Evolving Rule Induction Algorithms with Multi-objective Grammar-based Genetic Programming

Gisele L. Pappa[1] and Alex A. Freitas[2]

[1] A2SI - ESIEE - Université Paris-Est
Noisy-le-Grand, BP99, 93162
giselepappa@gmail.com
[2] Computing Laboratory
University of Kent
Canterbury, Kent, CT2 7NF, UK
+44 (0)1227 827220 / +44 (0)1227 762811
A.A.Freitas@kent.ac.uk

Abstract. Multi-objective optimization has played a major role in solving problems where two or more conflicting objectives need to be simultaneously optimized. This paper presents a Multi-Objective Grammar-based Genetic Programming (MOGGP) system that automatically evolves complete rule induction algorithms, which in turn produce both accurate and compact rule models. The system was compared with a single objective GGP and three other rule induction algorithms. In total, 20 UCI data sets were used to generate and test generic rule induction algorithms, which can be now applied to any classification data set. Experiments showed that, in general, the proposed MOGGP finds rule induction algorithms with competitive predictive accuracies and more compact models than the algorithms it was compared with.
Keywords: Grammar-based Genetic Programming, Pareto Optimization, Rule Induction Algorithms, Data Mining, Classification.

1 Introduction

Evolutionary algorithms (EAs) have been used to solve classification (supervised learning) problems for many years, and they became particularly popular in the data mining community for producing sets of classification rules [Falco et al., 2005a, Freitas, 2002, Wong, 1998]. In the context of EAs for rule induction, the individuals being evolved are designed to represent a single rule or a set of rules which best classifies instances in a *specific* data set.

Apart from EAs, rule models can be also produced by algorithms following the sequential-covering approach [Furnkranz, 1999] or they can be extracted from other representation models, like decision trees [Quinlan, 1990, 1993] or neural networks [Jacobson, 2005]. Although there is a large variety of methods available to produce rule models, there is still a great interest in studying new algorithms capable of producing them. The motivation for this is two-fold. First,

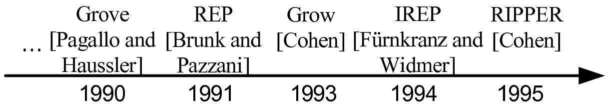

Fig. 1. The human-driven evolution of rule induction algorithms

the accuracy of rule models produced by existing rule induction algorithms can still be improved. Second, there are currently many powerful classifiers which can learn a variety of other types of classification models from data sets and very often obtain higher classification accuracies than rule models. Nevertheless, most of these very accurate classification models are black-boxes [Schölkopf and Smola, 2002], lacking interpretability. Interpretability has proved to be a key feature in many domains of current interest, such as bioinformatics or medical domains [Mirkin and Ritter, 2000, Clare and King, 2002, Pazzani, 2000, Fayyad et al., 1996]. In these domains, in particular, a model is not very useful if it cannot be understood, because the model is typically used to support a decision to be made by a domain expert.

This necessity of creating better rule induction algorithms has led to a human-driven "evolution" in their design. For instance, Ripper - a popular sequential-covering algorithm - was created by following the evolution line presented in Fig. 1. An analysis of the algorithms illustrated in Fig. 1 shows that each of them identifies and preserves the best components of its predecessors, and at the same time adds new or modifies components that will improve its performance. If evolution is a natural phenomenon in human-designed rule induction algorithms, and there is still room for research in the area of design of rule induction algorithms, why not to do it automatically? That is what we proposed in [Pappa and Freitas, 2006].

More precisely, in Pappa and Freitas [2006] we presented a Grammar-based Genetic Programming (GGP) method which makes use of the background knowledge about how humans design sequential-covering rule induction algorithms to automatically evolve new sequential-covering rule induction algorithms. It should be emphasized that, unlike conventional EAs that evolve a rule set for a *single given data set*, the GGP proposed in [Pappa and Freitas, 2006] evolves a *generic rule induction algorithm* which can be applied to any classification data set. In order to generate a generic classifier, the individuals (rule induction algorithms) evolved by the GGP are evaluated using a fitness function based on their accuracy on a set of data sets named meta-training set. At the end of the evolutionary process, the individuals are then validated in a new set of data sets (unseen during the evolution) named meta-test set.

Note that it is well known in the data mining literature that no classification algorithm will present the best possible performance in all kinds of data [Lim et al., 2000, Michie et al., 1994]. However, in the same way as human-designed rule induction algorithms are conceived to be robust, this work also aims to create rule induction algorithms as generic and robust as possible. Later, rule induction algorithms parameters – such as the number of candidate rules preserved during the search process, the amount of rule pruning, etc – can be tuned to improve the performance of the algorithm in more specific types of data. The algorithms evolved by the MOGGP also present this set of parameters and, more importantly, some parameters of the MOGGP itself (e.g., its meta-training set) can be set to generate rule induction algorithms tailored to a single data set.

The system presented in [Pappa and Freitas, 2006] had one disadvantage: the rule induction algorithms generated could not deal with continuous attributes. In [Pappa and Freitas, 2007] we introduced a new version of the system which solved this problem, and presented more extensive results involving a set of 20 data sets.

However, one of the main advantages of using rule induction algorithms is related to the comprehensibility of the knowledge models generated. In both [Pappa and Freitas, 2006] and [Pappa and Freitas, 2007], comprehensibility was not taken into account when automatically evolving the rule induction algorithms, since the fitness function was based only on the accuracy of the generated rule induction algorithms in the meta-training set. In this paper, however, we propose an extended version of the GGP, which uses the principle of multi-objective Pareto optimization to evolve rule induction algorithms which produce both high accuracy and compact rule models.

This extended GGP system was obtained by changing the fitness function, tournament selection and elitist mechanisms of the original system to consider both accuracy and the size of the models generated by the evolved rule induction algorithms, as will be explained later. The modifications introduced were inspired by a multi-objective genetic algorithm proposed in [Pappa et al., 2004].

The remainder of this paper is organized as follows. Section 2 introduces the basic concepts of GGP and reviews some works which used this kind of EA to solve problems in rule induction. Section 3 gives a brief introduction to sequential-covering rule induction algorithms and the problems which we aim to solve with the proposed system. Section 4 presents some related work, whilst Section 5 gives an overview of the system proposed in Pappa and Freitas [2006], and describes the extensions made to produce the proposed multi-objective GGP (MOGGP). Section 6 describes experiments performed, and shows comparisons between the single and multi-objective EAs and well-known human-designed rule induction algorithms. At last, Section 7 draws some conclusions and proposes some future work.

2 Grammar-based Genetic Programming

This section introduces Grammar-based GP (GGP). As the name suggests, the main difference between the classical GP approach and a grammar-based one is the presence of a grammar. In GGP systems, the set of terminals and functions is replaced by a grammar. The grammar guarantees that all the individuals are syntactically correct. Note that in GGP we do not use the terms functions and terminals, but rather terminals and non-terminals, where terminals and non-terminals refer to the symbols of the grammar.

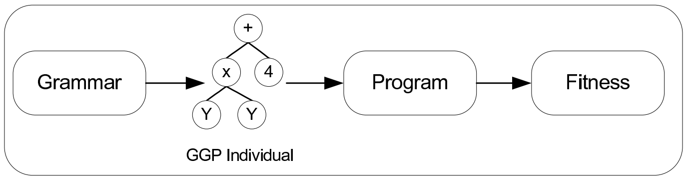

Fig. 2. GGP scheme with solution-encoding individual representation

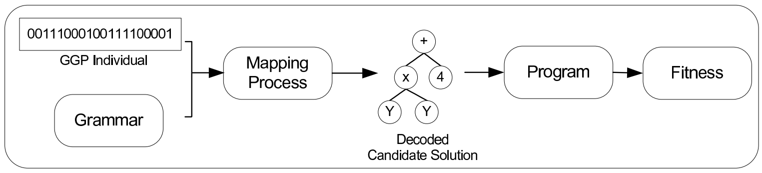

Fig. 3. GGP scheme with production-rule-sequence-encoding individual representation

The motivation for combining grammars and GP is two-fold [O'Neill and Ryan, 2003]. First, it allows the user to incorporate prior knowledge about the problem domain to help guide the GP search. Second, it guarantees the closure property through the definition of grammar production rules.

Grammar-based genetic programming has been used in a variety of application domains. One of its first domains of application was to develop the topology of neural networks [Gruau, 1996, Hussain and Browse, 1998]. It was also used in symbolic function regression [Keller and Banzhaf, 1996, O'Neill and Ryan, 2003, McConaghy and Gielen, 2006], to take into account the dimension of variables when evolving physical laws [Ratle and Sebag, 2000, 2001], to perform the clustering task of data mining [Falco et al., 2005b], for multiple instance learning [Zafra and Ventura, 2007], to evolve rule sets [Wong, 1998, Wong and Leung, 2000, O'Neill et al., 2001, Tsakonas et al., 2004, Hetland and Saetrom, 2005] and decision trees [Zhao, 2007].

GGPs can be divided into different classes according to two different criteria: (1) the representation used by the GGP individuals; (2) the type of grammar they are based on.

Considering the representation of the GGP individuals, grammar-based systems follow two different approaches, which we named solution-encoding individual and production-rule-sequence-encoding individual, and are represented in Figures 2 and 3, respectively. In the first approach (Figure 2), there is no difference between the individuals' genotype and phenotype. An individual is represented by a tree, which corresponds to a derivation tree produced when following the production rules of the grammar.

The second GGP approach (Figure 3) differs from the first because it uses a mapping between the individual's genotype and phenotype (the search and solution space). In this approach, the individuals are represented by a linear genome (usually a binary string or an array of integers), which is generated

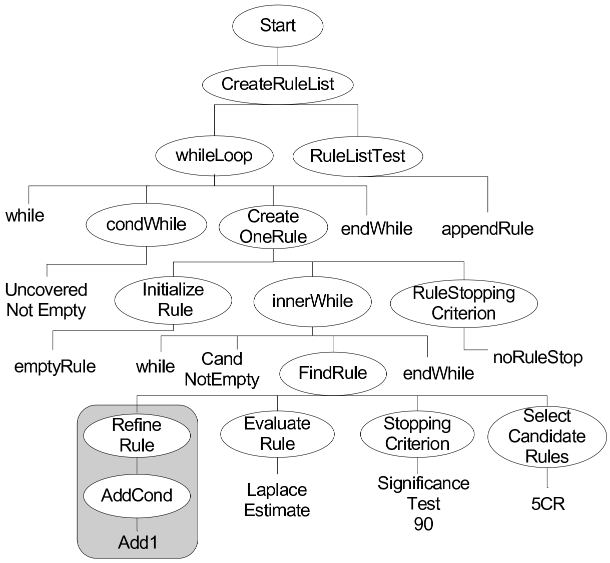

Fig. 4. Example of a GGP individual representing a rule induction algorithm

independently from the grammar. When evaluating the individuals, a genotype/phenotype mapping is made, and the genetic material is used to select appropriate production rules from the grammar.

Regarding the types of grammar used to guide the GP, the most popular are the context-free grammars (CFG) [Aho et al., 1986]. A CFG is represented by 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 ∈ N and y ∈ {T ∪ N}.

Regardless of the representation used by the GGP systems based on CFGs, either the individuals' genotype or phenotype will be represented by a grammar derivation tree, as shown in Fig. 4. Fig. 4 presents the tree containing the pseudocode of an individual which represents a rule induction algorithm. This derivation tree is the product of a set of derivation steps. A derivation step is the application of a production rule $p \in P$ to some non-terminal $n \in N$, and it is represented by the symbol $\Longrightarrow$. In order to exemplify, consider the two production rules:

```
<RefineRule> ::= <AddCond> | <RemoveCond>.
<AddCond> ::= Add1 | Add2.
```

where non-terminals are wrapped into "<>" symbols and "|" represents a choice. A derivation step starting in the non-terminal $<RefineRule>$ would be represented as $<RefineRule> \Longrightarrow <AddCond>$ – meaning that a rule is refined by adding one or more conditions to it, and could be followed by a second step $<AddCond> \Longrightarrow Add1$, meaning that exactly one condition is added to the rule. These two derivation steps are illustrated in the subtree indicated by the gray rectangle in Fig. 4.

In the same way the derivation steps just described create the subtree shown in the gray rectangle in Fig. 4, a derivation step starting from the start symbol $<Start>$ will generate the whole tree presented in Fig. 4. The pseudo-code for the algorithm this individual represents can be extracted by reading the leaf nodes of the tree, which are always represented by a grammar's terminal symbol.

Although CFGs are the most popular type of grammars used with GGP systems, after the popularization of these systems, works have been done using logic grammars [Wong, 1998], attribute grammars [Cleary, 2005], tree-adjunct and tree-adjoining grammars [Hoai et al., 2003] and, more recently, Christiansen grammars [Ortega et al., 2007]. While context-free grammars are used to restrict the syntax of the programs generated, logic grammars, attribute grammars and tree-adjoining/tree-adjunct grammars also consider context-information while generating trees (programs), and can express more complex representations. Christiansen grammars, in contrast, are particularly useful to account for semantic restrictions.

In this work, we are particularly interested in GGPs following the solution-encoding individual approach using a CFG, and how these algorithms were previously used in order to solve rule induction problems, as will be detailed in Section 2.1.

2.1 GGP with Solution-Encoding Individual

This section briefly reviews the main ideas of GGP systems following the approach described in Figure 2, named solution-encoding individual. In these systems, a GGP individual directly encodes the solution for the target problem, and does not require any mapping from the search (genotype) to the solution (phenotype) space. This type of individual representation requires a special procedure to initialize the first population of individuals, and to control crossover and mutation operations.

Figure 4 shows an example of a solution-encoding individual for the problem of evolving a rule induction algorithm. The individual is built through a set of derivation steps, and production rules are applied to the tree until all the leaf nodes are represented by terminals. As explained before, the solution represented by the GGP individual shown in Figure 4 is obtained by reading the leaf nodes of the tree, from left to right. This same procedure is used to generate all the individuals in the population.

Note that, when choosing a production rule from the grammar to form the individual's derivation tree, the initialization algorithm needs to check whether that particular production rule will be able to reach a terminal symbol in a number of derivation steps smaller than the maximum tree depth permitted.

Crossover and mutation operations are restricted to non-terminals, and different non-terminals might be assigned different crossover/mutation rates. In the case of crossover, a non-terminal N_x is randomly selected from the tree of the first individual I_1. After that, the system searches for the same non-terminal N_x in the tree of individual I_2. If N_x is present in I_2, the subtrees rooted at N_x

in individuals I_1 and I_2 are swapped (respecting the maximum individual size parameter). If N_x is not present in I_2, the operation is not performed.

Regarding the mutation operator, a random non-terminal N_x is selected from the derivation tree of the individual, the subtree rooted at N_x is deleted, and a new subtree is created by following the productions of the grammar (starting from N_x). The population initialization, individual representation and crossover and mutation operations just described were first introduced by Whigham [1995].

2.2 GGPs as Rule Induction Algorithms

Evolutionary algorithms, including GGPs, have been widely used as a tool to evolve a set of *if-then* rules to a specific data set [Freitas, 2002]. This section briefly describes just a few of these GGP systems which automatically evolve rule sets for a specific data set.

Whigham [1996], for instance, described a set of grammars to solve a classification problem named “greater glider density” . This set of grammars supported *if-then-else* statements and built programs, which can be read as a set of rules. In this same context, Wong and Leung [2000] combined GGP with ILP (Inductive Logic Programming) to produce a data mining classification system called LOGENPRO (The Logical grammar based Genetic Programming system). However, instead of a CFG, they worked with a logic grammar to create individuals. LOGENPRO can produce both decision trees and rule sets.

When evolving rule sets, LOGENPRO individuals (derivation trees extracted from a logic grammar) represent classification rules. The trees representing individuals might present some “frozen” nodes, which cannot be swapped during crossover operations. Crossover swaps subtrees rooted at two non-terminals to produce a single individual, and mutation recreates a subtree rooted at a random non-terminal according to the grammar. Reproduction is not used, but a third operator called the drop condition (which generalizes rules) was implemented. LOGENPRO also has a mechanism called token competition, which aims to maintain population diversity.

In a similar manner, Wong Tsakonas et al. [2004] proposed to evolve rule sets for medical domains using steady-state GPs and CFGs. They proposed two systems: one to evolve crisp rules and another to evolve fuzzy rules. Again, the population initialization and evolutionary operators are the conventional ones in GGPs with solution-encoding individual, as proposed by Whigham. In this system, each GGP individual is a rule list for a specific class. Hence, in a system with C classes, the GGP is executed C times, in each of them trying to find a rule list which separates the *i-th* class (i = 1,...,C) from the other *C-1* classes.

Also with the purpose of evolving prediction rules (although not classification rules), Romero et al. [2005] proposed a GGP system which uses both a single and a multi-objective approach to create a set of prediction rules to provide feedback for courseware authors. The system was compared with both sequential-covering algorithms and multi-objective algorithms when producing rules, showing promising results.

All the systems described so far were used to evolve rule sets for a specific data set. Going on step further, Wong [1998] used a GGP to automatically evolve the evaluation function (or scoring function) of the FOIL algorithm (an inductive logic programming algorithm). The GGP proposed by Wong creates a population of evaluation functions by following the production rules of a logic grammar, which uses terminals like the current information gain of the rule being evaluated, the number of positive and negative examples covered by the rule being evaluated, and random numbers. The individuals (evaluation functions) generated by the GGP are then incorporated into a generic version of a top-down first-order logic learning algorithm based on FOIL, and the learning algorithm as a whole is evaluated.

Out of the GGPs previously mentioned, the most related one to our GGP system is the GGP proposed by Wong [1998], because it is the only one of the above mentioned systems that evolves a component of a generic rule induction algorithm, rather than just evolving a set of rules specific to the dataset being mined as usual. Wong [1998] proposes to evolve a component of a learning system, i.e., the evaluation function of a first-order rule induction algorithm named FOIL.

The work proposed in this paper aims to evolve a generic and complete rule induction algorithm. Hence, while [Whigham, 1996, Wong, 1998, Wong and Leung, 2000, Tsakonas et al., 2004, Romero et al., 2005] trained the GGP with a single data set, this work proposes that the GGP is trained with several data sets (from different application domains) in the “meta-training set” in the same run of the GGP, in order to evolve a truly generic and robust rule induction algorithm.

Moreover, when comparing the proposed method with Wong’s work, whilst the search space for Wong’s GGP is just the space of evaluation functions for FOIL, our grammar is much more elaborate, and includes in its search space all the major components found in sequential-covering rule induction algorithms. To put it simply, our work consists of evolving a complete rule induction algorithm, whilst Wong’s work consists of evolving just one component (the evaluation function) of just one existing algorithm.

3 Sequential-Covering Rule Induction Algorithms

This section briefly describes one of the strategies most explored and most used to induce classification rules from data: the sequential-covering (also known as separate and conquer) strategy [Witten and Frank, 2005, Furnkranz, 1999]. It learns a rule from a training set, removes from it the examples covered by the rule, and recursively learns another rule which covers the remaining examples in the training set. A rule is said to cover an example e when all the conditions in the antecedent of the rule are satisfied by the example e. For instance, the rule “IF (salary > £ 100,000) THEN rich” covers all the examples in the training set in which the value of salary is greater than £100,000, regardless of the current value of the class attribute of an example. The learning process goes on until a pre-defined criterion is satisfied. This criterion usually requires that all or almost all examples in the training set are covered by a rule.

This basic algorithm used to learn rules from data is composed of four main elements: the representation of the candidate rules, the search mechanisms used to explore the space of candidate rules, the way the candidate rules are evaluated and the pruning method, although the last one can be absent [Furnkranz, 1999, Witten and Frank, 2005]. These four main components, in turn, can be represented at a lower level of abstraction by a set of operations, which we call building blocks. In the last 30 years, researchers in the rule induction area experimented with the combination of various building blocks in order to create better rule induction algorithms.

Consider for example the algorithms which appear in Fig. 1. Grove [Pagallo and Haussler, 1990] was created to produce decision lists (ordered rule sets) using a top-down approach to search for rules (i.e., it started the search with an empty rule – which covered all the examples in the training set – and iteratively specialized it). The produced rules are evaluated using the entropy measure. More importantly, Grove introduced a new idea of building a set of rules from a data sample, and later pruning the built rule set in a different data sample. Subsequently, REP [Brunk and Pazzani, 1991] was created based on this latter idea of growing and pruning a rule in different data samples but, by contrast with Grove, it uses a bottom-up search mechanism (i.e., it randomly selected an example from the training set to represent the initial rule and then iteratively generalized it), and evaluates the rules using their accuracy. Following this same line of reasoning, GROW [Cohen, 1993] combined REP with a top-down search (now using information gain to evaluate rules) to produce unordered rule sets. However, the application of this new algorithm in large domains was impracticable. Hence, GROW modified REP so that it simplifies every produced rule in the training set, and then re-grows the rule set, by adding the simplified rules to a new(empty) set of rules. Following GROW, IREP [Furnkranz and Widmer, 1994] used the basic ideas of REP but it prunes each rule right after they are generated. As an improvement to IREP, RIPPER [Cohen, 1995] added an optimization process to it, and changed the heuristic functions/methods used to prune the rules.

These examples illustrate the fact that many sequential-covering rule induction algorithms were designed by experimenting with the current building blocks present in successful algorithms and inserting some new features to them.

The idea of this work is conceptually similar. By defining a grammar with a large number of building blocks and adding to it some new features which might work in a rule induction algorithm, a GGP is able to automatically generate new sequential-covering rule induction algorithms.

4 Related Work on Multi-Objective Genetic Programming

The success of multi-objective evolutionary algorithms (MOEA) in solving data mining problems was already demonstrated in the literature. MOEAs were used, for instance, for classification [Hetland and Saetrom, 2005, Zhao, 2007, Zafra and Ventura, 2007, Handl et al., 2007], clustering [Handl and Knowles, 2004, Law et al., 2004] and attribute selection [Pappa et al., 2004], in order to optimize objectives such as accuracy and rule/tree size, sensitivity and specificity, precision and recall, etc.

However, regardless of the domain of application, MOEAs are commonly used to optimize the efficacy and the complexity of the solutions produced. In the case of genetic programming, for instance, multi-objective genetic programming (MOGP) has been successfully applied to solve one of the most well-known GP problems: bloating.

Bloating [Banzhaf et al., 1998, Soule and Foster, 1998] is the uncontrolled growth of genetic programming code. There are many possible reasons to explain this phenomenon, being the non-homologous nature of crossover one of them [Bleuler et al., 2001]. Controlling bloating means saving resources which otherwise would be wasted evaluating long individuals whose additional code does not lead to better fitness, and that have a weaker generalization power than smaller individuals.

Two types of techniques are commonly applied to reduce or remove the effects of bloating in GP: the first ones change the structure of the programs or the evolutionary operators, by using techniques such as automatically defined functions (ADF) and explicitly defined introns; the second ones implicitly consider the size of the individuals, such as by defining their maximum size or inserting a penalty term to their fitness (parsimony pressure).

Nevertheless, MOGPs have also proved to be an effective technique to control bloating by optimizing simultaneously the individual's size and fitness. For instance, by using different types of MOGPs both Bleuler et al. [2001] and De Jong et al. [2001] showed that they can outperform four conventional methods used to prevent bloating. While Bleuler et al. [2001] optimized programs' size and fitness, De Jong et al. [2001] also took into account the diversity of the individuals generated.

It is important to point out that, in this work, it is not the size of the individuals which is being optimized, but the size of the rule sets generated by the individuals, which are rule induction algorithms. Bloating is tackled by setting a maximum size to the individuals, based on the depth of the derivation trees which can be generated by the grammar. However, using the size of the individuals themselves as a way to prevent bloating is one of the tasks proposed for future work.

5 Grammar-based Genetic Programming for Automatically Evolving Rule Induction Algorithms

In Section 2, two types of GGP were introduced, and some systems used to evolve a rule set *specific* to a single data set were described. This section presents a multi-objective grammar-based genetic programming algorithm which evolves *generic* rule induction algorithms following the sequential-covering approach, introduced in Section 3. This system is an extension of the GGP system proposed in Pappa and Freitas [2006], and is based on the solution-encoding individual approach, where the grammar is used to create the individuals in the GGP initial population, instead of taking part in a genotype/phenotype mapping process (production-rule-sequence-encoding individual approach). To the best of our knowledge, there is no significant research which indicates that either the solution-encoding individual approach or the production-rule-sequence-encoding individual approach is superior to the other. Hence, in the absence of significant evidence in favor of any of these two approaches with respect to their effectiveness, we chose to use the solution-encoding-individual approach. This choice was based on the fact that, as this approach lacks a genotype-phenotype mapping process, there is no need to worry about how effective the mapping is, how large is the degree of epistasis (interaction among genes) at the genotype level, or how exactly the genotype should be defined.

However, we make no claim that the solution-encoding individual approach is superior for our problem domain. Running computational experiments comparing the relative effectiveness of this approach and the production-rule-sequence-encoding approach is a topic left for further research. The next two sub-sections describe the single-objective version of the GGP introduced in Pappa and Freitas [2006], and the extensions added to it to deal with multi-objective optimization.

5.1 The Single-Objective GGP (SGGP)

As explained before, in the SGGP first proposed in Pappa and Freitas [2006] each individual represents a new rule induction algorithm, potentially as complex as well-known algorithms, such as CN2 [Clark and Boswell, 1991], and is represented by a derivation tree such as the one illustrated in Fig. 4. In order to extract from the tree the pseudo-code of the corresponding rule induction algorithm, we read all the terminals (leaf-nodes) in the tree from left to right. The tree in Figure 4, for example, represents the pseudo-code of the "ordered version" of the CN2 algorithm (where the induced rules are used in a specific order when applied to the test set).

As the individuals are represented by derivation trees, both the crossover and mutation operators have to be adapted, so that only individuals which are valid according to the production rules of the grammar are generated, as described in Section 2.1.

The grammar used to generate the first population of individuals and guide the genetic operations contains knowledge about how humans design sequential-covering rule induction algorithms, and was created after an extensive survey of

Table 1. The grammar used by the GGP, adapted from [Pappa and Freitas, 2006]

```
<Start> ::= (<CreateRuleSet>|<CreateRuleList>)[<PostProcess>].
<CreateRuleSet> ::= forEachClass <whileLoop> endFor <RuleSetTest>.
<CreateRuleList> ::= <whileLoop> <RuleListTest>.
<whileLoop>::= while <condWhile> <CreateOneRule> endWhile.
<condWhile>::= uncoveredNotEmpty |uncoveredGreater
              (10| 20| 90%| 95%| 97%| 99%) trainEx.
<RuleSetTest> ::= lsContent |confidenceLaplace.
<RuleListTest>::= appendRule | prependRule.
<CreateOneRule>::= <InitializeRule><innerWhile>
                   [<PrePruneRule>][<RuleStopCriterion>].
<InitializeRule> ::= emptyRule|randomEx|typicalEx|<MakeFirstRule>.
<MakeFirstRule> ::= NumCond1| NumCond2| NumCond3| NumCond4.
<innerWhile> ::= while (candNotEmpty| negNotCovered)<FindRule>endWhile.
<FindRule> ::= (<RefineRule>|<innerIf>)
                <EvalRule>[<StopCriterion>]<SelectCandRules>.
<innerIf> ::= if <condIf> then <RefineRule> else <RefineRule>.
<condIf> ::=  <condIfExamples> | <condIfRule>.
<condIfRule> ::= ruleSizeSmaller (2| 3| 5| 7).
<condIfExamples> ::= numCovExp ( >| <)(90%| 95%| 99%).
<RefineRule> ::= <AddCond>| <RemoveCond>.
<AddCond> ::= Add1| Add2.
<RemoveCond>::= Remove1| Remove2.
<EvalRule>::= confidence | Laplace| infoContent| infoGain.
<StopCriterion> ::= MinAccuracy (0.6|0.7|0.8)|
                    SignificanceTest (0.1|0.05|0.025|0.01).
<SelectCandRules> ::= 1CR| 2CR| 3CR| 4CR| 5CR| 8CR| 10CR.
<PrePruneRule> ::= (1Cond| LastCond| FinalSeqCond) <EvaluateRule>.
<RuleStopCriterion> ::= accuracyStop (0.5| 0.6| 0.7).
<PostProcess> ::= RemoveRule EvaluateModel| <RemoveCondRule>.
<RemoveCondRule> ::= (1Cond| 2Cond| FinalSeq) <EvaluateRule>.
```

the rule induction literature. It also includes knowledge that we thought would be worth testing in the context of rule induction but, to the best of our knowledge, was not used in this context before (for instance, the concept of typical instance [Zhang, 1992] was borrowed from the instance-based learning literature in order to create a first rule to be refined by a candidate rule induction algorithm). At last, it also contains a set of traditional programming statements, such as conditional statements and *while* loops, which enrich the way rules are refined or pruned. The grammar is shown in Table 1, and is composed of 83 production rules, creating a search space of over a billion candidate rule induction algorithms [Pappa, 2007]. For a detailed explanation of the grammar the reader is referred to Pappa and Freitas [2006].

This same grammar and individual representation are used in this paper, but combined with a multi-objective approach, as described in the next subsection.

5.2 The Multi-Objective GGP

This section introduces an extended version of the single objective-GGP proposed in Pappa [2007]. The main modifications were introduced in the fitness calculation process and the selection and elitism procedures, besides the mechanism which selects the best individual to be returned to the user. Before describing these modifications, we introduce the multi-objective concept followed by the algorithm.

According to the multi-objective optimization concept, when many objectives are simultaneously optimized, there is no single optimal solution. Rather, there is a set of optimal solutions, each one considering a certain trade-off among the objectives [Coello, 1999, Coello et al., 2002]. Hence, a system developed to solve this kind of problem returns a set of optimal solutions, and can be left to the user to choose the one that best solves his/her specific problem. This means that the user has the opportunity for choosing the solution that represents the best trade-off among the conflicting objectives after examining several high-quality solutions. Intuitively, this is better than forcing the user to define a single trade-off before the search is performed. This is what happens when the multi-objective problem is transformed into a single-objective one by assigning weights to each objective and combining all objectives into a single weighted formula.

The Pareto multi-objective optimization concept is used to find the set of optimal solutions. According to this concept, a solution S_1 dominates a solution S_2 if and only if [Deb, 2001]:

- Solution S_1 is not worse than solution S_2 in any of the objectives;
- Solution S_1 is strictly better than solution S_2 in at least one of the objectives.

Using these concepts, the solutions which are not dominated by any other in an EA population are said to form the estimated Pareto front. Note that the Pareto front returned by the GGP is just an estimation of the true, unknown Pareto front. The best estimated Pareto front found by the GGP is the set of solutions returned to the user, who can then select the best one according to his/her preference.

The System's Fitness Evaluation Process When developing a new rule induction algorithm, there are two objectives which should be simultaneously optimized: the predictive accuracy obtained when classifying a set of test examples (unseen during training), and the size (complexity) of the model (rule set) used to classify these new examples. These objectives are often conflicting, since many very accurate models are also very complex.

In this work, the values of these two objectives are obtained through the process of computation of each individual's fitness showed in Fig. 5. As illustrated, during this process the GGP individuals are converted into rule induction algorithms through a GGP/Java interface. The resulting rule induction algorithm is then trained and evaluated in a set of data sets named the meta-training set. It is important to note that 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 (i.e., each GGP individual) is trained with 70% of the examples, and then validated in the remaining 30%. For each data set, a rule model is produced from the training set, and a classification accuracy in the validation set is computed. To avoid overfitting, at each generation the train and test sets in which the algorithms are trained change.

In the first, single-objective version of our system [Pappa and Freitas, 2006], only the predictive accuracy was used to calculate a measure of fitness. However, in the new multi-objective version presented in this paper, we aim at optimizing two objectives:

1. The value of a normalized accuracy rate used in Pappa and Freitas [2006], described in Eq. (1), where i corresponds to a data set in the meta-training set, should be maximized.
2. The number of rule conditions in the produced rule model should be minimized.

$$norm\_acc_i = \begin{cases} \frac{Acc_i - DefAcc_i}{1 - DefAcc_i}, \text{ if } \mathrm{Acc}_i > \mathrm{DefAcc}_i \\ \frac{Acc_i - DefAcc_i}{DefAcc_i}, \text{ otherwise} \end{cases} \quad (1)$$

According to the definition of $norm\_acc_i$, 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 $DefAcc_i$). The default accuracy is obtained when using the trivial strategy of assigning to each test example the most frequent class among the training examples.

Hence, $normacc_i$ returns a value between -1 (when $Acc_i = 0$) and 1 (when $Acc_i = 1$). The motivation for Eq. (1) is that the degree of difficulty of the classification task depends strongly on the value of $DefAcc_i$ (the default accuracy associated with the $i-th$ data set). The above fitness function recognizes this and returns a positive value of $norm\_acc_i$ when $Acc_i > DefAcc_i$. For instance, if $DefAcc_i = 0.95$, then $Acc_i = 0.90$ would lead to a negative value of fit_i, as it should.

Note that, in this case, we have to be able to treat one special case: individuals whose rule induction algorithms produce models consisting of only one empty rule, i.e., individuals predicting the most frequent (default) class among the training examples for all the new examples in the validation set. For these individuals, the number of conditions in the produced model will always be 0. As we are dealing with a minimization problem, 0 represents the minimum possible value, but in this case the results obtained by the candidate rule induction algorithm is not a good result, representing only a trivial "majority classifier". To avoid this situation, the number of rule conditions of a model with 0 conditions is set to 100,000.

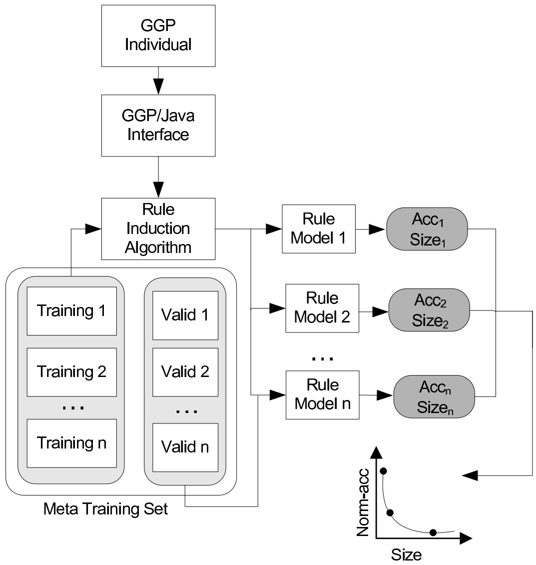

Fig. 5. Calculating the fitness

Observe that the MOGGP used in this work does not set a single aggregated value for the fitness of the individuals, like NSGA-2 [Deb et al., 2000] (which assigns fitness values based on rankings) or SPEA-2 [Zitzler et al., 2002] (which assigns fitness values based on the strength of the individuals' dominators). The individuals selected for tournament are simply compared among themselves using the two objectives to be optimized, and the tie-breaking criteria which will be introduced in the next subsection.

Selection Scheme and Elitism Apart from the fitness evaluation of the individuals, three other major modifications were introduced to the single-objective GGP. First, in the GGP selection process, the individuals have to be selected according to a relationship of Pareto dominance instead of a simple fitness value. In a tournament selection of size 2, for instance, which is the method used in this system, the winner of the tournament is the individual that dominates the other. In the case that neither of the individuals dominates the other, a tie-breaking criterion decides the winner.

This tie-breaking criterion considers the difference between the number of individuals in the entire population which are dominated by an individual and the number of individuals in the population which dominate that individual [Pappa et al., 2004]. This function is named ftb. If neither Ind_1 dominates Ind_2 nor vice-versa with respect to accuracy and rule set size, the winner of the tournament is the individual with the largest value of ftb (if the ftb values for both individuals are equal, one of them is selected at random).

While the MOGGP used in this paper uses the ftb as tie-braking criteria for tournament ties, other multi-objective algorithms - such as NSGA-2 and SPEA-2, give preference to solutions found in the less crowded regions of the solution space. Although our MOGGP does not implement any mechanisms which enforce the distribution of the solutions in different areas of the search space, its results in previous data mining applications [Pappa et al., 2004] showed a good diversity of solutions in the Pareto front. However, a comparative study between the MOGGP presented in this paper, NSGA-2 and SPEA-2 is left for future research.

The second modification concerns the elitism scheme used by the algorithm. The single-objective version of the GGP preserves the best individual found by the GGP during the evolution process. This individual is the one returned to the user.

In the case of the multi-objective GGP, at each generation, all the solutions in the current estimated Pareto front (individuals not dominated by any other individual in the population) are passed to the next generation by elitism, as long as their number does not exceed half of the size of the population. If they do, then the individuals with the highest value of ftb are preserved. If after applying ftb the number of individuals is still higher than half of the population size, the individuals with better $norm\_acc$ (function defined in Eq.(1)) are given priority. When selecting the best individual to be returned to the user, at the last generation, this same logic is applied. The returned individual is then tested in a meta-test set.

The evolution process Based on the grammar and individual representation just explained, the proposed GGP works as a classical GP. After the first population is created, the two objectives to be optimized are calculated, and the estimated Pareto front formed. The individuals in that Pareto front are passed through the next generation by elitism, as long as the number of individuals in the front is not greater than half of the number of individuals in the population. The two objectives are then used to selected a subset of individuals through a tournament selection of size two to breed and undergo reproduction, crossover and mutation operations. Recall that the individuals generated by the latter two operators have also to be valid according to the grammar. The number of individuals selected to undergo these operations is given by the population size minus the number of individuals selected by elitism. The individuals generated by these operations are inserted into a new population, representing a new generation of evolved individuals. The evolution process is carried out until a maximum number of generations is reached.

6 Experiments and Results

This section describes the experiments performed and the results obtained in order to evaluate the effectiveness of the MOGGP system to automatically evolve rule induction algorithms.

Table 2. Data sets used by the GGP in the Meta-Training Set

Data set	Examples	Attributes	Classes	Def. Acc.(%)
monks-2	169/432	6	2	67
monks-3	122/432	6	2	52
bal-scale-discr	416/209	4	3	46
lymph	98/50	18	4	54
zoo	71/28	16	7	43
glass	145/69	9	7	35.2
pima	513/255	8	2	65
hepatitis	104/51	20	2	78
vehicle	566/280	18	4	26
vowel	660/330	13	11	9

6.1 Methodology

As this paper presents an extended version of a single-objective GGP (from now on referred as SGGP) introduced in Pappa and Freitas [2006], we first compare the proposed system with its single objective version. Next, we compare the results obtained by the MOGGP-RIs (rule induction algorithms generated by the MOGGP) with three well-known human-designed rule induction algorithms.

The MOGGP system presented in Section 5 requires two sets of parameters: a set of standard EA parameters, such as population size and number of generations; and the meta-training and meta-test set parameters, which include the number of data sets in each of the meta-data sets and which data sets will be included in each of the meta-data sets. In order to make the comparison between the MOGGP and the SGGP fair, the two sets of parameters used in these experiments are the same for both versions of the GGP: population size of 100, evolved over 30 generations, tournament size of 2, crossover rate of 0.7, mutation rate of 0.25 and reproduction rate of 0.05.

Note that these parameter values were chosen after some initial experimentation, but cannot be considered optimal. A population composed of 200 individuals was previously tested, but the results obtained were not considered statistically better than the ones obtained with a population of 100 individuals. Considering this result and the time necessary to evaluate each individual, a population of 100 was used. Although this population size might seem small when compared to conventional GP values, we cannot forget that the GGP's search space is guided by the grammar. Hence, a smaller population might obtain the same results as a bigger population with no guiding mechanism. Ratle and Sebag [2001], for example, noticed that their SG-GP (Stochastic Grammar-based Genetic Programming) obtained better results when run with smaller populations and more generations than conventional GP systems.

Regarding the second set of parameters, i.e., the distribution of data sets in meta-training and meta-test sets, the 20 data sets used are presented in Tables 2 and 3. While Table 2 shows the data sets inserted in the meta-training set, Table 3 shows the data set used in the meta-test set. All the data sets were obtained

Table 3. Data sets used by the GGP in the Meta-Test Set

Data set	Examples	Attributes	Classes	Def. Acc.(%)
crx	461/229	15	2	67.7
segment	1540/770	19	7	14.3
sonar	139/69	60	2	53
heart-c	202/101	13	2	54.5
ionosphere	234/117	34	2	64
monks-1	124/432	6	2	50
mushroom	5416/2708	23	2	52
wisconsin	456/227	9	2	65
promoters	70/36	58	2	50
splice	2553/637	63	3	52

Table 4. Comparing the MOGGP-RIs to other rule induction algorithms, taking into account both accuracy and number of the conditions in the produced rule model according to the concept of Pareto dominance

	Neutral	Dominates	Dominated
MOGGP-RI *vs* SGGP-RIs	4	6	0
MOGGP-RI *vs* Baseline Algorithms	7	23	0

from the well-known UCI (University of California at Irvine) data set repository [Newman et al., 1998], which is often used to benchmark rule induction algorithms. In both Tables 2 and 3, the figures in the column *Examples* indicate the number of examples present in the training and validation data sets – numbers before and after the "/", respectively, followed by the number of attributes and classes. Note that both data sets in the meta-training and meta-test sets are divided into training and validation sets. In the case of the meta-training set – which is used to evaluate individuals in all generations – the training set is used by the evolved rule induction algorithm (individual) to build a model, while the validation data set is used to test the model and generate accuracy rates. These accuracy rates are then used to calculate the individuals' fitness. In the case of the data sets in the meta-test set, which are only used to test the best individual at the end of the evolutionary process, the numbers in the column Examples represent an estimation of the size of the training and validation sets, since a cross-validation procedure is executed. The last column shows the default accuracy. As explained before, it is the accuracy obtained when using the trivial strategy of assigning to each test example the most frequent class among the training examples.

We selected the data sets which compose the meta-training set based on the execution time of the rule induction algorithms, so that we included in the meta-training set the data sets leading to faster runs of the rule induction algorithms.

After the MOGGP returns the best evolved rule induction algorithm, that algorithm is applied to each data set in the meta-test set, using a well-known cross-validation procedure [Witten and Frank, 2005]. Hence, the measures of accuracy of the best evolved rule induction algorithm to be reported later refer to predictive accuracy on a test set unseen during the training of the algorithm, as usual. However, analyzing the results obtained for accuracy and number of conditions present in the rule model separately when analyzing the results of the MOGGP is not the best approach. This is because, as explained before, the main objective of using a multi-objective approach is to find the best trade-off between predictive accuracy and the simplicity of the models obtained. Therefore, taking into account both the predictive accuracy and the number of conditions in the rule model when evaluating the MOGGP-RIs is essential. The next subsections describe the results obtained by the MOGGP-RIs and show two of the algorithms evolved.

6.2 MOGGP-RIs Performance

As explained before, analyzing the results obtained for accuracy and number of conditions present in the rule model separately when analyzing the results of the MOGGP is not the best approach. Table 4 makes an analysis using both these objectives simultaneously, and uses a special terminology. When evaluating the MOGGP-RIs regarding accuracy and number of conditions in the rule model (2 objectives), we based our analysis of results on the Pareto dominance concept – adapted to consider statistically significant differences. This adapted Pareto dominance concept states that a solution S_1 dominates a solution S_2 if two conditions are satisfied. First, if every objective value of S_1 is not statistically significantly worse than the corresponding objective value in S_2. Secondly, if at least one of the objective values of S_1 is statistically significantly better than the corresponding objective value of S_2 (statistical significance is determined by the results of a Student's t-test with significance level 0.05).

Hence, Table 4 presents the number of classification models produced by the MOGGP-RIs which neither dominate nor are dominated by the classification models produced by the SGGP-RIs or the baseline algorithms (*Neutral* column), the number of models produced by the SGGP-RIs or the baseline algorithms which the MOGGP-RIs dominate (*Dominate* column), and finally the number of models produced by the MOGGP-RIs which are dominated by a model produced by the SGGP-RIs or a baseline algorithm (*Dominated* column). This table was built by applying the statistical significance-adapted Pareto dominance concept to the results presented in Tables 5 and 6.

It is important to emphasize that the results in Tables 5 and 6 are reported here only for the sake of completeness, as the objective of this paper is to generate rule induction algorithms where both accuracy and rule model comprehensibility are equally important. This approach is suitable particularly for applications where, in practice, the classification model will not be directly used to predict the class of individual data instances, but will rather be interpreted by a user expert in the application domain and the data, in order to try to provide the user with new insights about the domain or the data. There are several applications where classification models are induced mainly to be interpreted by users, such as biomedical applications and bioinformatics [Mirkin and Ritter, 2000,

Table 5. Comparing predictive accuracy rates (%) of the MOGGP-RI and the baseline algorithms for data sets in the meta-test set

Data Set	MOGGP-RIs	SGGP-RIs	OrdCN2	UnordCN2	C45Rules
crx	83.33±1.26	77.46±3.8	80.16 ± 1.27	80.6 ± 0.93	84.82 ± 1.53
segment	92±0.67	95.06±0.26	95.38 ± 0.28	85.26 ± 0.87	88.16 ± 7.72
sonar	68.04±1.74	72.34±1.91	70.42 ± 2.66	72.42 ± 1.4	72.4 ± 2.68
heart-c	76.46±1.82	76.72±1.5	77.9 ± 1.96	77.54 ± 2.85	74.2 ± 5.43
ionosphere	85.48±1.63	87.04±2.2	87.6 ± 2.76	90.52 ± 2.03	89.06 ± 2.71
monks-1	99.78±0.22	99.93±0.07	100 ± 0	100 ± 0	100 ± 0
mushroom	99.66±0.22	99.98±0.02	100 ± 0	100 ± 0	98.8 ± 0.06
wisconsin	92.1±0.71	95.58±0.74	94.58 ± 0.68	94.16 ± 0.93	95.9 ± 0.56
promoters	71.84±5.24	78.98±2.93	81.9 ± 4.65	74.72 ± 4.86	83.74 ± 3.46
splice	87.68±0.5	88.68±0.31	90.32 ± 0.74	74.82 ± 2.94	89.66 ± 0.78

Table 6. Comparing the number of the conditions in the rule sets of the MOGGP-RI and the baseline algorithms for data sets in the meta-test set

Data Set	MOGGP-RIs	SGGP-RIs	OrdCN2	UnordCN2	C45Rules
crx	13.52±0.72	99.4±5.98	101.4 ± 3.46	101.6 ± 2.38	34 ± 1.38
segment	25.64±1.22	83.2±6.13	73.8 ± 1.74	102.8 ± 2.15	96.8 ± 12.71
sonar	4.6±0.75	20.2±1.32	19.4 ± 0.87	50.2 ± 3.02	14.6 ± 4.3
heart-c	7.2±0.9	50.6±2.92	37.2 ± 1.24	70 ± 3.54	22 ± 5.63
ionosphere	7.88±0.62	24.2±1.53	20.2 ± 1.53	37 ± 1.84	10.2 ± 4.34
monks-1	11.64±0.39	13±2.05	11 ± 0.71	61 ± 0	61 ± 0
mushroom	15.16±0.38	15.2±0.58	15.6 ± 0.24	26 ± 0	18.6 ± 2.73
wisconsin	9.68±0.57	48±10.62	32.6 ± 1.36	53.8 ± 2.91	19.2 ± 0.86
promoters	3.96±0.38	14.6±2.27	10.4 ± 0.75	23.6 ± 1.36	10.8 ± 1.02
splice	42.52±2.3	271.8±12.02	256.2 ± 5.08	172.6 ± 9.75	119.8 ± 29.68

Clare and King, 2002, Pazzani, 2000, Fayyad et al., 1996]. In such applications, discovering comprehensible knowledge is desirable for several reasons [Szafron et al., 2004], such as increasing the confidence of the user in the system's results, leading to new insights about the data and the formulation of new hypothesis, and detecting errors in the data. For instance, several new insights about the application domain provided by a comprehensible classification model are discussed in [Karwath and King, 2002], in the context of protein function prediction. As another example of the usefulness of comprehensible models, even if the predictive accuracy is not very good, Wong and Leung [2000] discovered classification rules with a relatively low or moderate degree of accuracy (around 40-60%) that were considered, by senior medical doctors, novel and more accurate than the knowledge of some junior doctors.

Tables 5 and 6 present the predictive accuracies and the number of conditions per rule model obtained by the MOGGP-RIs, respectively. In both tables, the results obtained by the MOGGP-RIs (second column) are followed by the results obtained by the SGGP-RIs (rule induction algorithms produced by the single-objective GGP) and three baseline well-known rule induction algorithms, namely

Ordered-CN2, Unordered-CN2 and C4.5Rules. Observe that the C4.5Rules algorithm is out of the search space of the MOGGP, as it does not follow the sequential-covering approach. Instead, it first generates a decision tree and later extracts rules from it [Quinlan, 1993]. However, as C4.5Rules is a baseline of comparison for many studies involving rule induction algorithms, it is also considered here.

As pointed out before, all the results were obtained using a 5-fold cross-validation procedure, and the numbers after the symbol "±" are standard deviations. Results were compared using a two-tailed statistical Student's t-test with significance level 0.05, performed over 5 independent runs of the GGP for each data fold. Cells in dark gray represent statistically significant wins of the MOGGP-RIs against the other algorithms, while light gray cells represent MOGGP-RIs' statistically significant losses. Note that, although the MOGGP finds a set of non-dominated solutions, in practice we want to select a single solution (rule induction algorithm) to compare it with the baseline algorithms. Instead of manually choosing a single rule induction algorithm among the non-dominated ones, to keep the experiments as automated and unbiased as possible we have also automated that selection. That is, the MOGGP returns the non-dominated rule induction algorithm (in the last generation) having the largest value of the ftb function. The selected rule induction algorithm is then called MOGGP-RI.

To illustrate the logic behind the results presented at Table 4, let us consider the cases of *ionosphere* and *mushroom* with OrdCN2. In both cases the accuracies of the MOGGP-RIs and OrdCN2 are competitive – i.e, the difference in these two accuracies is not statistically significant. However, in *ionosphere* the model generated by the MOGGP-RIs has a significantly smaller number of rule conditions than the model generated by the OrdCN2, and so we say that the MOGGP-RIs dominates OrdCN2. A different situation occurs for *mushroom*, where the size of the models generated by both the MOGGP-RIs and CN2Ord are also competitive, and so we say that the MOGGP-RIs and CN2Ord have a neutral relationship. Finally, the MOGGP-RIs are said to be dominated by another algorithm if they are significantly worse in one objective (accuracy or model size) and not significantly better in another. In other words, if the MOGGP-RIs obtain a significantly smaller accuracy (or larger rule set) than the algorithm in question and at the same time does not discover a significantly smaller rule set (larger accuracy), then it is said to be dominated by the respective algorithm.

An analysis of statistical significance-based Pareto dominance taking into account both the accuracy and the size of the models produced shows that in 4 out of 10 cases the MOGGP-RIs and the SGGP-RIs present a neutral relationship, while in the other 6 cases the MOGGP-RIs' models dominate the SGGP-RIs' models. In contrast, when comparing the MOGGP-RIs with the baseline algorithms, in 23 out of 30 cases (3 algorithms $\times$ 10 data sets) the MOGGP-RIs' models dominate the baseline algorithms' models, and the former are never dominated by the latter.

As stated before, the objective of this paper is to generate rule induction algorithms which are both accurate and generate comprehensible rule models. Hence, Appendix 1 shows the rule lists obtained by both the MOGGP, the SGGP and the C4.5 algorithms for the data set *crx* (Tables 7, 8 and 9). The major problem in many data mining studies is that there is no specialist who can evaluate the interpretability and interestingness of the rules. That is the main reason most studies consider simplicity, measured in terms of rule set size, as a proxy for interpretability - although we understand this is not a perfect metric. However, just looking at the rules listed in Appendix 1 we realize that the rule lists generated by the MOGGP are simpler than the ones obtained by both the SGGP and the C4.5 algorithms. Importantly, the significant reduction in rule set size obtained by the MOGGP was achieved without any significant reduction in classification accuracy, by comparison with the rule set sizes and classification accuracies obtained by SGGP and C4.5 in this data set.

Figure 6 shows a graph representing the objective values for the last population of individuals evolved by a run of the MOGGP. The x-axis shows the average number of rule conditions in the models built from data sets of the meta-training set, and the y-axis shows the value of *norm_acc* as defined in Eq. (1). Each point in the graph represents an individual (a rule induction algorithm). The greater the accuracy (and consequently the value of *norm_acc*) and the smaller the number of rule conditions, the better. The graph also shows the estimated Pareto front found by the MOGGP, which is formed by the set of individuals which are not dominated by any other individual in the population of the last generation. The circle indicates the individual in the Pareto front which was returned as the best single solution for the problem. The rule induction algorithm represented by this individual was the one evaluated in the meta-test set (unseen during the evolution of the MOGGP).

All the experiments were performed on pentium 4 duo processor machines with 1GB RAM and running Linux. Each run of the GGP (including the execution of the evolved algorithms in the meta-test set) took from 18 to 34 hours. The variation in time is due to the variety of operations the rule induction algorithms (individuals) could perform. A few of these operation, such as inserting two conditions at a time to new candidate rules, had non-linear time-complexities.

In summary, when comparing the MOGGP-RIs with the SGGP-RIs, in more than half of the cases the MOGGP-RIs are able to obtain statistically similar accuracies as the ones obtained by the SGGP-RIs but using much simpler rule models. The same is true when comparing the MOGGP-RIs with the three human-designed rule induction algorithms. However, it is also important to mention that, if we had chosen to perform an analysis which would give more weight to accuracy than to model simplicity, in 2 out of 10 cases the SGGP would produce better accuracies than the MOGGP. At the same time, in 4 out of 30 cases the other three rule induction algorithms would produce better accuracies than the MOGGP. The MOGGP, on contrast, would be better than Unordered-CN2 in two cases, and better than C4.5Rules in one case.

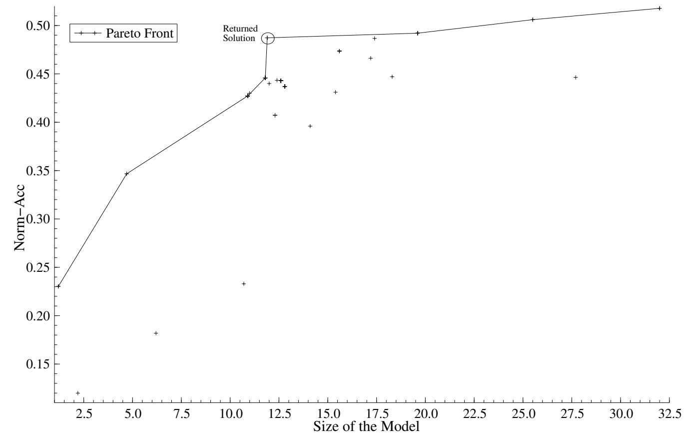

Fig. 6. Objective values for the last population of individuals evolved by the MOGGP in the meta-training set

Although in terms of accuracy the SGGP and the other three rule induction algorithms would be better than the MOGGP in two and four cases, respectively, the models produced by the MOGGP in 34 out of 40 cases are significantly simpler when compared with the SGGP and the human-designed rule induction algorithms. However, if accuracy is to be considered as a more important objective, a different type of MOGGP, such as the ones following a lexicographic approach [Freitas, 2004], should be used.

After presenting the results in terms of accuracy and rule model size, one question is left to answer: how different are the evolved MOGGP-RIs from the evolved SGGP-RIs and the baseline human-designed algorithms? This is the subject of the next subsection.

6.3 An Insight About the MOGGP-RIs

The most interesting fact to notice about the MOGGP-RIs is the impact that the inclusion of a measure of rule model size in the fitness function had in their design.

For instance, the five rule induction algorithms produced by the MOGGP (in 5 runs with different random seeds) use both a pre- and a post-pruning method. In contrast, none of the algorithms produced by the single-objective GGP used post-pruning, and only one out of the 5 algorithms pre-pruned rules by changing the final produced rule. Of course other forms of pre-pruning were used by these algorithms, such as considering as candidate rules only statistically significant rules, or rules with an accuracy greater than a predefined threshold.

Alg. 1 shows an example of a MOGGP-RI. The first feature to notice in this algorithm is that it works with three different sets of data. It first divides the training set into build and post-prune sets, and subsequently divides the build

Algorithm 1: Example of a MOGGP-RI (Rule Induction Algorithm evolved by the Multi-objective GGP)

```
Divide the training data in Build and PostPrune
RuleList = ∅
repeat
    Divide the Build data in Grow and PrePrune
    bestRule = an empty rule
    candidateRules = ∅
    candidateRules = candidateRules ∪ bestRule
    while candidateRules ≠ ∅ do
        for each candidateRule CR do
            newCR = ∅
            Add 1 condition-at-a-time to CR
            Evaluate CR using the Laplace Est. in Grow
            if CR is significant with 99% significance level then
                newCR = newCR ∪ CR
        candidateRules = best rule selected from newCR
    repeat
        bestRule' = Rule produced by removing the last condition from
        bestRule
        Evaluate bestRule' in PrePrune using Laplace Est.
        if bestRule' better than bestRule then
            bestRule = bestRule'
    until accuracy(bestRule') ≥ accuracy(bestRule)
    RuleList = RuleList ∪ bestRule
until there is a maximum of 20 examples in Grow not covered by any rule
RuleList' = RuleList
repeat
    Remove 1 condition-at-a-time from the last rule in RuleList'
until accuracy(RuleList') ≥ accuracy(RuleList) in PostPrune
```

set into grow and pre-prune sets. We recognize that this data division might be problematic if few examples are available from the training set. However, this algorithm obtained accuracies statistically competitive with the ones obtained by the well-known human-designed algorithms, together with much simpler rule models.

Alg. 1 works in three phases. First, it greedily builds rules and evaluates them using the Laplace estimation criterion in the grow set. It also tests the statistical significance of the rules before considering them as possible candidate rules, and only selects the best candidate rule to undergo further refinements. Once the best rule is found, it is pre-pruned. In this second phase, the algorithm pre-prunes a rule by removing only the last condition added to it. The new generated rules are again evaluated using the Laplace estimation criterion in the pre-prune set.

Rules are produced until the number of examples in the grow set is reduced to 20 or less. Once the rule list is complete, the third phase starts. The rule list is post-pruned by trying to remove, from the last to the first inserted rule, one

Algorithm 2: Example of a SGGP-RI (Rule Induction Algorithm evolved by the Single-objective GGP)

```
RuleList = ∅
repeat
    bestRule = an empty rule
    candidateRules = ∅
    candidateRules = candidateRules ∪ bestRule
    while candidateRules ≠ ∅ do
        for each candidateRule CR do
            newCR = ∅
            if size(bestRule) < 2 then
                Add 1 condition-at-a-time to CR
            else
                Add 2 conditions-at-a-time to CR
            Evaluate CR using the Laplace Est.
            if laplace(CR) > laplace(bestRule) then
                bestRule = CR
            newCR = newCR ∪ CR
        candidateRules = 5 best rules selected from newCR
    if accuracy(bestRule) < 0.7 then break
    else RuleList = RuleList ∪ bestRule
until all examples in the training set are covered
```

rule at-a-time. Rules are removed from the rule list while the accuracy of the system in the post-prune data set is not reduced.

The dynamics of Alg. 1 can be compared to Ripper's [Cohen, 1995]. Both algorithms produce, prune and "optimize" the entire rule list, with the following differences. Ripper works with rules ordered per class, produces only rules which do not cover any negative examples and uses the MDL (minimum description length) criterion when optimizing the set of discovered rules. Alg. 1 produces rule lists, evaluates rules using the Laplace estimation and "optimizes" rules by removing one condition-at-a-time from them. We do not know any algorithm in the literature which works in this way, so Alg. 1 is – to the best of our knowledge – considered a novel rule induction algorithm.

Alg. 2 shows an SGGP-RI produced by the SGGP when using the same random seed used by the MOGGP which produced Alg. 1. As observed, Alg. 2 is much simpler than Alg. 1. It does not use any advanced pruning technique, and simply stops producing rules when their predictive accuracy is lower than 70%. Note that Alg. 2 is also an innovative algorithm, since there is no rule induction algorithm in the literature which refines a rule according to the number of conditions already present on it.

The comparison between Alg. 1 and Alg. 2 shows how the addition of a measure of rule set complexity to the fitness of the GGP changed the evolved MOGGP-RIs in order to find a good balance of the accuracy/complexity trade-off. These results also show that the MOGGP is flexible enough to produce very different kinds of algorithms.

7 Conclusion and Future Work

This work presented a multi-objective version of a Grammar-based Genetic Programming (GGP) system which automatically evolves *generic* rule induction algorithms – rather than just rule sets to a *specific* application domain. Comparisons between the single-objective version (SGGP) and the proposed multi-objective version of the system (MOGGP) showed that the MOGGP produces rule induction algorithms with competitive accuracies and much more compact models. Considering application domains where model size (as an approximation to interpretability) is as important as predictive accuracy, these are positive results. When comparing the MOGGP with three well-known human-designed conventional (non-evolutionary) rule induction algorithms, similar conclusions were drawn, as in general the solutions produced by the MOGGP had competitive accuracies and significantly smaller models than the ones produced by Unordered-CN2, Ordered-CN2 and C45Rules.

As future steps in this research, other objectives could be optimized by the MOGGP, such as the complexity of the generated algorithm, and/or the time necessary to evaluate a data set with the evolved algorithms.

Moreover, as many works in the literature [Handl et al., 2007], this paper considers the accuracy of a model to be as important as the size of the generated rule models. However, in cases where the user chooses to give more weight to optimizing accuracy than the simplicity of the model, a lexicographic approach could be more appropriate [Freitas, 2004].

It would be interesting to implement a version of the system using a GGP following the production-rule-sequence-encoding individual, and compare its results with the ones obtained in this study. Finally, the current version of the grammar could be extended, so that the MOGGP will be able to build even more innovative rule induction algorithms.

Acknowledgment

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

References

A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques and Tools*. Addison-Wesley, 1^{st} edition, 1986.

W. Banzhaf, P. Nordin, R. Keller, and F. Francone. *GP – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, January 1998. ISBN 1-55860-510-X.

S. Bleuler, M. Brack, L. Thiele, and E. Zitzler. Multiobjective genetic programming: Reducing bloat using SPEA2. In *Proc. of the 2001 Congress on Evolutionary Computation - CEC2001*, pages 536–543, Korea, 27-30 2001. IEEE. ISBN 0-7803-6658-1. URL citeseer.ist.psu.edu/bleuler01multiobjective.html.

C. A. Brunk and M. J. Pazzani. An investigation of noise-tolerant relational concept learning algorithms. In L. Birnbaum and G. Collins, editors, *Proc. of the* 8^{th} *International Workshop on Machine Learning*, pages 389–393. Morgan Kaufmann, 1991.

A. Clare and R. D. King. Machine learning of functional class from phenotype data . *Bioinformatics*, 18(1):160–166, 2002.

P. Clark and R. Boswell. Rule induction with cn2: some recent improvements. In Y. Kodratoff, editor, *EWSL-91: Proc. of the Working Session on Learning*, pages 151–163, New York, NY, USA, 1991. Springer-Verlag. ISBN 0-387-53816-X.

R. Cleary. Extending grammar evolution with attribute grammars: An application to knapsack problems. Master's thesis, University of Limerick, Canberra, Australia, 2005.

C. A. Coello Coello. A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems*, 1(3): 129–156, 1999. URL citeseer.ist.psu.edu/article/coello98comprehensive.html.

C. A. Coello Coello, D.A. Van Veldhuizen, and G.B. Lamont. *Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, New York, USA, 2002.

W. W. Cohen. Efficient pruning methods for separate-and-conquer rule learning systems. In *Proc. of the* 13^{th} *Int. Joint Conf. on Artificial Intelligence (IJCAI-93)*, pages 988–994, France, 1993.

W. W. Cohen. Fast effective rule induction. In A. Prieditis and S. Russell, editors, *Proc. of the* 12^{th} *Int. Conf. on Machine Learning*, pages 115–123, Tahoe City, CA, 9–12, 1995. Morgan Kaufmann.

E. D. De Jong, R. A. Watson, and J. B. Pollack. Reducing bloat and promoting diversity using multi-objective methods. In L. Spector, E. Goodman, A. Wu, W.B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. Garzon, and E. Burke, editors, *Proc. of the Genetic and Evolutionary Computation Conference, GECCO-2001*, pages 11–18, San Francisco, CA, 2001. Morgan Kaufmann Publishers. URL citeseer.ist.psu.edu/dejong01reducing.html.

K. Deb. *Multi-objective Optimization using Evolutionary Algorithms*. Wiley Interscience series in Systems and Optimization, Berlin, 2001.

K. Deb, S. Agrawal, A. Pratab, and T. Meyarivan. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H. Schwefel, editors, *Proc. of the Parallel Problem Solving from Nature VI Conference*, pages 849–858, Paris, France, 2000. Springer. Lecture Notes in Computer Science No. 1917.

I. De Falco, A. Della Cioppa, A. Iazzetta, and E. Tarantino. An evolutionary approach for automatically extracting intelligible classification rules. *Knowledge and Information Systems*, 7(2):179–201, 2005a.

I. De Falco, E. Tarantino, A. Della Cioppa, and F. Fontanella. A novel grammar-based genetic programming approach to clustering. In *Proc. of the 2005 ACM Symposium on Applied Computing (SAC-05)*, pages 928–932, New York, NY, USA, 2005b. ACM Press. ISBN 1-58113-964-0.

U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery: an overview. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, 1996. ISBN 0-262-56097-6.

A. A. Freitas. *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer-Verlag, 2002. ISBN 3-540-43331-7.

A. A. Freitas. A critical review of multi-objective optimization in data mining: a position paper. *ACM SIGKDD Explor. Newsl.*, 6(2):77–86, 2004. ISSN 1931-0145.

J. Furnkranz. Separate-and-conquer rule learning. *Artificial Intelligence Review*, 13(1):3–54, 1999. ISSN 0269-2821.

J. Furnkranz and G. Widmer. Incremental reduced error pruning. In *Proc. the* 11^{th} *Int. Conf. on Machine Learning*, pages 70–77, New Brunswick, NJ, 1994.

F. Gruau. On using syntactic constraints with genetic programming. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 19, pages 377–394. MIT Press, Cambridge, MA, USA, 1996. ISBN 0-262-01158-1.

J. Handl and J. Knowles. Evolutionary multiobjective clustering. In *PPSN VIII: Proc. of the 8th Int. Conf. on Parallel Problem Solving from Nature*, pages 1081–1091, London, UK, 2004. Springer-Verlag.

J. Handl, D. B. Kell, and J. Knowles. Multiobjective optimization in bioinformatics and computational biology. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 4(2):279–292, 2007. ISSN 1545-5963.

M. L. Hetland and P. Saetrom. Evolutionary rule mining in time series databases. *Machine Learning*, 58(2):107–125, 2005.

N. X. Hoai, R. I. McKay, and H. A. Abbass. Tree adjoining grammars, language bias, and genetic programming. In C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, and E. Costa, editors, *Proc. of the* 6^{th} *European Conf. on Genetic Programming (EuroGP-03)*, volume 2610 of *Lecure Notes in Computer Science*, pages 335–344, Essex, 14-16 April 2003. Springer-Verlag. ISBN 3-540-00971-X.

T. Hussain and R. Browse. Network generating attribute grammar encoding. In *Proc. of IEEE Int. Joint Conf. on Neural Networks*, pages 431–436, 1998.

H. Jacobson. Rule extraction from recurrent neural networks: A taxonomy and review. *Neural Computation*, 17:1223–1263, 2005.

A. Karwath and R. King. Homology induction: the use of machine learning to improve sequence similarity searches. *BMC Bioinformatics*, 3:online publication, 2002.

R. E. Keller and W. Banzhaf. Genetic programming using genotype-phenotype mapping from linear genomes into linear phenotypes. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, editors, *Proc. of the* 1^{st} *Annual Conf. on Genetic Programming (GP-96)*, pages 116–122, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

M. H. Law, A. Topchy, and A.K. Jain. Multiobjective data clustering. In *Proc. of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 424–430, 2004. URL citeseer.ist.psu.edu/law04multiobjective.html.

T. Lim, W. Loh, and Y. Shih. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Machine Learning*, 40(3):203–228, 2000.

T. McConaghy and G. Gielen. Canonical form functions as a simple means for genetic programming to evolve human-interpretable functions. In *Proc. of the 8^{th} annual Conf. on Genetic and Evolutionary Computation (GECCO-06)*, pages 855–862, New York, NY, USA, 2006. ACM Press. ISBN 1-59593-186-4.

D. Michie, D. J. Spiegelhalter, C. C. Taylor, and J. Campbell, editors. *Machine learning, neural and statistical classification*. Ellis Horwood, Upper Saddle River, NJ, USA, 1994. ISBN 0-13-106360-X.

B. Mirkin and O. Ritter. A feature-based approach to discrimination and prediction of protein folding groups. In *Genomics and Proteomics*, pages 155–177. Springer, 2000.

D. J. Newman, S. Hettich, C.L. Blake, and C.J. Merz. UCI repository of machine learning databases, 1998. URL http://www.ics.uci.edu/∼mlearn/MLRepository.html.

M. O'Neill and C. Ryan. *Grammatical Evolution Evolutionary Automatic Programming in an Arbitrary Language*. Morgan Kaufmann, 2003.

M. O'Neill, A. Brabazon, C. Ryan, and J. J. Collins. Evolving market index trading rules using grammatical evolution. In E. J. W. Boers, S. Cagnoni, J. Gottlieb, E. Hart, Pier L. Lanzi, G. R. Raidl, Robert E. Smith, and H. Tijink, editors, *Applications of Evolutionary Computing*, volume 2037 of *LNCS*, pages 343–352. Springer-Verlag, 2001.

A. Ortega, M. de la Cruz, and M. Alfonseca. Christiansen grammar evolution: Grammatical evolution with semantics. *Evolutionary Computation, IEEE Trans. on*, 11(1):77–90, 2007. ISSN 1089-778X.

G. Pagallo and D. Haussler. Boolean feature discovery in empirical learning. *Machine Learning*, 5(1):71–99, 1990.

G. L. Pappa. *Automatically Evolving Rule Induction Algorithms with Grammar-based Genetic Programming*. PhD thesis, Computing Laboratory, University of Kent, Cannterbury, UK, Submitted in August 2007.

G. L. Pappa and A. A. Freitas. Automatically evolving rule induction algorithms. In J. Fuernkranz, T. Scheffer, and M. Spiliopoulou, editors, *Proc. of the 17th European Conference on Machine Learning*, volume 4212 of *Lecture Notes in Computer Science*, pages 341–352, Berlin, September 2006. Springer Berlin / Heidelberg.

G. L. Pappa and A. A. Freitas. Discovering new rule induction algorithms with grammar-based genetic programming. In O. Maimon and L. Rokach, editors, *Soft Computing for Knowledge Discovery and Data Mining*, pages 177–196. Springer-Verlag, 2007.

G. L. Pappa, A. A. Freitas, and C. A. A. Kaestner. Multi-objective algorithms for attribute selection in data mining. In C. A. Coello Coello and G.B. Lamont, editors, *Applications of Multi-Objective Evolutionary Algorithms*, pages 603–626. World Scientific, 2004.

M. J. Pazzani. Knowledge discovery from data? *IEEE Intelligent Systems*, 15 (2):10–13, 2000. ISSN 1541-1672.

J. R. Quinlan. Induction of decision trees. In J. W. Shavlik and T. G. Dietterich, editors, *Readings in Machine Learning*. Morgan Kaufmann, 1990. Originally published in *Machine Learning* 1:81–106, 1986.

J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann, 1993.

A. Ratle and M. Sebag. Genetic programming and domain knowledge: Beyond the limitations of grammar-guided machine discovery. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H. Schwefel, editors, *Proc. of the 6^{th} Int. Conf. on Parallel Problem Solving from Nature (PPSN)*, pages 211–220. Springer Verlag, 2000.

A. Ratle and M. Sebag. Avoiding the bloat with probabilistic grammar-guided genetic programming. In P. Collet, C. Fonlupt, J.-K. Hao, E. Lutton, and M. Schoenauer, editors, *5th Int. Conf., Evolution Artificielle, EA 2001*, volume 2310, pages 255–266, Creusot, France, October 29-31 2001. Springer Verlag. ISBN 3-540-43544-1.

C. Romero, S. Ventura, and P. De-Bra. Knowledge discovery with genetic programming for providing feedback to courseware authors. *User Modeling and User-Adapted Interaction*, 14(5):425–464, 2005. ISSN 0924-1868.

B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. The MIT Press, 2002.

T. Soule and J. A. Foster. Effects of code growth and parsimony pressure on populations in genetic programming. *Evol. Comput.*, 6(4):293–309, 1998.

D. Szafron, P. Lu, R. Greiner, D. S. Wishart, B. Poulin, R. Eisner, Z. Lu, J. Anvik, C. Macdonell, A. Fyshe, and D. Meeuwis. Proteome analyst: custom predictions with explanations in a web-based tool for high-throughput proteome annotations. *Nucl. Acids Res.*, 32(suppl-2):W365–371, 2004.

A. Tsakonas, G. Dounias, J. Jantzen, H. Axer, B. Bjerregaard, and D. G. von Keyserlingk. Evolving rule-based systems in two medical domains using genetic programming. *Artificial Intelligence in Medicine*, 32(3):195–216, 2004.

P. A. Whigham. Grammatically-based genetic programming. In J. P. Rosca, editor, *Proc. of the Workshop on GP: From Theory to Real-World Applications*, pages 33–41, Tahoe City, California, USA, 9 July 1995.

P. A. Whigham. *Grammatical Bias for Evolutionary Learning*. PhD thesis, School of Computer Science, University College, University of New South Wales, Canberra, Australia, 1996.

I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 2005. ISBN 0-12-088407-0.

M. L. Wong. An adaptive knowledge-acquisition system using generic genetic programming. *Expert Systems with Applications*, 15(1):47–58, 1998.

M. L. Wong and K. S. Leung. *Data Mining Using Grammar-Based Genetic Programming and Applications*. Kluwer, 2000.

A. Zafra and S. Ventura. Multi-objective genetic programming for multiple instance learning. In *Proc. of European Conf. on Machine Learning - ECML 2007*, pages 790–797, 2007.

J. Zhang. Selecting typical instances in instance-based learning. In *Proc. of the 9^{th} Int. Workshop on Machine Learning*, pages 470–479, San Francisco, CA, USA, 1992. Morgan Kaufmann.

H. Zhao. A multi-objective genetic programming approach to developing pareto optimal decision trees. *Decis. Support Syst.*, 43(3):809–826, 2007. ISSN 0167-9236.

E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In K.C. Giannakoglou, D.T. Tsahalis, J. Periaux, K.D. Papaliliou, and T. Fogarty, editors, *Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems. Proc. of the EUROGEN2001 Conf.*, pages 95–100. Int. Center for Numerical Methos in Engineering (CIMNE), 2002.

Appendix 1

In this Appendix, we show the rule lists produced by the MOGGP, the SGGP and the C4.5 algorithm for the data set *crx*. The rule lists produced by the Ordered-CN2 and the Unordered-CN2 algorithms are not reported due to space limitations, as they are always much longer than the ones produced by the MOGGP, with 99 and 102 conditions, respectively. Although the C4.5 algorithm is not in the search space of the MOGGP, we use it as a baseline to show that the rules generated by the MOGGP are actually more compact and simpler than the ones produced by the other algorithms, as the numbers reported in Table 6 show. Observe that this more compact rule list is obtained by the MOGGP-RIs while preserving the values of predictive accuracy, as showed in Table 5.

Table 7. Rule list generated by the MOGGP for the data set *crx*

```
     IF  (A9 = f)   AND
         (A14 <= 280) THEN -
ELSE IF  (A11 >= 4) AND
         (A5 = g)  THEN +
ELSE IF  (A15 >= 147) THEN +
ELSE IF  (A14 >= 411) THEN -
ELSE IF  (A14 <= 100) AND
         (A3 <= 10.5) THEN +
ELSE IF  (A3 <= 4) THEN -
DEFAULT CLASS: +
```

The *crx* data set has information about credit card applications, described by 15 attributes. As pointed out in Michie et al. [1994], even with a specialist the results obtained in this data set would be difficult to be interpreted, as the data was coded to preserve confidentiality. Tables 7, 8 and 9 show the rule lists for the data set *crx*. As we can observe, the rules in Table 7 look considerably simpler than the other two, as they are even visually much smaller. While the MOGGP produces a rule list with 9 rule conditions in total (considering all rules), the

SGGP and the C4.5 produce 97 and 32 rule conditions, respectively. Note that the C4.5 generates a rule list for each class, while the rule lists created by both versions of the GGP generate a single rule list.

Table 8. Rule list generated by the SGGP for the data set *crx*

```
     IF (A9 = f) AND
        (A4 = y)  AND
        (A3 >= 0) AND
        (A14 <= 411) THEN -
ELSE IF (A15 >= 2954) AND
        (A9 = t) THEN +
ELSE IF (A9 = f) AND
        (A10 = t) AND
        (A3 >= 0.165) AND
        (A8 >= 0) THEN -
ELSE IF (A11 >= 4) AND
        (A3 >= 1.46) AND
        (A11 <= 19) AND
        (A14 <= 290) THEN +
ELSE IF (A7 = ff) THEN -
ELSE IF (A9 = f) AND
        (A3 >= 1.585) AND
        (A8 >= 0.125) AND
        (A14 <= 640) THEN -
ELSE IF (A6 = cc) AND
        (A2 >= 20.75) AND
        (A11 >= 0) AND
        (A14 <= 583) THEN +
ELSE IF (A15 >= 444) AND
        (A8 >= 1.25) THEN +
ELSE IF (A10 = t) AND
        (A11 <= 1) AND
        (A3 <= 14.5) AND
        (A8 >= 0.04) THEN +
ELSE IF (A14 >= 760) THEN -
ELSE IF (A9 = f) AND
        (A7 = v) AND
        (A3 <= 0.625) AND
        (A8 >= 0.04) THEN -
ELSE IF (A7 = bb) AND
        (A2 >= 26.67) AND
        (A3 >= 0.25) AND
        (A8 <= 1.585) THEN -
ELSE IF (A3 <= 0.42) AND
        (A14 <= 400) THEN +
```

```
ELSE IF (A2 >= 62.5) AND
        (A1 = b) THEN -
ELSE IF (A11 >= 3) AND
        (A14 <= 100) THEN +
ELSE IF (A6 = j) THEN -
ELSE IF (A9 = f) AND
        (A14 <= 120) AND
        (A3 >= 1.08) AND
        (A15 <= 809) THEN -
ELSE IF (A14 <= 60) AND
        (A4 = u) AND
        (A2 >= 19.33) AND
        (A3 >= 1) THEN +
ELSE IF (A15 >= 551) AND
        (A3 <= 11.75) THEN +
ELSE IF (A6 = aa) AND
        (A1 = b) THEN -
ELSE IF (A2 <= 18.5) AND
        (A2 >= 17.5) THEN +
ELSE IF (A3 >= 13) THEN -
ELSE IF (A6 = i) AND
        (A2 <= 33.58) THEN -
ELSE IF (A14 <= 128) AND
        (A14 >= 40) AND
        (A2 >= 20.17) AND
        (A3 >= 0.665) THEN +
ELSE IF (A13 = s) AND
        (A8 <= 0.5) THEN -
ELSE IF (A8 >= 9.46) THEN -
ELSE IF (A8 >= 3.96) AND
        (A13 = g) THEN +
ELSE IF (A2 <= 22.5) AND
        (A8 <= 1.375) THEN -
ELSE IF (A8 <= 0.04) AND
        (A2 <= 32) THEN +
ELSE IF (A8 <= 0.125) AND
        (A2 >= 34.92) THEN +
ELSE IF (A6 = q) AND
        (A1 = a) THEN -
ELSE IF (A2 <= 23.33) AND
        (A2 >= 21.25) THEN +
ELSE IF (A15 >= 225) THEN -
ELSE IF (A10 = t) AND
        (A8 <= 3.165) THEN +
ELSE IF (A3 <= 0.67) THEN -
ELSE IF (A2 >= 35.17) AND
        (A9 = t) THEN -
ELSE IF (A3 >= 4) AND
        (A3 <= 8.665) THEN +
ELSE IF (A14 <= 200) THEN -
ELSE IF (A8 >= 2) AND
        (A9 = t) THEN -
```

```
ELSE IF (A8 >= 1.5) THEN +
ELSE IF (A4 = u) AND
        (A3 >= 0.835) AND
        (A2 >= 24.58) AND
        (A13 = g) THEN -
DEFAULT CLASS: +
```

Table 9. Rule list generated by the C4.5 algorithm for the data set *crx*

```
IF      (A9 = t) AND
        (A15 > 500) THEN +
ELSE IF (A4 = u) AND
        (A7 = v) AND
        (A9 = t) AND
        (A14 <= 112) THEN +
ELSE IF (A9 = t) AND
        (A10 = t) THEN +
ELSE IF (A6 = w) AND
        (A9 = t) AND
        (A12 = f) THEN +
ELSE IF (A7 = h) AND
        (A9 = t) THEN +
ELSE IF (A3 <= 1.375) AND
        (A7 = bb) AND
        (A13 = g) AND
        (A15 <= 500) THEN +
ELSE IF (A13 = p) THEN +.

     IF (A6 = ff) AND
        (A15 <= 501) THEN -
ELSE IF (A9 = f) THEN -
ELSE IF (A4 = y) AND
        (A7 = v) AND
        (A10 = f) AND
        (A12 = f) THEN -
ELSE IF (A3 > 1.375) AND
        (A7 = bb) AND
        (A10 = f) THEN -
ELSE IF (A7 = v) AND
        (A10 = f) AND
        (A14 > 112) AND
        (A15 <= 70) THEN -

DEFAULT CLASS: +
```