Adaption of XCS to multi-learner predator/prey scenarios

[Extended Abstract]

Clemens Lode
clemens@lode.de

Urban Richter
Karlsruhe Institute of Technology
Institute AIFB
76128 Karlsruhe, Germany
urban.richter@kit.edu

Hartmut Schmeck
Karlsruhe Institute of Technology
Institute AIFB
76128 Karlsruhe, Germany
hartmut.schmeck@kit.edu

ABSTRACT

Learning classifier systems (LCSs) are rule-based evolutionary reinforcement learning systems. Today, especially variants of Wilson's *eXtended classifier system (XCS)* are widely applied for machine learning. Despite their widespread application, LCSs have drawbacks, e. g., in multi-learner scenarios, since the *Markov property* is not fulfilled. In such scenarios, typical challenges of research are: How to divide team reward among the individual learners and how to cope with the problem of parallel learning agents (coadaptation)?

In this paper, an instance of the generic homogeneous and (non-) communicating predator/prey scenario is used to adapt an XCS to such a scenario with concurrent and cooperating learners. Results show that improvements in learning can be achieved by cleverly adapting the *multi-step* approach to the characteristics of the investigated scenario. Firstly, the environmental reward function is expanded to include sensory information. Secondly, the learners are equipped with a memory to store and analyze the history of local movements and given payoffs.

1. INTRODUCTION

The complexity of today's technical systems is continuously increasing. Future systems will consist of a multitude of autonomous soft- and hardware components that interact with each other to satisfy functional requirements of the global system. Since this trend bears the risk of unpredictable or even uncontrollable system behavior, Organic Computing (OC)[1] focusses on monitoring, analyzing, and controlling complex distributed systems to endow them with the ability of *controlled self-organization.* I. e., an organic system can self-organize to achieve its tasks while being simultaneously observed and – if necessary – influenced by a higher level component to avoid unwanted emergent system states. To this end, an *observer/controller architecture* has been proposed in [17]. Since it is in general impossible to foresee all possible configurations the *system under observation and control* takes on in a dynamic environment, it is important to endow the system with learning components so that system components can learn autonomously new control actions for unforeseen situations and evaluate their consequences.

An XCS [21] is a rule-based evolutionary on-line learning system that is suitable to perform this learning task. It evolves a set of condition-action rules (called *classifiers*) that contain a reward prediction estimating the benefits of their action for situations matching their condition. In general, LCSs combine aspects of evolutionary algorithms that are used for rule generation with reinforcement learning that is used for reward prediction. If an agent reaches a goal, a reward will be distributed among the classifiers. Usually, this depends on the type of scenario: In *single-step* environments, a reward is generated at each learning iteration, while in a *multi-step* environments (they are often characterized by local and restricted knowledge) the agent will construct the global information using multiple repetitions and iterations of the learning problem.

Here, LCSs are used to learn control rules for a *predator/prey scenario* [2]. Following the global task to observe one or up to n (randomly) moving preys, a number of predators has to achieve a common goal with local and distributed behavior, e. g., maximizing the global observation time or catching moving prey(s). The scenario shows technical relevance to OC scenarios, it seems to be very flexible in its parametrization, and it allows many variations [18]. Especially, the challenge of concurrent learning agents is addressed, where all predators are equipped with a single, independent XCS.

Since neither the classical single-step nor the multi-step implementation of an XCS [6] can be used to learn a dynamic observation task (cf. Section 3.3), the special focus of this paper is on an approach, how to handle such dynamic predator/prey scenarios. Thereby, the proposed idea is mainly based on a local cooperative reward function and some kind of temporary memory, which stores past action sets and their corresponding reward values.

The remainder of this paper is structured as follows: First, LCSs are shortly introduced and their usability in different scenarios is discussed (see Section 2). In Section 3, the predator/prey example is described, which has not been ad-

[1]http://www.organic-computing.de/spp

dressed in XCS literature yet. In such scenarios, a new reward function is needed, which is explained in detail in Section 4. Using this new reward function, several experiments are executed and their results are discussed in Section 5. The paper concludes with a summary and a discussion of future work in Section 6.

2. LEARNING CLASSIFIER SYSTEMS

The field of LCSs, introduced in the 1970ies [7, 8, 9], is one of the most active and best-developed form of genetic based machine learning [12, 13, 15]. As mentioned above, much of LCS's theory is inherited from the reinforcement learning literature.

LCSs are rule-based on-line learning systems that combine nature-inspired optimization heuristics and reinforcement learning techniques to learn appropriate actions for any input they get [21]. They are applicable to any problem, where a numerical reward reflecting the quality of an action can be obtained. The core component of an LCS is its *rule base* that contains rules (called *classifiers*) consisting of a condition, an action, a prediction value, etc. The selection of an appropriate action for a given input is a two-step process. From the rule base of all classifiers a subset called *match set* is computed that contains all classifiers, whose condition matches the current input. Then, for all distinct actions present in the match set the average prediction of all classifiers advocating the same action is calculated. The action with the highest average prediction is selected for execution (which will be called *best selection*) and all classifiers in the match set advocating that action form the *action set*. The reward received from the environment is subsequently used to update the prediction values of all classifiers in the action set. One alternative of *best selection* is *tournament selection* [3] where actions with a lower average prediction have a probability to be selected, too.

Classifiers forming the rule base are created in two different ways: Whenever the match set is empty, classifiers consisting of a condition matching the current input, a random action, and a default prediction value are inserted into the rule base by a process called *covering*. Furthermore, a randomized *evolutionary component* occasionally selects relatively good classifiers to be the parent individuals for a reproduction cycle. Crossover and mutation are applied to copies of the parents to form offspring, which are inserted into the rule base.

A variety of different LCS implementations has been proposed (cf. [12]), many are based on Wilson's XCS [21] system, which is an LCS implementation that maintains separate prediction and fitness values and where the fitness of a classifier is based on the *accuracy* of the prediction reward.

In certain problems (see Section 2.1), the environment's reaction on an action executed by an LCS is not instantaneous. Thus, further reinforcement learning cycles are required to build up an optimal (local) reward function (see Section 2.2). Also, more complex scenarios with aliasing positions require additional handling like an internal memory (see Section 2.3). In dynamic scenarios, e. g., the predator/prey scenario, additional measures are required. This will be subject of the following sections.

2.1 Single-Step vs. Multi-Step Problems

Literature about LCSs could be divided into *single-step* and *multi-step* approaches. This separation bases on how to solve the reinforcement learning problem and addresses a design decision, which has to be taken when implementing an LCS. It refers to the question, *when* a reinforcement signal (reward) is achieved from the environment and *how* this reward is distributed on the past action(s).

In *single-step* environments, the external reward is received on every time step and the environmental input for each time step has completely been independent of the prior time step in the case of environments with only a single entity with an LCS. When a decision is made, the reinforcement is directly received and measures the quality of the decision. Single-step environments generally involve categorization of data examples. A typical single-step benchmark problem is the *boolean multiplexer problem* [5, 21].

In *multi-step* environments, the external reward may not necessarily be received on any time step, since the environmental input on a time step depends on at least the prior input and the system's last action. Typical multi-step environments are known as sequential environments or so-called *Maze* problems, e. g., *Wood1* [20] or *Wood2* [21]. These examples model the adaptive interaction of an agent with its environment and have been studied using a variety of methods. Most often, a Maze is defined as a given number of neighboring cells in a grid-world. A cell is a bounded space formally defined and is the elementary unit of a Maze. Cells are either empty or can contain an obstacle, food, a so-called *animat*, and eventually a predator of the animat.

An animat is randomly placed in the Maze environment (which could be some kind of a labyrinth) and it tries to set its position to a cell containing food, which is sparsely located in the environment. To perform this task, it possesses a limited perception of the environment (often limited to the eight cells surrounding the animat's position) and it can also only move to an empty neighboring cell. Moving step by step through the Maze in order to fulfil its goal, the animat searches for a strategy (or adopt a policy), which minimizes the effort undertaken to find the food in the selected Maze environment. Maze environments offer plenty of parameters that allow to evaluate the complexity of a given system and also to evaluate the efficiency of a learning method. A full description of these parameters is available in [1].

2.2 Learning in Markov Environments

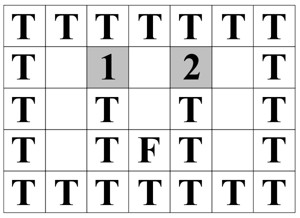
When the environment in question does have the Markov property, i. e., it is an environment, where the sensory data for an agent differs for each position, then an agent can acquire global information by visiting all positions of the environment. With this information, an LCS can find the optimal set of rules to find the goal on the shortest route from any starting position. The learning process itself is done by a random walk (the *explore* phase) in order to cover the search space, starting from a random position and repeating the process when reaching the goal position. Actions that lead to the goal are positively rewarded. Each action is rewarded by a portion of the reward value of the next action (or the maximal reward value when reaching the goal). The actual quality (the shortness of the path length) of the LCS is determined in the next phase (the *exploit* phase), where the agent only chooses actions with the highest product out of fitness and prediction values.

2.3 Non Markov Environments

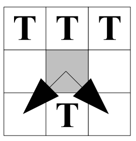
Some Maze problems offer perceptually identical situations that require different actions to reach the food. This problem is often studied in the context of *non Markov environments*. *Woods101* (see Figure 1(a)) is a typical example of a *partially observable Markov decision process (POMDP)*, where a *single-agent* cannot distinguish different situations due to a lack of global environmental information.

The animat is placed in a free random field, it can sense the eight adjacent cells, and has to learn to reach the food denoted with F, while trees T blocking any movement. Here, the aliasing positions marked with 1 and 2 share the same sensory configuration, but require different optimal actions (cf. Figure 1(b), each optimal action for the different positions is marked with an arrow). In position 1, the optimal action is to move to the bottom right, while the optimal action in position 2 is to move to the bottom left. Thus, an XCS cannot evolve an optimal policy for *Woods101* (without further modifications).

I. e., using records of past condition-action-mappings by adding temporary memory is a widespread approach to cope with such environments, as investigated in [14, 16].



(a) The aliasing positions are marked with 1 and 2.



(b) Sensory configuration of the aliasing positions

Figure 1: The *Woods101* environment: Trees are marked with *T*, food is marked with *F*.

A second non Markov property is still embedded in *multi-agent* environments and this is related to a change of an agent's internal state. In scenarios with more than one learning agent, an agent has to evaluate actions that may be caused by its own internal state or that are the result of other agent's actions. It is difficult to recognize an environmental change, which is caused by the change of another agent's internal state, due to a lack of the other agents' informations. Even if an agent stays in the same location, the agent cannot evaluate the environmental changes. In [19], this second non Markov property is defined as the *non observable Markov decision process (NOMDP)*.

In Section 3.3, it will be shown that the predator/prey scenario, presented in the following and used as testbed in this paper, includes both non Markov properties (POMDP and NOMDP). Thus, learning in such environments is more complex than learning in single-agent environments (where only one agent adapts to its dynamically changing environment) and requires a different approach, which is discussed in Section 4.

3. THE PREDATOR/PREY EXAMPLE

The predator/prey domain is an appropriate multi-agent example that has successfully been studied in a variety of instantiations. It does not serve as a complex real world domain, but as a test scenario for demonstrating and evaluating manifold research ideas. Introduced by [2], researchers have investigated different instantiations of its original formulation in the context of different application areas [18].

Both, predator and prey, typically can move into four different directions – north, east, south, and west. Mostly, predators follow a capturing strategy as a goal, while the prey randomly moves or stays still with a certain probability in order to simulate slower movements than the predators. A variation is that the prey moves faster than the predators. In this case, a strategic collaborative effort is required by the predators. An active escaping strategy, where the prey adapts and learns its behavior, may also be possible.

While predators and prey(s) have limited actions and follow well defined objectives, the predator/prey domain is simple to understand, easy to implement, and flexible enough to demonstrate a range of different scenarios, which have been emerged over the past decades.

Here, a simple implementation of the predator/prey scenario is examined, where the predators fulfil a cooperative task and are rewarded for just being very close to the prey. Thus, it is not necessary to surround and catch the prey. The overall scenario settings are described in Section 3.1 followed by a detailed description of the investigated obstacle configurations (see Section 3.2).

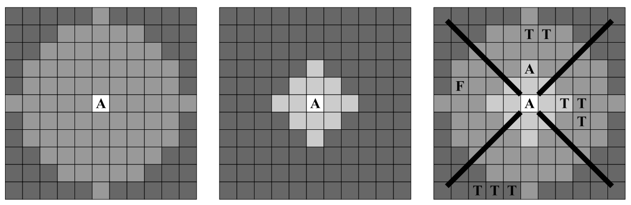
3.1 Scenario Settings

The different XCS implementations are tested in a simple predator/prey scenario on a discrete, quadratic, and toroidal world consisting of 16×16 squares. One non-learning prey randomly moves over the field. Eight adaptive predators do not follow a catching strategy, but cooperatively have to learn to keep the prey under observation. Thereby, every predator possesses its own XCS instance.

A cell of the two-dimensional grid-world can only be occupied by one agent. The quality of the observation task is determined by the amount of time any agent has the prey in its observation range. It is calculated by averaging the qualities of 100 experiments, each consisting of 10 runs. In detail, each run consists of 2 000 steps, after which the scenario is reset. The individual learning experiences of the agents are saved between each run, but not between each experiment.

In each time step, each agent can only move to one of the four neighboring fields, while the prey can move two fields, which allows for a faster movement. In addition, there are obstacles on the field and any movement to an occupied field fails (without any further consequences). Both, the prey and the predators, have 24 binary sensors that can sense their close environment, but their lines of sight can be blocked by objects. Half of the sensors can detect objects, which are two fields away, while the other half can detect objects up to five fields away (see Figure 2). The 12 sensors for each sight range are for the four directions and the three types of objects (prey, other agents, and obstacles). An example of a resulting sensor data string is shown in Figure 3.

The simulation is conducted in discrete time. At every time step, the goal object and the agents gather sensor data and decide their next actions. Then, all actions of all objects are executed in a random sequence. Since the prey can move two fields, it gathers new sensor data after the first step. If the prey is surrounded by agents in all four neighboring directions, it will jump to a random free field nearby, which basically means a restart of the simulation. Experiments have shown that this random jumping strategy only happens



(a) Sight range (b) Observation range (c) example with an agent, trees and food/prey

Figure 2: Sensor ranges of individual agents and a example situation: Obstacles/trees are marked with *T*, prey/food is marked with *F*, predators/agents are marked with *A*, and the observation and sight ranges are marked with light grey and grey color, respectively. Areas, which are out of sight of any predator, are marked with dark grey color.

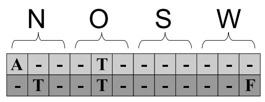


Figure 3: Matching sensor data string for example in Figure 2(c)

very seldom, i. e., it does not alter the simulation results significantly.

3.2 Scenario Configurations

Three different configurations of obstacles (trees) and starting positions of prey and predators have thoroughly been tested.

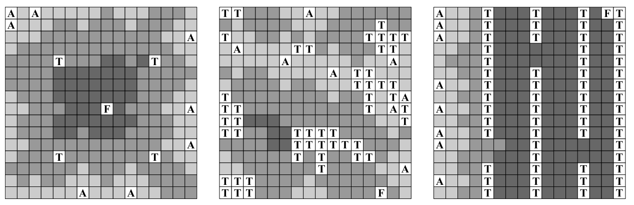

(a) *Pillar scenario* (b) *Random scenario* (c) *Difficult scenario*

Figure 4: Sample start configurations for the three types of scenarios: Obstacles/trees are marked with *T*, prey/food is marked with *F*, predators/agents are marked with *A*, and the observation and sight ranges are marked with light grey and grey color, respectively. Areas, which are out of sight of any predator, are marked with dark grey color.

In the *pillar scenario* (see Figure 4(a)), four obstacles are arranged in equal distance to each other, the prey starts in the middle of the whole field, and the predators starts randomly positioned along the borders. The idea of this scenario has been to use a minimal number of obstacles while still giving the agents some points of orientation.

In the second scenario, the *random scenario* (see Figure 4(b)), several obstacles are randomly distributed on the field with a certain tendency to build connected structures. In both scenarios, the *pillar scenario* and the *random scenario*, two kinds of prey implementations have been tested. The first one randomly moves, but moves away from predators when they become visible in sight. The second one uses a random walk strategy as well but tries to evade collisions with obstacles.

The third scenario, the *difficult scenario* (see Figure 4(c)), provides some kind of a labyrinth with several walls and small openings. Predators start on the left and have to find the openings that lead to the prey, while the prey only stays in the area on the right and continuously moves northward for two fields (or will jump to a free field nearby, if the direct path is blocked in northern direction).

3.3 Classification of the Scenario

As described in Section 2.3, environments can be classified as a MDP, a POMDP, or a NOMDP. The main characteristics of the predator/prey scenario, which is here investigated, could be summarized as follows:

1. An agent has only access to local information,
2. the whole field usually consists of open areas with randomly placed obstacles,
3. each agent has an internal state unknown to others,
4. the scenario is dynamic (agents act in parallel),
5. the agents (i. e., the predators) share and cooperatively contribute to a global observation task, and
6. the global observation task forces continuous agents' activities.

It is obvious that the three scenarios constitute being no MDPs, since sensory information is limited (1, 3) and different positions share the same sensory configuration (2). Moreover, adding the capability of storing information in a memory would not restore the Markov property (4), which let us conclude that the scenarios are also no POMDPs. Thus, this remains that the scenarios have to be NOMDPs.

Additionally, if each predator tries to learn its cooperative behavior using an XCS, a clearly defined (local) goal (5) will be needed to generate payoffs when agents have reached their goals. Furthermore, a typical multi-step scenario will be restarted, if an agent has reached its goal. But, the observation task is a continuous task (6). To conclude, a different approach is needed. Thus, a modified multi-step XCS variant is introduced in the following that can properly handle these issues.

In XCS literature, NOMDP environments are discussed in [11, 19]. There, the issue of non observability is either investigated by a very simplified scenario with only two agents or by using complex communication and central organization mechanisms, where all agents share a global LCS with global information. Both approaches seems not applicable here, since the three scenarios focus on more than two agents and (global) communication between the predators is not intended. Thus, new mechanisms are required to learn this cooperative and dynamic observation task using the XCS algorithm. The most important step to solve this problem is the design of an adequate reward function, which is discussed in the following section.

4. THE REWARD FUNCTION

The main prerequisite of an environment that can be solved by the single-step method is that the agent has global information. There, the optimal representation of the reward function by the XCS is also the solution of the actual problem. For example in the 6-multiplexer problem the reward function already contains the table of the 6-multiplexer itself.

In multi-step environments the environment only returns 1 when the agent stands on the goal position and 0 at all other positions. As there is only access to local information it is up to the individual agent to build up a reward function for all positions in order to be able to decide which direction is to be preferred at each step.

This is done by back propagating the reward of the environment to previous actions. During the back propagation the reward is discounted in order to favor shorter routes. When the goal position is reached the scenario is repeated a number of times. In order to find the optimal (shortest) route, i. e., the global reward function, the agent must be able to distinguish between all positions, i. e., the scenario must have the Markov property.

In the predator/prey scenario the nature of the reward function is not obvious. In such a dynamic environment with a continuous goal condition neither repetition is possible nor does that environment possess the Markov property. Other agents and the goal object may have moved and previously gathered information might be no longer valid.

Thus, global information and therefore a global reward function cannot be constructed and it is important to reward previous actions directly. In addition it is necessary to determine points in time when a reward is to be distributed because the simulation runs continuously.

In order to fulfil these requirements a new way of reward distribution for such a scenario has been developed. First the environment reward function is chosen by testing static strategies in the scenario (see Section 4.1). Agents then record the resulting reward values and create *events* when the values change or when there was no change for a specific amount of time (see Section 4.2). After that previously executed actions are rewarded according to the type of and the time difference between the current and the last event (see Section 4.3).

4.1 Environmental Reward Function

A "being in the goal position" in the context of a predator/prey scenario could be defined as "being in observation range of the goal object" which would be equal to the global goal. Another approach would be to make use of the increased abilities of the sensors of the agents instead of being restricted to a simple binary function as in some multi-step environments ("goal position reached" and "goal position not reached").

Because of the number of possible reward functions that include sensory information not all functions can be tested in a reasonable amount of time. So the focus was to explore the search space using static strategies modeled after a selection of reward functions and then test promising reward functions in connection with learning classifier systems. Although static strategies do not use a reward function, they still evaluate situations and corresponding actions either as "good" ("move in that direction") or as "bad" ("don't move in that direction"). The combinations of two main components have evolved out of these preliminary tests:

- Cooperation (goal object not in sight/observation range) : Move in a random direction without other agents in observation range.
- Egoism (goal object in sight/observation range): Move in the direction of the goal object.

Implementing a collaborative strategy in XCS would require multiple states. As the original implementation uses a binary reward function an approximation is used. The reward function returns 1 when there is no agent in observation range or when the goal object is in sight range (or when both is the case) and 0 otherwise.

As further adaptions to the reward is necessary the value this function returns will be called "base reward" and the function itself "environmental reward function". In the following section the actual reward for the rules of the XCS will be calculated.

4.2 Events

Above we concluded that it is important to determine points in time when the reward is distributed. In the usual implementation of XCS [6] this happens whenever a positive reward is generated and the scenario is then restarted. Here, we analyze past base reward values and generate so called *events* when either the *base reward* value has changed or when there was no change for a certain period of time.

Assuming that the agent did something right when it gets into sight range of the goal object (or leaving the sight range of all other agents) such a situation change will be called a "positive event" while loosing the goal object or getting into sight range of other agents will be called a "negative event". Thus a positive event occurs whenever the base reward changes from 0 to 1, a negative event occurs whenever the base reward changes from 1 to 0 (see Figure 5(a)).

As there can be cases where an agent never encounters an event the number of steps is limited to *maxStackSize* steps. If the step counter reaches that number then a "neutral event" occurs (see Figure 5(b)). In that case the step counter is reset to 0 and the classifier system waits for a new event or until the step counter again reaches the value of *maxStackSize*. In addition half of action sets in the stack are rewarded according to the *base reward* and then discarded.

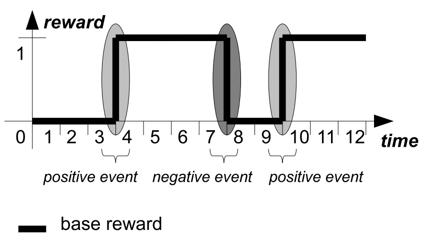
4.3 Reward Distribution

The standard implementation of XCS is based on the assumption that it learns within a MDP. This is expressed in the way the reward is distributed between the classifiers that contributed to reaching the goal. It requires several repetitions in order for the reward to be transferred to all classifiers that contributed to the solution.

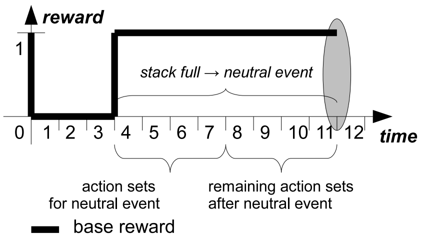
In a dynamic scenario such repetitions are not available, the scenario is not restarted and runs continuously. This is the reason why separate measures have to be taken in order to reward previous contributing steps as well.

The approach that was used in this paper is to record not only the last action but all past actions. Such a memory mechanism does not necessarily restore the Markov property because the scenario is a NOMDP. But it does restore a direct connection of the reward between the goal and previous steps that contributed to the success (or failure).

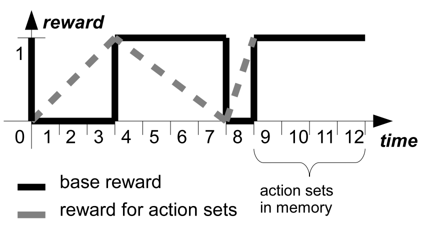
Thus, when (and only when) an event occurs, the reward is distributed among the entries of the action sets that were saved since the last event. With the idea in mind that recent actions probably contributed more to a positive (negative) event they are given a higher (lower) reward than to actions that were executed long time ago.



(a) Example of a series of base rewards that lead to positive and negative events.



(b) Example of a series of base rewards that lead to neutral event (with *maxStackSize* = 8).



(c) Reward distribution to the action sets over time after several positive and negative events

Figure 5: Calculation of the reward of individual action sets by analyzing the course of the base reward

For this a quadratic function which loosely resembles the transfer of the reward in the original implementation is used. With $r(a)$ being the *reward* for the *action set* with age a:

$$r(a) = \begin{cases} \frac{a^2}{\text{size}(\textit{action set})^2} & \textit{positive event} \\ \frac{(1-a)^2}{\text{size}(\textit{action set})^2} & \textit{negative event} \\ \textit{base reward} & \textit{neutral event} \end{cases}$$

Figure 5(c) shows a distribution of the reward for an exemplary distribution of the base reward. For simplicity a linear distribution is displayed in the other graphics. More sophisticated approaches are possible, this is merely the most straightforward approach. Compared to a linear distribution tests showed no significant difference.

5. EXPERIMENTAL RESULTS

Many different combinations of the reward function adaptions discussed above are possible. The main goal of this paper is to show that there are better implementations than XCS, not to find the best possible implementation. Therefore, this paper concentrates on the comparison of the XCS variants presented in the following Section 5.1.

In order to properly compare them it was important to determine good parameter settings for each variant. While most of the standard values given in [6] ("Commonly Used Parameter Settings") showed good results, some special settings need closer examination. Although thousands of different combinations were tested, the parameter discussion is not necessarily complete. Of main importance was that any score below the algorithm with randomized movements has to be dismissed as it is difficult to say if a parameter settings with better results (below the result of the random algorithm) just makes the agent move more randomly or if it actually learns better. The results of the parameter discussion is described in Section 5.2. Finally, in the subsequent Section 5.3, the results of a number of experiments are presented.

5.1 XCS Variants

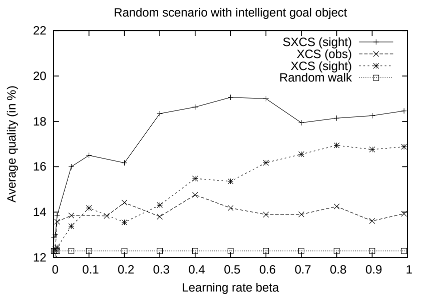
Many different combinations of the reward function options discussed in Section 4 are possible. Surprisingly no reward function option on its own showed a better performance. Using the collaborative environmental reward function from Section 4.1 even reduced the performance significantly. The best results showed a standard implementation of XCS with an environmental reward function that returns 1 when the goal object is in observation range and 0 at all other times. This will be referred to as "**XCS**".

Using events (see Section 4.2) in combination with the reward distribution based on the generated events (see Section 4.3) showed in some scenarios significant better performance than any of the other XCS variants. In addition it is able to perform well with any of the tested environmental reward functions while XCS only works with a reward function modeled after the global goal. This will be referred to as "**SXCS**" (*Supervising eXtended Classifier System*).

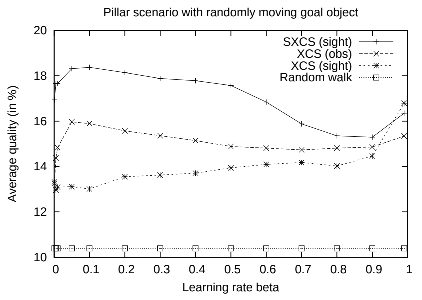
5.2 XCS/SXCS Parameters

The parameter *maxStackSize* that was introduced in section 4.2 determines when the stack overflows and a neutral event occurs. While further research is required, a relatively good value was determined by several experiments. Similar to XCS' prediction discount parameter γ the optimal value is a compromise between several conflicting factors: Larger values result in an inclusion of older - maybe irrelevant - actions in the reward of positive or negative events. Smaller values can reduce the delay between an event and the actual reward but they may also lead to a possible disregard of actions that were important in achieving the current event. As Figure 7(a) shows the optimal value depends on the complexity of the scenario. For comparison in all tests a value of 8 will be used.

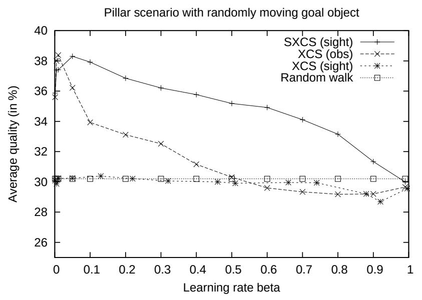
During the tests another important parameter was the learning rate β. In a similar type of scenario in [10] a value below the standard value was proposed ($\beta = 0.02$). The reason was that dynamic multi-agent systems can be described only by movement probabilities so the learning process has to be slow and careful. Tests (see Figure 6) showed an optimal value between 0.01 and 0.5 depending on the scenario. Very low values seem to always harm the learning process while larger values seem to harm the learning process in some scenarios while improving it in others (see Figure 7(b)). To maintain comparability between the scenarios and to other implementations of XCS a value of 0.05 was used for β in further tests. According to [6] the maximum number of classifiers N should be chosen big enough so that *covering* happens only in the beginning of a run. For the scenario in question tests have shown that a population size of around 512 fulfils this criteria. The classifier sets were filled with random classifiers [4] but no significant difference was seen. Instead sometimes a slower convergence was observed, probably because the corresponding system had to unlearn



(a) *Random scenario* with intelligent goal object

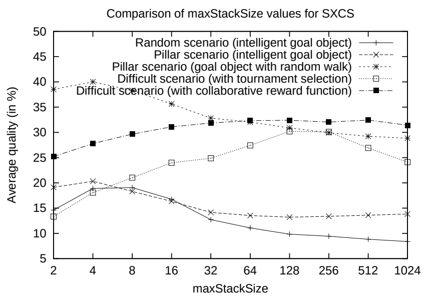


(b) *Pillar scenario* with intelligent goal object

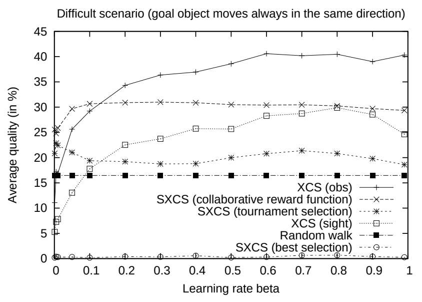


(c) *Pillar scenario* with randomly moving goal object

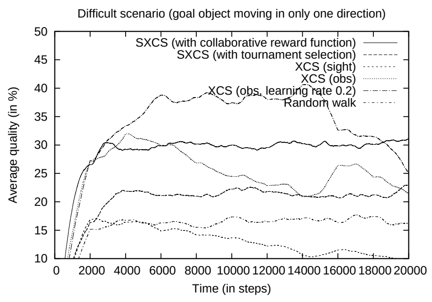
Figure 6: Comparison of different values for the learning rate β for different XCS variants



(a) Comparison of different *maxStack-Size* values for SXCS in different scenarios



(b) Comparison of different values for the learning rate β in the *difficult scenario*



(c) Comparison of the average quality over time in the *difficult scenario*

Figure 7: Comparison of different XCS variants

irrelevant classifiers. The *GA threshold* parameter θ_{GA} was set to 25, larger values seemed to reduce the quality of the algorithm. As SXCS itself does use the quadratic reward distribution, the parameter *reward prediction discount* γ is only needed to compare XCS with SXCS. Tests have been inconclusive so the standard value of $\gamma = 0.71$ was used. Only $\gamma = 1.0$ showed significant different results in some cases. It seems that while a reduction of the transfer of the reward is needed, the actual value is of little importance. Other parameters, like the subsumption threshold θ_{sub}, GA threshold θ_{GA} and the mutation probability μ values in the standard range were used (20.0, 25.0 and 0.05 respectively).

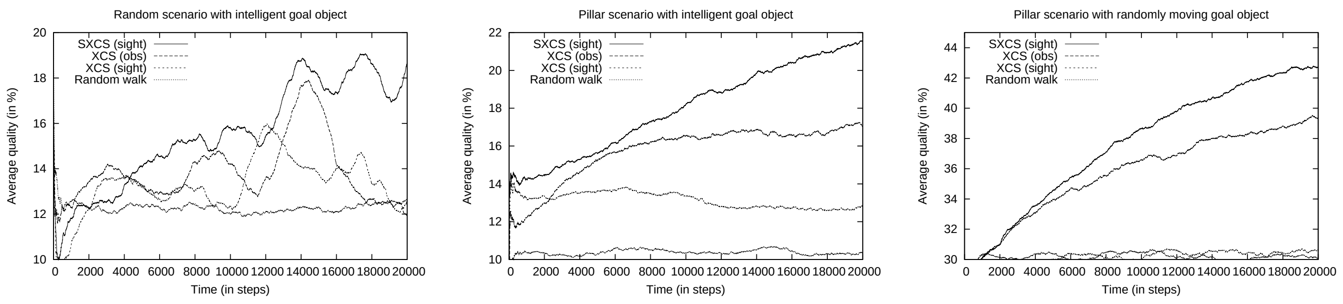
5.3 Experimental results

Figure 8(c), Figure 8(b) and Figure 8(a) show the average percentage of time steps where the goal object was in observation range (each averaged over the last 2000 steps). With an intelligent goal object SXCS clearly outperforms XCS, while in the case with the randomly moving goal object both variants show a similar learning curve. In all cases increasing the base reward function for XCS from observation range to sight range resulted in worse results.
The case with a randomly moving (but obstacle-evading) goal object in the *random scenario* is not displayed, none of the XCS variants are able to gain a significant advantage over a "random walk" strategy.

In the *difficult scenario* SXCS clearly fails (see Figure 7(b)). Alternative implementations of SXCS with *tournament selection* on the other hand solve the problem but reach only a lower level as XCS with *best selection* (see Figure 7(c)). But as in the *random scenario* with an intelligent goal object XCS seems to have problems with overlearning. Increasing the learning rate β to 0.2 increase the quality in short-term, but in long-term XCS still has problems while SXCS present a very stable learning curve. In connection with a collaborative base reward function SXCS is even able to outperform XCS in the long run.

6. CONCLUSION

In Section 5 it was shown that SXCS outperforms XCS in scenarios with few obstacles. This was mainly due to the fact that XCS is unable to handle base reward functions that differ from the global goal and because XCS seems to have problems reaching a stable level, possibly due to overlearning. SXCS with *best selection* showed serious problems in the *difficult scenario* while it was able manage the problem with a more random *tournament selection.* This points to a correctable flaw in the design of the algorithm (probably with the *neutral events*). But other than XCS SXCS *can* handle advanced base reward functions, e.g. with a collaborative element (evading other agents) which significantly helps for example in the *difficult scenario* though this is probably due to the fact that it simply encourages agents to move away from other agents, explore the grid and move through the openings. But in the end SXCS was designed to follow and



(a) *Random scenario* with intelligent goal object (b) *Pillar scenario* with intelligent goal object (c) *Pillar scenario* with randomly moving goal object

Figure 8: Comparison of the average quality over time of different XCS variants

observe a moving goal object, the *difficult scenario* is more like a labyrinth where finding the goal object in the first place is important.

Besides improving the SXCS algorithm itself future research could be pointed at a switching strategy between explore (when no goal object is in sight) and exploit phases (when it is near the goal object) and improvement can be expected from an adaptive *learning rate* and *maxStackSize* to fit the scenario's needs.

7. REFERENCES

[1] A. J. Bagnall and Z. V. Zatuchna. On the classification of Maze problems. In *Foundations of Learning Classifier Systems*, volume 183 of *Studies in Fuzziness and Soft Computing*, pages 305–316. Springer, 2005.

[2] M. Benda, V. Jagannathan, and R. Dodhiawala. An optimal cooperation of knowledge sources: An empirical investigation. Technical Report BCS-G2010–28, Boeing Advanced Technology Center, Boeing Computing Services, Seattle, Washington, United States of America, 1986.

[3] M. Butz, K. Sastry, and D. E. Goldberg. Tournament selection: Stable fitness pressure in xcs. In *GECCO*, pages 1857–1869, 2003.

[4] M. V. Butz. *The XCS Classifier System*, chapter 4, pages 51–64. Springer, 2006.

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

[6] M. V. Butz and S. W. Wilson. An algorithmic description of XCS. *Soft Computing*, 6(3–4):144–153, 2002.

[7] J. H. Holland. *Adaptation in natural and artificial systems.* University of Michigan Press, Ann Arbor, 1975.

[8] J. H. Holland. Adaptation. In *Progress in Theoretical Biology*, volume 4, pages 263–293. Academic Press, New York, NY, United States of America, 1976.

[9] J. H. Holland and J. S. Reitman. Cognitive systems based on adaptive algorithms. In *Pattern-Directed Inference Systems*, pages 313–329. Academic Press, 1978.

[10] H. Inoue, K. Takadama, and K. Shimohara. Exploring xcs in multiagent environments. In *GECCO '05: Proceedings of the 2005 workshops on Genetic and evolutionary computation*, pages 109–111, New York, NY, USA, 2005. ACM.

[11] S. K. K. Miyazaki, M. Yamamura. On the rationality of profit sharing in multi-agent reinforcement learning. In *Fourth International Conference on Computational Intelligence and Multimedia Applications*, pages 123–127, 2001.

[12] T. Kovacs. Learning classifier systems resources. *Soft Computing*, 6(3–4):240–243, 2002.

[13] T. Kovacs and P. L. Lanzi. A bigger learning classifier systems bibliography. In *Proceedings of the International Workshop on Learning Classifier Systems (IWLCS 2000)*, volume 1996 of *LNAI*, pages 213–249. Springer, 2000.

[14] P. L. Lanzi. Adding memory to XCS. In *Proceedings of the IEEE Conference on Evolutionary Computation (ICEC98).* IEEE Press, 1998.

[15] P. L. Lanzi. Learning classifier systems: Then and now. *Evolutionary Intelligence*, 1(1):63–82, 2008.

[16] P. L. Lanzi and S. W. Wilson. Toward optimal classifier system performance in non-markov environments. *Evolutionary Computation*, 8(4):393–418, 2000.

[17] U. Richter, M. Mnif, J. Branke, C. Müller-Schloer, and H. Schmeck. Towards a generic observer/controller architecture for organic computing. In *INFORMATIK 2006 – Informatik für Menschen!*, volume P-93 of *GI-Edition – Lecture Notes in Informatics (LNI)*, pages 112–119. Bonner Köllen Verlag, 2006.

[18] P. Stone and M. Veloso. Multi-agent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, 2000.

[19] K. Takadama, T. Terano, and K. Shimohara. Learning classifier systems meet multi-agent environments. In *Advances in Learning Classifier Systems*, volume 1996 of *LNAI*, pages 192–212. Springer, 2001.

[20] S. W. Wilson. ZCS: A zeroth level classifier system. *Evolutionary Computation*, 2(1):1–18, 1994.

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