Int. J. Appl. Math. Comput. Sci., 2007, Vol. 17, No. 4, 539–547
DOI: 10.2478/v10006-007-0044-x

amcs

REAL–VALUED GCS CLASSIFIER SYSTEM

ŁUKASZ CIELECKI, OLGIERD UNOLD

Institute of Computer Engineering, Control and Robotics, Wrocław University of Technology
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland
e-mail: {lukasz.cielecki,olgierd.unold}@pwr.wroc.pl

Learning Classifier Systems (LCSs) have gained increasing interest in the genetic and evolutionary computation literature. Many real-world problems are not conveniently expressed using the ternary representation typically used by LCSs and for such problems an interval-based representation is preferable. A new model of LCSs is introduced to classify real-valued data. The approach applies the continous-valued context-free grammar-based system GCS. In order to handle data effectively, the terminal rules were replaced by the so-called environment probing rules. The rGCS model was tested on the checkerboard problem.

Keywords: learning classifier systems, GCS, GAs, grammatical inference, context-free grammar.

1. Introduction

A Learning Classifier System (LCS) is an evolutionary algorithm that operates on a population comprised of rules referred to as the rule set, and these rules are used to attempt to classify a situation. The first learning classifier system was created by Holland (1976) shortly after he created Genetic Algorithms (GAs) (Holland, 1975). Many real-world problems are not conveniently expressed using the ternary representation typically used by LCSs (true, false, and the "don't care" symbol). To overcome this limitation, Wilson (2000) introduced a real-valued XCS classifier system for problems which can be defined by a vector of bounded continuous real-coded variables.

We propose a new model of LCS (the so-called rGCS) that makes it possible to represent continuous-valued inputs. The rGCS is an extension of the Grammar-based Classifier System (GCS) introduced by Unold (2005a) in which the knowledge about the solved problem is represented by a Context-Free Grammar (CFG) in Chomsky normal form productions. An integer-valued representation (in fact, the set of letters a-z) is used with the GCS. The GCS is described in detail in (Unold, 2005b; Unold and Cielecki, 2005a). This article extends (Cielecki and Unold, 2007) by examining the influence of the training set size on the grammar competence and behavior of the rGCS with initial knowledge.

The remainder of this paper is organized as follows: Section 2 describes the generic LCS. The next section contains the framework of the GCS. Section 4 introduces the new rGCS—an extended system prepared to work with real-valued inputs. Section 5 illustrates our experiments with the checkerboard problem, and the last one summarizes the paper and makes plans for the future.

2. Learning Classifier Systems

A learning classifier system, introduced by Holland (1976), learns by interacting with an environment from which it receives feedback in the form of a numerical reward. Learning is achieved by trying to maximize the amount of reward received. There are many models of LCSs and many ways of defining what a learning classifier system is. All LCS models, more or less, comprise four main components (Fig. 1):

- a finite population of condition-action rules (classifiers), which represent the current knowledge about the system,
- the performance component, which governs the interaction with the environment,
- the reinforcement component, called the credit assignment component, which distributes the reward received from the environment to the classifiers responsible for the rewards obtained,
- the discovery component, responsible for discove-

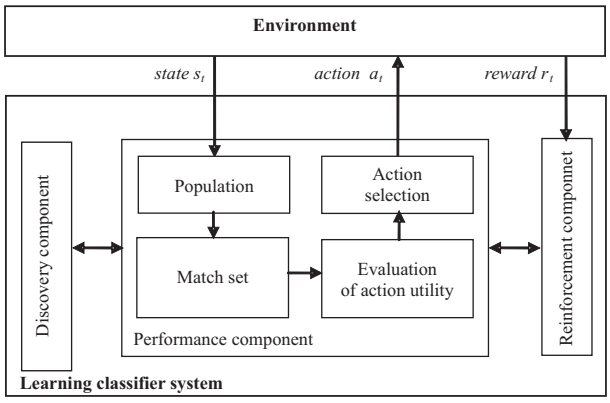

Fig. 1. Architecture of the learning classifier system (Holmes *et al.,* 2002).

ring of better rules and improving existing ones through a genetic algorithm.

Classifiers have two associated measures: prediction and fitness. Prediction estimates the classifier utility in terms of the amount of reward that the system will receive if the classifier is used. Fitness estimates the quality of the information about the problem that the classifier conveys, and it is exploited by the discovery component to guided evolution. A high fitness means that the classifier conveys good information about the problem and therefore it should be reproduced more through the genetic algorithm. A low fitness means that the classifier conveys little or no good information about the problem and therefore should reproduce less.

At each discrete time step t the LCS receives as the input the current state of the environment s_t and builds a match set containing the classifiers in the population whose condition matches the current state. Then, the system evaluates the utility of the actions appearing in the match set; an action a_t is selected from those in the match set according to some criterion, and sent to the environment to be performed. Depending on the current state s_t and on the consequences of the action a_t, the system eventually receives a reward r_t. The reinforcement component distributes the reward r_t among the classifiers accounting for the incoming rewards. This can be implemented either with an algorithm specifically designed for the learning classifier systems (e.g. a bucket brigade algorithm (Holland, 1986)) or with an algorithm inspired by traditional reinforcement learning methods (e.g. the modification of Q-learning (Wilson, 1995)). On a regular basis the discovery component (a genetic algorithm) randomly selects, with the probability proportional to their fitness, two classifiers from the population. It applies crossover and mutation generating two new classifiers.

The environment defines the target task. For instance, in autonomous robotics the environment corresponds roughly to the robot's physical surroundings and the goal of learning is to learn a certain behavior (Katagami and Yamada, 2000). In classification problems, the environment provides a set of preclassified examples. Each example is described by a vector of attributes and a class label. The goal of learning is to evolve rules that can be used to classify previously unseen examples with high accuracy (Holmes *et al.*, 2002; Unold and Dabrowski, 2003). In computational economics, the environment represents a market and the goal of learning is to make profits (Judd and Tesfation, 2005).

For many years the research on LCSs was done on Holland's classifier system. All implementations shared more or less the same features which can be summarized as follows: (i) some form of a bucket brigade algorithm was used to distribute the rewards, (ii) evolution was triggered by the strength parameters of classifiers, (iii) the internal message list was used to keep track of the past input (Lanzi and Riolo, 2000).

In recent years new models of Holland's system have been developed. Among others, two models appear particularly worth mentioning. The XCS classifier system (Wilson, 1995) uses Q-learning to distribute the reward to classifiers, instead of the bucket brigade algorithm. The genetic algorithm acts in environmental niches rather than on the whole population. The most important thing is that the fitness of classifiers is based on the accuracy of classifier predictions, instead of the prediction itself. Stolzmann's ACS (Stolzmann, 2000) differs greatly from other LCS models in that the ACS learns not only how to perform a certain task, but also an internal model of the task dynamics. In ACSs, classifiers are not simple condition-action rules, but they are extended by an effect part, which is used to anticipate the environmental state.

3. GCS

The GCS operates similarly to the classic LCS but it differs from it in (i) the representation of the classifier population, (ii) the scheme of classifiers' matching to the environmental state, (iii) methods of exploring new classifiers.

The population of classifiers has the form of a context-free grammar rule set in a Chomsky Normal Form (CNF). This is actually not a limitation because every context-free grammar can be transformed into an equivalent CNF. The Chomsky normal form allows only for production rules in the form of $A \rightarrow a$ or $A \rightarrow BC$, where A, B, C are non-terminal symbols and a is a terminal symbol. The first rule is an instance of the *terminal rewriting rule*. These are not affected by the GA, and are generated automatically as the system meets an unknown (new) terminal symbol. The left-hand side of a rule plays the role of the classifier's action while the right-hand side is the classifier's condition. The system evolves only one grammar according to the so-called Michigan approach. In this

approach each individual classifier (or a grammar rule in the GCS) is subjected to GA operations. All classifiers (rules) form a population of evolving individuals. In each cycle a fitness calculating algorithm evaluates the value (an adaptation) of each classifier and a discovery component operates only on a single classifier.

An automatic learning context-free grammar is realized with the so-called grammatical inference from text (Gold, 1967). According to this technique, the system learns using a training set that in this case consists of sentences both syntactically correct and incorrect. A grammar which accepts correct sentences and rejects incorrect ones is able to classify unseen sentences from a test set. The Cocke-Younger-Kasami (CYK) parser, which operates in $\Theta(n^3)$ time (Younger, 1967), is used to parse sentences from the sets.

The environment of a classifier system is substituted by an array of CYK parsers. The classifier system matches the rules according to the current environmental state (the state of parsing) and generates an action (or a set of actions in a GCS) pushing the parsing process toward the complete derivation of the sentence analyzed.

The discovery component in a GCS is extended in comparison with a standard LCS. In some cases a "covering" procedure may occur, adding some useful rules to the system. It adds productions that make it possible to continue parsing in the current system state. This feature utilizes, i.e., the fact that accepting 2-length sentences requires a separate, designated rule in the grammar in the CNF.

Apart from "covering", a GA also explores the space by looking for new, better rules. The first GCS implementation used a simple rule fitness calculation algorithm which appreciated the ones commonly used in correct recognitions. Later implementations introduced the "fertility" technique, which made the rule fitness dependent on the amount of the descendant rules (in the sentence derivation tree) (Unold, 2005b; Unold and Cielecki, 2005a). This approach is particularly useful since in a GCS population individuals must cooperate to parse sentences successfully. Appreciating linked rules, we help to preserve the structure of the evolved grammar. In both techniques the classifiers used in parsing positive examples gain the highest fitness values, unused classifiers are placed in the middle, while the classifiers that parse negative examples gain the lowest possible fitness values.

4. rGCS

4.1. Overview of the rGCS. Despite the fact that the GCS is able to solve grammar induction problems effectively, the area of its usage is strongly limited. Due to the nature of the tagged input data, most tasks it is employed for are connected with formal or natural languages. To overcome that limitation, we created an extension of the GCS that accepts any input data stored as a vector of real values.

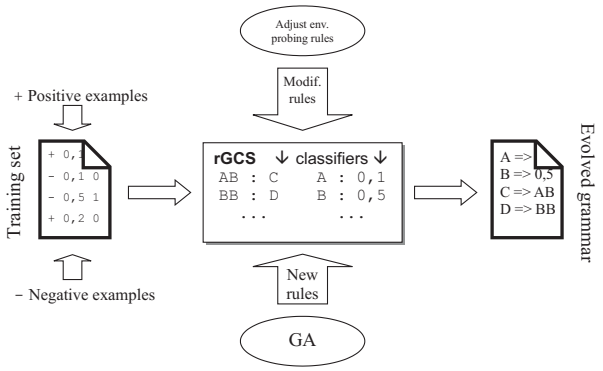

Fig. 2. rGCS block diagram.

The rGCS exploits the main idea of the classic GCS. The CYK table is the environment and the area where the rGCS operates. The learning process is divided into the cycles. During each cycle the evolved grammar is tested against each example of the train set, and then new rules are evolved or existing ones are modified and another cycle begins (Fig. 2). The rGCS employs two different kinds of rules—environment probing and regular ones—which are used in different phases of the learning process (Fig. 3).

4.2. Environment Probing Rules. The structure of the rules that are used in the very first row of the CYK table during parsing is the main difference between the classic GCS and the rGCS. Since their role is to sense the input data and then to launch the CYK process, we called them environment probing rules. In a classic GCS the environment situation consisted of input string terminals so these rules were called appropriately the terminal rewriting rules. Now in the rGCS the input data (an environmental situation) is formed by the vector of real numbers that may describe various kinds of data (Fig. 4). Each rule has the form

$$A \rightarrow f, \tag{1}$$

where A is a non-terminal symbol and f is a real number.

Here f is used during the matching process and the non-terminal A is to be put into the first row of the CYK table. Additionally, a special environment probing rule may be used—the most general one that accepts every single input value. This one is called the wildcard (the "don't care" symbol) and has the form

$$F \rightarrow *, \tag{2}$$

where F is the non-terminal symbol chosen to play the role of a wildcard (every time it appears in the CYK table, this means: put any value here) and the asterisk means that any real number is accepted here.

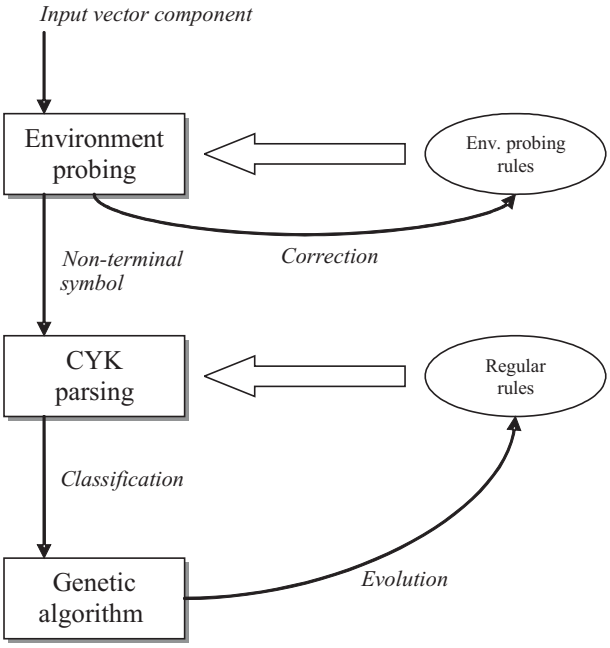

Fig. 3. rGCS, environment probing and regular rules usage.

4.3. Regular Grammar Rules. These rules are identical with the ones used in the classic GCS. They are used in the CYK parsing process and the GA phase. They are in the form of

$$A \rightarrow BC, \tag{3}$$

where A, B and C are non-terminal symbols.

4.4. Generating the Rules. Every time a new experiment is started, a set of random rules is generated. It contains a specified number of environment probing rules and regular grammar rules. All non-terminals in the set are from the range limited by a system parameter. In environment probing rules the system keeps equal numbers of

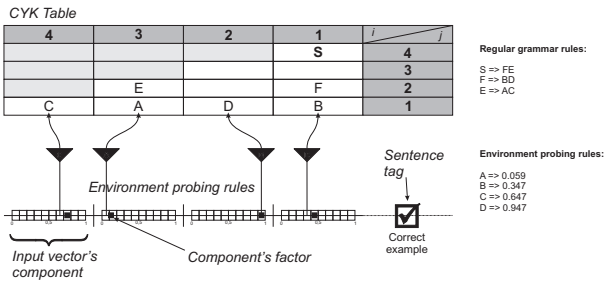

Fig. 4. Main idea of the rGCS. Environment probing rules and their cooperation with the CYK parsing procedure. A symbol in a CYK table's cell means that the particular non-terminal derives the substring of length j starting from position i, where i and j are cell indices.

various non-terminal symbols in the rules. Real values may be from the range determined by the minimum and maximum values of the input data to speed up the learning process, but this is not necessary.

4.5. Matching Phase. Environment Probing Rules.

Scheme 1. First, a list of distances between the elements of the input vector (real numbers) and each rule's real number is created. Then all values from the list are scaled using the equation

$$df_i = 1 - \left(\frac{dist_i}{maxdist} \right), \tag{4}$$

where df_i is the factor calculated for rule i (the distance factor), $dist_i$ is the distance of rule i and 'maxdist' is the maximum distance value (the distance value of the most distant rule).

The most distant rule receives factor 0 and the rule that is located exactly at the input vector's value receives 1. In the next step the rules are sorted from the nearest one to the most distant one. Finally, the equation

$$p_i = \frac{df_i}{pos_i}, \tag{5}$$

where df_i is the distance factor and pos_i is the rule's position in the sorted distance list, describes the probability of each rule to be chosen. This means that zero or more rules may be selected for each cell of the first row of the CYK table.

Scheme 2. In this scheme, just after creating the list of distances, simply the nearest rule is selected. As a result, always one rule is put into the CYK cell.

Both schemes use only a single real value from the rule—a point with no accepting range around it explicitly labeled. That approach differs from the one adopted in Wilson's XCSR (Wilson, 2000), where several interval predicates are defined to "catch" input values located inside its bounds. The main resulting difference is that in the rGCS a single value method always chooses at least one rule—no matter where the input value is located. This means that even with a limited number of environment probing rules there are no input values that are left unrecognized.

Wildcard rules. If a wildcard rule exists in the system, it is always used during the environment probing because it fits to every element of the input vector. The non-terminal desired to be the wildcard one appears then in the CYK table's cell.

During the matching phase, a bundle of non-terminals are defined to be put into the first row of the CYK table, achieving the goal of translating real input values into the string of symbols capable of parsing.

Regular grammar rules. These rules play the same role as in the classic GCS. They are used in the CYK parsing process and the matching follows the CYK algorithm procedure.

4.6. Adjusting Environment Probing Rules. As the environment probing rules match the input vector, some data about the environment is collected. Every single rule of this kind keeps a copy of its real number factor. At the beginning of each learning cycle, it is set to the same value as the factor itself. Just after the matching phase, if the rule was used, this copy is modified according to the equation

$$vn_i = vc_i + wsp \cdot g \cdot ch, \tag{6}$$

where vn_i is the copy of the factor value of the i-th classifier in the population, vc_i is the current factor's value of the i-th classifier in the population, wsp is the learning factor that depends on the learning cycle (see below), g is the neighborhood function dependent on the rule's position in the list sorted by the rule's distance from the environmental situation (see below), ch is the distance from the rule's factor to the environmental situation value, calculated according to the situation:

$$ch = ve - vc_i, \tag{7}$$

where ve is the input vector element's value.

The learning factor is calculated according to the rule

$$wsp = pMaxLearningRate \cdot \left(\frac{pMinLearningRate}{pMaxLearningRate}\right)^{\left(\frac{cycle}{pCycles}\right)} \tag{8}$$

where $pMinLearningRate$ is the minimal learning factor value (parameter), $pMaxLearningRate$ is the maximal learning factor value (parameter), $cycle$ is the current learning cycle and $pCycles$ is the desired learning cycles value (parameter).

The neighborhood function is calculated according to the equation

$$g = e^{\frac{-pos_i}{ps}}, \tag{9}$$

where pos_i is the rule's position in the list sorted by the rule's distance from the environmental situation, ps is the neighborhood radius calculated according to the equation

$$ps = pMaxNeigbourhoodRadius \cdot \left(\frac{pMinNeigbourhoodRadius}{pMaxNeigbourhoodRadius}\right)^{\left(\frac{cycle}{pCycles}\right)} \tag{10}$$

where $pMinNeigbourhoodRadius$ is the minimal radius value (parameter) and $pMaxNeigbourhoodRadius$ is the maximal radius value (parameter).

It is important to work on the copy of the real number factor since we want the system to classify each example in the learning set using the same rules. This enables us to estimate a correct competence of the current grammar evolved by the rGCS. As soon as the cycle terminates, the copy of the factor replaces the old one moving the factor towards the values the rules accept frequently. The change is more significant during the initial cycles of the learning process. As the induction goes on, only small adjustments of the factors take place.

4.7 Evolving Regular Grammar Rules. Regular grammar rules are evolved just like in the classic GCS during the evolutionary process. A genetic algorithm is then launched at the end of the learning cycle. Fitness evaluation uses the fertility measurement technique (see (Unold and Cielecki, 2005b) for a discussion) for the rules that were present in any complete parsing tree generated during the cycle:

$$f_i = FTrim + tf_i \cdot FertSig, \tag{11}$$

where f_i is a fitness measure of the i-th classifier in the population, $FTrim$ is a fitness trim parameter—a base value given to the unused classifier, tf_i is a pure fertility measure (see below) of the i-th classifier in the population, and $FertSig$ is a fertility significance parameter.

The pure fertility parameter is calculated according to the equation

$$tf_i = \frac{FertPos_i - FertNeg_i}{FertPos_i + FertNeg_i}, \tag{12}$$

where $FertPos_i$ is the number of positive fertility points of the i-th classifier in the population and $FertNeg_i$ is the number of negative fertility points of the i-th classifier in the population.

Rules that were unused in the complete parsing trees but still appeared in the CYK table take the following fitness measure:

$$f_i = FTrim + tn_i \cdot FSig, \tag{13}$$

where f_i is a fitness measure of the i-th classifier in the population, $FTrim$ is a fitness trim parameter—a base value given to unused classifier, tn_i is a pure fitness measure (see below) of the i-th classifier in the population and $FSig$ is a fitness significance parameter.

The pure fitness parameter is calculated according to the equation

$$tn_i = \frac{PosPoints_i - NegPoints_i}{PosPoints_i + NegPoints_i}, \tag{14}$$

where $PosPoints_i$ is the number of positive usage points of the i-th classifier in the population and $NegPoints_i$ is the number of negative usage points of the i-th classifier in the population.

It is important to note that the fitness of these rules is downgraded by the fitness significance parameter. Complete tree parsing rules are more valuable for the system

as they cooperate with the others. Finally, unused rules take a constant fitness trim value (the parameter is usually 0.5).

The GA in the rGCS chooses parents using the roulette wheel or random selection. Then crossover and mutation operators are applied to the offspring with the probability given by the system parameters. The crowding technique replaces rules in the population with the offspring.

5. Checkerboard Problem

The checkerboard problem was proposed as a benchmark in (Stone and Bull, 2003). It divides the n-dimensional space into hypercubes of two colors (i.e. black and white). Each hypercube has the same size and is surrounded by the others with alternate colors. This means that for the two-dimensional space it looks like a chess or checkers board. There are two parameters describing the problem complexity. The first is the space dimension (n). The second is the number of divisions of each dimension of the space (n_d). In the following, we use sets of checkerboard problem examples with $n = 3$ and $n_d = 3$. We evolve grammars telling us whether the point in the solution space of given coordinates is inside a black or a white hypercube. A single CNF grammar can only tell us if the given example is positive (belongs to the grammar language) or not. We have to decide if we evolve the grammar related to white or black hypercubes. There are only two classes of hypercubes so the example rejected by one class grammar is assumed to come from the other. Every example in the set consist of three real numbers which are coordinates and the example's tag telling us whether the example is positive or negative, depending on the color of hypercubes we want the grammar to be evolved for. Example sets consisted of 100 examples (50 positive and 50 negative ones).

Parameter seeking experiments (some of these are presented below) helped us to choose optimal parameters for the checkerboard problem. For $N = 3$ and $N_d = 3$ these include:

– Number of non-terminals = 6,

– Number of environment probing rules = 3,

– Number of regular grammar rules = 60,

– Crowding factor = 35,

– Crowding subpopulation = 5,

– Desired learning cycles = 50000.

When supplied with no initial knowledge, the rGCS was able to evolve a perfect grammar when accepting all positive and rejecting all negative sentences, at every single run after on the average 24506 learning cycles (the mean of 10 runs, min. 19898 cycles, max. 31893 cycles, see Fig. 5(a)). For results with some initial knowledge (Fig. 5(b)), see Section 5.3. Every run ended with a 100% grammar competence factor.

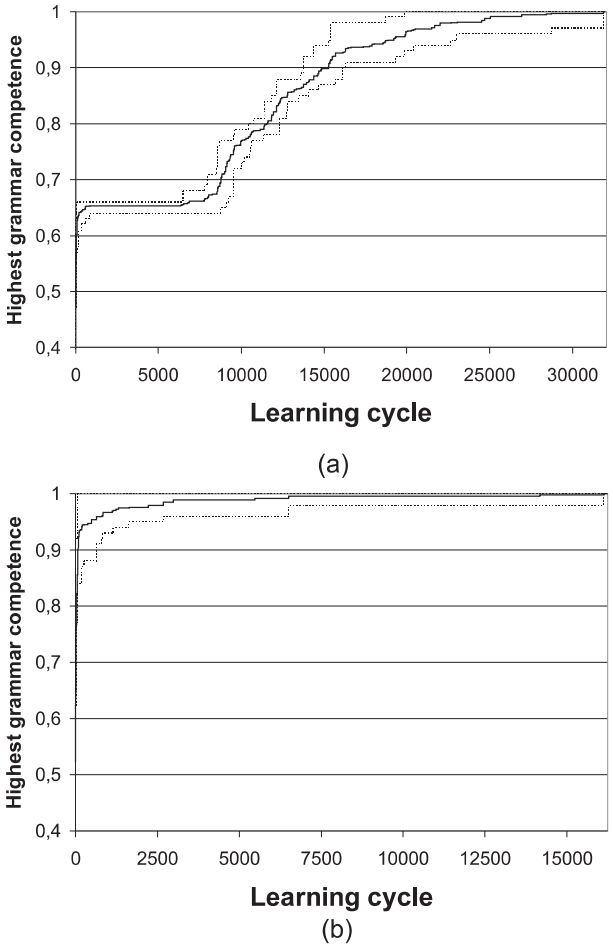

Fig. 5. Minimum, mean and maximum highest grammar competence achieved during the learning cycles without (a) and with (b) initial knowledge.

5.1. Parameter Seeking Experiments. The rGCS tends to inherit a GCS property—it is quite sensitive to the system parameter settings. It looks like a kind of disadvantage: however, correct settings may result in a rapid grammar evolution. Our previous work discussed many of these settings (see (Unold, 2005b; Unold and Cielecki, 2005b) for details). In this article we explain only rGCS-specific ones. In what follows, we investigate the best parameter settings for the checkerboard problem, $N = 3$, $N_d = 3$. A single experiment tests a range of settings for the examined parameter while the other (constant) parameters are set to random values.

Number of environment probing rules. Some preliminary experiments confirmed our first guess that we should generate at least as many rules as the number of classes the input vector elements are divided into. This means that the checkerboard problem requires at least N_d environment probing rules. Figure 6(a) shows the mean (averaged over 10 runs), minimum and maximum grammar competences of the grammar evolved as a function of environment probing rules. Setting the number of environment probing rules to a value less than N_d (3 in this case) does not allow the system to evolve an efficient grammar. However, higher values ($> Nd$) do not have any significant effect on the grammar evolution. Moreover, it slightly decreases the maximum competence of the grammar developed by the system.

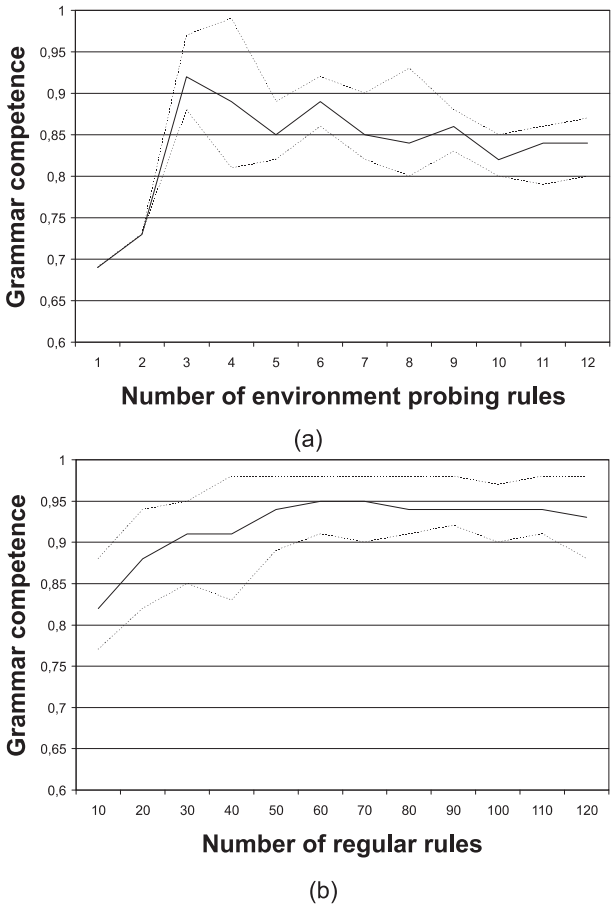

Fig. 6. Minimum, mean and maximum grammar competence as functions of the number of environment probing rules (a) and regular grammar rules (b).

Number of regular grammar rules. The number of regular grammar rules defines the size of the grammar evolved by the system. This parameter is used at the beginning of the learning process when random rules are generated. Since the rules evolved by the GA replace the old ones, the number of regular grammar rules remains constant. Figure 6(b) shows the mean (averaged over 10 runs), minimum and maximum grammar competences of the grammar evolved as a function of the number of regular rule parameters. More regular rules enable the system to evolve highly efficient grammars. However, complex grammars are difficult to analyze and consume more system resources during processing.

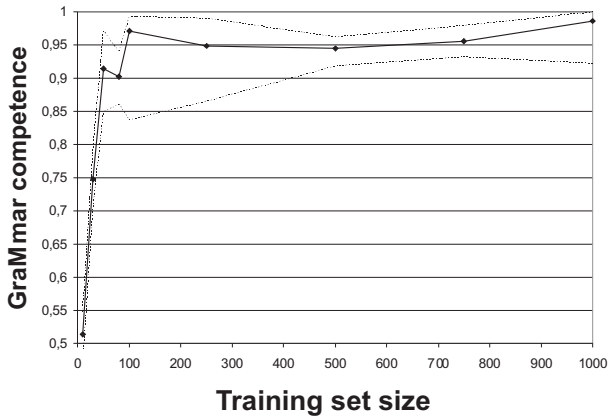

Fig. 7. Minimum, mean and maximum grammar competence as a function of the training set size.

5.2. Training Set Size. The training set size is not an actual system parameter, but it still strongly influences the system competence. This is strictly linked with the types of individuals that the rGCS evolves. Some preliminary experiments proved that the main problem during the grammar evolution is connected with the value of the factor in the environment probing rules. The relevant point is located during the adjustment phase of the learning process. Since the position is estimated using the random examples of training sets, it may be misleading when they do not cover regularly the whole area of possible values. The smaller the set, the bigger the chance of a misleading factor value. Figure 5.2 shows the minimum, mean and maximum competences of the grammar evolved using random train sets of different sizes. For each size, ten different grammars were evolved and then tested against one another (the 'test set') containing 1000 previously unseen, random examples. It is worth emphasizing that grammars evolved on the smaller train set sizes were still able to correctly classify every single sentence from its own training set, but they failed to pass generalization tests using a huge test set with unseen examples.

5.3. Initial Knowledge. As has been discussed above, the training set size proves that it is better to use bigger training sets with a large variety of examples. Howe-

ver, even when evolved with pretty extensive train sets, the main problem with grammars remains the same. The environment probing rule vectors seem to be the factors that keep grammars from passing the generalization test (using unseen examples) with the perfect competence. To prove this, we performed some tests with apopulation of individuals initialized with "perfect" environment probing rules. In this case "perfect" means: with a rule factor placed directly in the center of the area the rule should be launched for. For the checkerboard problem $N_d = 3$ and these points are

- $1/6 = 0.166\ldots$
- $3/6 = 1/2 = 0.5$
- $5/6 = 0.833\ldots$

As a result, the following initial rules were created at the beginning of every experimental run:

- $A \rightarrow 0,166\ldots$
- $B \rightarrow 0,5$
- $C \rightarrow 0,833\ldots$

This resulted in both an extremely fast grammar evolution during learning and a perfect grammar competence during testing. Introducing some initial knowledge to the system saves a lot of time at the beginning of the evolution process when normally it is necessary to evolve environment probing rules that are able to "read" input values from the training set correctly. That is why the system "stops" during the cycles 500–7500 in Fig. 5(a) and continues the evolution during the same period in Fig. 5(b). The perfect grammar competence during testing (using the same testing sets as in the previous paragraph) shows that when probing the environment's real values the input vector remains the main source of classification problems. In fact, only the input vectors with values located very close to the class borders tend to be misclassified and decrease the overall grammar competence when using "non-perfect" environment probing rules.

6. Conclusions

The rGCS, being a mutation of the grammar-based classifier system, constitutes a new tool that is able to solve problems represented by vectors of real numbers. That allows us to introduce grammar classification to a new set of problems. When tested with the common checkerboard benchmark, the rGCS evolves a perfect grammar using a similar number of learning cycles as the XCSR develops the perfect set of classifiers (approximately 20,000 (Stone and Bull, 2003)). However, rGCS knowledge representation, using grammar rules, is far easier to interpret by a human and may be processed easily.

It seems to be still possible to improve the environment probing technique to work with smaller data sets. That could be achieved by utilizing different probing schemes. We plan to introduce a rule radius that will limit the area of the input space affected. We also consider modifying the way the rule's factor behaves when used in the successful parsing. That will be especially useful when we apply the rGCS to some noisy data sets (as we plan to do).

References

Gold E. (1967): *Language identification in the limit.* Information Control, Vol. 10, No. 5, pp. 447–474.

Cielecki L. and Unold O. (2007): *GCS with real-valued input.* Lecture Notes in Computer Science, Vol. 4527. Berlin: Springer Verlag, pp. 488–497.

Holland J.H. (1975): *Adaptation in Natural and Artificial Systems.* Ann Arbor, University of Michigan Press.

Holland J.H. (1976): *Adaptation.* In: Progress in Theoretical Biology, (R.F. Rosen, Ed.) New York: Plenum Press, pp. 263–293.

Holland J.H. (1986): *Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems.* In: Machine Learning, an Artificial Intelligence Approach, Vol. II, (R.S. Michalski, J.G. Carbonell, T.M. Mitchell, Eds.), San Mateo, Morgan Kaufmann, pp. 593–623.

Holmes J.H. and Lanzi P.L., and Stolzmann W., and Wilson S.W (2002): *Learning classifier systems: New models, successful applications.* Information Processing Letters, Vol. 82, No. 1, pp. 23–30.

Judd K.L. and Tesfatsion L. (2005): *Agent-based computational economics.* In: Handbook of Computational Economics, Vol. 2, *Agent-Based Computational Economics* Elsevier Science B.V.

Katagami D. and Yamada S. (2000): *Interactive classifier system for real robot learning.* Proceedings of the IEEE International Workshop on *Robot–Human Interaction ROMAN–2000*, Osaka, Japan, pp. 258–263.

Lanzi P.L. and Riolo R.L. (2000): *A roadmap to the last decade of learning classifier system research*, Lecture Notes in Artificial Intelligence, Vol. 1813, Berlin: Springer-Verlag, pp. 33–62.

Stolzmann W. (2000): *An introduction to anticipatory classifier systems.* Lecture Notes in Artificial Intelligence, Vol. 1813, Berlin: Springer–Verlag, pp. 175–194.

Stone C. and Bull L. (2003): *For real! XCS with continuous-valued inputs.* Evolutionary Computation,Vol. 11, No. 3, pp. 299–336.

Unold O. (2005a): *Context-free grammar induction with grammar-based classifier system.* Archives of Control Science, Vol. 15 (LI), No. 4, pp. 681–690.

Unold O. (2005b): *Playing a toy-grammar with GCS*. Lecture Notes in Computer Science, Vol. 3562, Springer-Verlag, pp. 300–309.

Unold O. and Cielecki L. (2005a): *Grammar-based classifier system.* In: Issues in Intelligent Systems: Paradigms (O.Hryniewicz, J. Kacprzyk, J.Koronacki, S.T. Wierzchoń, Eds.), EXIT, Warsaw, pp. 273–286.

Unold O. and Cielecki L. (2005b): *How to use crowding selection in grammar-based classifier system.* In: Proceedings of the 5th International Conference on *Intelligent Systems Design and Applications* (H. Kwasnicka and M. Paprzycki M., Eds.), Los Alamitos, IEEE Computer Society Press, pp. 126–129.

Unold O. and Dabrowski G. (2003): *Use of learning classifier system for inferring natural language grammar.* In: Design and Application of Hybrid Intelligent Systems (A.Abraham, M.Köppen, K.Franke, Eds.), Amsterdam, IOS Press, pp. 272–278.

Wilson S.W. (1995): *Classifier fitness based on accuracy.* Evolutionary Computation, Vol. 3, No. 2, pp. 147–175.

Wilson, S.W (2000): *Get real! XCS with continuous-valued inputs.* In: Learning Classifier Systems. From Foundations to Applications (P.L. Lanzi and W. Stolzmann, and S.W. Wilson, Eds.), Lecture Notes in Artificial Intelligence, Vol. 813, Berlin: Springer-Verlag, pp. 209–222.

Younger D. (1967): *Recognition and parsing of context-free languages in time* n^3. Technical report, University of Hawaii, Department of Computer Science.

Received: 11 April 2007
Revised: 5 June 2007