The proposed method first uses C4.5 to find the large disjuncts, and then the remaining instances are classified using GA to create rules that cover the small disjuncts. Rules are represented using the Michigan approach and rule conditions consist of attribute-value pairs. Experiments performed on the *Adult* data set from the UCI machine learning repository [34] showed that the hybrid method was considerably better than only using a C4.5 tree.

Another approach to simplifying the creation of a decision tree is *rule extraction*. Rule extraction techniques are designed to handle the so called accuracy versus compressibility tradeoff, i.e., the fact that most powerful techniques like artificial neural networks, support vector machines and ensembles, all are opaque and impossible to interpret. The basic idea of rule extraction is to use the prediction of an accurate but opaque model as the target instead of the original class value when building a transparent model. The core idea is that the powerful technique can generalize and create a classification for the training instances that has less noise than the original data.

Genetic Rule Extraction (G-REX) is a rule extraction technique based on genetic programming. In several studies, e.g. [13],[35], G-REX have been shown to outperform standard decision tree algorithms such as C4.5 and CART, both in terms of accuracy and comprehensibility. Disregarding the rule extraction feature, G-REX is a typical GP classification system based on the Pittsburg representation and constrained syntactic structures.

III. METHOD

The following section will present a novel hybrid technique called DT Injection GP (DTiGP). The method starts with estimating a benchmark level for reasonable accuracy, based on decision tree performance on bootstrap samples of the training set. Next, a normal GP evolution is started with the aim of producing an accurate GP. At even intervals, the best GP in the population is evaluated against the accuracy benchmark. If the GP has higher accuracy than the benchmark, the evolution continues as normal until the maximum number of generations is reached. If the accuracy falls below the benchmark, two things happen. First, the fitness function is modified to allow larger GPs able to handle more relationships. Secondly, a decision tree with increasing size (trained on a bootstrap of the training data) is injected into the population.

The overall purpose of the method is to automatically force the GP to search for models of suitable size; i.e., as small as possible but not overly simplified. Because of this, after the threshold for reasonable accuracy is established, the GP starts out with small trees and a high length penalty, but is allowed to search for increasingly larger trees until the accuracy is higher than the threshold. In more detail, this is accomplished by decreasing the length penalty and by injecting accurate models into the population. Once the level of acceptable accuracy is met by the GP, normal GP optimization is used for the rest of the search.

As described above, the method is consists of three parts: estimating a reasonable level of accuracy, evolving GP trees and conditionally injecting decision trees.

Each part will be described in detail and motivated in the sections below. Evaluation of the method is done against normal GP, the decision tree algorithm used for the injected trees and J48.

A. Estimating Reasonable Accuracy

To obtain as comprehensible trees as possible, a high size punishment will initially be used to favor small solutions regardless of the data set. A smaller punishment could lead to unnecessary complex and specialized trees that may generalize poorly, when the data contains noise. However, some data sets require a large complex solution and do not contain noise. In these cases, a more complex solution is actually required. A typical example is the UCI dataset Tic-Tac-Toe, which encodes the complete set of possible board configurations at the end of tic-tac-toe games. Since no noise exists, and there is a true underlying concept, a larger solution will most often perform better.

However, since the true complexity of the solution is not known, it is impossible to directly set the length punishment in an optimal way. Instead, the proposed method will dynamically lower the size punishment, until a reasonable accuracy is achieved. Several GP runs, with varied size punishments, could of course be used to find the best setting, but since GP is computational expensive, this is normally not practical. Decision trees are, however, fast to train and usually perform quite well.

To get a good approximation of the performance, 10 bootstrap samples are first drawn from the training set. Training instances not selected for a specific bootstrap are instead used as an out of bag test set for that bootstrap. Next, decision trees are created using jaDTi [36], an open source implementation based on the C4.5 algorithm. The main difference from the real C4.5 algorithm is that jaDTi uses pre pruning based on an entropy threshold, and a score threshold instead of the usual sub tree based post pruning of C4.5. The entropy threshold is set to 0.5 and the score threshold is set to 0.01 * the *size* of the data set, i.e., the number of instances in the data set. When all trees have been created, both average training and test accuracies are calculated. However, since reasonable training accuracy is sought, the mean test accuracy would be a too pessimistic value, and the mean training accuracy would be too optimistic. Instead, the average of these values is used as the benchmark for reasonable training accuracy for the GP system.

B. Evolving Decision Trees

Since this study is a comparison of evolutionary classifier systems and decision tree algorithms, the Pittsburgh approach is the most suitable in view of the fact that each individual represents a single decision tree. Furthermore, a fair comparison also demands that the GP system uses the same internal representation, i.e., IF - THEN rules. The GP implementation of DTiGP is based on an open source framework for evolutionary data mining presented in [26]. GP settings used in the experiment are presented in Table I.

TABLE I
GP SETTINGS

Population size	200
Creation type	Ramped Half and Half
Selection type	Roulette wheel
Number of Generations	100
Crossover probability	80%
Mutation probability	1%

Equation 5 below defines the fitness function that is minimized in the experiments.

$$fitness = ErrorRate + ErrorRate * size * w \qquad (5)$$

ErrorRate is the percentage of misclassified instances, *size* is the number of nodes in the tree and w is a weight used to

balance the size punishment. Since it is the percentage of misclassified records that is minimized, the settings of *w* is independent of the number of instances in the data set. Note that the size punishment also is scaled with the error rate to make the size more important for less accurate trees.

Initially, *w* is 0.01, to ensure that small comprehensible rules are evolved. However, every five generations, the training accuracy of the best individual is compared to the benchmark for reasonable accuracy. If the best individual has a lower accuracy than the benchmark, *w* is decreased ten percent, i.e. *w=w*0.9,* otherwise the evolution continues as normal. As mentioned above the motivation for this adjustment is to allow larger more complex trees when necessary.

C. Injection of Decision Trees

If the search space is very large, and especially if the population has been evolved with a high size punishment, it can sometimes be hard for the GP search to find a good solution. To help the GP process in these situations, new individuals created using jaDTi will be injected in to the population. Similar to how the size punishment is handled, the best individual will be evaluated against the benchmark for reasonable training accuracy every 15th generation. If the best individual performs worse than the benchmark, a random tree in the population will be replaced with a jaDTi tree. However, larger jaDTi trees would often have a very high performance, compared to the GP population since they are optimized on the training set and could easily become dominant in the population. To avoid introducing unnecessary large trees, the injected jaDTi will have an ascending size, starting with compact trees. The tree size is decided by stepwise adjusting the score threshold of jaDTi from 0.05 to 0.001. The score threshold is then multiplied with the *size* of the dataset in the same way as when estimating reasonable accuracy. To make it easier to control the size of the produced trees, the influence of the entropy threshold is minimized by setting it to 0.0005. Even if different bootstrap samples and score thresholds are used for each tree, they may still become quite similar due to the deterministic nature of the decision tree search. Hence, to give the jaDTi trees some extra diversity they are mutated (using the standard GP mutation operation) before being injected into the population.

IV. Evaluation

The experimentation used 18 UCI data sets and 4-fold cross validation. Standard two tailed sign tests were used to test for significance in the result. To ensure a fair evaluation, all techniques were implemented in WEKA [37]. The reported results are the average accuracies over all folds. Comparisons were made against jaDTi, normal GP and WEKA's J48. The results of jaDTi and normal GP are reported to evaluate if the hybrid combination is better than just using the techniques separately.

In the experimentation, normal GP uses the same settings and fitness function as DTiGP, except from using a lower weight for the size punishment, (*w=0.001*). A lower weight is used since DTiGP is supposed to start looking for really short and comprehensible solutions, while normal GP must try to find accurate and reasonably sized tree.

J48 is included in the experiments since it is a well-known decision tree algorithm, and therefore should be a good benchmark. J48 is like jaDTi based on the C4.5, but uses the original post pruning, in difference to jaDTi. J48 is executed with the standard settings in WEKA.

A. Data sets

The experiments are performed on 18 well known data sets publically available from the UCI machine learning repository [38]. For a summary of the characteristics of the data sets, see Table II. *Size* is the number of instances in the data set. *Classes* is the number of output classes in the data set. *Num.* is the number of numeric attributes, and *Nom.* is the number of nominal attributes.

TABLE II
CHARACTERISTICS OF DATA SET

Data set	Size	Classes	Num.	Nom.
Breast-cancer	286	2	0	9
Breast-w	699	2	9	0
Colic	368	2	7	15
Credit-a	690	2	6	9
Credit-g	1000	2	7	13
Cylinderbands	540	2	20	19
Diabetes	768	2	8	0
Ecoli	336	8	7	0
Glass	214	7	9	0
Haberman	306	2	2	1
Heart-cleveland	303	2	6	7
Heart-statlog	270	2	5	8
Hepatitis	155	2	6	13
Iris	150	3	4	0
Labor	57	2	8	8
Liver-disorders	345	2	6	0
Lymph	148	4	3	15
Tic-tac-toe	958	2	0	9

V. Results

Table III shows the average test accuracies over 4 folds for each of the tested techniques. The best result for each dataset is marked with bold letters. DTiGP performs best and have the highest score (with three split victories) on 10 of the 18 datasets. Normal GP is the second best technique, with 7 overall wins, including two split victories. DTiGP also has the highest mean accuracy, followed by normal GP, J48 and jaDTi.

TABLE III
AVERAGE ACCURACY OVER FOUR FOLDS

Data set	jaDTi	GP	DTiGP	J48
Breast-cancer	68.8%	69.6%	**71.7%**	69.9%
Breast-w	93.6%	94.6%	**94.7%**	95.6%
Colic	75.4%	73.9%	82.6%	**86.0%**
Credit-a	82.2%	**86.7%**	86.4%	85.5%
Credit-g	67.7%	72.3%	72.0%	**73.0%**
Cylinderbands	**71.5%**	66.5%	**71.5%**	57.8%
Diabetes	68.0%	**73.7%**	**73.7%**	73.3%
Ecoli	75.3%	**81.5%**	79.5%	78.6%
Glass	**67.3%**	61.2%	**67.3%**	66.8%
Haberman	67.6%	**74.5%**	73.5%	73.9%
Heart-cleveland	72.6%	78.2%	**78.9%**	75.9%
Heart-statlog	72.6%	**80.0%**	77.8%	77.0%
Hepatitis	73.5%	73.5%	**83.2%**	80.6%
Iris	94.0%	94.7%	**96.0%**	95.3%
Labor	79.0%	**89.5%**	**89.5%**	73.7%
Liver-disorders	63.8%	62.0%	**64.1%**	62.0%
Lymph	78.4%	**79.7%**	78.4%	73.0%
Tic-tac-toe	**93.6%**	78.8%	89.9%	84.9%
Average	**75.8%**	**77.3%**	**79.5%**	**76.8%**

To evaluate if there are any statistically significant differences, a standard pair wise sign test was used. Using 18 data sets, a sign test requires 14 wins for statistical significance. Table IV shows wins, ties and losses for the row technique against the column technique. Significant results are marked in bold.

As seen in the table, DTiGP performs significantly better than both decision tree algorithms, having 14 wins against each. J48 has 12 wins against jaDTi, which is not significant.

TABLE IV
WINS/TIES/LOSSES

	jaDTi	GP	DTiGP	J48
jaDTi	-	5/1/12	**1/3/14**	6/0/12
GP	12/1/5	-	6/2/10	9/1/8
DTiGP	**14/3/1**	10/2/6	-	**14/0/4**
J48	12/0/6	8/1/9	**4/0/14**	-

VI. DISCUSSION

It is interesting to note that DTiGP is 11% better than normal GP on the *Tic-Tac-Toe* dataset. Obviously this is due to the high performance of jaDTi, which reaches an even higher accuracy by building a very large tree. Since this dataset does not contain any noise, a bigger tree will almost always be better. However, this is exactly what DTiGP was designed to achieve. Since the benchmark for reasonable performance also is calculated using jaDTi, it becomes rather high and forces DTiGP to increase its training accuracy by lowering the size punishment and injecting larger jaDTi trees. The exact same scenario also happens on *cylinder bands* and *glass*, where jaDTi clearly outperforms normal GP, thus forcing the GP to aim for higher training accuracy by using larger programs.

It is not surprising that DTiGP's accuracy increases when jaDTi outperforms normal GP, since similar trees will be injected into the population. However, DTiGP is not bounded by the accuracy of jaDTi. As seen in table IV, jaDTi only outperforms normal GP on five datasets, but DTiGP has 10 wins over normal GP. In fact, DTiGP is not bounded by the performance of either technique, since it has seven outright wins when compared to jaDTi and normal GP. *Colic* and *Hepatitis* are two good examples where DTIGP wins over both techniques with large margins (7% and 10%, respectively.).

J48 is significantly worse than DTiGP, but still better than jaDTi on 12 datasets. J48 wins over jaDTi is probably due to the more advanced pruning algorithm, since both use a creation method based on C4.5. This result is interesting since it shows a possible improvement by using J48 in DTiGP instead of jaDTi.

Finally it is interesting to note that normal GP and J48 perform comparable with nine and eight wins respectively (with one draw). Apparently, the settings used for normal GP is not sufficient to constantly outperform decision tree algorithms. In that light, the fact that DTiGP significantly outperforms J48 must be considered a strong result. Furthermore, DTiGP also performs clearly better than normal GP, losing on only six data sets of 18 against normal GP. In addition, in all six losses, the differences in accuracy obtained by the two techniques are all less than 2%. On the other hand, DTiGP has 10 wins against normal GP, where five of these wins are between 5% and 10% better than normal GP.

Even if the aim of this study was not to evaluate the size of the produced trees, intial results (not presented in detail here) show that DTiGP generally evolved trees that were shorter than trees produced by jaDTi and J48, but slightly larger than the original GP. This is a very promising results since DTiGP clearly outperforms the other techniques with regards to accuracy. Typically the trees are only considerably larger when DTiGP outperforms normal GP; i.e., when the algorithm is forced to look for larger trees in order to obtain sufficient accuracy

VII. CONCLUSIONS

The aim of this study was to present DTiGP, a novel technique aimed at improving GP classification accuracy. DTiGP's main strength is that it increases the robustness of normal GP, thus performing well over a large range of problems. In the experimentation, DTiGP was significantly better than J48, and clearly outperformed normal GP.

DTiGP robustness comes from the use of a benchmark level for reasonable training accuracy, but also from the injection of decision trees and an adjustable length punishment. It is clear that the use of the benchmark greatly improves DTiGP's result in at least three of the datasets where normal GP performs poorly. Nevertheless, the benchmark is not enough in itself when the search space is too big. Hence, a very important feature of DTiGP is the

injection of accurate trees to help the evolution, when the benchmark training accuracy cannot be reached.

Lowering the weight of the size punishment and injecting jaDTi trees greatly improves the robustness and accuracy of DTiGP. Furthermore, the method does not only improve the results on hard data sets where standard GP performs poorly, but also showed to increase the accuracy even when GP outperformed jaDTi. All in all, the experiments clearly show that the hybrid solution of injecting decision trees into a GP population gives synergetic effects, resulting in higher accuracies than using the basic techniques separately.

VIII. Future Work

Since J48 clearly outperformed jaDTi, it should be interesting to evaluate the effects of injecting J48 trees. J48 is more consistent, so should be expected to produce both better trees for injection and a better estimation of reasonable training accuracy. In addition, the optimal setup for when and how often, trees should be injected could be investigated further.

References

[1] Roiger, R. and Geatz, M., *Data mining: a tutorial-based primer*, Addison Wesley, Boston, 2003.

[2] Berry, M. J. A. and Linoff, G.. Data mining techniques: for marketing, sales and customer relationship management, Wiley, New York ; Chichester, 2004.

[3] Goodwin, P., Integrating management judgment and statistical methods to improve short-term forecasts, *Omega*, 30, 127-135, 2002.

[4] Davis, R., Buchanan, B. and Shortliffe, R., Production rules as a representation for a knowledge-based consultation program, *Artificial Intelligence*, 8, 15-45, 1977.

[5] Kim, Y. S., Toward a successful CRM: variable selection, sampling, and ensemble, *Decision Support Systems*, 41, 542-553, 2006.

[6] Frietas, A., A Survey of Evolutionary Algorithms for Data Mining and Knowledge Discovery, *Advances in evolutionary computing: theory and applications*, Springer-Verlag, New York, 2003, pp. 819-845.

[7] Murthy, S. K., Automatic Construction of Decision Trees from data: A Multi-Disciplinary Survey, *Data Mining and Knowledge Discovery*, vol. 2, Kluwer Academic Publishers, Boston, 1998, pp. 345-389,

[8] Hyafil, L. and Rivest, R., Constructing optimal binary decision trees is NP-complete*, Information Processing Letters*, 5(1):15–17, 1976.

[9] Eggermont J., Kok JN., Kosters W. A.: Genetic Programming for Data Classification:Partitioning the Search Space, *Proceedings of the 2004 ACM symposium on Applied computing*, New York, ACM Press; 2004:1001-1005, 2004.

[10] Koza, J. R., Genetic Programming: on the programming of computers by means of natural selection, MIT Press, Cambridge, 1992.

[11] Poli, R. , Langdon, W.B., McPhee, F. N., *A field Guide to Genetic Programming*, United Kingdom, Poli, R., Langdon, W.B., McPhee, F. N , 2008.

[12] Papagelis, A. and Kalles, D., Breeding Decision Trees Using Evolutionary Techniques, *In 18th International Conference on Machine Learning,* Williamstown, MA, pp. 393-400, 2001.

[13] Freitas, A. A., A Survey of evolutionary algorithms for data mining and knowledge discovery, Advances in evolutionary computing: theory and applications, Springer-Verlag, New York, pp. 819-845, 2003.

[14] Konig, R., Johansson, U. and Niklasson, L., Increasing rule extraction comprehensibility, *International Journal of Information Technology and Intelligent Computing*, 1, 303-314, 2006.

[15] Jong K. D. Learning with Genetic Algorithms: An Overview, Machine learning, 3, Kluwer academic publishers, Netherlands, pp 121-138, 1988. Quinlan, J. R., Induction of Decision Trees, *Machine Learning,* 1**,** 81-106, 1986

[16] Quinlan, J. R., *C4.5: programs for machine learning,* Morgan Kaufmann Publishers Inc., 1993.

[17] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. *Classification and Regression Trees,* Wadsworth, 1983.

[18] Berry, M. J. A., and Linoff, G. Data mining techniques: for marketing, sales and customer relationship management, Wiley, New York ; Chichester, 2004.

[19] Murthy, S. K. and Salzberg S. Lookahead and pathology in decision tree induction*, Proceedings of the 14th International joint conference on artificial Intelligence*, Morgan Kaufman, pp. 1025-1031, 1995.

[20] Wang, Qing Ren and Ching Y. Suen. Large tree classifier with heuristic search and global training. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-9(1):91–102, January 1987.

[21] Meisel, W. S. and Demetrios A. Michalopoulos. A partitioning algorithm with application in pattern classification and the optimization of decision trees. *IEEE Trans. on Comp*., C-22(1):93–103, 1973.

[22] Bennett, Kristin P. Global tree optimization: A non-greedy decision tree algorithm*. In Proc. of Interface 94: The 26th Symposium on the Interface, Research Triangle*, North Carolina, 1994.

[23] Koza, J. R. (1992). *Genetic Programming*, MIT Press/Bradford Books, Cambridge, MA, 1992.

[24] Montana, J. M., Strongly Typed Genetic Programming, *Evolutionary Computation*,3,pp. 199-230, 1993.

[25] Espejo, P., Ventura, S. & Herrera, F., 2010. A Survey on the Application of Genetic Programming to Classification. *IEEE Transactions on Systems*, Man and Cybernetics, 40(2), 121-144.

[26] Konig, R., Johansson, U. and Niklasson, L., G-REX: A Versatile Framework for Evolutionary Data Mining, *IEEE International Conference on Data Mining Workshop*s, Pisa, Italy, pp. 971-974, 2008.

[27] Bojarzuk, C.C., Lopes, H. S., Freitas, A. A and Michalkiewicz, E. L., A constrained-syntax genetic programming system for discovering classification rule: application to medical data sets. *Artificial Intelligence in Medicine,* vol. 30:1, pp 27-48, 2004.

[28] De Falco, i., Della Cioppa, A., Tarantino, E., Discovering interesting classification rules with genetic programming, *Applied soft computing,* vol. 1, pp. 257-269, 2002.

[29] Eggermont J., Eiben, A.E., and Hemert, J.I., Adapting the fitness function in GP for data mining. *Proceedings of Second European Workshop on Genetic Programming*, LNCS. Springer, Berlin, 1999.

[30] Tan K.C., Tay A., Lee T.H, Mining multiple comprehensible classification rules using genetic programming, *Proceedings of the 2002 congress on evolutionary computation,* pp. 1302-1307, 2002.

[31] Wo-Chiang, L., Genetic Programming Decision Tree for Bankruptcy Prediction, *Advances in intelligent systems research,* Atlantis Press. 2006.

[32] Oka, S. & Zhao, Q., 2007. Design of Decision Trees through Integration of C4.5 and GP. *Japan-Australia Joint Workshop on Intelligent and Evolutionary Systems*. pp. 128-135.

[33] Carvalho. D., Freitas. A., Ahybrid decision tree/genetic algorithm for coping with the problem of small disjuncts in data mining, *International Journal of Information Sience,*Elsevier Science Inc, New York, USA, vol. 163, pp. 13-35, 2004.

[34] Blake, C. L. and Merz, C. J. 1998. "UCI Repository of machine learning databases." University of California, Department of Information and Computer Science.

[35] Johansson U., Konig, R. and Niklasson, L., The truth is in there - Rule extraction from opaque models using genetic programming, International Florida Artificial Intelligence Research Society Conference, Miami Beach, Florida, USA, pp. 658-663, 2004.

[36] Francois, J. M., jaDTi – Decision Trees: a Java implementation, http://www.run.montefiore.ulg.ac.be/~francois/software/jaDTi/

[37] Weka 3: Data mining software in Java, http://www.cs.waikato.ac.nz/ml/weka/

[38] Blake, C. L. and Merz, C. J., UCI Repository of machine learning databases, University of California, Department of Information and Computer Science, 1998.