A Constrained-Syntax Genetic Programming System for Discovering Classification Rules: Application to Medical Data Sets

Celia C. Bojarczuk [1] ; Heitor S. Lopes [2] ; Alex A. Freitas [3]; Edson L. Michalkiewicz [4]

[1, 2] Laboratório de Bioinformática / CPGEI
Centro Federal de Educação Tecnológica do Paraná – CEFET-PR
Av. 7 de Setembro, 3165, 80230-901, Curitiba (PR), Brazil

[3] Computing Laboratory
University of Kent
Canterbury, CT2 7NF, UK

[4] Setor de Cirurgia Pediátrica - Hospital Erasto Gaertner
R. Dr. Ovande do Amaral, 201, 81520-060, Curitiba (PR), Brazil
{celiacri, hslopes}@cefetpr.br, A.A.Freitas@kent.ac.uk, emichalk@netpar.com.br

Abstract

This paper proposes a new constrained-syntax genetic programming (GP) algorithm for discovering classification rules in medical data sets. The proposed GP contains several syntactic constraints to be enforced by the system using a disjunctive normal form representation, so that individuals represent valid rule sets that are easy to interpret. The GP is compared with C4.5, a very well-known decision-tree-building algorithm, and with another GP that uses boolean inputs (BGP), in five medical data sets: Chest pain, Ljubljana breast cancer, Dermatology, Wisconsin breast cancer, and Pediatric Adrenocortical Tumor. For this last data set a new preprocessing step was devised for survival prediction. Computational experiments show that, overall, the GP algorithm obtained good results with respect to predictive accuracy and rule comprehensibility, by comparison with C4.5 and BGP.

Key words: genetic programming, data mining, classification.

1 Introduction

Classification is an important problem extensively studied in several research areas, such as statistical pattern recognition, machine learning and data mining [6, 14, 20, 29]. The basic idea is to predict the class of an instance (a record of a given data set), based on the values of predictor attributes of that instance.

Medical diagnosis can be considered a classification problem: a record is a given patient's case, predictor attributes are all patient's data (including symptoms, signals, clinical history and results of laboratory tests), and the class is the diagnosis (disease or clinical condition that the physician has to discover, based on the patient's data). Finally, in this analogy, the medical knowledge and past experience of the physician takes the role of the classifier. Notwithstanding, many times, physicians cannot easily state the final diagnosis of a patient using simple classification rules. This is usually due to the underlying complexity of the information necessary to achieve such a diagnosis. This paper proposes a new genetic programming (GP) system for discovering simple classification rules.

GP is a search algorithm based on the principle of natural selection [1, 17]. In essence, it evolves a population of individuals, where each individual represents a candidate solution for a given problem. In each "generation" (iteration) individuals are selected for reproduction. This selection is probabilistically biased towards the current best individuals. New offsprings are produced by operators that modify those selected individuals. This process of selection and creation of new individuals is repeated for a number of generations, so that the quality of the individuals (candidate solutions) is expected to improve with time. At the end of this evolution process, the best individual ever produced by GP is presented to the user as the final solution.

The GP algorithm proposed in this paper discovers classification rules in the following format: IF (a-certain-combination-of-attribute-values-is-satisfied) THEN (predict-a-certain-disease). Each individual represents a set of these IF-THEN rules. This rule format has the advantage of being intuitively comprehensible for the user. Hence, he/she can combine the knowledge contained in the discovered rules with his/her own knowledge, in order to make intelligent decisions about the target classification problem – for instance, medical diagnosis.

The use of GP for discovering comprehensible IF-THEN classification rules is much less explored in the literature when compared with other traditional rule induction and decision-tree-induction methods [21, 31]. We believe such a use of GP is a promising research area, since GP has the advantage of performing a global search in the space of candidate rules. In the context of classification rule discovery, in general this makes it cope better with attribute interaction than conventional, greedy rule induction and decision-tree-building algorithms [5, 8, 10, 11, 23].

The GP algorithm proposed in this paper is a constrained-syntax one in the sense that it contains several syntactic constraints to be enforced by the system, so that individuals represent rule sets that are valid and easy to interpret, due to the use of a disjunctive normal form representation.

The remainder of this paper is organized as follows. Section 2 presents an overview of the area of GP, in order to make this paper self-contained. Section 3 describes the proposed constrained-syntax GP for discovering classification rules. Section 4 reports the results of computational experiments comparing the GP with C4.5 and a "booleanized" GP [2]. Finally, section 5 presents the conclusions and suggested research directions.

2 An Overview of Genetic Programming (GP)

In GP, the basic idea is the evolution of a population of "programs"; i.e., candidate solutions to the specific problem at hand. A program (an individual of the population) is usually represented as a tree, where the internal nodes are functions (operators) and the leaf nodes are terminal symbols. More complex representations, such as graphs or multiple trees, were proposed in the past [13, 27], but they are far less popular than the tree representation, which was popularised by Koza's first book [17]. Both the function set and the terminal set must contain symbols appropriate for the target problem. The function set can contain arithmetic operators, logic operators, mathematical functions, etc., whereas the terminal set can contain the variables (attributes) of the problem. In the medical domain, these attributes could be, for instance, signals, symptoms and other information about a given patient to be diagnosed.

Each individual of the population is evaluated with respect to its ability to solve the target problem. This evaluation is performed by a fitness function, which is problem-dependent. In general, the fitness is computed by means of a well-defined mathematical function (or even a complex procedure) that assigns a scalar value to the outcome. In most cases, the fitness of an individual is evaluated over a set of different representative situations (called fitness cases), sampled from the problem space.

Individuals undergo the action of genetic operators such as reproduction and crossover. These operators are very briefly described below; a more detailed analysis about these and other genetic operators can be found in [1, 17].

The reproduction operator selects one individual of the current population in proportion to its fitness value, so that the fitter an individual is, the higher the probability that it will take part in the next generation of individuals. After selection, the individual is copied into the new generation. Reproduction reflects the principles of natural selection and survival of the fittest.

The crossover operator swaps randomly selected subtrees of two individuals. Therefore, crossover mimics the process of sexual reproduction found in the nature.

The application of crossover is independent of the tree topology, since it has to meet the closure property of GP. This property states that any function has to accept as operand any possible combination of terminals and functions. Figure 1 illustrates the application of the crossover operator to a couple of parents. The crossover point was chosen at random for both parents. Translating the trees into logical expressions, one can describe the exact results of this application of crossover, as follows:

- Parent 1: *(not A_1) and ($A_2 = 3$)*
- Parent 2: *((not A_1) or A_3) or (($A_4 \leq 2$) and A_5)*
- Offspring *1: (($A_4 \leq 2$) and A_5) and ($A_2 = 3$)*
- Offspring2: *((not A_1) or A_3) or (not A_1)*

Once genetic operators have been applied to the population according to given probabilities, a new generation of individuals is created. These newly created individuals are evaluated by the fitness function. The whole process is repeated iteratively for a fixed number of generations or until another termination criterion is met. The result of genetic programming (the best solution found) is usually the fittest individual produced along all the generations.

A GP run is controlled by several parameters, some numerical and some qualitative ones [17]. However, control is implementation-dependent; i.e., not all GP systems have to follow the same directives, especially those parameters concerning the genetic operators. If special genetic operators are defined, specific parameters have to be set to control their use throughout the run.

Summarizing, the use of GP to solve real-world problems encompasses the previous definition of the following:

- the structural components of the solution (i.e., both the function set and terminal set);
- the method for generating the initial population of solutions;
- the genetic operators used to modify stochastically a given solution;
- the measure of goodness of a given solution (fitness function);
- the criteria for terminating the evolutionary process and for designating the result;
- the major and minor parameters that control the GP.

2.1 Constrained Genetic Programming

The standard GP was not developed to deal with different data types at the same time. In fact, this limitation is due to the closure property [17], which has to be satisfied, that is, any non-terminal must be able to manipulate any data type returned from a terminal or non-terminal. In many applications, this is a serious limitation, leading to poor solutions to the problem dealt by GP. Several methods have been proposed in the past to circumvent this limitation, allowing GP to deal with different data types and satisfying the closure property. The two main approaches are discussed in the following sections.

2.1.1 Constrained syntactic structures

A method to enforce the closure property and allow multiple data types simultaneously is the use of “dynamic typing”. In this approach, each non-terminal has to manipulate arguments of any data type that can be returned from any terminal or non-terminal. There are two (non exclusive) methods to do so. The first method is to make every function of the function set to execute different actions for different data types received as input. The second method is to have these functions returning an error flag when the data types of the received arguments are incompatible, and then assigning an extremely bad fitness value to the corresponding tree [19].

A better way to satisfy data type restrictions is the use of ‘strong-typing’, that is, generating only trees that can satisfy data type restrictions. This, in fact, is essentially what Koza [17] proposes with his syntactically constrained structures. For problems that require variables of several data types, a set of syntactic rules defines, for each parent node, which kinds of nodes can be used as its child nodes. These constraints are always imposed when a new tree is generated as result of genetic operators [4, 17].

The central issue for syntactically-constrained structures is the definition of the syntactic rules that specify, for each non-terminal, which kinds of child node it can have [15, 19].

2.1.2 Grammar-based genetic programming

Another approach to go around the closure requirement of GP, being able to manipulate several different data types without producing invalid individuals, consists of using a grammar to enforce syntactic (and possibly semantic) constraints [7, 22, 30]. A grammar is composed by a set of non-terminal symbols, a set of terminal symbols, a starting symbol and a list of production rules. The starting symbol, usually a non-terminal, is designated to be the first symbol of the first rule. An individual is created by the complete derivation of the starting symbol, applying the production rules.

A grammar-based GP for rule discovery is described in detail in a recent book by [32]. One of the case studies described in this book is summarized here. The data being mined is a fracture data set containing 6,500 records and 8 attributes, namely: *Sex, Age, Admission Date (AdmDay), Length of Stay in Hospital (Stay), Diagnosis of Fracture (Diagnosis), Operation, Surgeon, Side of Fracture (Side)*. These attributes have the following dependence. The attribute *Diagnosis* depends on the attributes *Sex, Age* and *AdmDay*. The attribute *Operation* depends on the previous three attributes (*Sex, Age* and *AdmDay*) and on the attribute *Diagnosis*. The attribute *Stay* depends on all the other seven attributes.

A small part of the grammar for this data set, taking into account these dependencies, is shown in Figure 2. A more complete view of this grammar can be found in the cited reference. In Figure 2 terminal symbols of the grammar are written in **bold** font style. The first three grammar rules in the figure are justified by the fact that the user wants rules predicting one of the following three goal attributes: *Diagnosis, Operation* and *Stay*. The next three grammar rules specify that each of the above three prediction rules consists of an antecedent (Antec) and a consequent (Cons). The next grammar rules specify the structure of each antecedent and each consequent. Each antecedent is a conjunction of attribute-value descriptors, whereas each consequent is a single goal-attribute-value descriptor.

3 A Constrained-Syntax GP for Discovering Classification Rules

This section describes a constrained-syntax genetic programming (GP) system developed for discovering IF-THEN classification rules. Hence, the system addresses the classification problem, as discussed in the Introduction. The design of the system involved the following aspects: individual representation, genetic operators, fitness function, and classification of new instances. These subjects are discussed in the next subsections.

3.1 Individual Representation

The first step in designing a GP for classification is to choose between the Michigan or the Pittsburgh approach for individual representation [9, 10]. In essence, in the Michigan approach each individual corresponds to a single classification rule, so that a set of individuals - possibly the entire population - corresponds to a rule set, i.e. a complete solution for the target classification problem. (Note that we are using the term "Michigan approach" in a broad sense here, rather than in the narrow sense of referring only to classifier systems, as used by some authors.) By contrast, in the Pittsburgh approach each individual corresponds to a rule set, constituting a complete solution for the classification problem.

The Michigan approach has the advantage of working with simpler, shorter individuals, avoiding the need for more elaborate genetic operators that cope with rule sets. On the other hand, it has the disadvantage that each individual corresponds only to a partial solution for the classification problem, making it difficult to evaluate the quality of an entire rule set, whose component rules are contained in different individuals.

The Pittsburgh approach works with more complex, longer individuals, often requiring somewhat more elaborate genetic operators, but it has the advantage that each individual represents an entire solution (rule set) for the classification problem.

Therefore, the quality of a candidate rule set can be easily evaluated as a whole, by taking into account the interaction among all the rules composing the individual.

In this paper we follow a kind of hybrid Pittsburgh/Michigan approach, as follows. An individual can contain multiple classification rules, subject to the restriction that all its rules have the same consequent - i.e., they predict the same class. In other words, an individual consists of a set of rule antecedents and a single rule consequent. The rule antecedents are connected by a logical OR operator, and each rule antecedent consists of a set of conditions connected by a logical AND operator. Therefore, an individual is in disjunctive normal form (DNF) - i.e., an individual consists of a logical disjunction of rule antecedents, where each rule antecedent is a logical conjunction of conditions (attribute-value pairs).

The rule consequent specifies the class to be predicted for an instance that satisfies all the conditions of any of the rule antecedents.

The terminal set consists of the attribute names and attribute values of the data set being mined. The function set consists of logical operators (AND, OR) and relational operators ("=", "≠", "≤", ">").

Figure 3 shows an example of the genetic material of an individual. Note that the rule consequent is not encoded into the genetic material of the individual. Rather, it is chosen by a deterministic procedure, as will be explained later. In the example of Figure 3 the individual contains two rules, since there is an OR node at the root of the tree. Indeed, the tree shown in that figure corresponds to the following two rule antecedents:

IF ($A_1 \leq 2$)

OR

IF (($A_3 \neq 1$) AND ($A_5 > 1$))

Once the genetic material (set of rule antecedents) of an individual is determined, the rule consequent (predicted class) associated with the individual is chosen in such a way that the fitness of the individual is maximized. More precisely, let *k* be the number of classes. For each of the *k* classes, the system computes what would be the fitness of the individual if that class were chosen to be the class predicted by the individual. Then, the system chooses the class that leads to the best (largest) fitness value for the individual.

So far, this individual representation might be seem as a representative of the Pittsburgh approach, since an individual consists of multiple rules. However, in our approach an individual does not represent an entire solution for the classification problem, for the following reason. As mentioned above, all the rules of an individual have the same rule consequent - i.e., they predict the same class. This leaves us with the problem of how to discover rules predicting different classes. The most common solution for this problem in the literature is to run the GP *k* times, where *k* is the number of classes [2, 16]. In the *i-th* (*i=1,...,k*) run, the GP discovers rules predicting the *i-th* class. Instead of using this conventional approach, our system works with a population of individuals where different individuals may have different rule consequents. Hence, in our approach an entire solution for the classification problem consists of *k* individuals, each of them predicting a different class. In other words, at the end of the evolution, the solution returned by GP will consist of *k* individuals, each of them being the best individual (the one with the best fitness value) for a different class. Therefore, it seems fair to call this approach a hybrid Pittsburgh/Michigan approach.

To summarize, in our individual representation each individual consists of a set of rules predicting a given class, and an entire solution for the classification problem consists of *k* individuals, each of them predicting a different class.

The main advantage of our hybrid Pittsburgh/Michigan approach, by comparison with the previously mentioned conventional Pittsburgh approach, is that in the former we need to run the GP just once to discover rules predicting different classes. By contrast, in the latter we need to run GP *k* times, where *k* is the number of classes. Therefore, our approach is considerably more efficient, in terms of computational time.

In the next two subsections we specify syntactical and semantic constraints (respectively) associated with our individual representation. It should be stressed that these constraints are used throughout the GP evolution, including both the generation of individuals in the initial population and the production of new individuals via crossover.

3.1.1 Syntactic Constraints on the Individual Representation

Conventional GP systems must satisfy the property of closure, which means that the output of any function of the function set can be used as the input for any other function of it. This property is satisfied, for instance, if the function set contains only mathematical operators (like +, -, /, *) and all terminal symbols are real-valued variables or constants.

However, in a typical data mining scenario the situation is more complex, since we often want to mine a data set with a mixing of categorical (nominal) and continuous (real-valued) attributes. Hence, our individual representation includes several constraints useful for data mining applications, as follows.

First, we specify, for each function of the function set, what are the data types valid for the input arguments and the output of the function. As mentioned earlier, the function set of our GP consists of logical operators (AND, OR) and relational operators ("=", "≠", "≤", ">"). The valid data types for the input arguments and output of these operators is shown in Table 1.

Note that all operators of Table 1 take two input arguments, so that each GP individual is represented by a binary tree. Our GP can cope with attributes that are either categorical (nominal) or continuous (real-valued), which is a desirable flexibility in a data mining system. It can also cope with boolean attributes, of course, since boolean attributes are a special case of categorical attributes where the number of values in the attribute domain is just two.

The data type restrictions specified in Table 1 naturally suggest an individual representation based on a hierarchy of operators. Of course, at the deepest level in the tree, the leaf nodes are terminal symbols - i.e., attributes or their values.

One level up there are internal nodes containing relational operators. Each of those internal nodes will have two child nodes, one of them containing an attribute and the other containing a value in the domain of that attribute. If the attribute-value pair in question is categorical, the internal node will be either "=" or "≠". On the other hand, if the attribute-value pair in question is continuous, the internal node will be either "≤" or ">". In any case, the output of such an internal node will be a boolean value.

In the higher levels of the tree the internal nodes contain a logical operator, either AND or OR. Hence, the output of the tree (i.e., the output of the root node) will be a boolean value, indicating whether or not a data instance satisfies the set of rule antecedents encoded in the individual. An example of this hierarchical structure was previously shown in Figure 3. Note that the individual shown in that Figure satisfies all data type constraints specified in Table 1.

Another constraint of our hierarchical individual representation is that an AND node cannot be an ancestor of an OR node. Although this constraint is not essential for producing syntactically-valid individuals, we find it useful in practice, because it enforces the restriction that every individual represents a set of rule antecedents in disjunctive normal form (DNF). The DNF representation is not only intuitively simple, but also facilitates the enforcement of a semantic constraint, namely an attribute uniqueness constraint, as described next.

Finally, our system also enforces a constraint of uniqueness of an attribute in the rule antecedent, i.e., each attribute can occur only once in a given antecedent. This constraint is necessary to avoid invalid rule antecedents like: IF (*Sex* = *male*) AND (*Sex* = *female*).

To summarize, our hierarchical individual representation has the following syntactic constraints:

(a) A leaf (terminal) node can have as its parent only a relational operator - i.e., "=" or "≠" if the leaf node's data type is is categorical, "≤" or ">" if the leaf node's data type is continuous;

(b) An internal node containing a relational operator ("=", "≠", "≤" or ">") can have as its parent only a logical operator (AND or OR);

(c) An internal node containing an AND operator can have as its parent only a logical operator (AND or OR);

(d) An internal node containing an OR operator can have as its parent only another OR operator.

(e) An attribute can occur at most once in a rule antecedent.

3.2 Genetic Operators

Our GP uses the reproduction and crossover operators. The reproduction operator consists simply of passing a copy of an individual to the next generation, as discussed in section 2. The crossover operator used here is a variant of the standard tree-crossover operator also discussed in section 2. In our system that crossover operator is adapted to our constrained-syntax individual representation, as follows.

First, a crossover point (a tree node) is randomly selected in one of the parent individuals, here called the first parent. Then the crossover point (tree node) of the other parent individual, here called the second parent, is randomly selected among the nodes that are compatible with the crossover point of the first parent. The crossover is performed by swapping the subtrees rooted at the crossover points of the two parent individuals, as usual.

In essence, two tree nodes are compatible if their operators can be swapped without producing an invalid individual. More precisely, we define the following compatibility relation for the operators included in Table 1:

(a) A node containing either the operator "≤" or ">" is compatible only with another node containing either "≤" or ">";

(b) A node containing either the operator "=" or "≠" is compatible only with another node containing either "=" or "≠";

(c) A node containing the AND operator is compatible only with another node containing AND;

(d) A node containing the OR operator is compatible only with another node containing OR.

If the second parent does not contain any tree node that is compatible with the first parent's crossover point, then the crossover operation is aborted and both parents are reproduced, without any modification, to the next generation.

Note that, although swapping subtrees that return the same data type guarantees that all tree nodes have inputs and output of a valid data type as specified in Table 1, it does not guarantee that all new offspring are valid. Any of the two new offspring produced by crossover could still be an invalid individual, due, e.g., to a violation of the condition of uniqueness for each attribute in each rule antecedent. If one of the new child individuals is valid and the other one is invalid, the former is put in the next generation, but the latter is discarded. Instead of it, one of two parents is randomly selected to pass, without modification, to the next generation. If the two new child individuals are invalid, the crossover is aborted and both parents are reproduced, without modification, to the next generation.

Our GP also uses a form of elitism that we call classwise elitism. The basic idea of elitism is that the best (or a small set of best) individual(s) of a generation is passed unchanged to the next generation, to prevent the stochastic process of evolution from losing that individual. Recall that the population contains individuals predicting different classes. In our classwise elitism the best individual of each of the *k* classes is chosen to be passed unchanged to the next generation. In other words, *k* elite individuals are passed unaltered to the next generation. The *i-th* elite individual (*i* =1,...,*k*) is the best individual among all individuals predicting the *i-th* class.

3.3 Fitness Function

The fitness function used in this work is the same as that proposed in [2] for a “booleanized” GP (that is, a GP that accepts only boolean inputs). The major difference between the current work and that one is the constrained-syntax individual representation used here, which is more elaborate than the simple approach of working only with boolean attribute values used in the that former work.

The fitness function evaluates the quality of each individual (a rule set where all rules predict the same class) according to two basic criteria, namely its predictive accuracy and its simplicity.

Before the definition of the fitness function, a review of some basic concepts on classification-rule evaluation is necessary. When using a rule for classifying a data instance (a record of the data set being mined), depending on the class predicted by the rule and on the actual class of the instance, one of the following four types of result can be observed:

- *True positive* - the rule predicts that the instance has a given class and the instance does have that class;
- *False positive* - the rule predicts that the instance has a given class but the instance does not have it;
- *True negative* - the rule predicts that the instance does not have a given class, and indeed the instance does not have it;
- *False negative* - the rule predicts that the instance does not have a given class but the instance does have it.

Let *tp*, *fp*, *tn* and *fn* denote respectively the number of true positives, false positives, true negatives and false negatives observed when a rule is used to classify a set of instances. The first part of our fitness function, which measures the predictive accuracy of a rule set, combines two indicators commonly used for evaluating medical-diagnosis procedures, namely the sensitivity (*Se*) and the specificity (*Sp*), defined as follows:

$$Se = tp \,/\, (tp + fn) \tag{1}$$

$$Sp = tn \,/\, (tn + fp) \tag{2}$$

Finally, the measure of predictive accuracy used by our GP is defined as the product of these two indicators, i.e.:

$$\textit{predictive accuracy} = Se \cdot Sp \tag{3}$$

The goal of equation 3 is to foster the GP to maximize both the *Se* and the *Sp* at the same time. This is an important point, since it would be relatively trivial to maximize the value of one of these two indicators at the expense of significantly reducing the value of the other. Note that equation 3 has the advantages of being simple and returning a meaningful, normalized value in the range [0..1]. This measure of predictive accuracy seems to have been first used in an evolutionary algorithm by [18], and it has also been independently suggested for classification in general (independent of the paradigm of the classification algorithm) by [14].

The second criterion considered by our fitness function is the simplicity of the rule set represented by an individual. In fact, GP does not necessarily produce simple solutions. Considering that the comprehensibility of a rule is inversely proportional to its size, something has to be done to force GP to produce rules as short as possible. Therefore, we define a measure of simplicity (*Sy*) of a rule, given by:

$$Sy = (maxnodes - 0.5 \cdot numnodes - 0.5) / (maxnodes - 1) \tag{4}$$

where *numnodes* is the current number of nodes (functions and terminals) of an individual (tree), and *maxnodes* is the maximum allowed size of a tree (set to 45). Equation 4 produces its maximum value of 1.0 when a rule is so simple that it contains just one term. The equation value decreases until its minimum of 0.5, which is produced when the number of nodes equals the maximum allowed. The reason to set the lower bound to 0.5 is to penalize large-sized individuals without forcing them to disappear from the population. This is especially important in the early generations of a run, when most individuals will have very low predictive accuracy but can carry good genetic material capable of further improvement by means of the genetic operators.

Finally, the fitness function used by our GP is fully defined as the product of the indicators of predictive accuracy and simplicity, as follows:

$$fitness = Se \cdot Sp \cdot Sy. \tag{5}$$

Therefore, the goal of our GP is to maximize both the *Se* and the *Sp*, and to minimize simultaneously the rule set size. This fitness function has also the advantages of being simple and returning a meaningful, normalized value in the range [0..1].

3.4 Classification of New Instances

Recall that, after the GP run is over, the result returned by GP consists of a set of *k* individuals, where *k* is the number of classes. The *i-th* returned individual (*i*=1,...,*k*) consists of a set of rules predicting the *i-th* class for a data instance (record) that satisfies the rule set associated with the individual. An instance is said to satisfy a rule set if it satisfies all the conditions of at least one of the rules contained in the rule set. Recall that an individual contains a rule set in disjunctive normal form.

When the set of returned individuals is used to classify a new instance (in the test set), the instance will be matched with all the *k* individuals, and one of the following three situations will occur:

(a) The instance satisfies the rule set of exactly one of the *k* individuals. In this case the instance is simply assigned the class predicted by that individual;

(b) The instance satisfies the rule set of two or more of the *k* individuals. In this case the instance is assigned the class predicted by the individual with the best fitness value (computed in the training set, of course);

(c) The instance does not satisfy the rule set of any of the *k* individuals. In this case the instance is assigned a default class, which is the majority class, that is the class of the majority of the instances in the training set.

4 Computational Results

4.1 Data Sets and Running Parameters

Experiments were done with five data sets – namely Chest pain, Ljubljana breast cancer, Dermatology, Wisconsin breast cancer, and Pediatric adrenocortical tumor. Table 2 summarizes the information about the first four data sets: the number of examples (records), attributes and classes. Ljubljana breast cancer, Wisconsin breast cancer and Dermatology are public domain data sets, available from the Machine Learning Repository at the University of California at Irvine (http://www.ics.uci.edu/~mlearn/MLRepository.html). The Chest pain data set is described in more detail in [2]. The Pediatric adrenocortical tumor is discussed in details in section 4.3.

Two of these data sets (Ljubljana Breast cancer and Dermatology) had some examples with missing values. In our experiments all examples with missing values were removed from these data sets. The number of examples reported in the second column of Table 2 is the number after removal of examples with missing values.

The parameters used in all GP runs were: population size: 500, generations: 50 (excluding the first population that was generated randomly), crossover probability: 95%, reproduction probability: 5%, selection method for both parents: fitness proportional (roulette wheel), steady-state, maximum tree depth: 15, stop criterion: maximum number of generations. The initial population was generated using ramped-half-and-half method [17] and initial tree depth: 12.

4.2 Results Comparing the GP with C4.5 and BGP in Four Data Sets

For each data set shown in Table 2 the performance of the GP system was compared with C4.5 [24], a very well-known decision-tree induction algorithm often used for benchmarking purposes in the data mining and machine learning literature, and also a "booleanized" version of GP (BGP) described in [2]. In all the experiments the default parameters of C4.5 and BGP were used. In order to make the comparison as fair as possible, we have also used the default parameters of our GP system. In other words, we have made no attempt to optimize the parameters of either C4.5, BGP or our GP system.

The comparison between the GP proposed in this paper and BGP is particularly interesting because both algorithms use the same fitness function. Hence, differences in their performance can be mainly associated with their different individual representations and corresponding genetic operators. BGP uses a simpler individual representation where all attribute values are "booleanized" in a preprocessing step (see [2] for details) and the function set contains only boolean operators. By contrast, the GP proposed in this paper uses a more complex representation, where the function set includes both boolean (AND, OR) and relational ($=$, $\neq$, $<$, $>$) operators. This has the advantage of avoiding the need for booleanization in a preprocessing step, and can potentially increase predictive accuracy, since that preprocessing step could lead to the loss of some relevant detail in the data.

All the results obtained with our GP and with C4.5 and BGP reported in this section were obtained by performing a five-fold cross-validation procedure [14, 29]. The basic idea is that the data set is first divided into five mutually exclusive and exhaustive partitions. Then, each classification algorithm is run five times. In each of these runs a different partition is used as the test set and the other four partitions are grouped into a training set. In each of those five runs the accuracy rate on the test set is computed as the ratio of the number of correctly classified test instances over the total number of test instances, as usual in the classification literature. The accuracy rates on the test set are averaged over the five runs, and this average result is reported. We stress that the GP, C4.5 and BGP are given exactly the same training and test sets in each of the five runs of the cross-validation procedure, again making the comparison as fair as possible.

The accuracy rates obtained by our GP, C4.5 and BGP are shown in Table 3. The numbers after the "±" symbol are standard deviations.

In the Ljubljana breast cancer data set C4.5 and the GP proposed in this paper achieved about the same accuracy rate, since the minor difference between them is certainly not significant. However, in Chest pain and Dermatology data sets, the GP outperformed C4.5, and the difference is clearly significant. Note that the accuracy rate intervals (taking into account the standard deviations) do not overlap in the case of these two data sets. For the Wisconsin breast cancer data set C4.5 had a somewhat better performance than our GP, and the difference is also significant, since the corresponding accuracy rate intervals do not overlap.

For all four data sets, our GP outperformed BGP. However, the accuracy rate intervals do overlap (indicating that the differences are not significant) – except for the Dermatology data set, where our GP significantly outperformed BGP. C4.5 had an average performance better than BGP in all but the Chest pain data set. However, the accuracy rate intervals overlap both for Chest pain and Dermatology data sets. Therefore, only for Ljubljana cancer and Wisconsin cancer datasets there are significant differences in the accuracy rates of C4.5 and BGP.

4.3 Results Comparing the GP with C4.5 and BGP in the Pediatric Adrenocortical Tumor Data Set

In this subsection we compare the results of our GP with C4.5 and BGP in a new data set, called Pediatric Adrenocortical Tumor. We emphasize that preparing this data set for data mining purposes was a considerable challenge. We had to carry out a significant preprocessing of the available data, as described in the following [3].

The original data set consisted of 178 instances (records) and 10 attributes. This included 54 instances with missing value for one or more attributes. These 54 instances were removed from the data, so that the data set used in our experiments contained 124 instances. The attributes and their values are shown in Table 4.

The first step was to decide which attribute would be used as the goal (or class) attribute, to be predicted. It was decided to predict how long a patient will survive after undergoing a surgery. The corresponding goal attribute is hereafter called *Survival*. The values of this attribute for the instances were not directly available in the original data set. It had to be computed in an elaborate way, as follows.

First the system computed, for each instance (patient), the difference, measured in number of days, between the date of the surgery and the date of the last follow up of the patient.

Then the system checked, for each instance, the value of another attribute called *Status*, whose domain contained four values. One of these values indicated that the patient was *dead*, whereas the other three values indicated that the patient was still *alive*. (The difference in the meaning of those three values indicating *alive* patient reflect different stages in the progress of the disease, but this difference is not relevant for our discussion here.)

A major problem in predicting *Survival* is that, if the *Status* of a patient (as recorded in the hospital's database) is different from *dead*, this does not necessarily means that patient is still *alive* in real life. It might be the case that the patient has actually died, but this information was not yet included in the database, due to several reasons that might lead to a loss of contact between the family of the patient and the hospital. On the other hand, if the value of *Status* recorded in the hospital's database is *dead*, this information is presumably much more reliable.

As a result, for many of the patients, one cannot be sure about the true value of the *Survival* attribute. One can be sure about this value only when the value of the *Status* attribute is *dead*. When *Status* is different from *dead*, the value of *Survival* computed as described above is just an underestimate of the true value of that attribute. Hence, any attempt to directly predict the value of *Survival* would be highly questionable, being prone to produce unreliable results.

To circumvent this problem, our solution was to transform the original problem of predicting *Survival* for all patients into three separate problems, each of them carefully defined to lead, at least in principle, to more reliable results. We try to predict the value of *Survival* for each of three classes of this attribute separately. These three classes were defined by discretizing the *Survival* attribute (which was previously measured in number of days) into three ranges of values, namely less than one year, between one and two years, between two and five years. Hereafter these ranges are called class 1, class 2 and class 3, respectively, for short. This discretization was suggested by an oncologist, based on his experience in clinical practice.

This leads to three classification experiments, each of them aiming at discriminating between two classes, a "positive" class and a "negative" class. In the *i-th* experiment, $i = 1,2,3$, the instances having class *i* are considered as positive-class instances, and all the other instances are considered as negative-class instances.

The reason why we need to perform three separate classification experiments is as follows. As mentioned above, when the patient's *Status* is different from *dead*, one cannot be sure about the true value of the *Survival* attribute. For instance, suppose that a patient underwent surgery one and a half year ago. One cannot be sure if the patient has class 2 or 3, since (s)he might or not live until (s)he completes two years of survival after surgery. However, one can be sure that this patient does not have class 1. So, its corresponding instance can be used as a negative-class instance in the first classification experiment, aiming at predicting whether or not a patient has class 1. On the other hand, that instance cannot be used in the second or third classification experiments, because in those experiments there would be no means to know if the instance had a positive class or a negative class.

The key idea here is that an instance is used in a classification experiment only when one can be sure that it is either definitely a positive-class instance or definitely a negative-class instance, and for some instances (those having *Status* different from *dead*) this depends on the classification experiment being performed. It is important to note that, as a result of the previous scheme, the number of instances given as input to the classification algorithm varies across the three classification experiments, which is by itself a decisive argument for performing three separate classification experiments.

Finally, we now precisely specify how we have defined which instances were used as positive-class or negative-class instances in each of the three classification experiments.

The first experiment consists of predicting class 1, i.e. *Survival* less than one year. In this experiment the positive-class instances are the patients whose *Status* is *dead* and whose *Survival* is less than or equal to one year. The negative-class instances are the patients whose *Survival* is greater than one year. After this instance-filtering process the data set contained 22 positive-class instances and 83 negative-class instances.

The second experiment consists of predicting class 2, i.e. *Survival* between one and two years. In this experiment the positive-class instances are the patients whose *Status* is *dead* and *Survival* is greater than one year and less than or equal to two years. The negative-class instances are the patients whose *Status* is *dead* and *Survival* is either less than one year or greater than two years. After this instance-filtering process the data set contained 8 positive-class instances and 86 negative-class instances.

The third experiment consists of predicting class 3, i.e. *Survival* between two years and five years. In this experiment the positive-class instances are the patients whose *Status* is *dead* and *Survival* is greater than two years and less than or equal to five years. The negative-class instances are the patients whose *Status* is *dead* and *Survival* is either less than two years or greater than five years. After this instance-filtering process the data set contained 6 positive-class instances and 62 negative-class instances.

Table 5 reports the accuracy rate (on the test set) obtained by our GP, C4.5 and BGP in each of the three classification experiments. These experiments have followed the same methodology as the experiments reported in section 4.2. Hence, in all the experiments we have used the default parameters of the GP, C4.5 and BGP, making no attempt to optimize the parameters of the systems, and all the results reported here were obtained by performing a five-fold cross-validation procedure.

For all three classes, BGP had a worse performance than both GP and C4.5. In particular, for class 3, BGP did not correctly classify any case, which was very surprising. Based only on the results reported in this table, at first glance one might conclude that C4.5 outperforms our GP system in this data set. In two out of the three classes (namely, classes 2 and 3) the accuracy rate of C4.5 is significantly better than the one of the GP. However, this conclusion would be premature, as we now show.

The problem with the previous results is that they are based on classification accuracy rate. Although this measure of predictive accuracy is still the most used in the literature, it has some drawbacks [14]. The most important one is that it is relatively easy to achieve a high value of classification accuracy when one class (the majority class) has a high relative frequency in the data set. In one extreme, suppose that 99% of the examples have a given class, called c_1 (Such cases are not as rare as one might think. They are actually quite common in applications such as fraud detection, prediction of rare diseases, etc). In this case one can trivially achieve a classification accuracy rate of 99% by "predicting" class c_1 for all examples. Does that mean that the classification algorithm (a trivial majority classifier) is doing a good job? Of course not. What this means is that the measure of classification accuracy rate is too weak in this case, in the sense that it is too easy to get a very high value of this measure. One needs a more demanding measure of predictive accuracy, which assigns more importance to the classification of the examples that are really difficult to classify, namely the minority class examples.

Indeed, an analysis of the trees induced by C4.5 shows that the results of the last two rows of Table 5 referring to classes 2 and 3) are very misleading. In particular, C4.5 is *not* discovering better rules for these classes. More precisely, when predicting class 2 and class 3, C4.5 induces a degenerate, "empty" tree with no internal node; i.e., a tree containing only one leaf node, predicting the majority class. This has consistently occurred in all the five folds of the cross-validation procedure. Clearly, C4.5 opted for an "easy solution" for the classification problem, favoring the correct prediction of the majority of the examples at the expense of making an incorrect prediction of all the minority-class examples. Such an easy solution is useless for the user, since it provides no rules (i.e., no knowledge) for the user. Only when predicting class 1, C4.5 was able to induce a non-degenerate, non-empty tree on average. And even for this class an empty tree was induced in some folds of the cross-validation procedure.

By contrast, our GP system discovered, on average, rules with 2, 1.7 and 1.9 conditions, for rules predicting classes 1, 2 and 3, respectively. Overall, the rules were considered comprehensible by the user. Of course, this does not imply that the predictive accuracy of the rules discovered by our GP is good. What we need, as argued above, is a more demanding measure of predictive accuracy, which emphasizes the importance of correctly classifying examples of all classes, regardless of the relative frequency of each class. This is the basic idea behind the previously-explained measure that includes both sensitivity and specificity (equation 3).

Hence, we report in Table 6 the values of sensitivity (Se), specificity (Sp), and the product $Se \cdot Sp$ obtained by our GP, C4.5 and BGP in the Pediatric adrenocortical tumor data set. This table contains one row for each class being predicted.

As can be observed in this table, all systems failed to discover good rules predicting class 3 but, unlike C4.5, our GP and BGP succeeded in discovering good rules (with relatively good values of Se and Sp) predicting classes 1 and 2. In addition, in all the three classes, the value of the product $Se \cdot Sp$ obtained by our GP considerably outperforms the one obtained by C4.5. For class 1, BGP has achieved a slight better value for the product $Se \cdot Sp$ than our GP, due to a more balanced performance regarding sensitivity and specificity (almost the same value). However, for class 2 our GP obtained a considerably better value of the product $Se \cdot Sp$ than BGP.

5 Conclusions

We have proposed a constrained-syntax GP for discovering classification rules. This GP uses a function set consisting of both logical operators (AND, OR) and relational operators (“=”, “≠”, “≤”, “>”). Rule sets are represented by individuals where these operators are arranged in a hierarchical, tree-like structure. We have specified several syntactic constraints to be enforced by the GP, so that individuals represent rule sets that are valid and easy to interpret, due to the use of the disjunctive normal form representation.

The GP was compared with C4.5, a very well-known classification algorithm, and BGP, a “booleanized” version of GP, in five data sets. In two data sets the accuracy rate of the GP was significantly better than the one of C4.5. In one data set the two systems obtained similar accuracy rates. In one data set C4.5 significantly outperformed the GP system, and in the fifth data set the accuracy rate of C4.5 was significantly better than the one of the GP at first glance.

However, an analysis of the trees built by C4.5 showed that it was *not* discovering better classification rules in the fifth data set. It was just building a degenerate, empty tree, predicting the majority class for all data instances. We then performed a more detailed analysis of the predictive accuracy of both systems, measuring the sensitivity and the specificity rates for each class separately, and showed that, according to this measure of predictive accuracy, overall the GP obtained considerably better results than C4.5.

In general, BGP did not show a good performance when compared with both C4.5 and our GP. For all data sets our GP was better than BGP and for three out of five data sets, C4.5 was significantly better than BGP.

We believe the main contributions of this paper are as follows. First, we have proposed a constrained-syntax GP using a new set of constraints tailored for discovering simple classification rules. Note that, as discussed before, the idea of constrained-syntax GP is not new. Furthermore, most of the constraints proposed here could also be implemented by using a grammar-based GP. However, both constrained-syntax GP and grammar-based GP are general approaches for coping with the closure problem in GP. They were not developed for discovering classification rules. Our approach is less generic, since it was specifically developed for discovering classification rules. In this sense our approach can be considered a special case of constrained-syntax GP particularly tailored for discovering classification rules. This specificity has the advantage that we are able to incorporate constraints that would be difficult to incorporate in a general constrained-syntax GP or a grammar-based GP. A typical example is the constraint that an attribute can occur at most once in a rule antecedent. Second, we have shown that overall this algorithm obtained good results with respect to predictive accuracy, by comparison with C4.5 and BGP, on five medical data sets. We make no claim that our GP is better than C4.5 in general, since no classification algorithm is universally best across all application domains [20, 25, 26]. Third, we have proposed a new way of preprocessing a medical data set for the purpose of performing a computational prediction of how long a patient will survive after a surgery. This preprocessing method was designed to cope with a major problem in predicting the time of *Survival* of a patient. Namely, if the *Status* of a patient (as recorded in the hospital's database) is different from *dead*, this does not necessarily mean that patient is still *alive* in real life. It might be the case that the patient has actually died, but this information was not yet included in the database, due to several reasons that might lead to a loss of contact between the family of the patient and the hospital. The proposed preprocessing method copes with this problem by using two basic ideas, namely:

(a) transforming the original problem of classification into separate classification problems, each of them predicting whether or not the patient belongs to a given class (a time interval for survival); and

(b) performing a careful process of instance filtering, in such a way that an instance is used in a classification experiment only when one can be sure that it is either definitely a positive-class instance or definitely a negative-class instance, for each of the previously-defined classification experiments.

Although the proposed preprocessing method for predicting patient survival was applied in this work only to the Pediatric adrenocortical tumor data set, it is a relatively generic method. Of course, it cannot be applied to other kinds of problem such as discovering prediction rules for diagnosis. However, it can be applied to any other data set where the goal attribute to be predicted is the survival of the patient, regardless of the underlying disease(s) or event (for instance, surgery) causing the death of the patient. In this sense, this method has a good degree of generality in the context of medical applications.

In the future we intend to apply our GP to other data sets, and compare it with other classification algorithms. Other possible research direction consists of performing experiments with other function sets and evaluate the influence of the function set of the GP in its performance, across several different data sets.

References

[1] 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, San Francisco, 1998).

[2] C.C. Bojarczuk, H.S. Lopes, and A.A. Freitas. Genetic programming for knowledge discovery in chest pain diagnosis, IEEE Engineering in Medicine and Biology Magazine 19 (2000) 38-44.

[3] C.C. Bojarczuk, H.S. Lopes, and A.A. Freitas. An innovative application of a constrained-syntax genetic programming system to the problem of predicting survival of patients. Genetic Programming: Proc. 6th European Conf. (EuroGP-2003), LNCS 2610, (Heidelberg : Springer-Verlag, 2003) 11-21 .

[4] C. Clack and T. YU. PolyGP: A polymorphic genetic programming system in haskell. Genetic Programming 1998: Proceedings of the 3rd Annual Conference. (Morgan Kaufmann, San Francisco, 1998) 416-421.

[5] V. Dhar, D. Chou and F. Provost. Discovering interesting patterns for investment decision making with GLOWER – a genetic learner overlaid with entropy reduction, Data Mining and Knowledge Discovery Journal 4 (2000) 251-280.

[6] U.M. Fayyad, G. Piatetsky-Shapiro and P. Smyth. From data mining to knowledge discovery: an overview. In: U.M. Fayyad et al. (Eds.) Advances in Knowledge Discovery and Data Mining (AAAI/MIT, Menlo Park, 1996) 1-34.

[7] J.J. Freeman: A linear representation for GP using context free grammars, Genetic Programming 1998: Proceedings of the 3rd Annual Conference (Morgan Kaufmann, San Francisco, 1998) 72-77.

[8] A.A. Freitas. Understanding the crucial role of attribute interaction in data mining. Artificial Intelligence Review 16 (2001) 177-199.

[9] A.A. Freitas. A survey of evolutionary algorithms for data mining and knowledge discovery, In: A. Ghosh and S. Tsutsui. (Eds.) Advances in Evolutionary Computation. (Springer-Verlag, Berlin, 2002) 819-845.

[10] A.A. Freitas. Data Mining and Knowledge Discovery with Evolutionary Algorithms. (Springer-Verlag, Berlin, 2002).

[11] A.A. Freitas. Evolutionary Algorithms. In: J. Zytkow and W. Klosgen. (Eds.) Handbook of Data Mining and Knowledge Discovery. (Oxford University Press, Oxford, 2002) 698-706.

[12] D.P. Greene and S.F. Smith. Competition-based induction of decision models from examples, Machine Learning 13 (1993) 229-257.

[13] F. Gruau. Cellullar encoding of genetic neural networks. Technical Report 92-21, Laboratorie de l'Informatique du Parallelisme (Ecole Normale Superieure de Lyon, France, 1992).

[14] D.J. Hand. Construction and Assessment of Classification Rules. (John Wiley & Sons, Chicester, 1997).

[15] C.Z. Janikow and S. Deweese. Processing constraints in genetic programming with CGP2.1. Genetic Programming 1998: Proceedings of the 3rd Annual Conference (Morgan Kaufmann, San Francisco, 1998) 173-180.

[16] J.K. Kishore, L.M. Patnaik, V. Mani and V.K. Agrawal. Application of genetic programming for multicategory pattern classification. IEEE Transactions on Evolutionary Computation 4(3) (2000) 242-258.

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

[18] H.S. Lopes, M. S. Coutinho, R. Heinisch, J.M. Barreto and W.C. Lima. A knowledge-based system for decision support in chest pain diagnosis, Medical & Biological Engineering & Computing 35 (suppl., part I) (1997) 514.

[19] D.J. Montana. Strongly typed genetic programming, Evolutionary Computation 3 (1995) 199-230.

[20] D. Michie, D.J. Spiegelhalter and C.C. Taylor. Machine Learning, Neural and Statistical Classification (Ellis Horwood, New York, 1994).

[21] T.M. Mitchell. Machine Learning (McGraw-Hill, New York, 1997).

[22] P.S. Ngan, M.L. Wong, K.S. Leung and J.C.K. Cheng. Using grammar based genetic programming for data mining of medical knowledge. Genetic Programming 1998: Proceedings of the 3rd Annual Conference (Morgan Kaufmann, San Francisco, 1998) 254-259.

[23] A. Papagelis and D. Kalles. Breeding decision trees using evolutionary techniques. Proc. 18th Int. Conf. on Machine Learning (ICML-2001) (Morgan Kaufmann, San Mateo, 2001) 393-400.

[24] J.R. Quinlan. C4.5: Programs for Machine Learning (Morgan Kaufmann, San Mateo, CA, 1993).

[25] R.B. Rao, D. Gordon and W. Spears. For every generalization action, is there really an equal and opposite reaction? Analysis of the conservation law for generalization performance. Proceedings of 12th International Conference on Machine Learning (Morgan Kaufmann, San Mateo, CA, 1995) 471-479.

[26] C. Schaffer. A conservation law for generalization performance.Proceedings of 12th International Conference on Machine Learning (Morgan Kaufmann, San Mateo, CA, 1994) 259-265.

[27] A. Teller and M. Veloso. PADO: a new learning architecture for object recognition. In: K. Ikeuchi and M. Veloso (eds.), Symbolic Visual Learning, (Oxford University Press, Oxford, 1996) 81-116.

[28] H. Vafaie and K. DeJong. Evolutionary Feature Space Transformation. In: Liu, H. & Motoda, H. (Eds.) Feature Extraction, Construction and Selection, (Kluwer, Boston, 1998) 307-323.

[29] S.M. Weiss and C.A. Kulikowski. Computer Systems that Learn, (Morgan Kaufmann, San Mateo, 1991).

[30] P.A. Whigham. Grammatically-based genetic programming. In: Rosca, J.P.(ed.), Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications, (Morgan Kaufmann, Tahoe City, CA, 1995) 33-41.

[31] I.H. Witten and E. Frank. Data Mining: practical machine learning tools and techniques with Java implementations, (Morgan Kaufmann, San Mateo, 2000).

[32] M. L. Wong and K. S. Leung. Data Mining Using Grammar Based Genetic Programming and Applications, Genetic Programming 3, (Kluwer Academic Publishers, Netherlands, 2000).

Table 1: Valid data types for each operator's input arguments and output.

Operator	**Input arguments**	**Output**
AND, OR	(boolean, boolean)	boolean
"=", "≠"	(categorical, categorical)	boolean
"≤", ">"	(real, real)	boolean

Table 2: Main characteristics of the data sets used in the experiments.

Data Set	**No. of examples**	**No. of attributes**	**No. of classes**
Chest pain	138	161	12
Ljubljana breast cancer	277	9	2
Dermatology	358	34	6
Wisconsin breast cancer	683	9	2

Table 3: Classification accuracy rate (%) on the test set.

Data Set	**C4.5**	**GP**	**BGP**
Chest pain	73.2 ± 0.77	80.3 ± 3.90	78.06 ± 4.85
Ljubljana breast cancer	71.4 ± 0.60	71.8 ± 4.68	63.85 ± 5.67
Dermatology	89.1 ± 0.13	96.6 ± 1.14	86.2 ± 6.24
Wisconsin brest cancer	94.8 ± 0.06	93.5 ± 0.79	89.3 ± 4.37

Table 5: Classification accuracy rate (%) on the test set, for the Pediatric Adrenocortical Tumor data set.

Class	**C4.5**	**GP**	**BGP**
1	75.7 ± 1.22	73.3 ± 2.43	58.07 ± 8.08
2	88.2 ± 0.77	78.8 ± 2.81	47.26 ± 12.94
3	87.3 ± 1.01	67.8 ± 6.82	0

Table 6: Sensitivity (*Se*) and Specificity (*Sp*) on the test set, for the Pediatric Adrenocortical Tumor data set.

Class	**C4.5**			**GP**			**BGP**		
	Se	***Sp***	***Se · Sp***	***Se***	***Sp***	***Se · Sp***	***Se***	***Sp***	***Se · Sp***
1	0.1	0.916	0.079	0.79	0.725	0.560	0.76	0.77	0.585
2	0	1	0	0.9	0.781	0.693	0.7	0.741	0.518
3	0	1	0	0.1	0.735	0.074	0	0	0

Table 4 – Description of the Pediatric Adrenocortical Tumor data set.

Attributes	Value	Description
Relapse	{0, 1}	0 – no relapse, 1 – relapse
Sex	{0, 1}	0 – male, 1 – female
Age	{13 to 7043}	patient's age at diagnosis date, in days
Tumor_volume	{2.99 to 4368}	tumor volume in cm^3
Histology	{0, 1}	0 – adenoma, 1 – carcinoma
Viriliz	{0, 1}	virilization: 0 – no, 1 – yes
Cushing	{0, 1}	cushing: 0 – no, 1 – yes
Nonfunctional	{0, 1}	0 – functional, 1 – non-functional
Stage	{1, 2, 3, 4}	1 – tumor completely excised with negative margins, tumor weight < 200g, and absence of metastasis. The above plus tumor spillage during surgery 2 – tumor completely excised with negative margins, tumor weight > 200g, and absence of metastasis. The above plus tumor spillage during surgery. Gross tumor excision with microscopic residual tumor, any tumor weight, and absence of metastasis 3 – gross residual tumor or inoperable tumor without IVC thrombus, absence of metastasis. Gross residual or inoperable tumor with IVC thrombus, absence of metastasis 4 – hematogenous metastasis at presentation; primary tumor and metastasis completely excised. Hematogenous metastasis at presentation; primary tumor and/or metastasis cannot be completely excised.
Status	{1, 2, 3, 4}	1 – alive – disease free 2 – alive - relapse 3 – alive - residual 4 – dead
Survival	{0 to 20}	Survival time after the surgery, in years

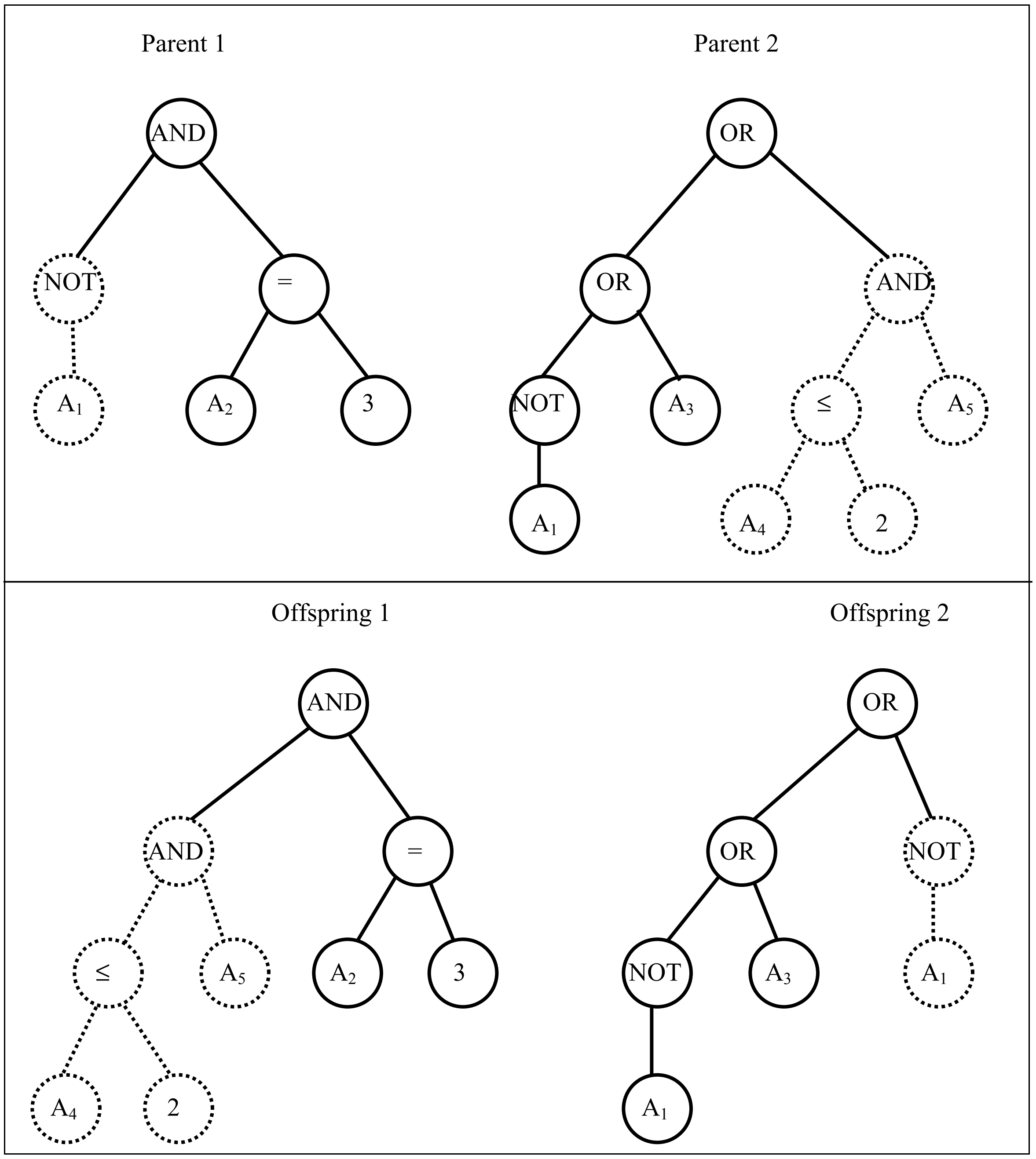

Figure 1 – Crossover operation in action. Top: original parents and crossover points. Bottom: offspring after crossover.

start => DiagnosisRule
start => OperationRule
start => StayRule
DiagnosisRule => **IF** DiagnosisAntec **THEN** DiagnosisCons
OperationRule => **IF** DiagnosisAntec **AND** OperationAntec
THEN OperationCons
StayRule => **IF** DiagnosisAntec **AND** OperationAntec **AND** StayAntec
THEN StayCons
DiagnosisAntec => SexDescriptor **AND** AgeDescriptor **AND** AdmDayDescriptor
DiagnosisCons => DiagnosisDescriptor
.
.

Figure 2 - Part of the grammar for a fracture data set

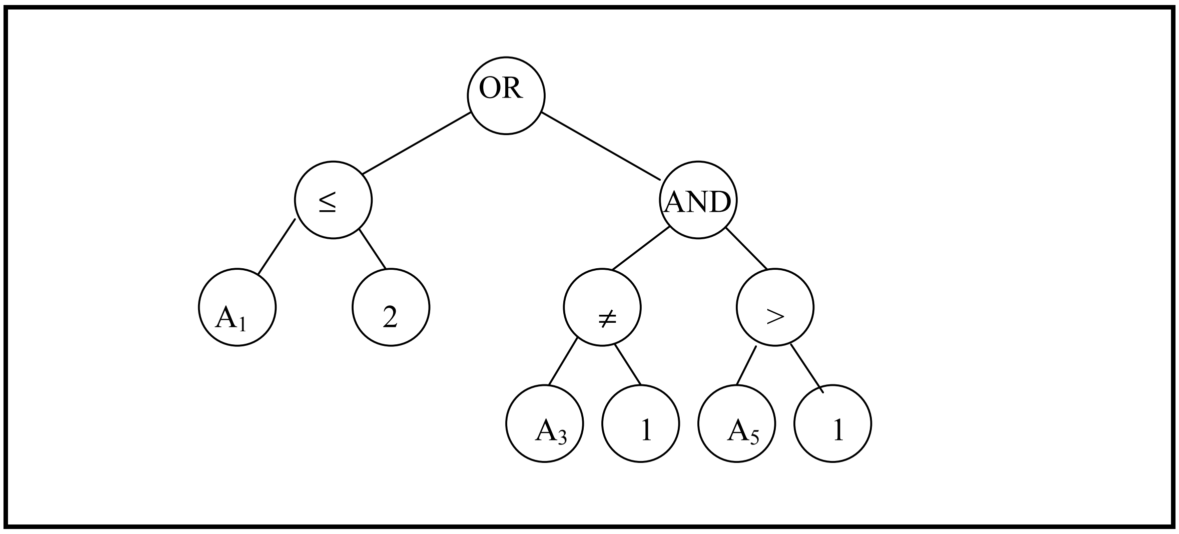

Figure 3 – Example of an individual.