Int. J. Appl. Math. Comput. Sci., 2010, Vol. 20, No. 1, 157–174
DOI: 10.2478/v10006-010-0012-8

SELF–ADAPTATION OF PARAMETERS IN A LEARNING CLASSIFIER SYSTEM ENSEMBLE MACHINE

MACIEJ TROĆ, OLGIERD UNOLD

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

Self-adaptation is a key feature of evolutionary algorithms (EAs). Although EAs have been used successfully to solve a wide variety of problems, the performance of this technique depends heavily on the selection of the EA parameters. Moreover, the process of setting such parameters is considered a time-consuming task. Several research works have tried to deal with this problem; however, the construction of algorithms letting the parameters adapt themselves to the problem is a critical and open problem of EAs. This work proposes a novel ensemble machine learning method that is able to learn rules, solve problems in a parallel way and adapt parameters used by its components. A self-adaptive ensemble machine consists of simultaneously working extended classifier systems (XCSs). The proposed ensemble machine may be treated as a meta classifier system. A new self-adaptive XCS-based ensemble machine was compared with two other XCS-based ensembles in relation to one-step binary problems: Multiplexer, One Counts, Hidden Parity, and randomly generated Boolean functions, in a noisy version as well. Results of the experiments have shown the ability of the model to adapt the mutation rate and the tournament size. The results are analyzed in detail.

Keywords: machine learning, extended classifier system, self-adaptation, adaptive parameter control.

1. Introduction

Learning classifier systems (LCSs) are rule-based systems which adapt themselves to the environment (Goldberg, 1989). They were introduced by John Holland in year 1975 (Holland, 1976). Since then, numerous types of LCSs have been proposed and used in many applications: from data-mining to robotics (Holmes *et al.*, 2002; Unold and Tuszynski, 2008; Stout *et al.*, 2008a; 2008b; Bull *et al.*, 2008). The majority of these models are based on Holland's original idea and they belong to the group of the Michigan approach. Every such a system consists of a set of condition-action rules called classifiers, where each represents a partial solution to the overall learning task, procedures for performing classifications, procedures for evaluation and for discovery rules. After detecting the state of an environment, the system uses classifiers to choose an action, performs this selected action and observes the results, which are called the payoff. The collected information is used afterwards to update the rule set. New classifiers are usually discovered with the help of the genetic algorithm (GA). In contrast, an individual of the Pittsburgh approach is a set of rules representing a complete solution to learning problem. Thus, the Michigan and the Pittsburgh models are quite different approaches to the learning. When trying to compare the performance of the two methods, it appears that in some cases the Pittsburgh approach is more robust, but it is also computationally very expensive compared with the Michigan method.

The extended classifier system (XCS) (Wilson, 1995) is probably still the most advanced and universal "Michigan-style" LCS. In every step, an XCS system tries to predict a payoff for each action which can be taken. Therefore, adapting the XCS system relies on building a "payoff map" of the environment in which the system acts (Butz *et al.*, 2004). The XCS has shown to be an effective and flexible method for solving both one-step problems (where the environmental payoff may be detected just after a single action of a system) and multi-step problems (where the final payoff occurs after some number of interactions with an environment). Theoretical analysis (Butz *et al.*, 2003; 2004) has shown how the XCS parameter should be set. Nevertheless, some parameters are still sensitive and they should be tuned with respect to the problem which is solved by the system. The adaptation or self-

adaptation of XCS parameters is to overcome this drawback (Troć and Unold, 2008; Hurst and Bull, 2002; Huang and Sun, 2004).

The XCS, like other LCSs, is not always implemented as a stand-alone system. Ensembles, which consist of several cooperating learning classifier systems, are also under research (Bull *et al*., 2007; Dam *et al*., 2005; Gao *et al*., 2007) and the possibility of parallel computing is not the only motivation. This work delivers a description of an architecture where adaptive parameter control is done in the framework of an ensemble machine built of XCS classifier systems. In this model, the XCS components learn in parallel using their own parameter values, and they cooperate in solving classification problems. The meta evolutionary algorithm (MEA) evolves the population of components and through this process it optimizes indirectly the XCS parameters. We consider that such an architecture makes it possible to adapt a majority of sensitive parameters used in the XCS system. We also suppose that the proposed self-adaptive ensemble may compete effectively with the one where self-adaptation of parameters is made at the classifier level in each component. Both methods are compared in this work.

The remainder of this paper is organized as follows: Section 2.1 provides an overview of the XCS system with a survey of adaptive parameter control in XCS, reviews self-adaptation in LCSs and classifier ensembles. In Section 3, we introduce a new model, the self-adaptive XCS-based ensemble machine. In Section 4, we use different one-step binary problems to compare the proposed model with two other XCS-based ensembles. Finally, Section 5 summarizes and concludes the work.

2. Background and related work

2.1. Extended classifier system. From its very beginning (Wilson, 1995), the XCS architecture has been evolving significantly and many varieties of it have been proposed (e.g., the XCSR, which processes real value inputs (Wilson, 2000)). Standard implementation of the basic XCS is described, among others, in (Butz, 1999). Nevertheless, in this section we take into account only the system which processes binary inputs and solves one-step problems. Solving one-step problem relies on simple classification of an input message in each cycle of system work. Every possible system action represents a class label.

As has been mentioned in Introduction, an XCS includes the population ($[P]$) of constant size linear rules (Goldberg, 1989) called classifiers and it applies procedures to adapt them, both in a parametric and structural way. There are two important data structures in an XCS apart from $[P]$: the match set $[M]$ formed out of the current $[P]$ and including all classifiers that match the current input, and the action set $[A]$ formed out of the current $[M]$ and including all classifiers from $[M]$ that propose the executed action.

Each classifier is a condition-action-prediction rule and consists of the following elements:

- the condition $C \in \{0, 1, \#\}^L$ specifies the subspace of the input space of dimensionality L in which the classifier is applicable (every "don't care" symbol $\#$ matches both 0 and 1);
- the action part A specifies the advocated action;
- the payoff prediction p estimates the average payoff expected if the classifier matches and its action is taken by the system;
- the prediction error ϵ estimates the average deviation of the payoff prediction p;
- the fitness f denotes the classifier fitness;
- the experience exp counts the number of cycles since its creation that the classifier has belonged to an action set;
- the time stamp ts denotes the number of cycles since the last GA occurred in the action set in which the classifier has attended;
- the action set size as estimates the average size of the action sets the classifier has belonged to;
- the numerosity num denotes the number of micro-classifiers aggregated in the classifier (the XCS stores identical classifiers as a single macro-classifier).

As the majority of learning classifier systems, in each cycle the XCS chooses an action as an answer for a current environmental state. Nevertheless, it may work either in the exploit or the explore phase, every cycle. During exploit cycles, the system selects an action which should cause the highest payoff (according to the prediction). In explore cycles, the system makes a random action to learn more about an environment (to build a payoff map of it).

At the beginning of each cycle, the system detects an environmental state and transforms it to a vector (input message s). After that, the XCS builds a match set $[M]$ including all classifiers which match the input. The classifier is considered to match the input when each symbol in its condition part C equals either a symbol at the corresponding position of input message or a "don't-care" symbol. Every possible action should be represented by at least one classifier in the match set. Otherwise, covering is done for all missing actions. The covering mechanism creates new classifier with each condition either taken from the input message or set to the "don't-care" symbol (with probability determined by the parameter $P_{\#}$). An action part is set to a missing action. After that, an action

a is selected and performed in the environment. While in explore cycles, the selection is random; in exploit cycles the system chooses an action with the highest value in the prediction array $P(a)$, which includes predictions for all possible actions. Prediction for an action $a \in \mathcal{A}$ is calculated as a weighted average of predictions p of classifiers which have a in their action parts. Classifier fitness f is treated as a weight.

After executing the action a, the reaction of the environment is detected, transformed to the scalar payoff R, and reinforcement learning of classifiers may be performed. Classifiers are usually trained in explore cycles only; nevertheless, some XCS implementations also carry it out in exploit cycles. At the beginning, an action set $[A]$ is created including those classifiers from $[M]$ which proposed an action a. The experience exp of all classifiers in $[A]$ is increased, and an update of parameters is made. The Widrow-Hoff delta rule (Widrow and Hoff, 1960) is used to update the prediction p, the prediction error ϵ, and the fitness f. Additionally, the first two parameters are updated with the help of the technique known as Moyenne Adaptive Modifée (MAM). In our implementation of the XCS, the prediction error is updated before the prediction, but an opposite order is often applied. The prediction is updated by $p \leftarrow p + \beta(R - p)$, where β ($\beta \in (0, 1]$) denotes the learning rate. The prediction error is updated by $\epsilon \leftarrow \epsilon + \beta(|R - p| - \epsilon)$. The use of MAM causes that, when classifier experience exp is lower than an inversion of learning rate β, the value of $1/exp$ is used instead of β. The fitness value of each classifier in $[A]$ is updated with respect to its current relative accuracy κ':

$$\kappa = \begin{cases} 1 & \text{if } \epsilon < \epsilon_0 \\ \alpha(\epsilon/\epsilon_0)^{-\nu} & \text{otherwise,} \end{cases} \tag{1}$$

$$\kappa' = \frac{\kappa \cdot num}{\sum\limits_{cl \in [A]} \kappa_{cl} \cdot num_{cl}}, \tag{2}$$

$$f \leftarrow f + \beta(\kappa' - f). \tag{3}$$

The parameter ϵ_0 ($\epsilon_0 > 0$) is a minimal classifier error considered. If $\epsilon < \epsilon_0$, the classifier is treated as an accurate one. Otherwise, the accuracy κ is a scaled reciprocal of an error, controlled by parameters ϵ_0, α ($\alpha \in (0, 1)$) and ν ($\nu > 0$). The set-relative accuracy κ' is counted with respect to accuracies of all classifiers in $[A]$. Classifier fitness f is updated according to the Widrow-Hoff delta rule, but without the use of the MAM technique. Finally, the action set size estimation as is updated (with the help of the Widrow-Hoff delta rule and MAM) by the current $[A]$ size.

Besides the covering mechanism, XCS applies the steady-state genetic algorithm for rule discovery. A GA is run if an average time from its last call in $[A]$ (counted based on the average time stamp ts parameter of classifiers) is greater than θ_{GA}. In early versions of the XCS model (Wilson, 1995; Butz, 1999), a roulete-wheel selection was used in the GA, but after that, tournament selection, proposed by Butz (Butz *et al.*, 2002; 2003), has gained in popularity, because of its attractive properties. In this selection method, two independent tournaments are created in an action set to select two parent classifiers. The tournament size is calculated as a fraction (controlled by the parameter τ) of the action set size. After reproduction, offspring classifiers are uniformly crossed (with the probability χ) and mutated (with the probability μ). We use simple free mutation (Butz, 1999), where an action part and each classifier condition can be changed into one of the remaining possible values. The parameters of offspring classifiers are in a majority derived from their parents. At the end, time stamps ts of all rules in the action set are updated to the current time.

GA subsumption (in the version proposed by (Butz *et al.*, 2002)) is fired for each offspring classifier. It checks if there exist experienced ($exp > \theta_{sub}$) and accurate rules in $[A]$ which logically subsume (with respect to the conditional parts) the new classifier. If so, the numerosity of the most general one is increased. Otherwise, the offspring classifier is inserted into the population. During the insertion, the classifier is compared with all individuals in $[P]$. If an exactly identical classifier is found, its numerosity is increased. If not, a new macro-classifier is added to the population with the numerosity set to 1, the experience set to 0 and the fitness divided by 10.

If a population size (in the sense of the number of micro-classifiers) is greater than the maximal value N, a deletion process in the whole $[P]$ is performed. Proportional selection is made with respect to two factors: action set estimation as of the classifier and the inversion of its relative fitness. The second factor is taken into account if the classifier is experienced ($exp > \theta_{del}$) and has very low fitness in relation to the average fitness in $[P]$. After the deletion, a new cycle can begin.

As has been described above, the XCS system is controlled by large numbers of parameters. They are as follows:

- β: learning rate,
- α: accuracy function fall-off rate,
- ν: accuracy function exponent,
- p_I, F_I, ϵ_I: initial values of classifier parameters,
- θ_{GA}: GA trigger threshold,
- τ: relative tournament size,
- χ: crossover probability,
- μ: mutation rate,
- $P_\#$: probability of using a dont-care symbol during covering,

- θ_{del}: deletion experience threshold,
- θ_{sub}: subsumption experience threshold,
- δ: mean population fitness fraction below which a classifier fitness is considered in the deletion-vote function.

Parameter influence on system adaption was investigated, among others, in (Butz *et al.*, 2004), where some important assumptions for parameter tuning were. In (Butz *et al.*, 2002; 2003; Kharbat *et al.*, 2005), it was shown that using tournament selection (instead of proportional one) makes parameters less sensitive. If tournament selection is applied, the values of some parameters (like α or ν), proposed in (Butz *et al.*, 2003), are always appropriate and they do not have to be changed. Nevertheless, some other XCS parameters need to be tuned with respect to the type and scale of the problem being solved by the system, the population size N, the dynamics of the environment, the level of payoff noise, or even the properties (like fitness or generality) of classifiers controlled by these parameters.

The value of tournament size may be taken from quite a broad range $[0.2, 0.8]$ in most cases (Butz *et al.*, 2002; 2003), but sometimes extremely high τ of 1 is required (Kharbat *et al.*, 2005). In (Dawson, 2002), it was shown that high evolutionary pressure is necessary if classifier population is very small, which also suggests using $\tau = 1$ in such circumstances. The mutation rate μ is probably the most sensitive XCS parameter, because its optimal value depends on the properties of the environment and those of the classifier being mutated (Hurst and Bull, 2002). In some environments, fixed μ should be either lower or higher than 0.04, which is the value proposed in (Butz *et al.*, 2003) for classification problems. Nevertheless, only the adaptive mutation rate, which takes into account the current content of classifier population, may support optimal search of the rule space. The learning rate β is another XCS parameter which requires tuning. In (Butz *et al.*, 2002; 2003) it was shown that, if β is about 0.05, the XCS using tournament selection adapts effectively to static environments even when the Gaussian noise is added to environmental payoffs. Whereas adaption to a dynamic environment requires a higher learning rate to enable fast recalculation of classifier parameters in response to changes of the payoff landscape (Dam *et al.*, 2007). Additionally, the optimal value of the learning rate depends on the generality and accuracy of the classifier being learned. The parameter β should be decreased for overgeneral rules, in which large fluctuations of the prediction p, prediction error ϵ and fitness f may occur (Butz *et al.*, 2005; Orriols-Puig *et al.*, 2009).

Because of the sensitivity, various methods for adaptive or self-adaptive parameter control have been proposed (Hurst and Bull, 2002; Butz *et al.*, 2005; Dam *et al.*, 2007). In the next subsection, we focus on self-adaptive ones.

2.2. Self-adaptation in learning classifier systems. Self-adaptation in evolutionary algorithms has been investigated in numerous projects (Meyer-Nieberg and Beyer, 2007). In genetic algorithms, the main goal is to control the mutation rate and the crossover operator. The latter is realized mainly as an adaptation of the place of crossing or the number of crossing points (one-point, many-points or uniform crossover) (Spears, 1995; Meyer-Nieberg and Beyer, 2007). In recent years, an attempt has been made to adapt the parameters which influence not only a single individual, but also the whole population. In (Eiben *et al.*, 2006a; Eiben *et al.*, 2006b), the voting mechanism was proposed to control population-level parameters with the help of individual-level adaptation (according to the Angelines classification (Meyer-Nieberg and Beyer, 2007)). The self-adaptation of the tournament size and the population size was performed, producing interesting results.

The self-adaption of parameters has been performed in various LCSs concerning both genetic and reinforcement parameters. The meta-EP method (Fogel, 1992) has been used to adapt the mutation rate μ, the learning rate β and some other reinforcement parameters in two systems solving multi-step problems: the extended classifier system (Hurst and Bull, 2002) and the zeroth-level classifier system (Hurst and Bull, 2003). In this method, the values of parameters are stored in each classifier as real-valued genes. During the action of the genetic algorithm, the parameter genes are passed to child classifiers, recombined and mutated using Gaussian distributions. For example, the mutation rate μ in each offspring classifier is mutated with the help of its own value, $\mu = \mu + N(0, \mu)$ (Hurst and Bull, 2002), and then applied to the classifier condition and action. The model presents the classic self-adapting attitude, individual-level in the Angelines classification. Experiments were carried out in static and dynamic Woods environments. The results showed competitive performance of the system using the self-adaptive mutation rate in comparison with the classic one for some difficult (complex or dynamic) environments. The adaptation of the learning rate (in both systems) and other reinforcement parameters (in ZCS) was made with coevolutionary modification of the meta-EP method called "enforced cooperation" (see (Hurst and Bull, 2002; 2003) for a description). It gave good results for some cases of ZCS adaption.

In (Howard *et al.*, 2008), the mutation rate μ of the neural XCS was changed before being copied to an offspring using the formula $\mu = \mu + e^{N(0,1)}$. It was shown that a self-adaptive neural XCS can perform optimally in more complex and noisy versions of two well-known simulated maze environments. However, the authors notice

that self-adaptation does not significantly influence the performance of the whole system (t-test value > 0.01).

In (Huang and Sun, 2004), co-adaptation between two learning classifier systems—the Main-LCS (which aimed at solving the problem) and the Meta-XCS (which aimed at control parameters in the Main-LCS)—was used. The Meta-XCS is based on two architectures: XCS and Dyna (Sutton, 1991). It learns rules which anticipated the future metrics of the main system (like performance or the population size) based on the recent metrics and the action of changing the parameters in the Main-LCS. Latent learning is applied. Thanks to this solution, a complete model of Main-LCS behavior with respect to the values of parameters is built. As was noted by the authors, the described co-adaptive architecture combines both an adaptive and a self-adaptive approach for parameter control. Based on the Angelines classification, we could also say that it is a population-level type of adaptation. The model was tested on the adaptation of the mutation rate in a six-bit multiplexer environment and showed high performance.

Many self-adaptation methods (like meta-EP) assume that individuals using more optimal parameter values are usually better evaluated and they have greater opportunities for being reproduced. The mutation rate complies with this assumption, causing the self-adaptation of μ perform well. Nevertheless, some parameters have direct influence on classifier evaluation and cannot be simply self-adapted. For example, the learning rate controls updates of classifier parameters (among others, the fitness updates) and an incorrect value of β makes inaccurate classifiers over-fitted (Orriols-Puig *et al.*, 2009). In (Hurst and Bull, 2003) it was shown that the adaptation of β at the individual level is "selfish" indeed and therefore the "enforced cooperation" method, which is dedicated to systems solving multi-step problems, was proposed.

Some parameters are even more difficult for self-adaption, because they control operations made on sets of classifiers or on the whole classifier population. The tournament size τ and the deletion threshold θ_{del} may be given as an example of such parameters in the XCS. Although an algorithm for the self-adaptation of the tournament size and the population size in genetic algorithms has already been proposed (Eiben *et al.*, 2006; 2006b), we do not know works which confirm the effectiveness of this method in a broad range of problems.

Owing to problems with self-adaptive control of many important parameters, we think that using the MEA for parameter adaptation in LCSs should be considered. In this group of methods, which are in a majority derived from the meta genetic algorithm (meta-GA) (Grefenstette, 1986), the additional, distinguished evolutionary process is applied for searching the parameter space. In the versions proposed for parallel genetic algorithms (Tongchim and Chongstitvatana, 2002; Takashima *et al.*, 2003), the model based on a population divided into several subpopulations evolving in parallel is applied. Each of them uses its own vector of parameter values. The meta evolutionary process operates on these vectors evaluating, reproducing and recombining them. An adaptation of the tournament size and other important parameters has been made in this way, giving promising results.

2.3. Classifier ensembles. A classifier ensemble is a group of classifiers (components) which are trained individually but used together to realize the classification task. The basic architecture of the classifier in the ensemble may be a neural network, a decision tree, etc. (Opitz and Maclin, 1999). To conduct classification, the outputs of the classifiers must be combined, and the simplest (but commonly used) way is by voting (Bahler and Navarro, 2000). For example, plurality voting relies on every classifier making a classification (the vote), and the class with the largest number of votes becomes the output of the whole ensemble. It has been shown in many works (Opitz and Maclin, 1999; Dietterich, 2000) that the ensemble exhibits better performance than a single classifier. Moreover, classifiers may be trained in parallel on separate computer machines.

Two factors are important for the effectiveness of an ensemble: the correctness of answers of the average component and the diversity of the answers of the components. Because classifications made by the components are not perfect, it is obvious that the ensemble will be effective if it makes mistakes for different inputs. To increase the diversity, in many models (e.g., in the bagging method (Breiman, 1996)), components are trained with separated learning sets. Applying heterogeneous classifiers differing in type or using different parameter values is also practiced and it makes the ensemble more independent from the problem being solved (Opitz and Maclin, 1999; Bahler and Navarro, 2000; Tsoumakas *et al.*, 2004). Note that a single XCS system (or a similar classifier system) is also some kind of classifier ensemble. Rules (classifiers) cooperate to type the best action for the current environmental state and fitness-based weighted voting is applied. Nevertheless, there are several reasons for using ensembles built of learning classifier systems.

In (Dam *et al.*, 2005), an XCS ensemble is applied for data-mining in a physically distributed data set. Each subset of data is used for training a local classifier system. An additional XCS learns how to combine outputs of local components.

In (Bull *et al.*, 2007), an ensemble built of YCS (yet another) classifier systems was proposed. In the exploration phase, every YCS system makes a random action and learns the rules, but in the exploitation phase all systems type an action by voting. Moreover, the migration mechanism is applied to move classifiers among systems (based on their fitness). Because the classifiers in each

YCS system are created by a genetic algorithm and migration is used, the whole YCS-based ensemble resembles an island model of parallel genetic algorithms (PGAs) (Bull *et al*., 2007). The performance of the rule sharing ensemble of YCSs was tested on 20-bit and 70-bit multiplexer problems. The ensemble improved learning speed in comparison with a single YCS.

Another approach to using an ensemble of LCSs was described in (Gao *et al*., 2007). The proposed system consists of two levels: the first level is comprised of a set of XCSRs (XCS with real-value attributes), the second one uses a vote module to combine the results of the XCSRs. The whole system mined medical data and performed image steganalysis. The ensemble of XCSRs had a better generalization ability and prediction performance than a single XCSR and other comparable supervised learning methods; however, differences were not statistically tested.

3. Self-adaptive XCS-based ensemble machine

LCS-based ensembles are usually made of homogeneous components (Dam *et al*., 2005; Bull *et al*., 2007; Gao *et al*., 2007). In this work, we investigate a model where XCS components use various values of parameters and the meta evolutionary algorithm (MEA) is applied for adaptive control of these parameters in the ensemble. Each component containing both the classifiers and the vector of parameter values is treated as an individual, which can be evaluated, reproduced and mutated in the population of XCS systems. Our approach is somewhat similar to that of (Opitz *et al*., 1996), where the ensemble of neural networks is optimized by means of a genetic algorithm. Because each XCS component applies the GA for rule discovery, our model is also similar to some parallel genetic algorithms, where an adaptation of parameters is made at the subpopulation level (Tongchim and Chongstitvatana, 2002; Takashima *et al*., 2003). The sensitiveness of XCS parameters and problems with their adaptation at the classifier level are the key motivation for our model.

The proposed ensemble consists of a fixed number of XCS systems, which cooperate to solve one-step (classification) problems and learn in parallel. The ensemble size will be denoted as N_C. Like in (Bull *et al*., 2007), we investigate a "coarse grained" ensemble ($N_C = 10$ in most experiments). The main loop of system work follows the description placed in (Bull *et al*., 2007). The exploitation and exploration phases go one after another and the components are trained during the second ones. The generic framework of the proposed ensemble model is given in Fig. 1, while the pseudo code and a description of basic components in Fig. 2.

During exploitation phases, all XCS systems compute the same input and make deterministic classifications. An ensemble class is chosen by non-weighted plurality voting. This means that a class which is pointed by the majority of the components is an output of the whole ensemble. Note that in the case of a binary class, plurality voting is equivalent to majority one. In the case when two (or more) classes have got the same number of votes, an output class is selected randomly among them.

During every explore phase, components are trained independently (in contrast to our previous works (Troć and Unold, 2008)). Thus, they process different inputs and after that they apply reinforcement learning on the basis of randomly selected classes. Moreover, the fitness f_C of each component is updated in this phase and the MEA may be invoked one or more times. It will be described in details in the following subsection. No rule migration among XCS systems is applied.

3.1. Component learning and calling the meta evolutionary algorithm. At the beginning of an explore cycle, every component creates the match set $[M]$ and the prediction array $P(a)$ to predict the class of the received input instance (as is done in exploit phases). The predicted class is compared with the target one and the result of the comparison updates the component fitness f_C, which is the proportion of the correct classifications done in the last s_C explore cycles. Thereafter, another class is selected randomly and used to learn component classifiers in a usual way.

After the training of components, the MEA may be invoked in the ensemble. At first a set is formed, which includes experienced components existing in the ensemble at least s_C explore cycles. The best fitted individual is selected among them as a candidate for reproduction. Each component in the set where fitness is lower than that of the selected one by some threshold value θ_{MEA} is deleted from the ensemble. Empty places shall become occupied by the offsprings of the most fitted (selected) individual. Details of component reproduction and mutation are given in the next subsection. Now, we will try to justify the proposed scheme of calling the MEA. It seems to be necessary because, in our previous model (Troć and Unold, 2008), the MEA is executed in some fixed number of iterations of components in which parameter values are optimized. Similarly, in related works (Tongchim and Chongstitvatana, 2002; Takashima *et al*., 2003), the step of meta genetic algorithm takes place in a predefined number of generations or fitness evaluations of individuals in subpopulations.

The component fitness f_C, which is the estimated probability of correct classification based on last s_C trials, fluctuates, disturbs component evaluation. The lower the parameter s_C the higher the variance of f_C. Note that a similar problem has also been observed at the rule level in a stand-alone XCS system (Butz *et al*., 2002; Orriols-Puig *et al*., 2009), where the learning rate β is a sensitive

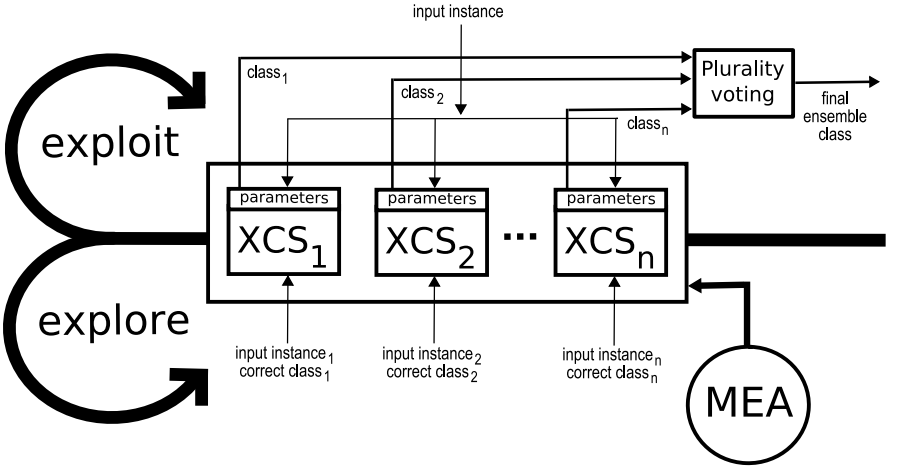


Fig. 1. Framework of the self-adaptive XCS-based ensemble machine.

parameter. Beside the fitness fluctuations, another problem appears in the proposed XCS-based ensemble. The influence of parameter values owned by an XCS component may be reliably observed only after some number of learning trials during which these values are used. It is hard to determine which number is large enough. The speed of component adaptation depends on its goal, and in some complex cases no change in system performance may be detected for many iterations. This problem makes component evaluation even more difficult. Therefore, we consider that executing the MEA in a predefined number of explore trials is not an optimal solution. Too short a period between executions results in random reproductions, which push a small component population in a random direction (Troć and Unold, 2008). Too long one may unnecessarily slow down the adaption. Because of these problems, in our recent model, the MEA is called if there is a significant difference between the fitness of a candidate for reproduction (the best fitted component) and the fitness of a candidate for deletion (other experienced component). The threshold value θ_{MEA} several times greater than the maximal fitness deviation guarantees high probability of reliable comparison of the components. Note that the sensitivity of s_C is reduced this way. For $s_C = 2000$, we use θ_{MEA} of 0.06.

The proposed method has a chance to be independent of the problem being solved by an ensemble and other factors, which influence the learning of components. Moreover, it limits the number of reproductions. For example, when all components are close to maximal performance, no reproductions are made. It is an important advantage, because every reproduction evokes some computational cost and reduces diversity in an ensemble as well.

3.2. Reproduction and mutation of a component. The reproduction of a component relies on making an exact copy of it with respect to both elements, i.e., the classifier population and the vector of parameter values. Thereafter, the offspring XCS component is placed at the empty position in an ensemble (done after deletion). If an ensemble is run in the network of computers, the new component is transmitted to the free computing node. After that, the mutation is performed. The operator changes only the vector of parameter values but not the classifiers.

In our recent research, the vector consists of two elements. The mutation rate μ and the tournament size τ are adapted. The first parameter is coded as a real value gene, which is mutated according to the meta-EP formula: $\mu = \mu + N(0, \mu)$. In contrast to that, the binary coded tournament size can have only two values: 0.4 or 1.0. This scheme is motivated by the works on tournament selection in XCS (Butz *et al*., 2002; Butz *et al*., 2003; Kharbat *et al*., 2005), where it was shown that either τ belonging to the range [0.2, 0.8] or τ of 1.0 should be set in relation to the problem being solved. The broad range of values ($[0.2, 0.8]$), which are appropriate in most cases, suggests that they all have similar influence on system performance. Therefore, an evolutionary search of the optimal tournament size could be difficult in the real-value space. In our model, the bit-flip mutation with the rate μ_τ is applied to mutate the binary gene of τ. We use a high value of the rate ($\mu_\tau = 0.25$) to keep the high diversity of the τ parameter in the ensemble. Let us assume that only the tournament size is adapted. If all components use the same tournament size, no significant difference between their fitness values will be observed and the MEA cannot be executed. Of course, parameter adaptation gets stack in such circumstances. High μ_τ is to minimize the risk of that.

Both adapted parameters are set in the initial component population simply by the mutation of their commonly used values (μ = 0.04, $\tau = 0.4$).

```
Algorithm
ensemble—the set of XCSs in the ensemble

for each XCS in ensemble
   initialize the vector of parameter values
end for each
do while (not reach the maximum learning step)
   if exploit phase
      distribute the same input instance to each XCS in ensemble
      for each XCS in ensemble
         select best class
      end for each
      choose the final class by plurality voting between XCSs
         in ensemble
   else
      for each XCS in ensemble
         get own input instance accompanied by a correct class
         select best class
         compare selected class with a correct class
         update f_c as a proportion of correct classifications done
            in the last s_c explore cycles
         select random class
         get reward for selected random class
         perform reinforcement learning and optionally GA
      end for each
      invoke MEA
   end if
neg exploit phase
end do while

procedure MEA
experienced ← the set of XCSs existing in ensemble
   at least s_c explore cycles
best ← select XCS from experienced with a highest f_c
exchange ← select XCSs from experienced
   for which f_c < f_{c_best} − θ_MEA
ifnot empty exchange
   ensemble ← ensemble / exchange
   for each XCS in exchange
      replace XCS with a copy of best
      set experience exp of XCS to 0
      mutate the vector of parameter values
   end for each
   ensemble ← ensemble ∪ exchange
end if
```

Fig. 2. Pseudocode of the self-adaptive XCS-based ensemble machine.

4. Experiments

The proposed architecture has been compared with the other two XCS-based ensembles, which only differ in parameter control. In the first of them, the fixed values of parameters ($\mu = 0.04$ and $\tau = 0.4$) are used without any parameter adaptation. In the second one, the self-adaptation of the mutation rate is performed at the classifier level in each XCS component. The meta-EP method (Fogel, 1992; Hurst and Bull, 2002; 2003) is used for that, and μ is initialized around the value of 0.04 in classifiers created by covering. In this ensemble, the tournament size remains fixed ($\tau = 0.4$).

All three ensembles are equal-sized and they solve binary problems in the same way. Their components are trained with two payoff levels, which are 0 (for bad classifications) and 1000 (for good classifications), respectively. The performance of every investigated ensemble is measured during exploit cycles as a moving average of the last 50 classifications (1 for every correct classification and 0 for the incorrect one). The performance curves are averages of ten independent runs. Only some presented results are averaged over 50 runs.

4.1. Implementation and parameters of the system. As has been noted, we investigate an ensembles consisting of 10 components. The parameters of the MEA are tuned as follows: $s_C = 2000$, $\theta_{MEA} = 0.06$, $\mu_\tau = 0.25$.

An implementation of a component is based on the description of the XCS system (the second section of this work) and it differs slightly from (Butz, 1999). Among others, we use tournament selection, uniform crossover and free mutation. Moreover, the method of fast classifier matching (proposed in (Llorà and Sastry, 2006)) is applied. The fixed values of XCS parameters, used in most of the experiments, are as follows: $\beta = 0.2$, $\alpha = 1.0$, $\nu = 5$, $\epsilon_0 = 1.0$, $\theta_{GA} = 25$, $\chi = 0.8$, $P_\# = 1.0$, $\theta_{del} = 20$, $\delta = 0.1$, $\theta_{sub} = 20$. They are taken from (Butz *et al.*, 2002). As mentioned above, the mutation rate is set to 0.04 and the tournament size is set to 0.4, if they are not adapted. Note that, like in (Butz *et al.*, 2003), $P_\#$ of 1.0 is applied to exhibit genetic algorithm activity. The size of the classifier population depends on the experiment.

4.2. Binary classification problems. The l-bit multiplexer (MP-l) is the basic benchmark problem which is used to verify our approach. Nevertheless, some experiments with a count ones l/k problem and a hidden parity l/k problem are also performed. All of these problems were analyzed before in (Butz *et al.*, 2003). In all of them, a system classifies each received string to one of the possible classes, which are labeled as: "0" and "1", respectively.

In the MP-l problem, where $l = k + 2^k$, the system is receiving binary strings, where the first k bits represent the address (index) of the binary position in the next 2^k data bits. In every step, the goal of the system is to determine the bit value at the position pointed by the address.

Solving the count ones l/k problem can be described as follows. A system processes binary strings of length l, which contain k significant bits at the predefined positions. If more than a half of these bits equal 1, the correct

class of a string is "1". In the opposite case, the string should be classified as "0". The remaining l-k bits have no impact on the class.

In the hidden parity l/k problem, the goal of classification is to determine if there is an odd number of ones among significant k bits in the string of length l. If so, the correct class is "1". Otherwise, the class is "0". Like in the count ones problem, significant bits are placed at fixed positions, which are the same for every classified string. The remaining l-k bits should be ignored by the classifier system.

Beside the benchmark problems described above, randomly generated Boolean functions were used during experiments (Butz and Pelikan, 2006). The goal of classification is to determine the function value calculated from input strings of length l (every bit in a string is a Boolean variable). The generated functions can be expressed in disjunctive normal form (DNF) with a fixed number of closures s and a fixed number of literals k in every closure. For example, the function $y = x_1x_4 \vee x_1\neg x_3 \vee \neg x_2x_4$ consists of three closures with two literals in each.

4.3. Learning of binary problems. First, we present results of experiments with the multiplexer problem for three problem sizes, i.e., MP-11 (Fig. 3), MP-20 (Fig. 4) and MP-37 (Fig. 5). As has been noted, all investigated ensembles, which differ in the method of parameter adaptation, consist of ten XCS components ($N_C = 10$). In the first two experiments (MP-11 and MP-20 problems), the population size N of 2000 is applied in every single component. In the third experiment, N is enlarged to 5000, which is the value commonly used in XCS systems solving MP-37 problems (Butz *et al*., 2004; Kharbat *et al*., 2005). Results show that both methods of parameter adaptation cause faster learning of the MP-11 (Fig. 3) and the MP-37 problem (Fig. 5). The ensemble using fixed values of parameters ($\mu = 0.04$ and $\tau = 0.4$) adapts very slowly to MP-37. In (Kharbat *et al*., 2005), the tournament size of 1.0 was suggested for this multiplexer. In the case of MP-20 (Fig. 4), an adaptation of parameters is not needed at all, because fixed values are quite well tuned to the problem size. Observe that, in all experiments the ensemble, which uses the MEA for parameter adaptation, shows better performance than the ensemble which applies the self-adaptive mutation rate accordingly to the meta-EP method. The average μ in the population of components converges to values which can be theoretically explained (Buts *et al.*, 2003; 2004) with respect to the problems being solved. Opposite to that, the average tournament size remains close to its lower level ($\tau = 0.4$) all the time. Even in the case of MP-37, a low mutation rate is more important than a high tournament size for effective learning.

Figure 6 shows the learning of the count ones 100/7 problem, where the population size N of each component is set to 3000 (as suggested in (Butz *et al*., 2003)). The mutation rate of 0.04 is too high to solve the problem and therefore average μ falls for both methods of parameter adaptation (Fig. 6(b)). Nevertheless, the meta-EP gives slightly better results than the MEA.

In the experiment with the hidden parity 20/5 problem, we use the population size of 1900, which is one of the values used for this problem in (Butz *et al*., 2003). The results (Fig. 7) are averaged over 50 runs, because of a high variance of system performance. Observe that the ensemble, which applies classifier-level self-adaptation of the mutation rate learns slower than the other two architectures. After a rapid growth at the beginning of the learning, average μ continuously falls. Contrary to that, the second method of parameter adaptation gives very good results. The average values of both parameters, which are controlled by the meta evolutionary algorithm, increase during the learning.

The adaptation of the mutation rate is much more important than that of the tournament size in all the investigated problems. This seems obvious, because mutation is an important XCS operator, which enables the discovery of accurate classifiers. An accurate classifier has specific symbols (either 0 or 1) at all essential positions in its conditional part. These positions are important to determine the class of a matching string. As reported in (Butz *et al*., 2003), free mutation causes pressure towards the average rate of specific symbols in classifiers of 0.66. This "specialization pressure" is balanced by the "set pressure", which favors less-specific rules (Butz and Pelikan, 2001; Butz *et al*., 2004). It is a well-known fact (Wilson, 1995; Butz and Pelikan, 2001; Butz *et al*., 2003; Butz *et al*., 2004) that, in the XCS system, the more general classifiers match more often input strings and therefore these classifiers are more often reproduced in an action set.

The mutation rate should be high enough to find specific and accurate classifiers. On the other hand, if classifiers in the population are too specific, even accurate ones may have no chance for reproduction in the limited classifier population. This mechanism is described in detail in already cited works (Butz *et al.*, 2003; 2004).

The experiment with the hidden parity problem (Fig. 7) will be further analyzed, because it gave the most interesting results. As reported in (Butz *et al*., 2003), there is no "fitness guidance" when an XCS system adapts to the hidden parity problem starting from overgeneral rules. This is caused by the fact that every classifier which may be created during learning is either perfectly accurate or completely inaccurate. Accurate classifiers have in their conditional parts specific symbols at all k positions (in our experiment, $k = 5$). These classifiers are well fitted, because their prediction errors converge to 0. Analogically, a prediction error of each classifier which has a "don't care" symbol at one or more significant positions is close to the

maximum value, even when only one significant position is generalized. There are no "partially accurate" classifiers. Under such circumstances, the only way to discover the accurate rules starting from the overgeneral (and inaccurate) ones is random exploration of the rule space. This may be effectively done with an appropriately high mutation rate. The more interested reader is referred again to (Butz *et al.*, 2003).

The self-adaption of the mutation rate at the classifier level can now be explained in the case of the hidden parity problem. All XCS components have too general classifiers at the beginning of the learning, and there is no "fitness guidance" until accurate classifiers are discovered. Instead of that, the rules which use lower mutation rates are usually more general, and they are more often reproduced because of the "set pressure". Accordingly, an average mutation rate decreases when the meta-EP method is applied (Fig. 7(b)). Under such circumstances, the "specialization pressure" becomes weaker and it takes more time to find the accurate classifiers. We can say that, in the case of Hidden Parity problem, the self-adaptation of μ goes in exactly the opposite direction than it should.

This drawback is not observed when MEA is applied for parameter adaptation. The MEA is not executed at the beginning of learning, because all XCS components have similar, poor fitness (f_C is about 0.5). For this reason, the initial distribution of the mutation rate (with the mean of about 0.04) is fixed in the component population. As soon as some component finds several accurate classifiers, it is reproduced by the MEA. It is highly probable that this component uses a mutation rate higher than 0.04 and therefore average μ in the ensemble increases (Fig. 7(b)). If accurate classifiers are found in an XCS component, the new accurate rules may be created from them by reproduction and mutation. A strong selection pressure makes reproductions of accurate components more likely. This may be the reason why XCS components which use the tournament size of 1.0 become over-represented in the component population (Fig. 7(b)). Finally, after all components reach maximum performance, the MEA is no longer executed, and the average values of both adapted parameters do not change.

Opposite to the hidden parity problem, the count ones problem enables strong "fitness guidance" (see (Butz *et al.*, 2003) for more details). In this case, meta-EP is an effective method of self-adaptation of the mutation rate. It gives better results than the application of the MEA because of two main reasons: only μ is adapted, while the fixed tournament size ($\tau = 0.4$) is appropriate for the problem; components are not reproduced and there is no loss in ensemble diversity. When the XCS learns the multiplexer problem, "fitness guidance" is considered to be weak (Butz *et al.*, 2003). This may be the reason why the meta-EP method is not very effective in this case. Particularly, for the MP-20 problem, an average mutation rate seems to decrease prematurely (Fig. 4(b)).

The experiments described above focus on the mutation rate. Nevertheless, we are particularly interested if the tournament size can be effectively adapted at the component level. In the next experiment, the MP-37 problem is solved again, but only the tournament size is optimized by the MEA, while the mutation rate is fixed at 0.04. The results are presented in Fig. 8. An average tournament size increases and exceeds the level 0.7, in which both values of the τ parameter (i.e., 0.4 and 1.0) are equally represented in the component population. This confirms the observation given in (Kharbat *et al.*, 2005) that the tournament size of 1.0 is appropriate for solving the MP-37 problem when the mutation rate is approximately 0.04. Nevertheless, before the 150000-th exploit cycle, the average tournament size begins to decrease, which means that the strong selection pressure is no longer necessary. Because we treat the XCS component rather as a "black-box", the interested reader is referred to (Butz *et al.*, 2002; Butz *et al.*, 2003; Kharbat *et al.*, 2005) to learn more about tournament selection in the XCS system.

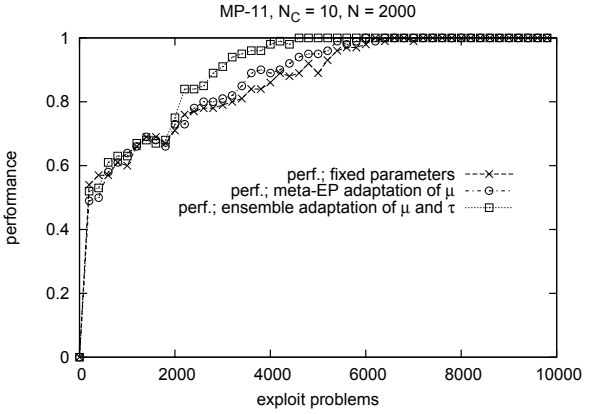
The adaptive tournament size is also useful when an ensemble solves smaller multiplexer problems but a very low population size N is set in each XCS component. Experiments with MP-11 and M-20 were performed for $N = 100$ and $N = 400$, respectively. Both parameters, i.e., the mutation rate and the tournament size, were adapted. In the case of MP-11 and $N = 100$, none of the investigated ensembles is able to completely learn the problem; however, the ensemble which uses the MEA for parameter adaptation significantly achieves better performance than the other two (Fig. 9). The superiority in the performance is caused by the components which have the tournament size of 1.0. Observe that they are in majority during the whole experiment (Fig. 9(b)). This result is also in agreement with (Dawson, 2002), where it was shown that a strong evolutionary pressure is needed if the classifier population is very limited in size.

When the MP-20 problem is solved with $N = 400$, the self-adaptive ensemble shows the highest learning speed as well (Fig. 10). Nevertheless, the adaptation of the tournament size is not so important in this case. The ensemble in which the mutation rate is adapted by the MEA but the tournament size is fixed at 0.4 shows worse performance only at the very beginning of learning (results not shown).

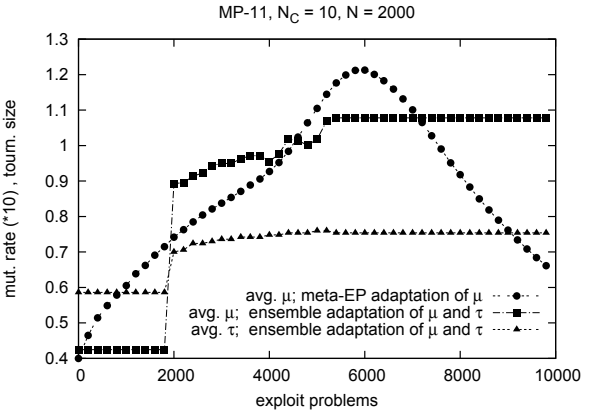
In the next experiment, we investigate learning randomly generated Boolean functions defined over binary strings of length 20 ($l = 20$). Each function, which is expressed in disjunctive normal form, consists of ten clauses ($s = 10$) and there are five literals in every clause ($k = 5$). Moreover, to make the problem more challenging, an alternating noise $P_X = 0.1$ is applied (Butz *et al.*, 2002). In this kind of noise, a wrong class of a string (learning instance) is passed with probability P_X to the XCS com-

ponent during an explore cycle. The noise disturbs both the reinforcement learning of classifiers and component evaluation. Results (Fig. 11) are averaged over 50 runs (Boolean functions). It seems not enough; nevertheless, it was shown in (Butz and Pelikan, 2006) that randomly generated Boolean functions of the same complexity are similarly difficult for XCS systems, and standard deviation of average system performance is rather small.

Contrary to our previous experiments (described above), in the recent one, the $P_{\#}$ parameter is set to 0.6. This is the value typically used for the input size of the investigated problem (see (Butz *et al.*, 2004) for a detailed explanation). Note that, because covering is enabled, the problem should be easier to solve by an XCS-based ensemble. Nevertheless, the ensemble with fixed parameters does not learn at all (Fig. 11). Both the ensembles that adaptively control the mutation rate are able to learn, but none of them reaches maximum performance. Observe that the MEA gives better results, though the tournament size does not need to be adapted. An average mutation rate decreases continuously for both methods of parameter adaptation. We suppose that a low mutation rate is needed because of alternating noise. It is worth noting that in (Kharbat *et al.*, 2005) the lowered mutation rate was suggested in the XCS, when the Gaussian noise is added to payoffs. Although it is a completely different kind of noise than the alternating noise, some analogy may be found.



(a)



(b)

Fig. 3. Solving the MP-11 problem by three ensembles: the ensemble using fixed parameters ($\mu = 0.04$, $\tau = 0.4$), the ensemble including components with the self-adaptive mutation rate (τ of 0.4 is used), and the proposed ensemble with meta evolutionary adaptation of μ and τ. The performance curves (a) and average values of adapted parameters (b) are presented.

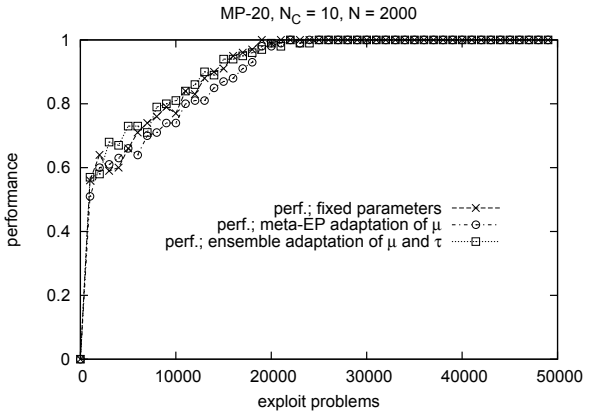
4.4. Consequences of the meta evolutionary algorithm. Though the MEA seems to be an effective adaptive method, it causes some unwanted effects. During every component reproduction, a copy of an XCS component has to be sent from one computing node to another, which results in extra computational cost. At the same time, some unique component has to be removed, which results in the loss of ensemble diversity. Figure 12(a) shows a total number of reproductions made before the i-th exploit trial (in relation to i) in the self-adaptive ensemble which solves one of three problems: MP-20, MP-37 or hidden parity 20/5. Observe that reproductions stop at some moment, when the MP-20 problem or the hidden parity 20/5 problem is being solved. This is because all components are about maximum fitness (maximum performance).

Much more iterations are needed to learn the MP-37 problem than the two remaining ones and therefore only the initial period of adaptation can be seen in Fig. 12(a). During this period, XCS components are close to a local optimum (see (Butz *et al.*, 2003) for more details) and have similar fitness values. Accordingly, reproductions are rather rare. When some components discover better classifiers, the number of reproductions grows rapidly, and finally it achieves a total value of about 35 (not shown). The results presented in Fig. 12(a) show that the frequency of reproductions depends on the state of learning. Moreover, the total number of component reproductions required for parameter adaptation is not as large as could be expected.

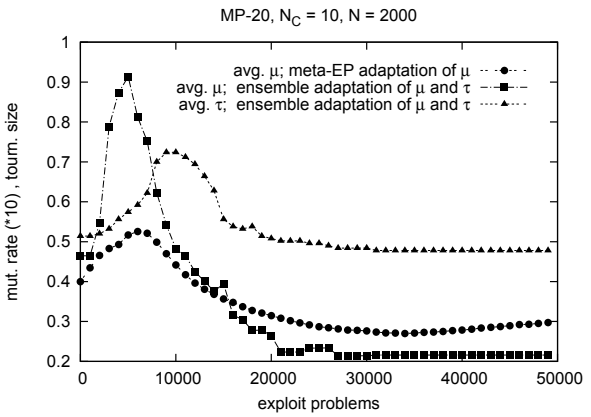
Figure 12(b) illustrates the influence of MEA on the diversity of the ensemble when the MP-20 problem is being solved. To measure the diversity, we used an entropy measure E (Kuncheva and Whitaker, 2003). By simplifying and adapting the formula (8) from (Kuncheva and Whitaker, 2003) to our model, we have

$$E = \frac{1}{Exploits} \sum_{i=0}^{Exploits} \frac{\min(L_0(i), L_1(i))}{N_C - \min(L_0(i), L_1(i))}, \quad (4)$$

where $Exploits$ represents the number of the most recent exploitation problems which are taken into account. The presented results are for $Exploits$ of 50. $L_0(i)$ and $L_1(i)$

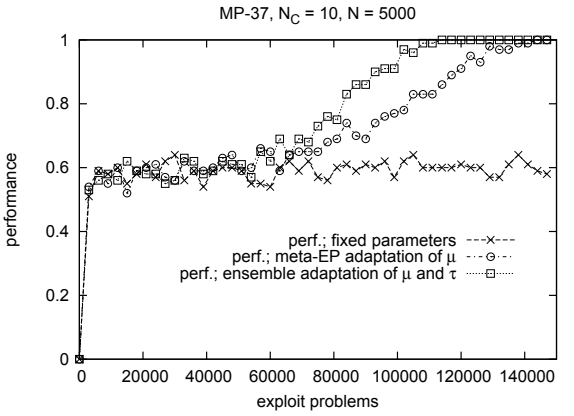


(a)

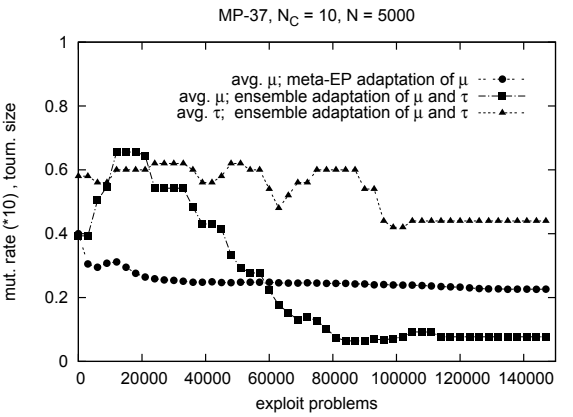


(b)

Fig. 4. Solving the MP-20 problem by the three ensemble architectures. The fixed values of parameters are $\mu = 0.04$, $\tau = 0.4$. The performance curves (a) and average values of adapted parameters (b) are presented.



(a)

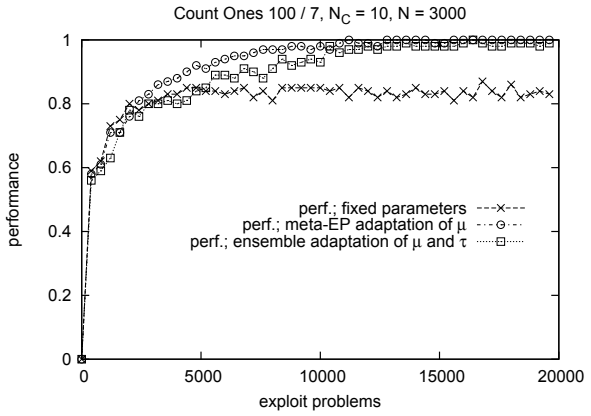


(b)

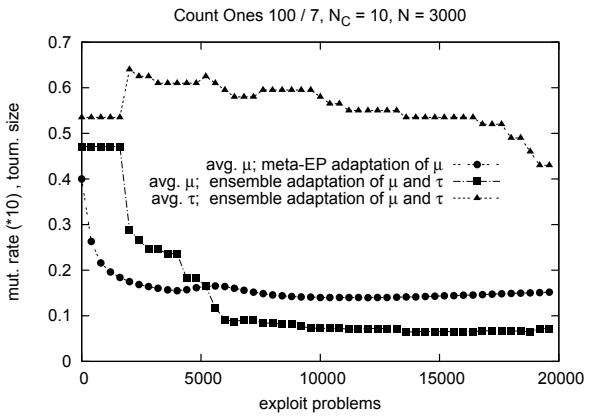
Fig. 5. Solving the MP-37 problem by the three ensemble architectures. N is 5000 in each XCS component. The fixed values of parameters are $\mu = 0.04$, $\tau = 0.4$. The performance curves (a) and average values of adapted parameters (b) are presented.

are the numbers of components choosing class 0 or 1, respectively, during the i-th exploit. Note that an entropy $E \in [0, 1]$, and it is the larger the larger the diversity in an ensemble (Kuncheva and Whitaker, 2003). Obviously, the diversity is maximal when both classes are chosen by the same number of components. The diversity decreases along with learning the problem, because the number of components which choose a correct class in an explore cycle increases. Nevertheless, the diversity in the ensemble with the MEA is lower than in the remaining two ensembles (without MEA) from the very beginning of learning (Fig. 12(b)). As has been shown (among other, in Fig. 4), this loss does not affect significantly the performance for $\theta_{MEA} = 0.06$, and therefore using the MEA is still reasonable. Lower θ_{MEA} results in more frequent component reproductions and lower diversity. In Fig. 13(a) the total number of reproductions is presented for three values of θ_{MEA} (0.006, 0.06, 0.12) when the MP-20 problem is being solved. For $\theta_{MEA} = 0.006$, more than a hundred of reproductions will be made before the ensemble achieves maximum performance, but the learning speed is not very high (Fig. 13(b)). In this case, the reproductions base on unreliable comparisons of components, because of a small minimal difference (θ_{MEA}) between their fluctuating fitness values. Many of these reproductions are unnecessary or even detrimental. On the other hand, if θ_{MEA} is set to 0.12, the probability of a "missed" reproduction is very low but the rare reproductions result in a slow adaptation of parameters (Fig. 13(b)). θ_{MEA} of 0.06 is a compromise which has occurred to be effective in the investigated problems.

Because the MEA operates on components, the ensemble size (N_C) is a critical factor, which determines algorithm effectiveness. It seems obvious that the more components, the more vectors of parameter values may be simultaneously evaluated. Moreover, the appropriate large ensemble prevents the adaptation process from stacking.

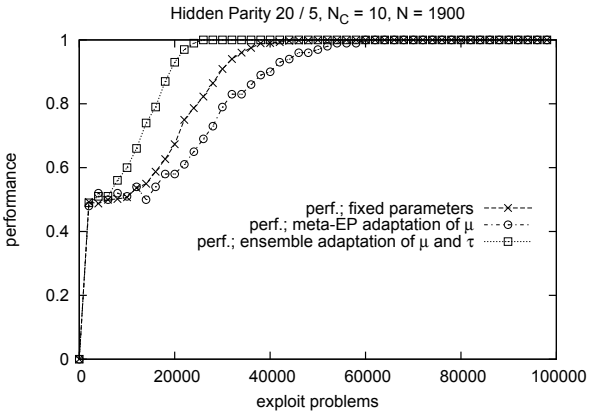


(a)

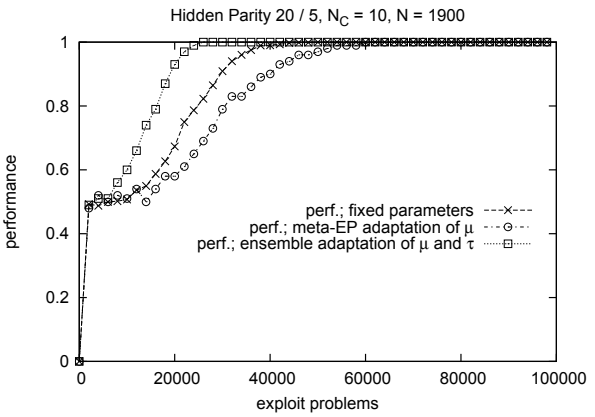


(b)

Fig. 6. Solving the count ones 100/7 problem by the three ensemble architectures. N is 3000 in each XCS component. The fixed values of parameters are $\mu = 0.04$, $\tau = 0.4$. The performance curves (a) and average values of adapted parameters (b) are presented.



(a)



(b)

Fig. 7. Solving the hidden parity 20/5 problem by the three ensemble architectures. N is 1900 in each XCS component. The fixed values of parameters are $\mu = 0.04$, $\tau = 0.4$. The performance curves (a) and average values of adapted parameters (b) are presented. The results are averaged over 50 independent trials.

Note that, if all the components in the ensemble have similar parameter values, they will be similarly fitted as well and the MEA will not be executed for a long time. The risk that such an undifferentiated component population will arise is the lower the higher the population size. In our model, this risk is additionally minimized by strong mutation of parameters.

In the following experiments, the ensembles consist of only three components. Note that N_C of 3 is a minimal configuration for which any voting can be done. The methods of parameter adaptation (the meta-EP and the MEA) are compared in these experiments. When the MP-37 problem is solved, parameter adaptation at the classifier level (meta-EP) gives significantly better results (Fig. 14). Note that this method is independent of the number of components. In the second experiment, the ensemble with the MEA seems to learn faster the hidden parity 20/5 problem (Fig. 15). Nevertheless, this ensemble achieves maximum average performance after more cycles than the ensemble with meta-EP achieves. As it has been explained before, when the MEA is used for the hidden parity problem, the parameter values in the initial component population play the key role. The components with high mutation rates, which have the biggest chances to find accurate classifiers, should be represented in the initial population. Nevertheless, if the mutation rate is initialized around the μ_0 value, the probability that no component has a mutation rate higher than μ_0, is 0.5^{Nc}. Note that, for $N_C = 10$, this probability is about 0.001, while for $N_C = 3$, it is as high as 0.125. That is why, occasionally, an ensemble with a MEA learns slower the hidden parity problem than an ensemble with a fixed or self-adaptive (at the classifier level) mutation rate. As has been shown, this risk decreases dramatically with the ensemble size N_C.

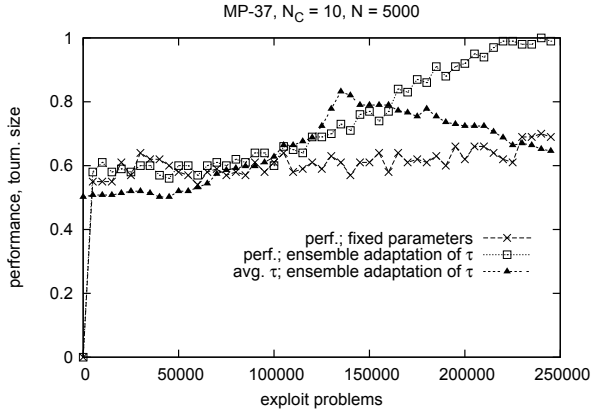
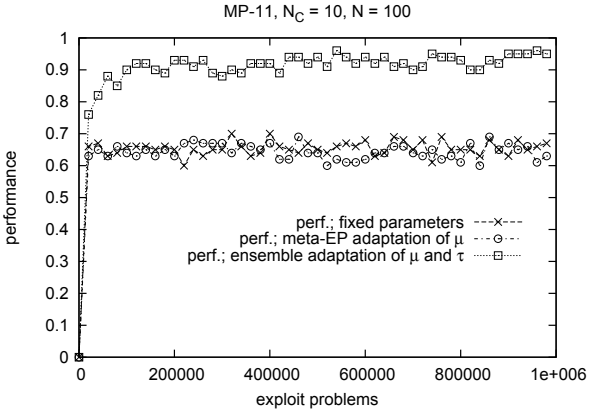


Fig. 8. Comparison of the adaptive tournament size and the fixed one ($\tau = 0.4$) in an ensemble solving the MP-37 problem. Fixed μ of 0.04 is used in both cases. The performance curves and the change of the average tournament size are shown.

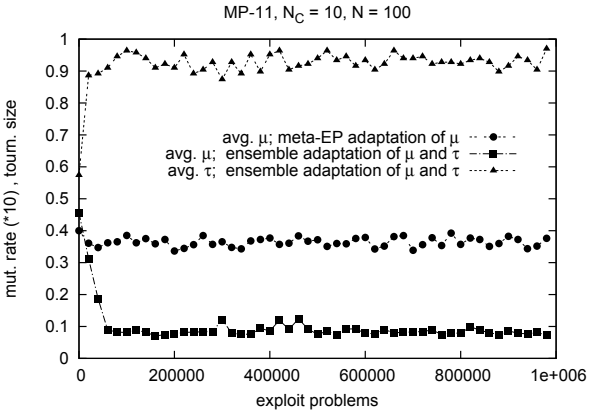
5. Summary and future work

In this work we have presented a model of an XCS-based ensemble machine, in which adaptive parameter control is performed by means of the MEA operating on components. A detailed experimental study demonstrated a possibility of adapting important XCS parameters, i.e., the mutation rate and the tournament size. The proposed model was compared with two other XCS-based ensembles: the ensemble with the self-adaptation of the mutation rate at the classifier level (meta-EP method) and the ensemble without any parameter adaptation. Advantages and disadvantages of both adaptive techniques (i.e., MEA and meta-EP) were discussed. Limitations of the MEA, which are caused by small component populations and difficulties with reliable evaluation of components, were analysed in detail. Despite these drawbacks, in most examined cases, the XCS-based ensemble with the MEA outperforms the compared architectures, and we believe that applying the MEA may make the XCS-based ensemble a more universal approach.

In future works, we will investigate how the migration of classifiers between components (proposed in (Bull *et al*., 2007)) reduces parameter sensitivity. We will try to join both the MEA and classifier migration in one XCS-based ensemble.



(a)



(b)

Fig. 9. Learning the MP-11 problem with small classifier populations ($N = 100$) in each component. Three architectures are compared. The fixed values of parameters are $\mu = 0.04$, $\tau = 0.4$. The performance curves (a) and average values of adapted parameters (b) are presented.

References

Bahler, D. and Navarro, L. (2000). Methods for combining heterogeneous sets of classifiers, *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI 2000), Workshop on New Research Problems for Machine Learning, Austin, TX, USA*, `http://www4.ncsu.edu/~bahler/aaai2000/aaai2000.pdf`.

Breiman, L. (1996). Bagging predictors, *Machine Learning* **24**(2): 123–140.

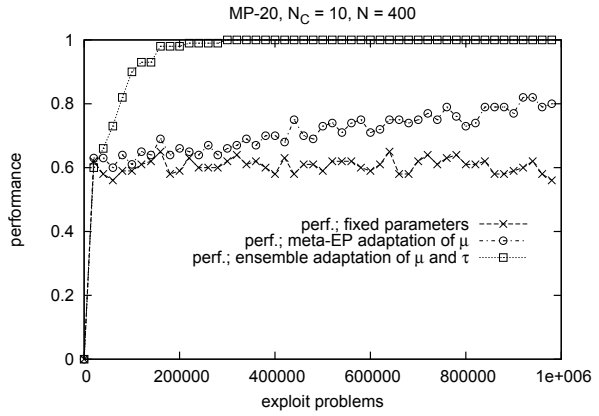
Bull, L., Mansilla, E. B. and Holmes, J. (Eds) (2008). *Learning Classifier Systems in Data Mining*, Springer, Berlin/Heidelberg.

Bull, L., Studley, M., Bagnall, A. and Whittley, I. (2007). Learning classifier system ensembles with rule-sharing, *IEEE Transactions on Evolutionary Computation* **11**(4): 496–502.

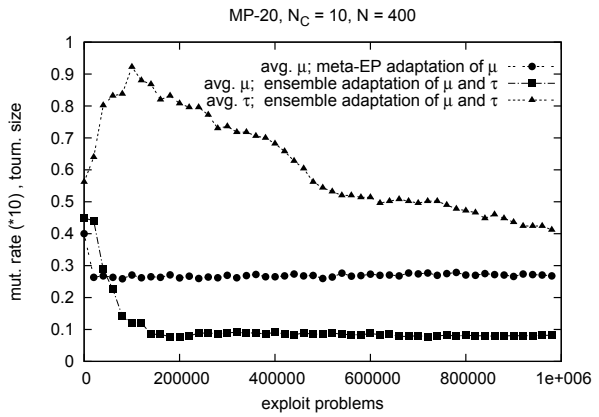
Butz, M. V. (1999). An implementation of the XCS classifier system in C, *Technical Report 99021*, Illinois Genetic Algorithms Laboratory, University of Illinois, Urbana-Champaign, IL.

Butz, M. V., Sastry, K., Goldberg, D. E. (2002). Tournament selection in XCS, *Technical report*, Proceedings of the Fifth Genetic and Evolutionary Computation Conference (GECCO-2003), pp. 1857–1869.

Butz, M. V., Goldberg, D. E. and Lanzi, P. L. (2005). Gradient descent methods in learning classifier systems: Improving

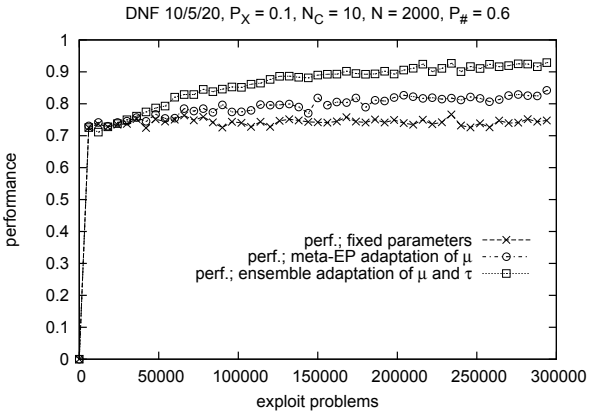


(a)

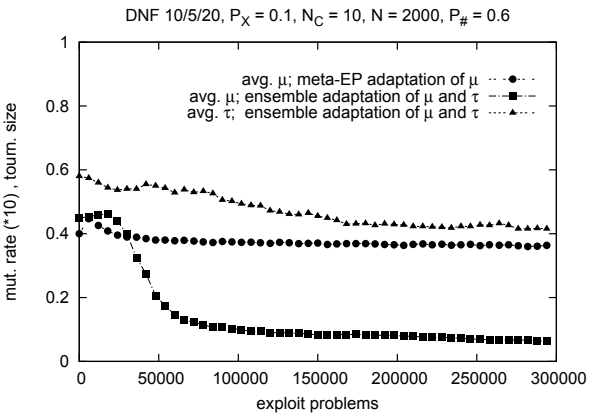


(b)

Fig. 10. Learning the MP-20 problem with small classifier populations ($N = 400$) in each component. Three architectures are compared. The fixed values of parameters are $\mu = 0.04$, $\tau = 0.4$. The performance curves (a) and average values of adapted parameters (b) are presented.



(a)



(b)

Fig. 11. Learning random Boolean functions in DNF representation (10 closures, k = 5, l = 20). The altering noise with $P_x = 0.1$ is applied. Three architectures are compared. The $P_\#$ parameter is set to 0.6 in every component. The fixed values of parameters are $\mu = 0.04$, $\tau = 0.4$. The performance curves (a) and average values of adapted parameters (b) are presented.

XCS performance in multistep problems, *IEEE Transactions on Evolutionary Computation* **9**(5): 452–473.

Butz, M. V., Goldberg, D. E. and Tharakunnel, K. (2003). Analysis and improvement of fitness exploitation in XCS: Bounding models, tournament selection, and bilateral accuracy, *Evolutionary Computation* **11**(3): 239–277.

Butz, M. V., Kovacs, T., Lanzi, P. L. and Wilson, S. W. (2004). Toward a theory of generalization and learning in XCS, *IEEE Transactions on Evolutionary Computation* **8**(1): 28–46.

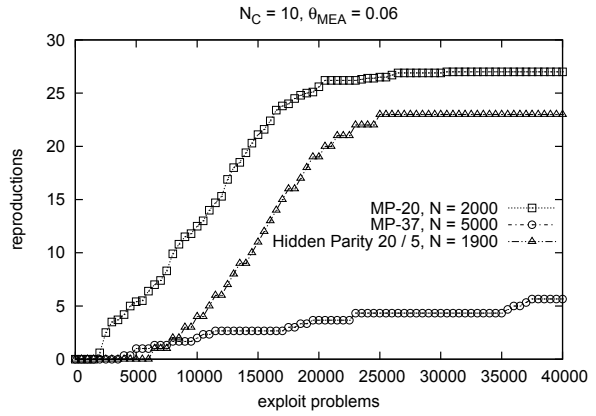
Butz, M. V. and Pelikan, M. (2001). Analyzing the evolutionary pressures in XCS, *in* L. Spector, E. Goodman, A. Wu, W. Langdon, H. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. Garzon, and E. Burke (Eds), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO2001)*, Morgan Kaufmann, San Francisco, CA, pp. 935–942.

Butz, M. V. and Pelikan, M. (2006). Studying XCS/BOA learning in boolean functions: Structure encoding and random boolean functions, *GECCO '06: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, Seattle, WA, USA*, pp. 1449–1456.

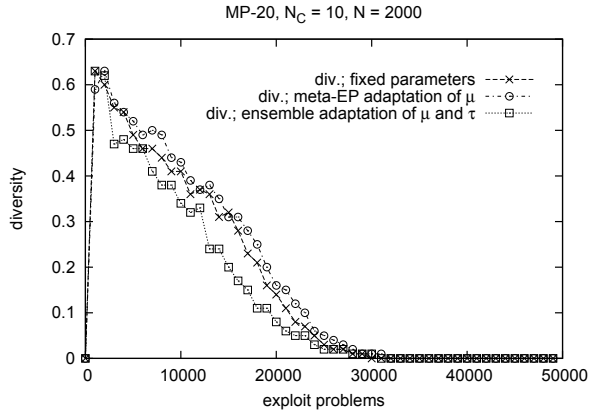
Dam, H. H., Abbass, H. A. and Lokan, C. (2005). DXCS: An XCS system for distributed data mining, *in* H.-G. Beyer and U.-M. O'Reilly (Eds), *GECCO*, ACM, New York, NY, pp. 1883–1890.

Dam, H. H., Lokan, C. and Abbass, H. A. (2007). Evolutionary online data mining: An investigation in a dynamic environment, *in* S. Yang, Y.-S. Ong and Y. Jin (Eds), *Evolutionary Computation in Dynamic and Uncertain Environments*, Studies in Computational Intelligence, Vol. 51, Springer, Berlin/Heidelberg, pp. 153–178.

Dawson, D. (2002). *Improving extended classifier system performance in resource-constrained configurations*, Master's

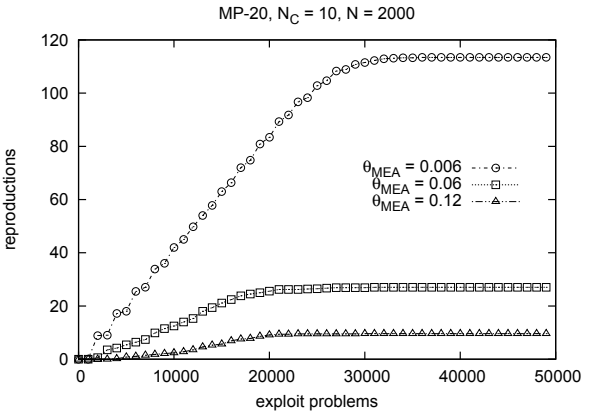


(a)

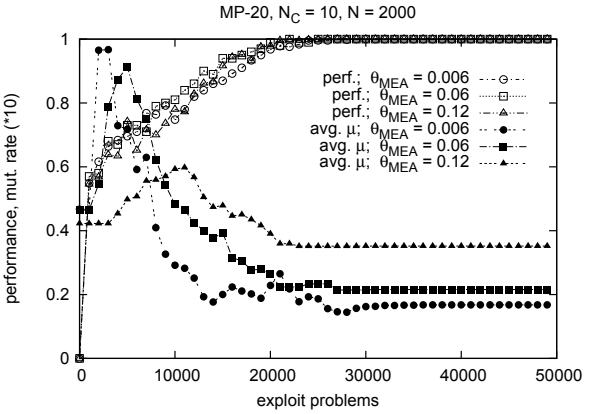


(b)

Fig. 12. Consequences of the MEA: the total number of component reproductions made by the MEA (in relation to the number of exploit trials) in the ensemble which solves the MP-20, thte MP-37 and hidden parity 20 / 5 problems (a), the diversity in the ensembles solving the MP-20 problem (b).



(a)



(b)

Fig. 13. Different values of the θ_{MEA} parameter in the adaptive ensemble solving the MP-20 problem. Performance, the average mutation rate and the total number of reproductions are presented for $\theta_{MEA} = 0.006, 0.06, 0.12$.

thesis, California State University, Chico, CA.

Dietterich, T. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization, *Machine Learning* **40**(2): 139–158.

Eiben, A., Schut, M. and de Wilde, A. (2006a). Boosting genetic algorithms with (self-) adaptive selection, *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2005), Vancouver, BC, Canada*, pp. 1584–1589.

Eiben, A., Schut, M. and de Wilde, A. (2006b). Is self-adaptation of selection pressure and population size possible? A case study, *in* T. Runarsson, H.-G. Beyer, E. Burke, J. J. Merelo-Guervs, L. D. Whitley and X. Yao (Eds), *Parallel Problem Solving from Nature (PPSN IX)*, Lecture Notes in Computer Science, Vol. 4193, Springer, Berlin/Heidelberg, pp. 900–909.

Fogel, D. B. (1992). *Evolving artificial intelligence*, Ph.D. thesis, US San Diego, La Jolla, CA.

Gao, Y., Huang, J. Z. and Wu, L. (2007). Learning classifier system ensemble and compact rule set, *Connection Science* **19**(4): 321–337.

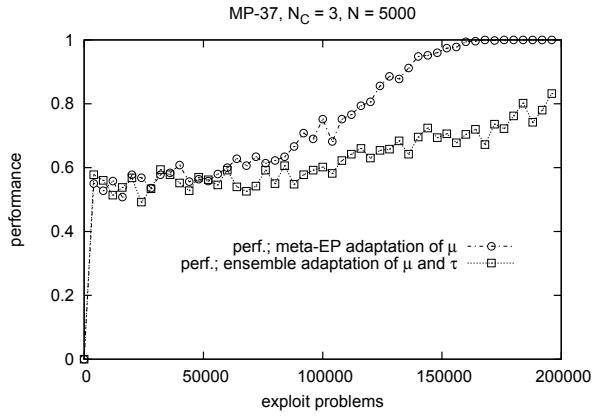
Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Professional, Reading, MA.

Grefenstette, J. J. (1986). Optimization of control parameters for genetic algorithms, *IEEE Transactions on Systems, Man, and Cybernetics* **SMC–16**(1): 122–128.

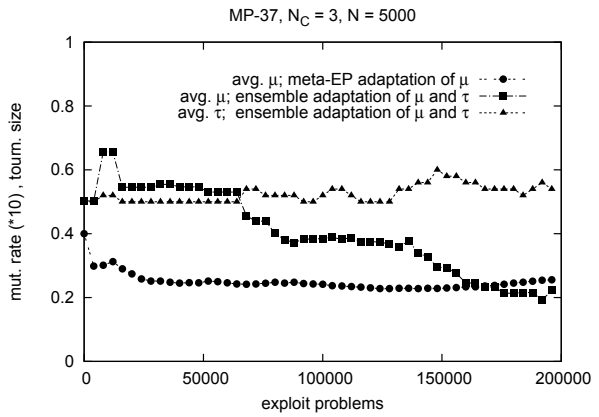
Holland, J. (1976). Adaptation, *in* R. Rosen (Ed.), *Progress in Theoretical Biology*, Plenum Press, New York, NY, pp. 263–293.

Holmes, J. H., Lanzi, P. L., Stolzmann, W. and Wilson, S. W. (2002). Learning classifier systems: New models, successful applications, *Information Processing Letters* **82**(1): 23–30.

Howard, D., Bull, L. and Lanzi, P. (2008). Self-adaptive constructivism in neural XCS and XCSF, *in* M. Keijzer, G. Antoniol, C. Congdon, K. Deb, N. Doerr, N. Hansen,

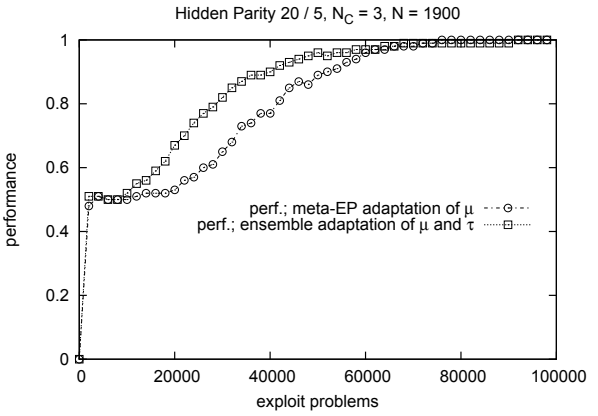


(a)

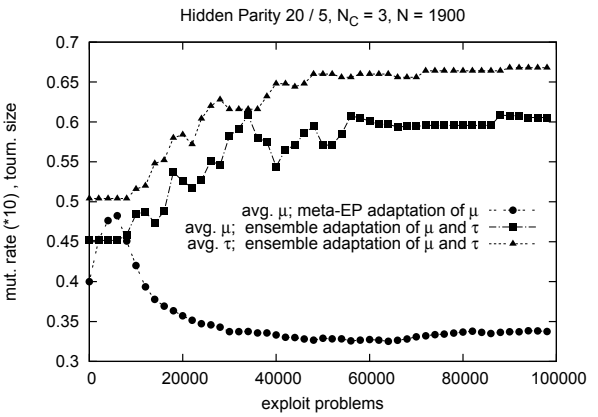


(b)

Fig. 14. Solving the MP-37 problem by ensembles consisting of only three components.



(a)



(b)

Fig. 15. Solving the hidden parity problem by ensembles consisting of three components. The results are averages from 50 independent trials.

J. Holmes, G. Hornby, D. Howard, J. Kennedy, S. Kumar and F. Lobo (Eds), *GECCO-2008: Proceedings of the Genetic and Evolutionary Computation Conference, Atlanta, GA, USA*, pp. 1389–1396.

Huang, C.-Y. and Sun, C.-T. (2004). Parameter adaptation within co-adaptive learning classifier systems, *in* K. Deb, R. Poli, W. Banzhaf, H.-G. Beyer, E. Burke, P. Darwen, D. Dasgupta, D. Floreano, J. Foster, M. Harman, O. Holland, P. L. Lanzi, L. Spector, A. Tettamanzi, D. Thierens and A. Tyrrell (Eds), *Genetic and Evolutionary Computation—GECCO-2004, Part II*, Lecture Notes in Computer Science, Vol. 3103, Springer-Verlag, Berlin/Heidelberg, pp. 774–784.

Hurst, J. and Bull, L. (2002). A self-adaptive XCS, *IWLCS '01: Revised Papers from the 4th International Workshop on Advances in Learning Classifier Systems*, Lecture Notes in Artificial Intelligence, Vol. 2321, Springer-Verlag, London, pp. 57–73.

Hurst, J. and Bull, L. (2003). Self-adaptation in classifier system controllers, *Artificial Life and Robotics* **5**(2): 109–119.

Kharbat, F., Bull, L. and Odeh, M. (2005). Revisiting genetic selection in the XCS learning classifier system, *Congress on Evolutionary Computation*, Vancouver, BC, Canada, pp. 2061–2068.

Kuncheva, L. I. and Whitaker, C. J. (2003). Measures of diversity in classifier ensembles, *Machine Learning* **51**(2): 181–207.

Llorà, X. and Sastry, K. (2006). Fast rule matching for learning classifier systems via vector instructions, *GECCO '06: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, Seattle, WA, USA*, pp. 1513–1520.

Meyer-Nieberg, S. and Beyer, H.-G. (2007). Self-adaptation in evolutionary algorithms, *in* F. G. Lobo, C. F. Lima and Z. Michalewicz (Eds), *Parameter Setting in Evolutionary Algorithms*, Springer, Berlin.

Opitz, D. and Maclin, R. (1999). Popular ensemble methods: An empirical study, *Journal of Artificial Intelligence Research* **11**: 169–198.

Opitz, D. W., Shavlik, J. W. and Shavlik, O. (1996). Actively searching for an effective neural-network ensemble, *Connection Science* **8**(3–4): 337–353.

Orriols-Puig, A., Bernado-Mansilla, E., Goldberg, D. E., Sastry, K. and Lanzi, P. L. (2009). Facetwise analysis of XCS for problems with class imbalances, *IEEE Transactions on Evolutionary Computation* **13**(5): 1093–1119.

Spears, W. M. (1995). Adapting crossover in evolutionary algorithms, *in* J. R. McDonnell, R. G. Reynolds and D. B. Fogel (Eds), *Proceedings of the Fourth Annual Conference on Evolutionary Programming, San Diego, CA, USA*, pp. 367–384.

Stout, M., Bacardit, J., Hirst, J. and Krasnogor, N. (2008a). Prediction of recursive convex hull class assignment for protein residues, *Bioinformatics* **24**(7): 916–923.

Stout, M., Bacardit, J., Hirst, J. and Krasnogor, N. (2008b). Prediction of topological contacts in proteins using learning classifier systems, *Journal of Soft Computing* **13**(3): 245–258.

Sutton, R. S. (1991). Reinforcement learning architectures for animats, *in* J. Meyer and S. W. Wilson (Eds), *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, MIT Press, Cambridge, MA, pp. 288–296.

Takashima, E., Murata, Y., Shibata, N. and Ito, M. (2003). Self adaptive island GA, *Proceedings of the 2003 Congress on Evolutionary Computation (CEC 2003), Newport Beach, CA, USA*, Vol. 2, pp. 1072–1079.

Tongchim, S. and Chongstitvatana, P. (2002). Parallel genetic algorithm with parameter adaptation, *Information Processing Letters* **82**(1): 47–54.

Troć, M. and Unold, O. (2008). Self-adaptation of parameters in a XCS-based ensemble machine, *Proceedings of the Eighth International Conference on Hybrid Intelligent Systems (HIS 2008), Barcelona, Spain*, pp. 893–898.

Tsoumakas, G., Katakis, I. and Vlahavas, I. (2004). Effective voting of heterogeneous classifiers, *Proceedings of the 15th European Conference on Machine Learning*, Lecture Notes in Artificial Intelligence, Vol. 3201, Springer, Berlin/Heidelberg, pp. 465–476.

Unold, O. and Tuszynski, K. (2008). Mining knowledge from data using anticipatory classifier system, *Knowledge-Based Systems* **21**(5): 363–370.

Widrow, B. and Hoff, M. E. (1960). Adaptive switching circuits, *1960 IRE WESCON Convention Record*, pp. 96–104.

Wilson, S. W. (1995). Classifier fitness based on accuracy, *Evolutionary Computation* **3**(2): 149–175.

Wilson, S. W. (2000). Get real! XCS with continuous-valued inputs, *in* P.L. Lanzi, W. Stolzmann, and S.W. Wilsin (Eds), *Learning Classifier Systems, From Foundations to Applications,* Lecture Notes in Artificial Intelligence, Vol. 1813, Springer-Verlag, Berlin/Heidelberg, pp. 209–219.

Maciej Troć received the M.Sc. degree in computer science from the Wrocław University of Technology in 2004. Presently he works on his Ph.D. thesis concerning adaptation of parameters in learning classifier systems. His scientific interests include self-adaptation in evolutionary algorithms, genetic-based machine learning, classifier ensembles, and evolutionary art. Professionally, he is interested in software engineering and system modeling.

Olgierd Unold is an assistant professor in the Institute of Computer Engineering, Control and Robotics of the Wrocław University of Technology. He received the M.Sc. degree in automation systems in 1989, the M.Sc. degree in information science in 1991, and the Ph.D. in computer science in 1994. His current research interest is the development of adaptive machine learning tools such as learning classifier systems and fuzzy-immune rule-based systems for grammatical inference and bioinformatics.

Received: 19 December 2008
Revised: 1 September 2009