Automatically Evolving Rule Induction Algorithms Tailored to the Prediction of Postsynaptic Activity in Proteins

Gisele L. Pappa and Alex A.Freitas*

University of Kent,

Computing Laboratory,

Canterbury, UK, CT2 7NF,

Tel.: +44 (0)1227 82-7220

giselepappa@gmail.com, A.A.Freitas@kent.ac.uk

Abstract

It is well-known that no classification algorithm is the best in all application domains. The conventional approach for coping with this problem consists of trying to select the best classification algorithm for the target application domain. We propose a refreshing departure from this approach, consisting of automatically creating a rule induction algorithm tailored to the target application domain. This work proposes a grammar-based genetic programming (GGP) system to perform "algorithm construction". The GGP is used to build a complete rule induction algorithm tailored to a protein data set, which predicts whether or not a protein presents postsynaptic activity. The results show that the rule

*Corresponding author

induction algorithms automatically constructed by the GGP are competitive with well-known human-designed rule induction algorithms, and that the former was more successful than the latter in discovering accurate rules predicting the minority class – whose prediction is more difficult and tends to be more important to the user than the prediction of the majority class.

Keywords: rule induction algorithms, genetic programming, postsynaptic proteins, classification.

1 Introduction

Several decades of research in classification have shown that no classification algorithm is the best in all application domains [14, 16]. The impact the choice of a suitable algorithm has in the classification model generated from the data is so big that meta-learning [26] emerged as a whole new research area dedicated to study this problem.

In particular, the STATLOG [16] and METAL [22] projects put together the efforts of many researchers to learn how to characterize data sets via "meta-attributes", i.e., attributes describing an entire data set, rather than describing an individual example. Then they used these meta-attributes to create a classification "meta-model" capable of selecting the most suitable classification algorithm for each data set.

Despite the progress obtained with these efforts, the choice of which classification algorithm to apply to a specific data set is still an open problem. This is because there are two major limitations in almost all of the meta-learning techniques. First, they have to identify the best meta-attributes which characterize the data. Choosing the right set of meta-attributes can be an extremely difficult task, given the huge diversity of classification data sets. Second, they

perform "algorithm selection". They try to select the best algorithm out of a small pre-defined set of algorithms.

In order to bypass these two limitations associated with meta-learning approaches, this work focuses on automated "algorithm construction". We propose the use of a grammar-based genetic programming (GGP) [27] system to automatically construct rule induction algorithms tailored to a specific application domain.

The proposed approach avoids the limitations of conventional meta-learning, and presents two main advantages over the latter: (a) it can produce potentially better algorithms than the available ones, and the system can be always updated to produce more new rule induction algorithms by simply modifying the grammar the GGP works with; (b) it presents a much cheaper alternative to the manual design of rule induction algorithms, specially because it can produce a rule induction algorithm tailored to a specific data set.

We chose to construct rule induction algorithms instead of any other type of classification algorithms for two main reasons: first, because of the type of human-comprehensible knowledge they generate. Knowledge comprehensibility is in general important in data mining [9, 23, 28]. Secondly, because research in the rule induction field has being carried out for more than 30 years and certainly produced a large number of algorithms [11]. However, these algorithms 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.

Hence, in some sense, we can say there was a "natural evolution" of rule induction algorithms in the past decades. We want to take advantage of this "natural evolution" and extend it to a new type of evolution, by automating the

design of new rule induction algorithms by means of an evolutionary algorithm – more precisely, a grammar-based genetic programming (GGP) system.

GGP is a special type of genetic programming [13] that incorporates in its search mechanism prior knowledge (expressed in the form of a grammar) about the problem being solved. Intuitively, GGP is an appropriate tool for automatically evolving rule induction algorithms for two main reasons. First, it makes use of what we already know about the design of rule induction algorithms (background knowledge accumulated through several decades of research). Second, it 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.

The GGP system described in this work was first proposed in [21]. However, this work has four important differences with respect to [21], as follows. First, the work described in [21] involved a completely different framework: to automatically evolve *robust* rule induction algorithms, which could be later applied to virtually any classification data set. We emphasize that, in this work, although the basic GGP system is the same as in [21], we propose a very different framework, where the GGP automatically evolves rule induction algorithms tailored to a specific data set. Second, this paper addresses a case study in bioinformatics, namely the prediction of postsynaptic function in proteins, instead of using just well-studies UCI data sets, as in [21]. Third, we study the impact that changing the fitness function of the GGP has on the rule induction algorithms generated, which was not done in [21]. Fourthly, we also investigate in detail the differences between the biases of the automatically evolved and manually-created rule induction algorithms, which again was not done in [21].

The remainder of this paper is organized as follows. Section 2 briefly discusses rule induction algorithms. Section 3 gives a brief overview of GGP.

Section 4 introduces the proposed GGP, while Section 5 describes some related work. Section 6 reports the results of automatically constructing rule induction algorithms for the postsynaptic data set. Finally, Section 7 presents the conclusions and describes future research directions.

2 Rule Induction Algorithms

This work focuses on classification models consisting of a set of IF-THEN rules, produced by rule induction algorithms following the sequential covering strategy. The sequential covering strategy (also known as separate and conquer) is certainly the most explored and most used strategy to induce rules from data. It was first employed by the algorithms of the AQ family [15] in the late sixties, and over the last few decades was applied again and again as the basic strategy in rule induction systems.

The separate and conquer strategy works as follows. 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. 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.

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. The methods based on this approach differ from each other in four main points [11], although the last one can be absent:

1. The representation of the candidate rules, which can be done with 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), in a post processing step (post-pruning) to help avoiding over-fitting and handling noisy data, or even on an integrated fashion.

The overall structure of the grammar which the GGP is based on was conceived by taking into account these four main elements and the way they can be combined.

3 Overview of Genetic Programming

Evolutionary computation is a research area dedicated to the study of computational intelligence algorithms inspired by Darwin's concepts of evolution and survival of the fittest. Its application is being very successful because of its global search and associated implicit parallelism and noise tolerance [2, 3]. Genetic Programming (GP) [13, 3] is an area of evolutionary computation which aims to automatically evolve computer programs. Its success is backed up by a list of 36 human-competitive solutions, where two created patentable new inventions [1].

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 (near-) optimum solution is found, or a pre-established number of generations is reached.

In this work, we are particularly interested in one type of GP: grammar-based genetic programming (GGP) [27]. As the name suggests, the main difference between a traditional GP and a grammar-based one is the definition and use of a grammar. The motivation to combine grammars and GP is two-fold [19]. 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[1] 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 $y \in \{T \cup N\}$.

Grammars are usually described using the Backus Naur Form (BNF) [17]. When using the BNF notation, production rules have the form <expr> ::= <expr><op><expr>, and symbols wrapped in "<>" represent the non-terminals of the grammar. Three special symbols might be used for writing the production rules in BNF: "|","[ ]" and "( )". "|" represents a choice, like in <var> ::=$x|y$, where <*var*> generates the symbol x or y. "[ ]" 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 $yz \Longrightarrow 0z$.

[1] In conventional GP, an individual consists of functions and terminals. The closure property states that every function in the function set has to be able to handle all the values it receives as input. For more details see [13].

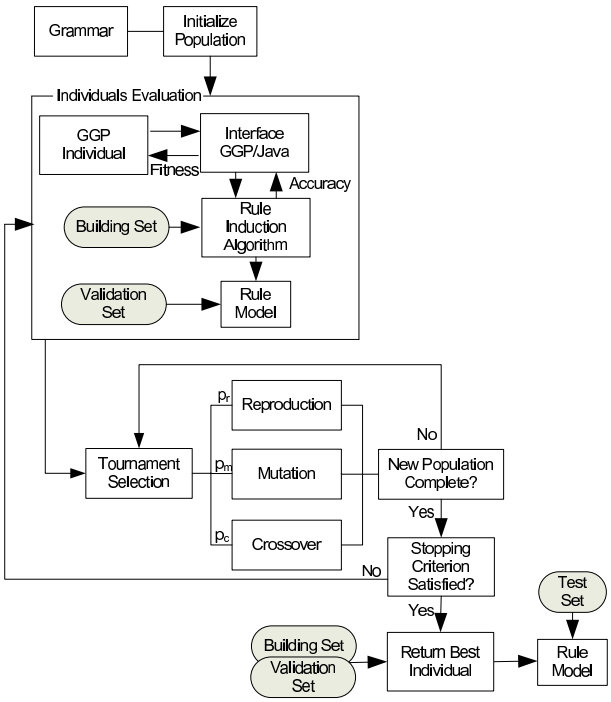

Figure 1: Scheme of the proposed GGP method

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 [27].

4 Constructing rule induction algorithms tailored to a specific application domain

This works proposes the use of a GGP system to automatically construct rule induction algorithms tailored to a specific application domain: the postsynaptic protein data set. In contrast with projects that use GP to discover a *set of rules* for a specific data set, like [30], this project aims to automatically invent a *complete rule induction algorithm*. Hence, each individual in our population

represents a new rule induction algorithm, potentially more sophisticated than well-known algorithms like CN2 [4].

Figure 1 shows an scheme of the proposed GGP method. As illustrated in Figure 1, the first GGP population is randomly generated by following a set of derivations steps from the grammar. In the proposed system, the grammar contains background knowledge about how humans designed sequential-covering rule induction algorithms so far, and also other components that we thought might work in rule induction, but were not tried before. The grammar is shown in Table 1, and is defined using the BNF notation described in Section 3. For a complete description of the grammar and its components the reader is referred to [21].

Each individual in the GGP population is represented by a grammar derivation tree, which can be "translated" to a new rule induction algorithm. Figure 2 shows an example of a GGP individual, whose pseudo-code is explained later in Alg.1. The pseudo-code that a GGP individual represents can be extracted from the GGP derivation tree by reading its leaf nodes from left to right.

In addition to the grammar and the individual representation, another important component of the system is the evaluation of the GGP individuals. As observed in Figure 1, the system implements a GGP/Java interface, which "translates" the GGP individual to Java code. After that, the GGP individuals are evaluated by running the rule induction algorithm they represent. The classification model is built from a building set, and validated into a validation set. Both the building and validation sets represent subsets of the targeted training set. The test set is not accessed during the GGP run, since it is reserved for measuring the generalization ability of the evolved rule induction algorithm.

This GGP/Java interface is also responsible for reading the classification accuracy obtained by the rule induction algorithms in the GGP validation set,

Table 1: The grammar used by the GGP, adapted from [21]

```
<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| randomExample| typicalExample|
                     <MakeFirstRule>.
<MakeFirstRule> ::= NumCond1| NumCond2| NumCond3| NumCond4.
<innerWhile> ::= while (candNotEmpty| negNotCovered) <FindRule> endWhile.
<FindRule> ::= (<RefineRule>|<innerIf>)<EvaluateRule>
               [<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.
<EvaluateRule>::= 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>.
```

and generating a fitness value from it. The fitness function of the individuals is calculated by the formula described in Eq. 1, where *Acc* represents the accuracy obtained by the rules discovered by the rule induction algorithm. *DefAcc* represents the default accuracy (the accuracy obtained when using the class of the majority of the building set examples to classify new examples in the validation set).

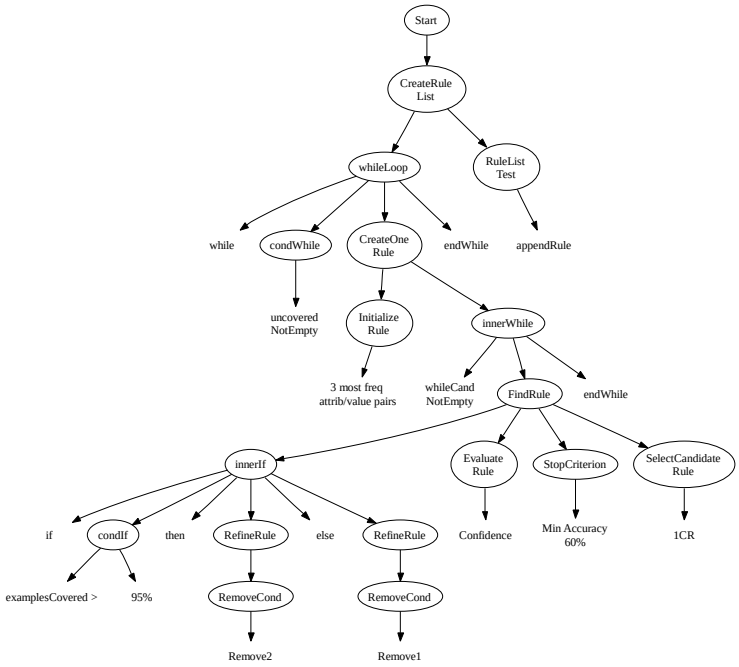

Figure 2: Example of a GGP individual

$$fit = \begin{cases} \frac{Acc - DefAcc}{1 - DefAcc}, & \text{if Acc} > \text{DefAcc} \\ \frac{Acc - DefAcc}{DefAcc}, & \text{otherwise} \end{cases} \quad (1)$$

According to the definition of fit, 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 $DefAcc$). Hence, fit returns a value between -1 (when $Acc = 0$) and 1 (when $Acc = 1$).

After the population evaluation, a tournament selection scheme is used to select the individuals which will produce the new population. The tournament

selection method works by randomly obtaining k individuals from the population. These k individuals will compete against each other in a tournament. The individual with the best fitness value defeats the other individuals. The GGP proposed in this work uses k equals to 2. Following selection, the winners of the tournaments undergo either reproduction, mutation, or crossover operations, depending on user-defined rates (p_r, p_m and p_c in Figure 1). The individuals resulting from these operations have also to be valid according to the grammar.

The evolutionary process is conducted until a maximum number of generations is reached. At the end of the evolutionary process, the best individual (highest fitness) produced along the run of the GGP system is returned as the solution for the problem. The chosen rule induction algorithm is then evaluated in a new set of data, which appears in Figure 1 as the test set. In order to build the model to be applied to the test set, the entire GGP training data (i.e. building set plus validation set) is used.

5 Related Work

Evolutionary algorithms were used before to construct other complete classification algorithms tailored to a specific application domain, such as neural networks [31, 24], but not, to the best of our knowledge, to construct complete rule induction algorithms. However, both Wong [29] and Suyama *et* al.[25] tried some related approaches before.

Wong [29] used a GGP to automatically evolve the evaluation 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 indi-

viduals (evaluation functions) generated by the GGP are then incorporated into a generic version of a top-down first-order learning algorithm based on FOIL, and the learning algorithm as a whole is evaluated.

Our work goes considerably beyond that work, as follows. In [29] the GGP was used to evolve only the evaluation function of a rule induction algorithm. By contrast, in our work the 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 Wong's GGP is just the space of evaluation functions for FOIL. As a result, our grammar is much more elaborated than the grammar used by Wong.

Suyama *et* al. [25] also used a hybrid of a GP and a local search method to evolve a classification algorithm. CAMLET uses an ontology rather than a grammar to guide its search. The ontology used in [25] has 15 coarse-grained building blocks, where a leaf node of the ontology is a full classification algorithm, like a decision tree, a genetic algorithm or a neural network. 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 [25].

6 Automatically Constructing GGP-RIs for the Postsynaptic Data Set: A case study

This section presents computational results obtained when applying the proposed GGP to construct rule induction algorithms tailored to a postsynaptic data set [20]. The type of problem described in this data set involves the prediction of the function of proteins based on a set of predictor attributes rep-

resenting biological "motifs" (patterns or "signatures" typically found in some proteins). A protein is essentially a sequence of amino acids. This sequence is called the "primary sequence".

As its name indicates, the postsynaptic data set is used to predict whether or not a protein presents postsynaptic activity, based on its primary sequence. Identifying proteins with postsynaptic activities is of great intrinsic interest because they are connected with functioning of the nervous system. Each of the 4303 examples in this data set represents a protein, which is characterized by a set of binary attributes. Each attribute indicates the presence or absence of a motif in the protein sequence.

The postsynaptic data set originally had 444 attributes [20]. However, it was pre-processed by using an attribute selection method. This pre-processing step was executed for two reasons. First, due to the large number of predictive attributes in the original data set, intuitively there are many attributes that are (at least to some extent) irrelevant or redundant. Second, attribute selection usually reduces significantly the size of the data sets. This makes the application of the GGP algorithm much more efficient.

There is some evidence that feature selection in the postsynaptic data set improves its predictive accuracy [6]. As it is not the purpose of this work to perform an optimal attribute selection pre-process, we borrowed the 10 attributes selected by [6] to perform experiments with the GGP. However, in order to show that the attribute selection process does not reduce the predictive accuracy of either the GGP-derived rule induction algorithms (GGP-RIs) or the baseline rule induction algorithms, we report results comparing the GGP-RIs tailored to the postsynaptic data set with selected attributes against the baseline methods applied to both the complete data set and the data set with selected attributes.

For all the experiments reported in this section, the parameters of the GGP

Table 2: Comparing the predictive accuracies (%) obtained by the GGP-RIs in the postsynaptic data set with selected attributes against the predictive accuracies (%) obtained by the baseline methods when using all attributes or just the 10 selected attributes

Algorithm	Postsynaptic	
	All Attr.	Selec. Attr.
GGP-RI	NA	98.32±0.24
CN2Ord	98.7±0.22	98.4 ± 0.2
CN2Unord	98.42±0.12	98.4 ± 0.2
Ripper	98.3±0.22	98.21 ± 0.22
C45Rules	97.82±0.32	98.4 ± 0.2

were set as follows: 100 individuals, evolved in 30 generations, using crossover rate of 0.7, mutation rate of 0.25 and reproduction rate of 0.05. For each data set, the GGP was run 25 times: 5 runs with different random seeds $\times$ 5 fold cross validation. Hence, the entire data set was divided in 5 partitions and, at each GGP run, four data partitions were used for training the GGP (being three of them used in the building set and one in the validation set), and one (unseen during the GGP evolution) for testing the rule induction algorithm constructed by the GGP.

The predictive accuracies (on the test set) obtained by the GGP were compared to the predictive accuracy of four baseline human-designed rule induction algorithms, namely the ordered and unordered versions of CN2 [4], Ripper [5] and C4.5Rules [23]. Note that both versions of CN2 are included in the search space of the GGP, while the only components present in Ripper and absent in the GGP search space are the minimum description length heuristic [28] and part of Ripper's optimization phase. Although C4.5Rules is not a sequential covering rule induction algorithm, since its rules are extracted from a decision tree, it was included in the comparisons because it is frequently used as a baseline comparison method in the machine learning literature.

Table 2 shows the predictive accuracies obtained by the GGP-RIs in the postsynaptic data set with the selected attributes, followed by the accuracies

obtained by the baseline methods when using all the attributes or just the 10 selected attributes. Two kinds of comparisons can be made in this table. First, we can compare the accuracy of the GGP-RIs evolved using only the selected attributes against the accuracy of the baseline methods using the complete set of attributes (last four rows in the second column of the table). Secondly, we can compare the accuracy of the GGP-RIs constructed using only the selected attributes against the accuracy of the baseline methods using the same selected attributes (last four rows in the third column of the table). All the reported results are statistically the same according to a 2-tailed Student's t-test with 0.01 significance level.

These results based on accuracy are reported for the sake of completeness, but there is another characteristic of the postsynaptic data set which is worth noticing. The class distribution in postsynaptic is very unbalanced, with only 6.04% of the examples having the positive class. This means that, as a baseline solution for this classification problem, the "majority classifier" - which predicts the majority (negative) class for all examples - would trivially obtain an accuracy rate of 93.96%. This value could be obtained without providing any insight about the relationship between the predictor attributes and the classes.

For data sets in which the class distribution is very unbalanced, an analysis based on the true positive rate (sensitivity) and true negative rate (specificity) is more effective [12]. Another alternative would be to use a measure based on ROC curves, such as the area under the curve. We preferred a measure based on sensitivity and specificity to be consistent with the results for this data set reported in [20]. The sensitivity $\times$ specificity measure calculates the product of the true positive and true negative rates. We used this measure to reevaluate the results obtained by the GGP-RIs for the data set postsynaptic.

Table 3 shows the sensitivity, specificity and the product sensitivity $\times$ speci-

Table 3: Sensitivity x Specificity for the data set postsynaptic using the 10 selected attributes

Algorithm	Sensitivity	Specificity	Sensit×specif
GGP-RIs	0.74±0.004	0.99±0.0001	0.736±0.004
Ordered-Cn2	0.76±0.02	0.99±0.001	0.758±0.02
Unordered-Cn2	0.76±0.02	0.99±0.001	0.758±0.02
C45Rules	0.75±0.03	0.99±0.001	0.748±0.03
Ripper	0.7±0.04	0.99±0.001	0.702±0.04

ficity obtained by the GGP-RIs and the other baseline methods for the postsynaptic data set. All the results in this table were produced by using the data set with the selected attributes. All the results showed in Table 3 present no significant difference according to a 2-tailed Student's t-test with 0.01 significance level.

The analysis of the GGP-RIs results for the data set postsynaptic using a sensitivity × specificity approach confirmed that the GGP-RIs produced are competitive with the other baseline methods. Both the GGP-RIs and the baseline methods not only obtain a high accuracy, but also have a good ability to separate objects from the positive and negative classes – rather than simply predicting the majority class for all the test examples – as shown by the relatively high values of sensitivity. After this new analysis, a new question came up. For data sets like postsynaptic, where the class distribution is very unbalanced, would it be worth to actually evolve the GGP with a different fitness function? That is, would it be worth to consider the sensitivity × specificity measure, for instance, as the GGP evaluation function during the evolution? This would make sense, once we know that, in the context of very unbalanced class distributions, predictive accuracy is not a very effective measure to evaluate the predictive power of classification models.

During preliminary results executed when designing the GGP, a fitness function based on the sensitivity × specificity measure was tried, and shown to be

not as effective as the fitness function defined in Eq. 1 – based on the normalized value of accuracy. But would a GGP constructing a rule induction algorithm tailored to the postsynaptic data set, and a fitness function based on sensitivity $\times$ specificity, generate better results than the GGP with the normalized accuracy fitness shown in Eq.(1)?

In order to find answers for this question, we ran a set of experiments almost identical to the ones described so far in this section. However, we replaced the current fitness of the GGP by the sensitivity $\times$ specificity measure. Surprisingly, the average predictive accuracy obtained over the 25 runs of the GGP was 92.85$\pm$0.03. This predictive accuracy value is slightly smaller than the default accuracy provided by the classification model using the class of the majority of the examples (93.96%), and it is significantly worse than the values of accuracy obtained by any of the baseline methods (and the GGP-RIs) presented in Table 2.

However, as explained before, in the case of the postsynaptic data set, an analysis based on the sensitivity $\times$ specificity measure is more appropriated than one based on predictive accuracy. The sensitivity $\times$ specificity value (measured on the test set) obtained for these experiments using sensitivity $\times$ specificity as the fitness function of the GGP was 0.789$\pm$0.04. This value is statistically the same as all the values in the last column of Table 3. On the other hand, a more detailed analysis of the sensitivity and specificity values separately showed a specificity (the proportion of negative (majority class) examples that are correctly predicted as negative) of 0.93$\pm$0.026 and a sensitivity (the proportion of positive (minority class) examples that are correctly predicted as positive) of 0.84$\pm$0.008. If we compare these values to the ones presented at Table 3, we notice that the specificity dropped from 0.99 to 0.93 and the sensitivity increased from 0.75 to 0.84. According to a 2 tailed Student's t-test with 0.05 significance

Algorithm 1: Example of a decision list algorithm created by the GGP – with a normalized accuracy fitness function – tailored to the data set *postsynaptic*

```
RuleList = ∅
repeat
    bestRule = rule created using the 3 most frequent attribute/values in
    the training data
    candidateRules = ∅
    candidateRules = candidateRules ∪ bestRule
    while candidateRules ≠ ∅ do
        for each candidateRule CR do
            newCandidateRules = ∅
            if number of covered examples in RuleList > 95% then
                Remove 2 conditions-at-a-time from CR
            else
                Remove 1 condition-at-a-time from CR
            Evaluate CR using confidence
            if accuracy(CR) > 60% then
                newCandidateRules = newCandidateRules ∪ CR
        candidateRules = best rule selected from newCandidateRules
    RuleList = RuleList ∪ bestRule
    Remove the examples covered by bestRule from the training set
until all examples in the training set are covered
```

level, the sensitivity of the GGP-RIs is significantly better than the sensitivity of all the baseline methods presented in Table 3, while the specificity of the GGP-RIs is significantly worse than the specificity of all the baseline methods.

From this we conclude that the GGP-RIs found with the sensitivity × specificity fitness produced algorithms which are better when predicting the class of the minority of the examples, which is more difficult to predict and tends to be a prediction more useful to the user, by comparison with a prediction of the majority class [7, 18]. At the same time, these GGP-RIs are not able to preserve a high specificity – true negative rate for the majority class.

This last experiment also confirmed that the fitness of the current GGP is robust enough to produce robust algorithms even for data sets with very unbalanced classes. But what was so different in the GGP-RIs produced by

these two versions of the GGP using different fitness functions?

An analysis of the rule induction algorithms produced by the GGP when using the normalized accuracy or the sensitivity $\times$ specificity measure as the fitness function revealed algorithms following two completely different approaches.

On one hand, the majority of the GGP-RIs produced with the normalized accuracy fitness follows one of the following two main approaches: (1) they create an initial empty rule and add conditions to it, or (2) they build an initial rule using 3 or 4 of the most frequent attribute/value pairs found in the training data, and remove conditions from it. Algorithms following any of these two approaches produce very compact and general rules, usually with a maximum of 3 or 4 conditions each. Alg. 1 shows an example of one of these algorithms produced by the GGP (which pseudo-code was extracted from the individual represented in Fig. 2). It creates the first rule with the 3 most frequent attribute/values pairs in the data, and starts by removing one condition-at-a-time from it. As soon as the number of examples covered by the rule list is greater than 95%, it changes its refinement strategy by removing 2 conditions-at-a-time from the candidate rules. Rules are evaluated using the rule confidence measure, which is required to be at least equal to 60% to enable the candidate rule to undergo further refinements (recall that in this context rule confidence and rule accuracy are synonyms). Only the best rule is selected to be refined. Rules are produced until all the examples in the training set are covered. The output of the algorithm is an ordered list of rules, called a decision list.

In contrast with the types of GGP-RIs produced when using the normalized accuracy measure as the fitness function, most of the GGP-RIs produced when using the sensitivity $\times$ specificity measure as fitness followed a bottom-up approach. Rules were initialized using a random example or a typical example of a class, and most of the actual rule models had very specific rules (rules with

many conditions). The selection of a typical example [32] is based on some principles used in instance-based learning algorithms, an innovative feature of the grammar that is not found in any manually-designed rule induction algorithm, to the best of our knowledge. An example is said to be typical if it is very similar to the other examples belonging to the same class it belongs to, and not similar to the other examples belonging to other classes. Looking at the predictive accuracies obtained by these algorithms we notice that most of them seemed to be over-fitting the training data. Regardless of that, as reflected by the sensitivity (true positive rate) obtained by these algorithms, they were able to generate significantly better rules to predict the minority class. Alg. 2 shows an example of a GGP-RI produced by the GGP using the sensitivity $\times$ specificity fitness function.

Alg. 2 creates a set of rules for each class in turn. It chooses a typical example of a class from the training set and removes one condition-at-a-time from it. When 95% of the examples belonging to the current class are covered, the algorithm starts to remove 2 conditions-at-a-time from the candidate rules. The 4 best rules in the current iteration are selected to undergo further refinements, which are carried out until the best rule found so far covers no negative examples. Candidate rules are evaluated using the information content, and are also required to have an accuracy of at least 60% to be considered as candidate rules.

In Alg. 2, rules are pre-pruned before being inserted into the rule set by removing a final list of conditions from them. Conditions are removed from the best produced rule until the Laplace correction value of the new pruned rule is worse than the Laplace correction value of the best rule in the prune set. Rules are produced until at least 97% of the examples in the current class are covered. The output of the algorithm is an unordered set of rules, unlike the ordered list of rules produced by Alg. 1.

Algorithm 2: Example of a rule set algorithm created by the GGP – with a sensitivity × specificity fitness function – tailored to the data set *postsynaptic*

```
RuleSet = ∅
for each class C in the training set do
    repeat
        Divide the training data in Grow and Prune
        bestRule = rule created from a typical example
        candidateRules = ∅
        candidateRules = candidateRules ∪ bestRule
        while negative examples covered by bestRule ≠ ∅ do
            for each candidateRule CR do
                newCandidateRules = ∅
                if number of covered examples in class C > 95% then
                    Remove 2 conditions-at-a-time from CR
                else
                    Remove 1 condition-at-a-time from CR
                Evaluate CR using information content in Grow
                if accuracy(CR) > 80% then
                    newCandidateRules = newCandidateRules ∪ CR
            candidateRules = 4 best rules in newCandidateRules
        repeat
            bestRule' = bestRule
            bestRule' = Rule obtained by removing the last condition
            from bestRule
        until laplace(bestRule') ≥ laplace(bestRule) in Prune
        RuleSet = RuleSet ∪ bestRule
        Remove examples from class C covered by bestRule from training
        set
    until at least 97% of the examples of class C in the training set are
    covered
Class clashes when classifying new examples are solved using the
ls-content criterion
```

Note that both pseudo-codes described in Algs. 1 and 2 represent innovative algorithms, which use different approaches from the well-known human-designed rule induction algorithms to start searching for rules (i.e., the 3 most-frequent attribute/value pairs in the training set or the typical example of a class) and later refine them (rules can be refined according to the number of examples covered by the rules produced so far). These two examples show that the proposed GGP system can potentially construct customized rule induction algorithms

very different from the ones available in the literature.

7 Conclusions and Future Work

This paper presented a grammar-based genetic programming (GGP) system which automatically constructs rule induction algorithms tailored to a specific application domain: the postsynaptic data set. The GGP system works with a grammar, which contains previous knowledge about how humans design rule induction algorithm, and some other interesting components that, to the best of our knowledge, were not used by rule induction algorithms so far.

In summary, the results showed that the GGP can produce GGP-RIs tailored to a specific real world data set which are competitive with well-known human designed rule induction algorithms. It also showed that the system coped well with the problem of unbalanced classes, in particular when using the sensitivity $\times$ specificity fitness function, which led to a better prediction of the minority class (whose prediction is more difficult and tends to be more important to the user than the prediction of the majority class). This new method is an interesting and promising alternative to the traditional meta-learning approach of trying to select the best classification algorithm to the target application domain.

An analysis of the evolved GGP-RIs showed that, besides being competitive with human-designed algorithms, many of them present some innovative way of refining rules and/or integrating pre and post-pruning techniques.

One research direction to be considered is how to apply the described GGP system to construct rule induction algorithms tailored to a set of data sets with similar characteristics where all data sets come from similar application domains, instead of tailored to a single application domain. In this approach, data sets would be grouped according to some common properties, and only

data sets belonging to this group would be used for training the GGP. For instance, in the bioinformatics field, there are a lot of data sets which contain a huge number of binary attributes, very sparse data and very unbalanced classes (the postsynaptic data set used in our case study is one of such data sets). This data sets would be used together to train and test the GGP, aiming to obtain rule induction algorithms tailored to that type of data sets.

Another research direction would be to use the GGP to create ensembles of rule induction classifiers, i.e., a set of classifiers whose individual predictions are combined to classify new examples. At each generation, 100 individual are evaluated by the GGP. At the last generation, one single rule induction algorithm is chosen out of 100 to be returned by the used. But isn't it a waste of time to evolve 100 different algorithms and ignore 99 of them?

Research in the area of ensembles of classifiers has demonstrated that combining classifiers can really obtain better prediction accuracies than using a single one [8]. We could combine the GGP-RIs produced in the last GGP's generation into a voting or a stacking framework, and check if the results obtained would be superior to the ones obtained by the single best GGP-RI.

Yet another research direction would be to modify the fitness of the current GGP system. In particular, we are interested in evaluating the GGP-RIs using the area under the ROC curve (AUC) [10]. This measure has being widely disseminated in the past years, and a comparison of the results of the GGP with the current fitness function and with the AUC would be interesting.

8 Acknowledgments

The first author was financially supported by CAPES, a Brazilian government's research support agency, process 165002-5.

References

[1] *Genetic Programming*. http://www.genetic-programming.org/, May, 2007.

[2] T. Baeck, D. B. Fogel, and Z. Michalewicz. *Evolutionary Computation 1 Basic Algorithms and Operators*. Institute of Physics Publishing, 2000.

[3] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, Jan. 1998.

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

[5] 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, jul 1995. Morgan Kaufmann.

[6] E. Correa, A. Freitas, and C. Johnson. A new discrete particle swarm algorithm applied to attribute selection in a bioinformatics data set. In M. K. et al. (Eds.), editor, *Proc. Genetic and Evolutionary Computation Conference (GECCO-2006)*, pages 35–42. ACM Press, July 2006.

[7] B. de la Iglesia, J. C. W. Debuse, and V. J. Rayward-Smith. Discovering knowledge in commercial databases using modern heuristic techniques. In

Proc. of the 2^{nd} *ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 44–49, 1996.

[8] S. Dŵeroski and B. Zenko. Is combining classifiers better than selecting the best one. In *Proc. of the* 19^{th} *Int. Conf. on Machine Learning*, pages 123–130, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[9] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining.* AAAI/MIT Press, 1996.

[10] P. Flach. The geometry of roc space: understanding machine learning metrics through roc isometrics. In *Proc.* 20^{th} *Int. Conf. on Machine Learning*, pages 194–201. AAAI Press, January 2003.

[11] J. Fürnkranz. Separate-and-conquer rule learning. *Artificial Intelligence Review*, 13(1):3–54, 1999.

[12] D. J. Hand. *Construction and Assessment of Classification Rules.* Wiley, 1997.

[13] J. R. Koza. *Genetic Programming: On the Programming of Computers by the means of natural selection.* The MIT Press, Massachusetts, 1992.

[14] 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.

[15] R. S. Michalski. On the quasi-minimal solution of the general covering problem. In *Proc. of* 5^{th} *Int. Symposium on Information Processing*, pages 125–128, Bled, Yugoslavia, 1969.

[16] 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.

[17] P. Naur. Revised report on the algorithmic language algol-60. *Communications ACM*, 6(1):1–17, 1963.

[18] C. Nguyen and T. B. Ho. An imbalanced data rule learner. In *Proc. of PKDD-05*, pages 617–624, 2005.

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

[20] G. L. Pappa, A. J. Baines, and A. A. Freitas. Predicting post-synaptic activity in proteins with data mining. *Bioinformatics*, 21(Suppl. 2):ii19–ii25, September 2005.

[21] G. L. Pappa and A. A. Freitas. Automatically evolving rule induction algorithms. In J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, editors, *Proc. of the 17th European Conf. on Machine Learning*, volume 4212/2006, pages 341–352, 2006.

[22] Y. H. Peng, P. A. Flach, C. Soares, and P. Brazdil. Improved dataset characterization for meta-learning. In *The 5th Int. Conf. on Discovery Science*, pages 141–152. Springer-Verlag, January 2002.

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

[24] D. Rivero, J. Dorado, J. R. Rabuñal, A. Pazos, and J. Pereira. Artificial neural network development by means of genetic programming with graph codification. *Transactions on Engineering, Computing and Technology*, 16:209–214, 2006.

[25] A. Suyama, N. Negishi, and T. Yamaguchi. CAMLET: A platform for automatic composition of inductive learning systems using ontologies. In *Pacific Rim Int. Conf. on Artificial Intelligence*, pages 205–215, 1998.

[26] R. Vilalta and Y. Drissi. A perspective view and survey of meta-learning. *Artificial Intelligence Review*, 18(2):77–95, 2002.

[27] P. A. Whigham. Grammatically-based genetic programming. In J. P. Rosca, editor, *Proc. of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 33–41, 1995.

[28] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann, second edition, 2005.

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

[30] M. L. Wong and K. S. Leung. *Data Mining Using Grammar-Based Genetic Programming and Applications.* Kluwer, Norwell, MA, USA, 2000.

[31] X. Yao. Evolving artificial neural networks. *Proc. of the IEEE*, 87(9):1423–1447, 1999.

[32] 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 Publishers Inc.