Automatically Evolving Rule Induction Algorithms

Gisele L. Pappa and Alex A. Freitas

Computing Laboratory
University of Kent
Canterbury, Kent, CT2 7NF, UK
{glp6, A.A.Freitas}@kent.ac.uk
http://www.cs.kent.ac.uk

Abstract. Research in the rule induction algorithm field produced many algorithms in the last 30 years. However, these algorithms are usually obtained from a few basic rule induction algorithms that have been often changed to produce better ones. Having these basic algorithms and their components in mind, this work proposes the use of Grammar-based Genetic Programming (GGP) to automatically evolve rule induction algorithms. The proposed GGP is evaluated in extensive computational experiments involving 11 data sets. Overall, the results show that effective rule induction algorithms can be automatically generated using GGP. The automatically evolved rule induction algorithms were shown to be competitive with well-known manually designed ones. The proposed approach of automatically evolving rule induction algorithms can be considered a pioneering one, opening a new kind of research area.

1 Introduction

Research in the rule induction field has being carried out for more than 30 years and certainly produced a large number of algorithms. However, these are usually obtained from the combination of a basic rule induction algorithm (typically following the sequential covering approach) with new evaluation functions, pruning methods and stopping criteria for refining or producing rules, generating many "new" and more sophisticated sequential covering algorithms.

We cannot deny that these attempts to improve the basic sequential covering approach have succeeded. Among the most successful and popular rule induction algorithms are, for example, CN2 [1] and RIPPER [2]. The CN2 algorithm was developed following the concepts of the successful ID3[3] and AQ algorithms. Its current version is a product of small modifications on its first rule evaluation function, together with the added feature of producing ordered or unordered rules. RIPPER is an improvement of IREP [4], which in turn is an improvement of REP, which was created to improve the performance of pFOIL in noisy domains. From IREP to RIPPER, for instance, the metric to evaluate the rules during the pruning phase and the rule stopping criterion were changed. A post-processing phase was also included to optimize the set of learned rules.

If changing these major components of rule induction algorithms can result in new, significantly better ones, why not keep on trying systematically? Our idea is to do this, but not by using the manual, ad-hoc approach of the previous research in the area. Rather, we propose the ambitious idea of automating the process of designing a rule induction algorithm.

Genetic Programming (GP) [5] is a suitable tool for automatically evolving computer programs. The program evolved by a GP can produce the same solution humans use to solve the target problem; but it can also produce something completely new and perhaps better than the "conventional" manually designed solution. Examples of human-competitive GP solutions can be found at [6].

Automatically evolving a rule induction algorithm "from scratch" would certainly be an extremely hard task for a GP. However, we can provide the GP with background knowledge about the basic structure of rule induction algorithms, making the task more feasible.

Grammar-based GP (GGP) [7] is a special type of GP that incorporates in its search mechanism prior knowledge about the problem being solved. Intuitively, GGP is an appropriated tool to automatically evolve rule induction algorithms.

The motivation to design a GP algorithm for automatically evolving a rule induction algorithm is three-fold. First, although there are various rule induction algorithms available, their accuracy in many important, complex domains is still far from 100%, and it is not clear how much, if any, improvement is still possible with current methods [8]. Hence, extensive research has been done to try to improve the results obtained by current rule induction systems. GP provides an automatic way of performing a global search that evaluates, in parallel, many combinations of elements of rule induction algorithms, which can find new, potentially more effective algorithms.

Second, all current rule induction algorithms were manually developed by a human being, and so they inevitably incorporate a human bias. In particular, the majority of rule induction algorithms select one-attribute-value-at-a-time, in a greedy fashion, ignoring attribute interactions. A machine–developed algorithm could completely change this kind of algorithm bias, since "its bias" would be different from the kind of algorithm bias imposed by a human designer.

At last, it has already been shown that no classification algorithm is the best to solve all kinds of tasks [9]. Therefore, a GP algorithm could be used to generate rule induction algorithms targeting specific sets of data, which share similar statistical features, or even generating a rule induction algorithm tailored for a given data set. It would allow us to generate different classifiers for different types of data, just by changing the training data given to the GP.

In [10] we presented the first concepts about automatically evolving a rule induction algorithm at a high level of abstraction. In this paper, we refine those ideas in much greater detail. The remainder of this paper is organized as follows. Section 2 briefly discusses rule induction algorithms. Section 3 gives an overview of GP and GGP. Section 4 introduces the proposed GGP, and Section 5 reports the results of several computational experiments. Section 6 presents the conclusions and describes future research directions.

2 Rule Induction Algorithms

There are three common strategies used to induce rules from data [11]: (1) The separate and conquer strategy [12]; (2) Generate a decision tree–using the divide and conquer strategy–and then extract one rule for each leaf node of the tree [3]; (3) The use of evolutionary algorithms, like genetic algorithms and genetic programming, to extract rules from data [13, 14].

Among these three strategies, the separate and conquer is certainly the most explored. The separate and conquer strategy (also known as sequential covering) learns a rule from a training set, remove from it the examples covered by the rule, and recursively learns another rule that covers the remaining examples, until all or almost all examples are covered. It is the most common strategy used for rule induction algorithms, and the methods based on this approach differ from each other in four main points [15, 12], although the last one can be absent:

1. The representation of the candidate rules : propositional or first-order logic.
2. The search mechanisms used to explore the space of candidate rules (Usually a bottom-up, top-down or bi-directional strategy combined with a greedy, beam or best-first search).
3. The way the candidate rules are evaluated, using heuristics such as information gain, information content, Laplace accuracy, confidence, etc.
4. The pruning method, which can be used during the production of the rules (pre-pruning) or in a post processing step (post-pruning) to help avoiding over-fitting and handling noisy data.

These 4 points will be the starting point for the definition of the grammar which the proposed GP will use, as described in Section 4.1.

3 Overview of Genetic Programming

Genetic Programming (GP) [5, 16] is an area of evolutionary computation which aims to automatically evolve computer programs. Together with other evolutionary algorithms, its application is successful because of its problem independency, global search and associated implicit parallelism [16].

Essentially, a GP algorithm evolves a population of individuals, where each individual represents a candidate solution to the target problem. These individuals are evaluated using a fitness function, and the fittest individuals are usually selected to undergo reproduction, crossover and mutation operations. The new individuals produced during these processes create a new population, which replaces the old one. This evolution process is carried out until an optimum solution is found, or a pre-established number of generations is reached.

In this work we use a Grammar-based GP (GGP). As the name suggests, the major difference between a GP and a GGP is the definition and use of a grammar. The motivation to combine grammars and GP is two-fold [17]. First, it allows the user to incorporate prior knowledge about the problem domain in

the GP, to guide its search. Second, it guarantees the closure property through the definition of grammar production rules.

Grammars are simple mechanisms capable of representing very complex structures. Context Free Grammars (CFG), the focus of this work, can be represented as a four-tuple $\{N, T, P, S\}$, where N is a set of non-terminals, T is a set of terminals, P is a set of production rules, and S (a member of N) is the start symbol. The production rules have the form $x ::= y$, where $x \in N$ and $y \in \{T \cup N\}$.

There are three special symbols used in the notation to write production rules: "|","[ ]" and "( )". "|" represents a choice, like in $x ::= y|z$, where x generates the symbol y or z. "[ ]" wraps an optional symbol which may or may not be generated when applying the rule. "( )" is used to group a set of choices together, like in $x ::= k(y|z)$, where x generates k followed by y or z.

A derivation step is the application of a production rule from $p \in P$ to some non-terminal $n \in N$, and it is represented by the symbol $\Longrightarrow$. Consider the production rules $x ::= yz$ and $y ::= 0|1$. A derivation step starting in x would be represented as $x \Longrightarrow yz$ and $yz \Longrightarrow 0z$.

In the GGP algorithm used in this work, each individual of the population is generated by applying a set of derivation steps from the grammar, guaranteeing that only valid programs (individuals) are generated [7], as detailed in Section 4.

4 Grammar-based Genetic Programming for Rule Induction

This work proposes the use of Grammar-based Genetic Programming (GGP) to automatically evolve rule induction algorithms. In contrast to projects that use GP to discover a set of *rules for a specific data set*, like [14] and [13], this project aims to automatically invent a *generic rule induction algorithm*, that is, a rule induction algorithm that can be applied to data sets in general, regardless of the application domain. Hence, each individual in our population represents a new rule induction algorithm, potentially as complex as well-known algorithms.

To the best of our knowledge, there has been just two attempts in the literature to use a GGP for improving the design of a sequential covering rule induction algorithm. Wong [18] used a GGP to automatically evolve the evaluation function of the FOIL algorithm. Our work goes considerably beyond that work, as follows. In [18] the GGP was used to evolve only the evaluation function of a rule induction algorithm. By contrast, in our work GGP is used to evolve virtually all components of a sequential covering rule induction algorithm. Hence, the search space for our algorithm is the space of sequential covering rule induction algorithms, whilst the search space for [18]'s GGP is just the space of evaluation functions for FOIL. Suyama *et* al. [19] also used a GP to evolve a classification algorithm. However, the ontology used in [19] has coarse-grained building blocks, where a leaf node of the ontology is a full classification algorithm. By contrast, our grammar is much more fine-grained; its building blocks are programming constructs ("while","if", etc), search strategies and evaluation procedures not used in [19]. Finally, in both [18] and [19], the GP was trained with a single data

set, like in any other use of GP for discovering classification rules. By contrast, in this work the GGP is trained with 6 data sets in the same run of the GGP, because the goal is to evolve a truly generic rule induction algorithm, and not just a rule induction algorithm for one particular data set.

The GGP method proposed was implemented as follows. In the first generation of the GGP a population of individuals is created using a grammar. The grammar contains background knowledge about the basic structure of rule induction algorithms following the separate and conquer approach.

Each individual in the population is represented by a derivation tree, built from a set of derivation steps produced by using the grammar. The individuals (rule induction algorithms) are evaluated using a set of data sets, named the meta-training set. The classification accuracies obtained from the runs of the rule induction algorithms represented by the individuals in the meta-training set are used to generate a fitness measure, as will be explained later.

Table 1. Grammar definition

```
1-<Start>::=(<CreateRuleList>|<CreateRuleSet>)[<PostProcess>].
2-<CreateRuleSet>::=forEachClass <whileLoop> endFor <RuleSetTest>.
3-<CreateRuleList>::=<whileLoop> <RuleListTest>.
4-<whileLoop>::=while <condWhile> <CreateOneRule> endWhile.
5-<condWhile>::=uncoveredNotEmpty|uncoveredGreater(10TrainEx|20TrainEx|
                90%TrainEx|95%TrainEx|97%TrainEx|99%TrainEx).
6-<RuleSetTest>::=lsContent|laplaceAccuracy.
7-<RuleListTest>::=appendRule|prependRule.
8-<CreateOneRule>::=<InitializeRule><innerWhile>[<PrePruneRule>]
                    <RuleStoppingCriterion>.
9-<innerWhile>::=while(candNotEmpty|negNotCovered)<FindRule>endWhile.
10-<InitializeRule>::=emptyRule|randomEx|typicalEx|<MakeFirstRule>.
11-<MakeFirstRule>::=NumCond1|NumCond2|NumCond3|NumCond4.
12-<FindRule>::=(<RefineRule>|<innerIf>)<EvalRule><StoppingCriterion>
                <SelectCandidateRules>.
13-<RefineRule>::=<AddCond>|<RemoveCond>.
14-<AddCond>::=Add1|Add2.
15-<RemoveCond>::=Remove1|Remove2.
16-<innerIf> ::=if <condIf> then <RefineRule> else <RefineRule>.
17-<condIf>::=<condIfExamples>|<condIfRule>.
18-<condIfExamples>::=(numCovExpSmaller|numCovExpGreater)(90p|95p|99p).
19-<condIfRule> ::=ruleSizeSmaller(2|3|5|7).
20-<EvalRule>::=accuracy|purity|laplace|infoContent|informationGain.
21-<RuleStoppingCriterion>::=noStop|purityStop|accuracyStop|nCoveredStop.
22-<StoppingCriterion>::=noStop|SignifTest90|SignifTest95|SignifTest99|
                PurityCrit60|PurityCrit70|PurityCrit80|defaultAccuracy.
23-<SelectCandidateRules>::=1CR|2CR|3CR|4CR|5CR|8CR|10CR.
24-<PrePruneRule>::=Prune1Cond|PruneLastCond|PruneFinalSeqCond.
25-<PostProcess> ::=RemoveRule EvaluateModel|<RemoveCondRule>.
26-<RemoveCondRule>::=(Remove1Cond|Remove2Cond|RemoveFinalSeq)<EvalRule>.
```

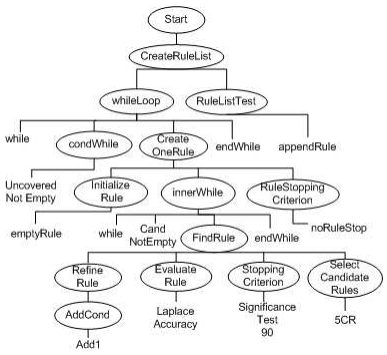

Fig. 1. Example of an GGP Individual (a complete rule induction algorithm)

After evaluation, a tournament selection scheme is used to select the individuals for the new population. Before being inserted in the new population, the winners of the tournaments undergo either reproduction, mutation, or crossover operations, depending on user-defined probabilities.

The evolution process is conducted until a maximum number of generations is reached. At the end of the process, the best individual (highest fitness) is returned as the solution for the problem. The chosen rule induction algorithm is then evaluated in a new set of data sets, named the meta-test set, which contains data sets different from the data sets in the meta-training set.

4.1 The Grammar

The grammar is the most important element in a GGP system, since it determines the search space. Table 1 presents the grammar. It uses the terminology introduced in Section 3, and the non-terminal *Start* as its Start symbol. The symbols that appear between "<>" are the grammar non-terminals.

The grammar is made of 26 production rules (PR), each one representing a non-terminal. For simplification purposes, this first version of the grammar does not include all the possible terminals/non-terminals we intend to use, but it is still an elaborate grammar, allowing the generation of many different kinds of rule induction algorithms.

According to PR 1 in Table 1 (*Start*), the grammar can produce either a decision list (where the rules are applied to an unclassified example in the order they were generated) or a rule set (where there is no particular order to apply rules to new examples). The derivation trees which can be obtained applying the production rules of the grammar will create an algorithm following the basic sequential covering approach. However, the non-terminals in the grammar will define how to initialize, refine and evaluate the rules being created. They also specify a condition to stop the refinement of rules and the production of the rule set/list, and define how the search space will be explored.

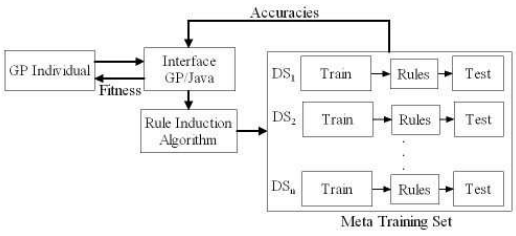

Fig. 2. Fitness evaluation process of a GGP Individual

4.2 The Design of the GGP Components

Individual Representation In our GGP system, an individual is represented by a derivation tree. This derivation tree is created using a set of production rules defined by the grammar described in Section 4.1. Recall that an individual represents a complete rule induction algorithm. Figure 1 shows an example of an individual's derivation tree. The root of the tree is the non-terminal *Start*. The tree is then derived by the application of PRs for each non-terminal. For example, PR 1 (*Start*) generates the non-terminal *CreateRuleList*. Then the application of PR 3 produces the non-terminals *whileLoop* and *RuleListTest*. This process is repeated until all the leaf nodes of the tree are terminals.

To extract from the tree the pseudo-code of the corresponding rule induction algorithm, we have to read all the terminals in the tree from left to right. The tree in Figure 1 represents an instance of the CN2 algorithm [1], with the beam-width parameter set to 5 and the significance threshold set to 90%.

The fitness function Evolution works selecting the fittest individuals of a population to reproduce and generate new offspring. In this work, an individual represents a rule induction algorithm. Therefore, we have to design a fitness function able to evaluate an algorithm RI_A as being better/worse than another algorithm RI_B.

In the rule induction algorithm literature, an algorithm RI_A is usually said to outperform an algorithm RI_B if RI_A has better classification accuracy in a set of classification problems. Hence, in order to evaluate the rule induction algorithms being evolved, we selected a set of classification problems, and created a meta-training set. In the meta-training set, each "meta-instance" represents a complete data set, divided into conventional training and test sets.

As illustrated in Figure 2, each individual in the GP population is decoded into a rule induction algorithm (implemented in Java) using a GP/Java interface. Each terminal in the grammar is associated with a block of Java code. When the evaluation process starts, the terminals in the individual are read, and together they generate a rule induction algorithm.

The Java code is compiled, and the rule induction algorithm is run on all the data sets belonging to the meta-training set. It is a conventional run where, for each data set, a set or list of rules is built using the set of training examples and evaluated using the set of test examples.

After the rule induction algorithm is run on all data sets in the meta-training set, the fitness of the individual is calculated as the average of the values of function f_i for each data set i in the meta training set. The function f_i is defined:

$$f_i = \begin{cases} \frac{Acc_i - DefAcc_i}{1 - DefAcc_i}, \text{ if } \mathrm{Acc}_i > \mathrm{DefAcc}_i \\ \frac{Acc_i - DefAcc_i}{DefAcc_i}, \text{ otherwise} \end{cases}$$

In this definition, Acc_i represents the accuracy (on the test set) obtained by the rules discovered by the rule induction algorithm in data set i. $DefAcc_i$ represents the default accuracy (the accuracy obtained when using the class of the majority of the examples to classify new examples) in data set i. According to the definition of f_i, if the accuracy obtained by the classifier is better than the default accuracy, the improvement over the default accuracy is normalized, by dividing the absolute value of the improvement by the maximum possible improvement. In the case of a drop in the accuracy with respect to the default accuracy, this difference is normalized by dividing the negative value of the difference by the maximum possible drop (the value of $DefAcc_i$).

Hence, f_i returns a value between -1 (when $Acc_i = 0$) and 1 (when $Acc_i = 1$). The motivation for this elaborate fitness function, rather than a simpler fitness function directly based only on Acc_i (ignoring $DefAcc_i$) is that the degree of difficulty of the classification task depends strongly on the value of $DefAcc_i$. The above fitness function recognizes this and returns a positive value of f_i when $Acc_i > DefAcc_i$. For instance, if $DefAcc_i$= 0.95, then Acc_i=0.90 would lead to a negative value of f_i, as it should.

Crossover and Mutation Operators In GGP, the new individuals produced by crossover and mutation have to be consistent with the grammar. For instance, when performing crossover the system cannot select a subtree *EvaluateRule* to be exchanged with a subtree *SelectCandidateRules*. Therefore, crossover operations have to exchange subtrees whose roots contain the same non-terminal, apart from *Start*. Crossing over two individuals swapping the subtree rooted at *Start* (actually, the entire tree) would generate exactly the same two individuals, and so it would be useless.

Mutation can be applied to a subtree rooted at a non-terminal or applied to a terminal. In the former case, the subtree undergoing mutation is replaced by a new subtree, produced by keeping the same label in the root of the subtree and then generating the rest of the subtree by a new sequence of applications of production rules, so producing a new derivation subtree. When mutating terminals, the terminal undergoing mutation is replaced by another "compatible" symbol, i.e., a (non-)terminal which represents a valid application of the production rule whose antecedent is that terminal's parent in the derivation tree.

5 Results and Discussion

The experimentation phase started with the definition of the meta-training and meta-test sets mentioned in Section 4.2. The current version of the system does

Table 2. Data sets used in the meta-training set

Data set	Examples	Attributes	Classes
Monks-2	169/432	6	2
Monks-3	122/432	6	2
Balance-scale	416/209	4	3
Tic-tac-toe	640/318	9	2
Lymph	98/50	18	4
Zoo	71/28	16	7

Table 3. Data sets used in the meta-test set

Data set	Examples	Attributes	Classes
Monks-1	556	6	2
Mushroom	8124	23	2
Promoters	106	58	2
Wisconsin	683	9	2
Splice	3190	63	3

not support continuous attributes. Hence, we used 10 public domain data sets having only categorical attributes. Out of the 10 data sets available, we arbitrarily chose 6 for the meta-training set and the other 4 for the meta-test set.

Tables 2 and 3 show respectively the data sets used in the meta-training and meta-test sets. During the evolution of the rule induction algorithm by the GGP, for each data set in the meta-training set, each candidate rule induction algorithm (individual) was trained with 70% of the examples, and then tested in the remaining 30%. In order to avoid overfiting, these sets of data were merged and then randomly divided in 70-30% for each of the generations of the GGP. In Table 2, the figures in the column *Examples* indicate the number of instances in the training/test sets used by each rule induction algorithm during the GGP run, respectively. In the meta-test set, data sets were processed using a 5-fold cross validation process. Hence, in Table 3, *Examples* indicates the total number of examples in the data set.

Once the meta data sets have been created, the next step was the choice of the GGP parameters. Population size was set to 100 and the number of generations to 30. These two numbers were empirically chosen based on preliminary experiments, but are not optimized. Considering crossover, mutation and reproduction probabilities, GPs usually use a high probability of crossover and low probabilities of mutation and reproduction. However, the balance between these three numbers is an open question, and may be very problem dependent [16]. In our experiments we set the reproduction probability as 0.05, and vary the balance between the crossover and mutation probabilities in order to choose appropriate values for these parameters. The empirically adjusted values were 0.8 crossover probability and 0.15 mutation probability. Sections 5.1 and 5.2 report the results obtained for the meta-training and meta-test sets respectively.

5.1 Results in the Meta-Training Set

First, we report results about the accuracy of the evolved rule induction algorithms in the test set of each of the data sets in the meta-training set. It should be stressed that this is not a measure of "predictive accuracy" because each test set in the meta-training set was seen during the evolution of the GGP. Nonetheless, the accuracy on the test sets of the meta-training set is useful to evaluate the success of the training of the GGP, and so it is reported here.

Table 4. Accuracy rates (%) for the Meta-training set

Data set	Def.	CN2Un	CN2Ord	RIPPER	C4.5R	GGP-RI
Monks-2	67.1	67.1	72.9	62.5	69.4	85.5±0.56
Monks-3	52.7	90.7	93.3	90.28	96.3	98.16±0.38
Balance-scale	45.9	77.5	81.3	77.03	78	80.48±0.68
Tic-tac-toe	65.4	99.7	98.7	98.43	100	96.16±1.14
Lymph	54	80	82	76	88	76.26±1.83
Zoo	43.3	96.7	96.7	90	93.3	99.34±0.66

Table 5. Accuracy rates (%) for the Meta-test set

Data set	Def.	CN2Un	CN2Ord	RIPPER	C4.5R	GGP-RI
Monks-1	50	100±0	100±0	93.84±2.93	100±0	100±0
Mushroom	51.8	100±0	100±0	99.96±0.04	98.8±0.06	99.99±0.01
Promoters	50	74.72±4.86	81.9±4.65	78.18±3.62	83.74±3.46	78.83±2.16
Wisconsin	65	94.16±0.93	94.58±0.68	93.99±0.63	95.9±0.56	94.54±0.56
Splice	51.8	74.82±2.94	90.32±0.74	93.88±0.41	89.66±0.78	89.24±0.32

Table 4 shows the default accuracy (accuracy obtained when using the class of the majority of the examples to classify any new example) of the data sets in the meta-training set, followed by the results of runs of CN2-Unordered and CN2-Ordered (using default parameters), RIPPER and C4.5 Rules. These results are baselines against which we compare the accuracy of the rule induction algorithms evolved by the GGP. Table 4 also reports the results obtained by the GGP-RI (Rule Induction algorithms evolved by the GP).

In Table 4 the numbers after the symbol "±" are standard deviations. Results were compared using a statistical t-test with confidence level 0.05. Cells in dark gray represent winnings of GGP-RI against a baseline method, while light gray cells represent GGP-RI losses. In total, Table 4 contains 24 comparative results between GGP-RI and baseline methods – 6 data sets × 4 classification algorithms. Out of theses 24 cases, the accuracy of GGP-RI was statistically better than the accuracy of the baseline methods in 15 cases, whilst the opposite was true in only 5 cases. In the other 4 cases there was no significant difference.

5.2 Results in the Meta-Test Set

The results obtained by the GGP-RIs for the data sets in the meta-training set were expected to be competitive with other algorithms, since the GGP evolved rule induction algorithms based on the data sets in that meta-training set. The challenge for the GGP is to evolve rule induction algorithms that obtain at least a competitive performance for data sets in the meta-test set, which were not used during the evolution of the rule induction algorithm. As in the previous section, Table 5 shows the default accuracy and the accuracies obtained by baseline methods in the data sets in the meta-test set, followed by the results obtained by the GGP-RI. Recall that in the meta-test set every algorithm was run using

a 5-fold cross-validation procedure, and the results reported are the average accuracy on the test set over the 5 iterations of the cross-validation procedure.

As shown in Table 5, most of the results obtained in the 5 data sets used in the meta-test set are statistically the same as the ones produced by the baseline methods. The only exceptions are *Mushroom* and *Splice*. In both data sets, GGP-RI gets statistically better results than one of the baseline methods. *Splice* is the only data set in which RIPPER produces a better result than GGP-RI.

One of the main goals of this project was to automatically evolve rule induction algorithms that perform as well or better than human designed rule induction algorithms. Another goal was to automatically produce a rule induction algorithm different from human-designed ones. Out of the 5 GGP-RI discovered (in the 5 runs of the GGP with different random seeds), the one most different from the human designed ones can be summarized as follows.

It searches for rules starting from an example chosen from the training set using the typicality concept [20], and removes 2 conditions at a time from it (bottom-up approach). It evaluates rules using the Laplace accuracy and stops refining them when the rules' accuracy is smaller then 70%.

6 Conclusions and Future work

This work showed that effective rule induction algorithms can be automatically generated using Genetic Programming. The automatically evolved rule induction algorithms were show to be competitive with well-known manually designed (and refined over decades of research) rule induction algorithms. The proposed approach of automatically evolving rule induction algorithms can be considered a pioneering one, opening a new kind of research area, and so there are still many problems to be solved.

One research direction is to create a more powerful version of the grammar, which could potentially lead to the discovery of more innovative rule induction algorithms. Another possible research direction is to design a fitness function that considers not only the accuracy of the rules discovered by the rule induction algorithms, but also a measure of the size of the discovered rule set. However, this introduces the problem of coping with the trade-off between accuracy and rule set simplicity in the fitness function, an open problem.

Yet another possible research direction is to evolve rule induction algorithms for specific kinds of data sets. Instead of using very different kinds of data sets in the meta-training set, we can assign to the meta-training set several data sets that are similar to each other, according to a pre-specified criterion of similarity. Then, in principle, the GGP algorithm would evolve a rule induction algorithm particularly tailored for that kind of data set, which should maximize the performance of the rule induction algorithm in data sets of the same kind, to be used in the meta-test set. However, this introduces the problem of defining a good measure of similarity between the data sets: an open problem. It is also possible to evolve a rule induction algorithm tailored for one given data set, as in [18, 19].

Acknowledgments

The first author is financially supported by CAPES, a Brazilian research-funding agency, process number 165002-5.

References

1. Clark, P., Boswell, R.: Rule induction with cn2: some recent improvements. In: EWSL-91: Proc. of the Working Session on Learning on Machine Learning. (1991)
2. Cohen, W.W.: Fast effective rule induction. In: Proc. of the 12^{th} International Conference on Machine Learning. (1995)
3. Quinlan, J.R.: C4.5: programs for machine learning. Morgan Kaufmann (1993)
4. Furnkranz, J., Widmer, G.: Incremental reduced error pruning. In: Proc. the 11^{th} Int. Conf. on Machine Learning, New Brunswick, NJ (1994) 70–77
5. Koza, J.R.: Genetic Programming: On the Programming of Computers by the means of natural selection. The MIT Press, Massachusetts (1992)
6. Koza, J.: http://www.genetic-programming.org/. (June, 2006)
7. Whigham, P.A.: Grammatically-based genetic programming. In: Proc. of the Workshop on GP: From Theory to Real-World Applications. (1995)
8. Domingos, P.: Rule induction and instance-based learning: A unified approach. In: Proc. of the 14^{th} International Joint Conference on Artificial Intelligence. (1995) 1226–1232
9. Lim, T., Loh, W., Shih, Y.: A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. Machine Learning **40**(3) (2000) 203–228
10. Pappa, G.L., Freitas, A.A.: Towards a genetic programming algorithm for automatically evolving rule induction algorithms. In Furnkranz, J., ed.: Proc. ECML/PKDD-2004 Workshop on Advances in Inductive Learning. (2004) 93–108
11. Mitchell, T.: Machine Learning. Mc Graw Hill (1997)
12. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann (1999)
13. Freitas, A.A.: Data Mining and Knowledge Discovery with Evolutionary Algorithms. Springer-Verlag (2002)
14. Wong, M.L., Leung, K.S.: Data Mining Using Grammar-Based Genetic Programming and Applications. Kluwer (2000)
15. Furnkranz, J.: Separate-and-conquer rule learning. Artificial Intelligence Review **13**(1) (1999) 3–54
16. Banzhaf, W., Nordin, P., Keller, R., Francone, F.: GP – An Introduction; On the Automatic Evolution of Computer Programs and its Applications. Morgan Kaufmann (1998)
17. O'Neill, M., Ryan, C.: Grammatical Evolution Evolutionary Automatic Programming in an Arbitrary Language. Morgan Kaufmann (2003)
18. Wong, M.L.: An adaptive knowledge-acquisition system using generic genetic programming. Expert Systems with Applications **15**(1) (1998) 47–58
19. Suyama, A., Negishi, N., Yamaguchi, T.: CAMLET: A platform for automatic composition of inductive learning systems using ontologies. In: Pacific Rim International Conference on Artificial Intelligence. (1998) 205–215
20. Zhang, J.: Selecting typical instances in instance-based learning. In: Proc. of the 9^{th} Int. Workshop on Machine Learning. (1992)