Cellular Genetic Programming Algorithm Applied to Classification Task

Alexandra Takac

Institute of Informatics
Faculty of Mathematics, Physics and Informatics
Comenius University, Bratislava, Slovakia
takaca@ii.fmph.uniba.sk

Abstract. The focus of this paper is the application of the genetic programming framework in the problem of knowledge discovery in databases, more precisely in the task of classification. Genetic programming possesses certain advantages that make it suitable for application in data mining, such as robustness of algorithm or its convenient structure for rule generation to name a few. This study concentrates on one type of parallel genetic algorithms – cellular (diffusion) model. Emphasis is placed on the improvement of efficiency and scalability of the data mining algorithm, which could be achieved by integrating the algorithm with databases and employing a cellular framework. Cellular model of genetic programming that exploits SQL queries is implemented and applied to the classification task. Achieved results are presented and compared with other machine learning algorithms.

Keywords: genetic programming, cellular genetic algorithms, data mining, classification, evolutionary algorithms, knowledge discovery in databases, machine learning

1 Introduction

This study examines Genetic Programming (further only GP) and the possibilities of applying it to the classification task. GP is relatively new, domain-independent, stochastic method inspired by evolution. Although in the absence of a strong theoretical background, this method has made significant success in different application areas, such as, optimisation problem. Striking thing about Genetic Algorithms (GA) and various parallel models is the richness of this form of computation, where small changes in the algorithm often result in surprising kinds of emergent behaviour [28]. Here we will combine GP algorithm with the cellular parallel model of genetic algorithms, which is also referred to as “diffusion” or “fine-grained” genetic algorithm.

Second area of our interest will be machine learning and data mining problems. Algorithms for Data Mining (DM) and Knowledge Discovery in Databases (KDD) can reveal rules, relations between certain data or other interesting information, which does not have to be obvious at first, but can be very useful in many areas, such as:

medicine, economics or marketing. This task is however difficult to solve for the number of reasons, even though there are several different commonly used techniques. One of the challenges is huge amount of data to be processed. Until now, it has not been found a method, which would be proven to be the best for DM and KDD. Applications of learning algorithms in the knowledge discovery in databases, for example, are promising and relevant area of research. It is both interesting in business circles, offering new possibilities and benefits in real-world applications, and in the scientific field, helping us understand better mechanisms of our own methods of knowledge acquisition.

Comparing to the standard genetic algorithms, GP has the characteristics which are beneficial for application in DM, such as convenient structure for rule generation. The given framework is suitable for parallelization of DM algorithm, which can improve the efficiency of computation. On the other hand, like all genetic algorithms it offers wide range of variations and modifications of algorithm that may also lead to improved overall performance of the application. We will examine application of the cellular GP framework to the classification task. Emphasis will be placed on the improvement of efficiency and scalability of the algorithm, achieved by integration with databases and employing the cellular framework. Furthermore, implemented classification algorithm will be tested on the real-world datasets for machine learning, and the results will be compared with available results of other learning algorithms.

2 Genetic Programming (GP)

Genetic algorithms use the concept of an *individual*, which encodes a potential solution to a specific problem (for instance, the classification rule), and a set of individuals constitute a *population.* Every individual can be a potential solution of the algorithm. Algorithm runs for limited number of *epochs*, where the first epoch begins with initial randomly created population. Genetic operators (*crossover*, *mutation* and *selection* to name the basic ones) are applied to the individuals from the population in order to create the population of the next epoch. The genetic algorithm finishes either when the maximum number of epochs is reached, or when the satisfactory solution is found[1]. Genetic operator – *selection* is applied to the population to designate the individuals that will enter the "reproductive" process (which includes the operations of crossover and/or mutation). *Fitness function* is used to evaluate the quality and "goodness" of each solution, so that better ones have more chances to enter the reproductive process than poorer solutions, and leads to the emergence of the best solution to the given problem ("survival of the fittest"). In GP, individuals are represented as parse trees, where mutation and crossover operators are defined on the tree structures.

Mutation is performed by choosing a random node in the tree to be mutated. Sub-tree in the chosen node is deleted and a new one is created ("grown") randomly in the chosen mutation point.

[1] Other end criteria can be used as well, only the basic ones are mentioned in the text.

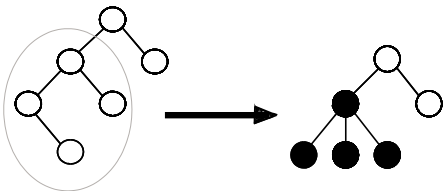

Fig. 1. Mutation of a tree

Crossover – Two candidate trees must be selected first for the crossover to be applied. Similarly to mutation, one random node is chosen in each parent. Then the sub-trees in the chosen nodes are recombined, i.e. exchanged between two parents in the corresponding places.

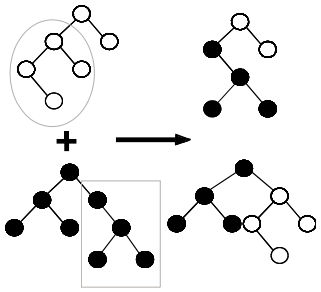

Fig. 2. Crossover on trees

Selection operator in GP does not differ from the one in genetic algorithms. This operator should ensure that the fitter individual in the population will have higher probability of entering the reproductive process. This means that the probability of selecting one individual for the reproduction is based on his evaluation, i.e. fitness function. One manner of implementing this function can be *Roulette Wheel procedure* (for more details about genetic algorithms and operators see [3] and [4], also [23]).

3 Cellular Framework

In cellular model of genetic algorithms each individual has spatial location and defined its (small) neighbourhood. An individual interacts only with its neighbourhood. Designed after cellular automata, this model is also composed of individuals in a regular spatial lattice, toroidal multidimensional net, where all the individuals have the same number of neighbours in the net. There are more possibilities for defining the neighbourhood, some of them that are common in 2-dimensional lattice, are 4-neighbourhood (von Neumann) and 8-neighbourhood (Moore). In the experiment, we will use the Moore model of the neighbourhood. Due to the definition of the neighbourhood, *selection* operator is significantly simplified

with no need to rank individuals in the population and applying *Roulette Wheel procedure* (see previous section of this paper).

Programme (pseudo) code for cellular model of GP (GA):

```
for every cell i do parallel
   generate_random_individual(c_i);
   evaluate(c_i);
end for
while not LastGeneration do
   for every cell i do parallel
      Probability:=random;
      if (Probability<Pr_Crossover) then
         c_j:=fittest_neighbour(c_i);
         x:=crossover(c_i, c_j);
         c_i:=best_fitness(x,y);
      else if (Probability<Pr_Mutation+Pr_Crossover) then
         c_i:=mutation(c_i);
      else
         c_i:=copy(c_i);
      end if
   end for
end while
```

Obviously, cellular model brings us new definition of selection operator, while mutation and crossover operators stay unchanged. On the other hand, genetic programming applies the standard genetic algorithm approach to the selection operator, and modifies "reproductive" operators. Therefore, combining GP and cellular model, we obtain a new hybrid of the GA, having the new qualities from the both frameworks.

Cellular and island models belong to the parallel genetic algorithms. The first model is inspired by the cellular automata and the behaviour of the biological processes in the nature, which are parallel. At the first look, what seems to be a small change of an selection operator, brings us interesting results and algorithm behaviour[2]. Although island model might be more suitable for parallel implementation, because of the smaller communication demands, I found the cellular model interesting and less examined parallel genetic approach, and decided to compare and implement it. Future research could compare the results of these two approaches in parallel genetic algorithms on the GP and the classification task.

Cellular automata framework for GP applied on the problem of classification can be found in the paper [10], which inspired this work as well. The authors of [10] claim that advantages of this model are handling well large populations, fast convergence leading to lower number of iterations and reduced execution time. This approach also avoids the problem of premature convergence of algorithm that occurs

[2] See results of research at http://www.ii.fmph.uniba.sk/~takaca/thesis/ where classical GA/GP approach to selection operator is compared with cellular approach, and where cellular model achieved significantly better results than the standard one

in some GP applications. In the cellular model of GP created information drifts and spreads in the population space. Discovered information of the fit individual slowly diffuses through the whole population, giving enough time to other individuals to develop their own different schemes. Diversity of the population is also preserved, because of the non-greedy search performed.

4 Classification Problem

Many companies collect vast amounts of data, however they fail to extract necessary information to support managerial decision-making. Knowledge discovery in databases comes to attention in the early nineties with the growing need to analyze data and turn it to the useful information. For example, in evaluating loan applications, by improving ability to predict bad loans, the company can substantially reduce its loan losses. In medical area, we could predict the probability of presence of the heart disease in a new patient by learning from historical data. Or in the industry field, reducing fabrication flaws of certain product can be achieved by processing the large quantity of data collected during the fabrication process.

Data mining is the search for relationships and global patterns “hidden” in the large quantity of data in databases. These systems can find meaningful relationships that might take years to find with conventional techniques. Data mining is an interdisciplinary field that uses methods from several research areas (above all machine learning, databases and statistics) to extract high-level knowledge from real-world datasets. It is an essential part of the broader process: Knowledge Discovery in Databases (KDD), which also includes several preprocessing methods for preparing data for data mining as well as postprocessing methods in order to refine and improve the discovered information.

Classification task is the one of the most studied in the area of data mining. The objective of this task is to predict the value (the class) of a user-specified goal attribute considering the values of the other attributes. In the other words, this process finds the mutual properties among a set of elements in a dataset, and classifies them into different classes, according to the classification rule (or procedure in some cases).

Desirable property of the discovered rule is its predicting power in assigning (predicting) the class of the new elements, for example, evaluating new loan applicant based on the data available about him. There are more methods used for testing of predicting ability. Task is simple when the dataset is already divided into the training and the test set. One technique commonly used for prediction testing when there is only one dataset (to test and train the algorithm) is *N-fold cross validation*. This method divides the dataset to N mutually exclusive sub-datasets of the same size. In each step of total N steps of the algorithm, different sub-dataset is used for the testing, and the rest of the sub-datasets for learning of the algorithm (i.e. for creating the classification rule). In the learning part of the algorithm, the goal attribute values are available, while in the testing part these values are used for evaluation of the predicting ability of the rule. The goal of the classification process is to process the training data first, then develop an accurate rule or a model for each class, based on

the values of predicting attributes, and after that to classify the future data. The prediction quality is verified on the test data.

For the sake of better understanding of the classification rules, commonly used representation is:

IF <some_conditions_are_satisfied> **THEN** <predicted_value_of_goal_attribute>

One simple illustration of possible classification rule could be:

IF (*Job*=”yes”) and (*UnpaidLoan*=”no”) **THEN** (*Credit*=”good”)

In this example predicting attributes are *Job* and *UnpaidLoan*, goal attribute is *Credit*, predicting attributes values are {“yes”, “no”} and goal attribute value (class) is “good”.

5 Motivation

There are several properties of GP and genetic algorithms in general, which make them more convenient for application in DM comparing to the other techniques. One of them is their robustness and ability to work on large and “noisy” datasets. While most of the classification algorithms apply greedy search of the solution space, GP performs global search, therefore coping well with attribute interaction problem. Thanks to variety of possible modifications and parallel approaches to genetic algorithms, scalability of these algorithms can be achieved. Together with robustness, these characteristics are of great importance in DM. Moreover, these algorithms have the high degree of autonomy that enables a discovery of knowledge previously unknown by the user.

However, a drawback of genetic algorithms is the necessity of frequent evaluation of individuals, i.e. possible solutions of the task, on the given dataset. If we have n individuals and m epochs of the algorithm, the number of evaluations of individuals will be $n*m$. If we consider that datasets in DM tend to have gigabytes and terabytes of data, evaluation of individual against the dataset is the most time consuming operation of the algorithm. We will try to address this problem by integrating our algorithm with databases, utilizing Structured Query Language (SQL). Another way to tackle the problem (which could be combined with previous) could be parallel implementation.

GP and genetic algorithms possess characteristics that make them easily parallelized. Parallel processing holds the key to extracting the maximum potential of knowledge discovery in databases [19]. Possible parallel approaches in GP are discussed in [14], [15] and [16].

Reason for selecting GP over genetic algorithms for applying to classification problem is their flexibility and possibility of straightforward representation of high-level rules. More diverse logical conditions (in the *IF* part) of the rule can be created by tree representation in contrast to fixed-length strings used by standard genetic algorithms. Manipulating and applying genetic operators on the functions we can solve the problem of attribute interaction. For example, some trees that are created

straightforward with GP would be developed significantly harder by the standard genetic algorithms.

The problem of the comprehensibility of the discovered rules could be addressed by adjusting properly the fitness function. Beside the accuracy of the classification rule on the dataset, the function should also take into the account a size of the tree, penalizing the trees with the high number of nodes.

6 Closure Problem

In the standard relational databases, we usually encounter two general types of attributes that are used for DM: qualitative and numerical. Some DM algorithms work only with qualitative attributes, so numerical values must be divided into categories. GP can deal with any combination of types. However, the task is much simpler if only one type is used. If we had, for example, only numerical data, then a GP individual would encode numerical function, and the evaluation of the function (individual) against one example from the dataset would return a numerical value. Considering the classification task, we can select intervals for each class and the obtained numerical value would determine the class to which example from the dataset should belong to.

One of the possibilities of combining two types of attributes in the DM algorithm is actually by using so-called atomic representation of trees [8]. This representation can be used with only one type of attributes as well. The idea is to use Boolean functions for all inner nodes, while leaves are represented as functions that return Boolean value:

$$Operator\ (Attribute, AttributeValue) \quad (1)$$

More details about this representation can be found in [8]. Moreover, in the next experiment this representation is used and therefore will be described here in more detail (it slightly differs from the one described in [8]). The advantage of atomic representation is an uncomplicated solution of the closure problem and also production of the rules that are more comprehensible to the user. On the other hand, its drawback is that good quality of the attribute interaction is not being exploited.

Other option is using some kind of constrained syntax GP. In some datasets, interaction between any combinations of attributes does not need to create meaningful construction. Can we suppose that just evolutionary process will let survive only the combinations that will make sense? Better solutions could be achieved by creating domain-related semantics, defining which attributes can interact. However, this would certainly require deeper understanding of a given dataset.

7 Fitness Function

Selection of the fitness function clearly depends on the type of the task and desired characteristics of the discovered solution. In DM tasks, discovered knowledge should have following characteristics: accuracy, comprehensibility and interestingness. First of all, we will focus on accuracy.

In classification problem, our objective is simple: creation of one rule that will classify one class[3]. The evaluation function can therefore simply represent the percent of correctly classified elements from the dataset (in our case we are maximizing the fitness function). Some datasets also have a cost matrix that defines different penalties for the incorrect classification of different attributes (values). This cost can also be included in the function if it is specified. Suppose that there are 2 classes (*C* and *not C*) and the following cost matrix (see Table I).

Table I. Cost matrix for the classification task

		Predicted Class	
		C	*Not C*
Actual Class	*C*	*0*	*Cost(C, not C)*
	not C	*Cost(not C, C)*	*0*

We will illustrate the classes *C* and *not C* on the dataset example of loan applicants. Class *C* will represent: a good applicant for a loan, and the class *not C* will symbolize a bad loan applicants. Dataset contains history data about the clients that in the past applied for a loan, and were either granted a loan (class *C*) or not (class *not C*).

From the Table I, *Cost(not C, C) > Cost(C, not C)* would mean that it is more important to classify correctly elements that actually belong to class *not C* (i.e., bad applications for a loan), then the elements that actually belong to class *C*.

By evaluating the discovered rule ***IF*** *Condition* ***THEN*** *Class=C* (i.e. we are evaluating the *Condition* part) against the dataset, four numbers will be obtained representing the following values: *TP*, *FP*, *FN* and *TN*. *TP* is the number of the Truly Positive elements, i.e. number of elements satisfying *Condition* that actually belong to the class *C*. *FP* is the number of the False Positive elements, i.e. the number of elements satisfying *Condition* that actually do not belong to the class *C*. *FN* is the number of the False Negative elements and *TN* is the number of the Truly Negative elements.

Table II. Matrix of results of the rule evaluation against the dataset

		Predicted Class	
		C	*Not C*
Actual Class	*C*	*TP*	*FN*
	not C	*FP*	*TN*

To evaluate the results obtained by the discovered rule, we will use the cost function *CF* (2):

[3] There are other possibilities as well, like creating decisions trees by genetic programming [8], or more rules for covering one class [11]

$$CF = \frac{FP \cdot Cost(not\ C, C) + FN \cdot Cost(C, not\ C)}{TP + FN + FP + TN} \quad (2)$$

We can suppose that $Cost(x, y) \geq 1$, where x, y are from $\{C, not\ C\}$. Further, we define *Count(C)*, representing the number of elements from the dataset belonging to class *C*, and *Count(not C)*, representing the number of elements belonging to the class *not C*. Clearly

$$Count(C) = TP + FN\ ,\ \text{and}\quad Count(not\ C) = FP + TN \quad (3)$$

If a classification rule is the perfect classifier, its cost will be 0 ($CF_{\min} = 0$). On the other side, if all the examples are classified incorrectly, we will have TP = 0 and TN = 0, so

$$CF_{\max} = \frac{Count(not\ C) \cdot Cost(not\ C, C) + Count(C) \cdot Cost(C, not\ C)}{Count(C) + Count(not\ C)} \quad (4)$$

Since we want to maximize the fitness function, which should favour the rules with the lowest cost, we can obtain it as function complementary to *CF*, i.e. while we maximize the fitness function, the *CF* will be minimized, which was our objective. Therefore we can define function E_1 for fitness evaluation:

$$E_1 = CF_{\max} - CF \quad (5)$$

Applying the equations (2) and (4) to equation (5), and using the definitions from (3), we can obtain the following formula:

$$E_1 = \frac{TN \cdot Cost(not\ C, C) + TP \cdot Cost(C, not\ C)}{TP + FN + FP + TN} \quad (6)$$

Function E_1 will help us to discover the classification rule with the high accuracy. Nevertheless, we did not consider the demand for comprehensibility and interestingness of the found rules. In order to ensure that algorithm favours the smaller and therefore more comprehensible rules, we can construct a simple function:

$$E_2 = \frac{1}{Size} \quad (7)$$

Where the *Size* could simply be the number of the nodes in the tree representing the *Condition*, for instance. The final function to evaluate each individual from a population could then be (8), where w_1 and w_2 would be the weights defined by the user, balancing between the demands for simplicity of the rules and their accuracy.

$$FitnessFun = w_1 \cdot E_1 + w_2 \cdot E_2 \quad (8)$$

8 Integration with Databases

To achieve scalability and efficiency of DM algorithm, Structured Query Language (SQL) can be utilized. SQL is a powerful tool for relational databases and can significantly improve the performance of our algorithm. Moreover, there is a possibility of using massively parallel SQL servers. If we have in mind that evaluation of the individuals in a population is the operation that consumes most of the time used by the GP algorithm applied to DM, reduction of the evaluation time would have a significant impact on overall performance of the application. Further benefits of integrating SQL are avoiding physical access to the data. We can just send the queries that need to be evaluated to the database server and receive only the results of the evaluation. In this way genetic DM application can benefit from the control of security of the data already provided by relational database management system. Additionally, it simplifies the problem of accessing the dataset, because SQL is standardized and drivers for all types of databases are available, enabling simple adaptation of DM algorithm to whichever database is necessary.

Papers [13] and [26] describe different GP frameworks for the evolution of SQL queries and parallel database server.

8.1 Classification Rule Evaluation

Consider the task of classification with the two classes: *C* and *not C*, where objective is to evolve a classification rule of the form (9).

$$IF\ Condition\ THEN\ GoalAttribute=C \tag{9}$$

Since there are only two classes, we can evolve only one function: *Condition* that should return *true* if the evaluated element from the dataset belongs to the class *C*, and *false* in the case that the element belongs to the class *not C*. *Condition* will be represented as a parse tree and GP will be used to evolve the tree that classifies correctly the highest number of elements of the given dataset. To evaluate each individual, a query from the Figure 1 will be used.

```
SELECT GoalAttribute, Count(*)

FROM MyRelation

WHERE Condition

GROUP BY GoalAttribute
```

Fig. 3. SQL query for evaluation of an individual (*Condition*)

GoalAttribute will represent attribute from the dataset that has two possible values: class *C* and class *not C*. Task is simplified so that entire dataset is located in one database relation *MyRelation*. Result of query evaluation will be Table III.

Table III The result of evaluating SQL query for one individual (*Condition*)

GoalAttribute Value (Class)	*Count*
C	*Count(Condition, C)*
Not C	*Count(Condition, not C)*

If Table III is compared with Table II hereunder relation (10) can be observed.

$$Count(Condition, C) = TP$$
$$Count(Condition, not\ C) = FP \quad (10)$$

In order to evaluate the fitness of each individual using the functions described in the chapter 7 we need to have the exact values of *TP*, *TN*, *FP* and *FN*, and also *Count(C)* and *Count(not C)*. If the last two numbers are not given in advance, we can obtain them straightforward by evaluating modified SQL query from the Figure 3, where condition ("WHERE Condition" part) is left out.

Now we have acquired further values:

$$Count(C) = TP + FN$$
$$Count(not\ C) = FP + TN \quad (11)$$

As we already have the values of *TP* and *FP*, by the simple calculation we can have the missing numbers *FN* and *TN* as well. Finally, we can evaluate the fitness function of our individual, using the equation (8).

The problem gets more complicated when there are more then two classes in the classification task. Let us suppose that there are *n* classes, where $n>2$: $\{C_1,\ldots, C_n\}$, which means that the *GoalAttribute* has *n* possible values. We can choose one of the options:

- Either the algorithm is easily adapted[4] to run for two classes (like we described above) *n-1* times, where two classification classes will be C_i and *not* C_i, $i=1,..,n-1$. Finally each of these n-1 algorithms will discover one best rule for one class from $\{C_1,\ldots, C_{n-1}\}$. Data elements that are not "covered" by any of the found rules will belong to default class C_n.
- Another approach is to try to create classification rules for *n-1* classes with one run of evolutionary algorithm. Again, class C_n can be considered as default class. This time evolutionary DM algorithm must evolve simultaneously *n-1* classification rules, one per each (different) class. These rules will be the results of the algorithm. In order to induce evolution of individuals that predict different classes, it is beneficial to use a method to enforce creation of the rules for all n-1 classes and prevent from losing classification rules for any of these classes. For instance, a type of elitist algorithm can be applied to copy the best individuals for each class to the next generation without the changes, ensuring a good representative of every class in each population.

[4] There will be a small difference in evaluating the fitness from the SQL query, since Table III will have more rows depending on the number of classes

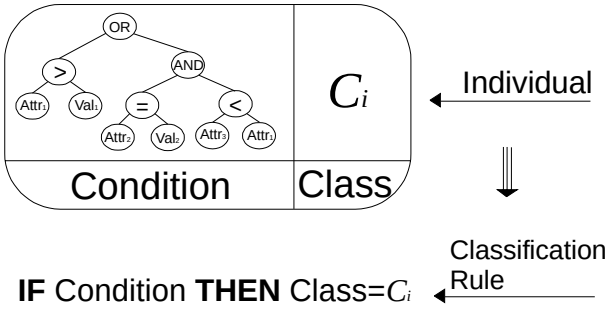

Fig. 4. GP algorithm individual and the corresponding classification rule

Using the analogy with classification problem with two classes, we can evaluate corresponding fitness value for every individual in the population. Evidently, in this more complicated classification task, every individual in the population, beside the tree representing the *Condition*, carries also the value of the class (Figure 2). When evaluating an individual, we can assign to the given individual the class C_k that has the highest value of *Count*:

$$Count(Condition, C_k) > Count(Condition, C_i),\ \forall i \neq k \qquad (12)$$

8.2 Encoding

General encoding of individuals is already described before, however we did not mention the *Condition* encoding yet. Since we are using GP framework, as expected, *Condition* will be represented as a parse tree. The terminal set includes names of predicting attributes and values of their corresponding domains. Nonterminals, represented as inner nodes of the tree, can be Boolean operators: {*AND, OR, NOT*} and comparison operators: {>, >=, <, <=, #} for instance. Also, numerical operators {+, -, *, /} can be used on a given numerical attributes. Moreover, other operators supported by SQL can be considered as nonterminals as well, but one should first consider the pros and cons of applying the new functions and what effect on the solution could they have. The important issue is the problem of closure. If we consider atomic representation, inner nodes corresponding to the nonterminals will include only operators {*AND, OR, NOT*}. Leaves of the *Condition* tree will be one of the two types: numerical or qualitative, and will have the form as in (1).

In the case of qualitative attributes: *Operator* is from {=, #}, *Attribute* is the name of one predicting qualitative attribute and *AttributeValue* is the constant from the *Attribute*'s domain, i.e. one of the n (n>0) values that are defined and found in the dataset for the given attribute.

For the numerical attributes, *Attribute* is defined in the same way as for the qualitative leaf. On the other hand, *AttributeValue* is randomly selected number from the interval [*min(Attribute), max(Attribute)*]. More operators are applicable on numerical attributes, but having more operators included does not necessarily mean better results of the algorithm. Even though we can choose all the comparison operators from the set {>, >=, <, <=, =, #}, the use of {=, #} would mean comparing

the given numerical attribute with the constant from that attribute domain on equality (inequality). Obviously, the expression would have very low informational value and including operators {=, #} would just unnecessary increase the search space. In order to decrease the search space, only numerical operators: {>, <} were included in the experiment.

9 Experiment

In this section, an experiment and the results obtained by applying GP to the classification are presented. The cellular model of GP is implemented, where individuals are mapped into the 2-dimensional array. *Crossover* is not applied inside of the leaves. If the *crossover* creates an individual that has the larger depth than the maximal depth allowed, the operation is considered unsuccessful and a fitter parent is copied to the new generation. Implemented *mutation* operator has two types: (a) a mutation of the whole sub-tree, and (b) inside of a chosen leaf. In the first case (a) we have the standard *mutation* defined for the genetic algorithms, and in the latter (b) we can modify the leaf in the three possible ways. 1) Either another operator of the corresponding type (qualitative or numerical) can be randomly selected, or 2) a different attribute value can be randomly chosen among the possible values or 3) a new attribute can be generated leading to a random creation of the a corresponding operator and an attribute value as well. When performing the mutation, decision about the mutation type (a or b case) to be carried out is made by creating a random number. Algorithm accepts both qualitative (discrete, nominal) and numerical (continuous) data.

The following parameters (obtained in the previous testing) were used in the experiment: the crossover probability: 0.7, the mutation probability: 0.2, the reproduction probability: 0.1, population size: 20 x 20 (400), number of epochs: 200, maximum depth of trees allowed: 7, branching factor: 2. Number of classes is 2 and attribute types are qualitative and numerical. SQL queries are used for evaluation of individuals, and for evaluating the fitness equations (6), (7) and (8) were used, where w_1 is 1 and w_2 is 0.05.

Three datasets from the STATLOG project[5] have been used: German Credit, Australian Credit and Heart Disease, but due to a lack of space here are presented only results with the first dataset, which is the largest. German Credit dataset contains historical data about loan applicants and their credibility, having 1000 records divided in 2 classes (*Good* and *Bad Credit*). First class has 700 elements, second 300. Number of qualitative attributes is 13 and continuous 7. Cost matrix from Table I is used, where the class *C* is *Good Credit*, and clearly the class *not C* represents *Bad Credit*, *Cost(C, not C)* = 1 and *Cost(not C, C)* = 5. For evaluation of prediction ability of algorithm 10-fold cross validation is used and equation (2).

The program was developed under the operating system MS Windows 98 and has been implemented in Borland Delphi 6, while datasets are stored in MS Access 97

[5] http://www.liacc.up.pt/ML/StatLog/

database. For accessing the data, implemented program uses SQL queries and the ActiveX Data Objects (ADO) component in Delphi[6].

9.1 Results

Obtained results of the cellular GP are compared to the results of the best performing algorithms (of more then 20 algorithms that participated) in the STATLOG project. In the brackets are shown the ordinal numbers of each algorithm's performance on the German Credit dataset in the project. Bellow are presented the results of the well known algorithms that participated in the project, such are K-nearest neighbours (KNN), C4.5 - the most famous decision tree based classification method, or BackProp, which is short from back propagation method for learning using neural networks. Complete results of the performance of all algorithms in the project can be found at [9].

Table IV. German Credit dataset – results

Algorithm	Average Train Cost	Average Test Cost
Cellular GP	0.472	0.539
Discrim[7] (1.)	0.509	0.535
LogDisc (2.)	0.499	0.538
Castle (3.)	0.582	0.583
Alloc80 (4.)	0.597	0.584
KNN (10.)	0	0.694
BackProp (14.)	0.446	0.772
BayesTree (15.)	0.126	0.778
C4.5 (22.)	0.64	0.985

Even though we used relatively small population of 400 individuals (20 x 20) and only 200 epochs, the results in the Table IV show very good performance of the cellular GP algorithm. Obtained results are comparable to the results of the best performing algorithms from the StatLog project and significantly better then some known algorithms like C4.5 or the back propagation algorithm with neural networks, which is a promising fact.

In the figure bellow we can observe learning curve of the cellular GP algorithm in the 7th run of the 10-fold testing. We can observe one interesting characteristic: the fitness value has the tendency to be a non-decreasing function, i.e. the best individual is either kept or is replaced by the individual with the higher fitness value. Mentioned behaviour can be noticed on the other results charts of this algorithm representing the fitness value of the best individual in the population. Note that we did not use any elitist model that would transfer automatically few best individuals to the next epoch.

[6] Developed application is available at http://www.ii.fmph.uniba.sk/~takaca/thesis

[7] The best performing algorithm in StatLog project on the German Credit dataset, a statistical classifier

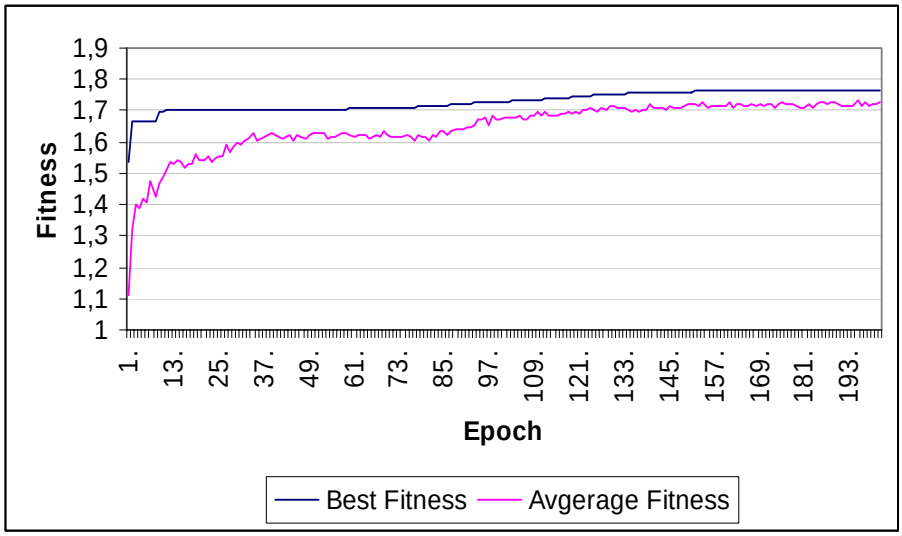

Fig. 5. Learning Results of Cellular GP in Classification Task on German Credit Dataset

10 Conclusion

The possibility of applying Genetic Programming in Data Mining was surveyed, examining its integration with databases and possibilities of parallelization. Cellular model of GP that uses SQL queries was applied to the classification task, and the obtained results were compared with the other machine learning algorithms.

Implemented cellular GP algorithm achieved promising results on the used datasets. GP was suitable and convenient model for encoding the classification rules, because of the tree representation. Cellular approach seems to be an interesting genetic parallel model with the certain advantages like: faster convergence and better results, simplified selection method with less calculation, prevention of loss of the good solutions[8] and a structure convenient for parallel implementation.

However, further investigation and testing should be performed, with larger datasets and employing the parallel SQL servers. Furthermore, comparison of cellular model with other parallel GA's on the similar problem could bring interesting results.

[8] Non-decreasing property of the fitness function in a cellular model has been observed in the results

References

[1] Agrawal, R., Imielinski, T., Swami, A.: Database Mining: A Performance Perspective. IEEE Trans. Knowledge and Data Eng., vol. 5, no. 6, pp. 914-925, Dec. 1993.

[2] Araujo, D.L.A., Lopes, H.S., and Freitas, A.A.: *Rule Discovery with a Parallel Genetic Algorithm*. Proc. 2000 Genetic and Evolutionary Computation (GECCO-2000) Workshop Program, 89-92. Las Vegas, NV, USA. July 2000.

[3] Bäck, T., Fogel, D. B., Michalevicz, Z.: *Evolutionary Computation 1 – Basic Algorithms and Operators*, Institute of Physics Publishing, Bristol and Philadelphia, 2000.

[4] Bäck, T., Fogel, D. B., Michalevicz, Z.: *Evolutionary Computation 2 – Advanced Algorithms and Operatorss*, Institute of Physics Publishing, Bristol and Philadelphia, 2000.

[5] Bennett, F.H., Koza, J.R., Keane, M.A., Andr'e, D.: *Genetic Programming: Biologically Inspired Computation that Exhibits Creativity in Solving Non-Trivial Problems*. In AISB'99 Symposium on AI and Scientific Creativity, pages 29--38, Edinburgh, Scotland, April 1999.

[6] Berka, P.: *Metody dobyvani znalosi z databazi.* Data Mining – Jak z vasich dat vytezit maximum: Sbornik k seminarum (Praha, Bratislava, podzim 2002), 22-34. StatSoft CR s.r.o., Praha, Czech Republic, 2002.

[7] Chen, M.S., Han, J., Yu, P.S.: *Data Mining: An Overview from Database Perspective*, IEEE Trans. Knowledge and Data Eng., Vol. 8, No. 6, Dec. 1996, pp. 866-883.

[8] Eggermont, J., Eiben, A.E., van Hemert, J.I.: *A Comparison of Genetic Programming Variants for Data Classification*. Proc. Third International Symposium on Intelligent Data Analysis (IDA-99). Amsterdam, The Netherlands. August 1999. Springer-Verlag, Berlin.

[9] Esprit Project 5170 StatLog (1991-94): http://www.niaad.liacc.up.pt/statlog/

[10] Folino G., Pizzuti C., Spezzano G.: *A Cellular Genetic Programming Approach to Classification*, Proc. Of the Genetic and Evolutionary Computation Conference GECCO99 , Morgan Kaufmann, pp. 1015-1020, Orlando, Florida, July 13-17, 1999

[11] Freitas, A.A.: *A Survey of Evolutionary Algorithms for Data Mining and Knowledge Discovery*. To appear in: A. Ghosh and S. Tsutsui. (Eds.) Advances in Evolutionary Computation. Springer-Verlag, 2002.

[12] Freitas, A.A.: *Evolutionary Computation*. To appear in: J. Zytkow and W. Klosgen. (Eds.) Handbook of Data Mining and Knowledge Discovery. Oxford University Press, 2001.

[13] Freitas, A.A.: *A Genetic Programming Framework for Two Data Mining Tasks: Classification and Generalized Rule Induction*. Genetic Programming 1997: Proc. 2nd Annual Conf. (Stanford University, July 1997), 96-101. Morgan Kaufmann, 1997.

[14] Freitas, A.A.: *A Survey of Parallel Data Mining*. Proc. 2nd Int. Conf. on the Practical Applications of Knowledge Discovery and Data Mining, 287-300. London: The Practical Application Company, Mar. 1998.

[15] Freitas, A.A., Lavington, S.H.: *Parallel Data Mining for Very Large Relational Databases*. In: H. Liddel et al. (Ed.) LNCS 1067: Proc. Int. Conf. on High-Performance Computing and Networking (HPCN-96, Brussels, Belgium, Apr./96), 158-163. Springer-Verlag, 1996.
[16] Freitas, A.A., Lavington, S.H.: *A Framework for Data-Parallel Knowledge Discovery in Databases*. IEE Colloquium on Knowledge Discovery and Data Mining. Digest No. 96/198, pp.6/1-6/4. London: IEE, Oct./96.
[17] Gordon, V.S., Whitley, D.: *Serial and Parallel Genetic Algorithms as Function Optimizers* , in: S. Forrest, (Ed.), Fifth International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, 1993, pp. 177--183.
[18] Han, J., Kamber, M.: *Data Mining: Concepts and Techniques.* Morgan Kaufmann Publishers, San Francisco, USA, 2001.
[19] Hedberg, S.R.: *Parallelism Speeds Data Mining*. (Industrial Spotlight) IEEE Parallel & DistributedTechnology, Winter 1995, 3-6.
[20] Holsheimer, M., Kersten, M., Mannila, H., Toivonen, H.: *A Perspective on Databases and Data Mining*. In Proc. of 1st Intl. Conf. on Knowledge Discovery and Data Mining (KDD), August 1995.
[21] Koza, J.R.: *Future Work and Practical Applications of Genetic Programming*. In Handbook of Evolutionary Computation, page H1.1:3. IOP Publishing Ltd and Oxford University Press, 1997.
[22] Koza, J.R.: *Concept Formation and Decision Tree Induction Using the Genetic Programming Paradigm.* Parallel Problem Solving from Nature --- Proceedings of the first Workshop, PPSN 1, volume 496 of Lecture notes in computer Science, pp. 124-128, Springer-Verlag, 1991.
[23] Kvasnicka, V., Pospichal, J., Tino, P.: *Evolucne algoritmy.* Vydavatelstvo STU, Bratislava, Slovakia, 2000.
[24] Mannila, H.: *Methods and Problems in Data Mining*. In Proc. of ICDT, Delphi, Greece, January 1997.
[25] Watson, R.T.: *Data Management: Databases and Organizations (Third edition).* John Wiley & Sons, Inc., New York, NY, USA, 2001.
[26] Watson, T., Rakowski, T.: *Data Mining with an Evolving Population of Database Queries.* p. 169-174, In: Proceedings of MENDEL'95 - the International Conference on Genetic Algorithms, Brno (CR), Sept. 26-28, 1995.
[27] Whigham, P.A.: *Grammatically-based Genetic Programming.* Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications (Conference proceedings), 1995, p. 33-41.
[28] Whitley, D.: *A Genetic Algorithm Tutorial*, Statistics and Computing 1994, Volume 4, pp. 65-85.