Adaption of XCS to multi-learner predator/prey scenarios

[Track: Genetics Based Machine Learning]

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 this paper, LCSs are investigated in an instance of the generic homogeneous and non-communicating predator/prey scenario. A group of predators collaboratively observe a (randomly) moving prey as long as possible, where each predator is equipped with a single, independent XCS. Results show that improvements in learning are achieved by cleverly adapting a *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 actions and given payoffs.

Categories and Subject Descriptors

I.5.2 [**Design Methodology**]: Classifier design and evaluation

General Terms

Algorithms, theory

Keywords

Agent cooperation, evolutionary learning, XCS, multi-agent learning, coordination task, non-Markovian environments

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* [17] focuses on monitoring, analyzing, and controlling complex distributed systems to endow them with the ability of *controlled self-organization*. 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 autonomously learn 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-mappings (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, which are used for rule generation, with reinforcement learning, which is used for reward prediction (e. g., 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 *multi-step* environments (which are often characterized by local and restricted knowledge) multiple steps are usually required to solve a problem and to generate a reward.

Here, LCSs are used to learn control rules for agents in 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 (i. e., maximizing the global *observation time*). The scenario shows technical relevance to organic computing scenarios concerning the learning aspect, it seems to be very flexible in its parametrization, and it allows many variations [18]. Thus, the challenge of concurrent learning agents is specially 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* (see argumentation in Section 2.5), 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 a *temporary memory*. This memory stores past action sets and their corresponding reward values. Although LCSs have been investigated in dynamic environments [14, 16], specifically predator/prey scenarios have not really been

GECCO '10 Portland, Oregon USA

addressed in XCS literature.

Thus, the remainder of this paper is structured as follows: First, LCSs are shortly introduced and their usability in different scenarios is discussed in Section 2. Additionally, the characteristics of a predator/prey scenario are presented, which require a new reward function, as introduced in Section 3. Using this new reward function, several experiments are executed according to the methodology, as described in Section 4. Experimental results are presented 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. Hence, 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 a LCS is its *rule base* that contains rules (called *classifiers*) consisting mainly of a condition, an action, and a prediction value. 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, which 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. After that, the action with the highest average prediction is selected for execution (called *best selection*) and all classifiers in the match set advocating that action form the *action set*. An alternative of *best selection* is *tournament selection* [5], where the probability of being selected depends on the average prediction value. Finally, the reward received from the environment is subsequently used to update the prediction values of all classifiers in the action set.

Classifiers forming the rule base are created in two ways: Whenever the match set is empty, classifiers are inserted into the rule base by a process called *covering*, which consists of a condition that matches the current input, a random action, and a default prediction value. Furthermore, a randomized *evolutionary component* occasionally selects 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 is 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 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 to 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 predator/prey scenarios, additional measures are required, as discussed in the following.

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).

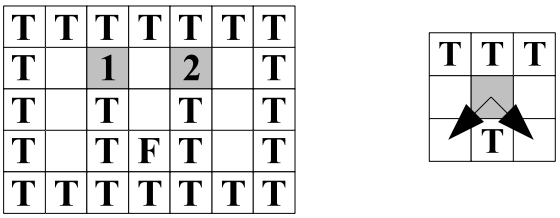
In *single-step* environments having a single (centralized) learning instance, the external reward is received on every time step and the environmental input for each time step has completely been independent of the prior time step. When a decision is made, the reinforcement is directly received and 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* [4, 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 a single-agent with its static 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, formally defined space and it is the elementary unit of a Maze. Cells are either empty or can contain an obstacle, food, a so-called *animat*, or eventually a predator of the animat. Then, an animat is randomly placed in the Maze environment (which could be some kind of 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 move to an empty neighboring cell. Moving step by step through the Maze in order to fulfill its goal, the animat searches for a strategy (or adopts 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 the efficiency of a learning method (cf. [1]).

2.2 Learning in Markov Environments

When the sensory data from the view of an agent differs for each position in the investigated Maze environment, the Markov property is fulfilled (i. e., an agent can acquire global information to solve the whole learning problem by visiting all positions of the environment). Having global information, an LCS can learn the optimal set of rules to find the goal position from any starting position by computing the shortest route. 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 position are positively rewarded by the maximal reward value. Other actions are rewarded by a portion of the reward value of the next action. Then, 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



(a) 1 and 2 mark the two aliasing positions.

(b) Sensory configuration of the two aliasing positions

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

However, 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 an animat 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 (see 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). Using records of past condition-action-mappings by adding temporary memory is a common approach to cope with such environments, see [14, 16].

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 agent's information. Even if an agent stays in the same location, the agent cannot evaluate the environmental changes. Moreover, memory-based approaches will fail, since any memory, which stores information about the environment, becomes invalid after each step. This second non-Markov property is often defined as a *non-observable Markov decision process (NOMDP)*, see [19].

As shown in Section 2.5, 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, as discussed in Section 3.

2.4 The Predator/Prey Example

The predator/prey domain, introduced by [2], is an appropriate multi-agent example that has successfully been studied in a variety of instantiations [18]. It does not serve as a complex real world domain, but as a test scenario for demonstrating and evaluating manifold research ideas. 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. Here, a simple instantiation of the predator/prey scenario is examined. Predators fulfill a *cooperative observation task*, where agents can sense other agents, but not communicate with each other. Then, predators are rewarded for the amount of time just being very close to the prey (i. e., it is not necessary to surround and catch it).

2.5 Classification of the Example

Environments can be classified as an MDP, an POMDP, or an 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, as intended in Figure 5, constitute being no MDPs, since agents' sensory information is limited (1, 3) and different positions share the same sensory configuration (2). Moreover, adding the capability of storing information would not restore the Markov property (4), as demonstrated on *Woods101* in [14, 16]. This let us conclude that the scenarios could not be any POMDPs. Finally, this also 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 XCS 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 establishing complex communication and centralized organization mechanisms, where all agents share and contribute to a global LCS with global information. Both approaches seem not applicable here, since the investigated

scenarios focus on more than two agents and (global) communication between the predators has not been intended. Thus, new mechanisms are required to learn this collaborative and dynamic observation task using the XCS algorithm. The most important step to solve this problem is the design of an adequate, *cooperative reward function*.

3. THE REWARD FUNCTION

Learning in single-step environments mainly requires that the agent has global information about the whole learning task, i. e., the optimal representation of an XCS's *reward function* is also the solution of the actual problem. In the 6-multiplexer problem, the reward function already contains the table of the 6-multiplexer itself, which provides a straightforward evaluation of classifiers.

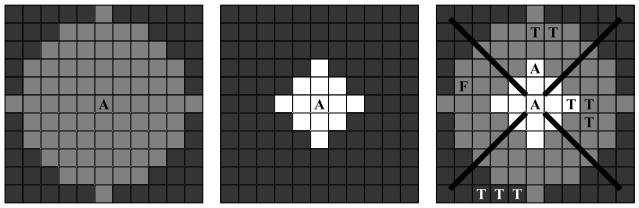
In multi-step environments, the environment will only return a reward value equal to 1, if the animat reaches the goal position and 0 at all other positions. Since an animat has only access to local information, it is the individual agent's task to compute different reward values for all possible positions in a Maze to differentiate, which movement is preferred at each step. In general, this is achieved by *back propagating* the reward of the environment to previous actions. The reward is discounted in order to favor shorter routes. If the animat reaches the goal position, the scenario will be repeated a number of times. In other words, 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 fulfill the Markov property).

In the predator/prey scenario, the nature of the reward function is not obvious. Since the prey continuously moves, repetition of the learning process seems not possible. Moreover, the environment does not possess the Markov property. Since all agents move, decide, and learn in parallel, previously gathered information does not seem valid any longer. Thus, global information and therefore a global reward function cannot be constructed from the local agent's view. Still, previous actions require a direct reward process. Additionally, points in time have to be determined, at which a reward is distributed on past condition-action-mappings.

To fulfill these requirements, a new mechanism of reward distribution in such dynamic predator/prey scenarios is proposed. Firstly, the sensor abilities of individual agents are presented in Section 3.1. Secondly, an expansion of the reward function is implemented by including these sensory information (see Section 3.2). Then, predators will record the resulting reward values of previous actions and will create *events*, if the reward values change or if no change occurs for a specific amount of time (see Section 3.3). Finally, previously executed actions will be rewarded according to the *type* of the event and the *time difference* between events (see Section 3.4). We explain this modified multi-step learning in the following.

3.1 Sensory Abilities

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, as depicted in Figure 2. Thus, 12 binary-coded sensors are used for both sight and observation range to encode every possible situation, as shown in Figure 3. Three bits are used to



(a) Sight range (b) Observation range (c) Example situation

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

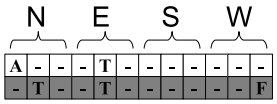


Figure 3: Matching sensor data string of the centralized agent, as depicted in Figure 2(c)

characterize a specific situation (prey, other predators, and obstacles) for each direction (north, east, south, and west).

3.2 Environmental Reward Function

In typical Maze environments, an animat will be rewarded by the environment, if it reaches a sparsely distributed goal position. In the context of a predator/prey scenario implementing a dynamic observation task, the state *reaching the goal position* could be interpreted as *having the prey in the observation range*. Then, the local goal would be equal to the global goal from a single-agent's view. Another approach could utilize more complex sensor data on an agent's level instead of being restricted to using a reward function modeled after the global goal (e. g., a *goal position is reached* and the *goal position is not reached*). Since the number of possible reward functions, which include more sensor data, seems unlimited and not all functions can be tested in a reasonable amount of time, the investigated approach follows a simpler idea. The search space is explored by using *static strategies* and then promising strategies are tested on learning predators. In the following, a selection of three static predator strategies is presented.

- ***Selfish behavior (observation range):*** Move towards the prey, when it is in the ***observation range,*** otherwise move in a randomly chosen direction.
- ***Selfish behavior (sight range):*** Move towards the prey, when it is in the ***sight range,*** otherwise move in a randomly chosen direction.
- ***Cooperative behavior:*** Move towards the prey, when it is in the ***sight range,*** otherwise move in a randomly chosen direction without having other agents in the individual ***observation range.***

Although static strategies do not use a reward function, they still evaluate condition-action-mappings either as *good*

(i. e., move in that direction) or as *bad* (i. e., do not move in that direction). Thus, an appropriate reward function can be implemented. We define the return value of such a reward function as the *base reward* and the function itself as the *environmental reward function*.

While the implementation of a *selfish behavior* strategy with only one prey on the field is trivial, the *cooperative behavior* strategy requires multiple return values of the reward function (e. g., situations with only one agent should be rewarded better than situations with three agents in observation sight). Since the typical XCS implementation [6] uses only a binary-coded reward function, an approximation will be used to differentiate between these different system states: The reward function will return 1, if no other predator is in the individual observation range or if the prey is in the individual sight range (or both will be true), and 0 otherwise.

3.3 Events

In the usual XCS implementation [6], any positive *base reward* is distributed, whenever an animat reaches a goal position 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 no change occurs for a certain period of time.

Assuming that a predator has taken a good decision, when the prey comes into the sight range of a predator (or the predator leaves the sight ranges of all other predators), we define such a situation change as a *positive event*. Moreover, loosing the prey or coming into the sight ranges of other predators will be defined as a *negative event*. In other words, a positive event always occurs, whenever the *base reward* changes from 0 to 1, and a negative event occurs, whenever the *base reward* changes from 1 to 0 (see Figure 4(a)).

Nevertheless, a predator also contributes to a cooperative group behavior, if the prey is never seen in the personal observation range and other predators are also stayed away. Hence, a predator will never encounter an event and no reward will be computed to evaluate past actions. Thus, the number of learning cycles – that can occur without encountering an event – have been limited by a variable `maxStackSize`. If this step counter reaches a predefined threshold, a *neutral event* will occur (see Figure 4(b)). Then, the step counter is reset to 0 and the LCS waits for a new event or the step counter again passes the threshold of the `maxStackSize`. Additionally, half of the action sets in the stack receive rewards according to the actual *base reward* value and are then discarded.

3.4 Reward Distribution

As already discussed, the standard XCS implementation is based on the assumption that it learns within an MDP. This is expressed in the way the reward is distributed on the classifiers that have contributed to reach the goal. Generally, several repetitions are required to correctly distribute the reward to all classifiers that have contributed to the solution.

In a dynamic scenario, such repetitions are not possible. The scenario is not restarted and the *observation task* is continuously performed. Thus, a separate mechanism is introduced to reward previous (contributing) action sets as well. This new approach stores not only the last, but all previous action sets. We admit that such a memory mechanism does not necessarily restore the Markov property (the scenario is obviously an NOMDP). However, it directly associates a changing base reward value with previous steps, which have contributed to the success (or to a failure), since the last change of the *base reward* has been observed. Thus, when, and only when, an event occurs, the reward will be distributed among the entries of the action sets that have been stored, since the last event has been occurred. Moreover, recent action sets have probably contributed more to a positive (negative) event than older action sets. Hence, they are evaluated with a higher (lower) reward than those action sets that were executed several steps before. To realize this *reward distribution* mechanism, a quadratic function is used, which extends the rewarding procedure of the original XCS implementation with $r(a)$ being the *reward* for the *action set* with an *age* of a:

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

In Figure 4(c), an example illustrates this reward distribution mechanism. To make it simple as possible, a linear distribution of the reward on previous action sets is displayed, but more sophisticated approaches seems possible. However, significant differences between a linear or a nonlinear distribution have not been investigated so far.

4. METHODOLOGY

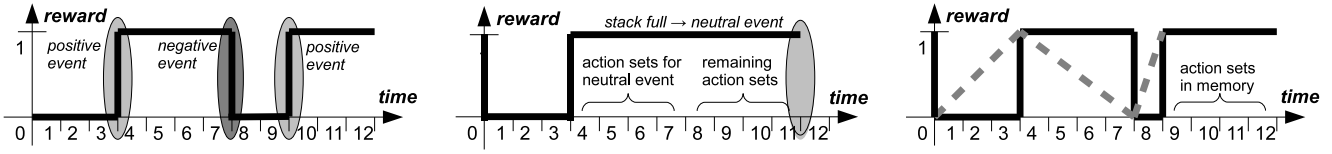
To analyze this new XCS approach, the implementation of the XCS variants is explained in Section 4.1, the specific scenario settings are presented in Section 4.2, and the obstacle configurations are discussed in Section 4.3.

4.1 XCS Variants

All here investigated XCS variants have in common that the corresponding scenario will not be restarted, if a positive reward is attained (as it appears in the standard XCS implementation). The first variant, which will be referred to as **XCS (obs)**, is equal to the standard implementation in all other implementation aspects, i. e., the global goal is equivalent to the local goal and predators act according to the *selfish behavior (observation range)* strategy (see Section 3.2). Including more sensory information to the predators' behavior by using the *selfish behavior (sight range)* strategy is denoted with **XCS (sight)**. Combining this variant with the event handling mechanism (see Section 3.3) and using a reward distribution based on these generated events (see Section 3.4) is referred by **eventXCS** as a third variant. Replacing also the selection strategy *best selection* (see Section 2) with *tournament selection* (with $p = 0.84$, see [5]) results in an XCS variant, as denoted with **eventXCS (ts)**. Finally, the usage of the *cooperative behavior* strategy is marked with **eventXCS (coop, ts)**.

4.2 Scenario Settings

The different XCS implementations are tested on a discrete, quadratic and toroidal world consisting of 16×16 squares (since bigger grids require more processing power). A cell of the two-dimensional grid-world can only be occupied by one agent. Eight adaptive predators follow an observation strategy (i. e., they cooperatively have to learn to keep the moving, non-learning prey under observation). Thereby, every predator possesses its own XCS instance.



(a) Example of a series of *base rewards*, which lead to both *positive* and *negative events*.

(b) Example of a series of *base rewards*, which force a *neutral event* (`maxStackSize` = 8).

(c) Reward distribution mechanism for a number of *positive* and *negative events*

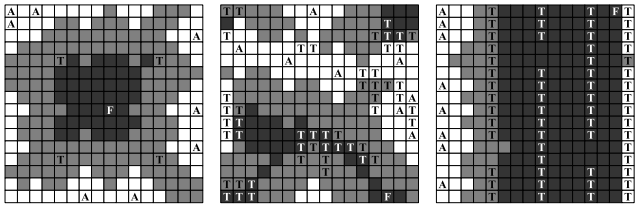
Figure 4: Calculation of the reward of individual action sets by analyzing the *base reward*

The *average quality* of the observation task is determined by the amount of time any predator has the prey in its *observation range*. It is calculated by averaging the qualities of 100 experiments, each consisting of 10 runs. The *continuous average quality* denotes the average quality of the last 2 000 steps only. 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 predator 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, obstacles are on the field and any movement to an occupied field fails (without consequences).

The simulation is conducted in discrete time. At every time step, the prey and the predators gather sensor data and decide their next actions. Then, all actions of all agents are executed in a random sequence. Since the prey can move two fields per each simulation step, it gathers new sensor data after the first step. If the prey is surrounded by predators 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 very seldom (i. e., it does not significantly alter the simulation results).

4.3 Scenario Configurations

Three configurations of obstacles (trees) and starting positions of prey and predators have thoroughly been tested. In the *pillar scenario* (see Figure 5(a)), four obstacles are arranged in equal distance to each other, the prey starts in the middle of the whole field, and the predators start randomly positioned along the borders. The idea 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 5(b)), several obstacles are randomly distributed on the whole grid 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 evaluated. The first one tries to move away from predators (the *predator-evading prey*), while the second one tries to evade collisions with obstacles (the *obstacle-evading prey*). If there are several alternatives, both types will move into a random direction, in which the prey does not see any predators or obstacles. The third scenario, the *difficult scenario* (see Figure 5(c)), provides a simple Maze with walls and openings. Predators start on the left and have to find the openings that lead them into the direction of the prey, while the prey only stays in the area on the right by moving only northward and evading an agent if necessary. Since this prey ignores sensory information, it will be referred to as the *blinded prey*. While this constraint makes the observation task easier, this scenario focuses on the agents' ability to find a way through a Maze.



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

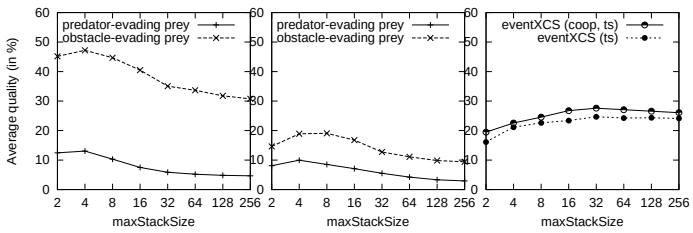
Figure 5: Sample start configurations

5. EXPERIMENTAL RESULTS

Many different design combinations of a reward function seem possible. This paper does not try to identify the best solution. Instead, it shows that there exists better implementations than the conventional XCS algorithm. Therefore, this paper concentrates on a comparison of XCS variants, as presented in Section 4.1. To properly compare these variants, it has been important to determine good parameter settings for each variant. While most of the standard values given in [6] and known as *commonly used parameter settings* provide good results, some special settings need closer examination (see Section 5.1). Thereby, thousands of different combinations have been tested and the parameter discussion could not completely presented in here. Configurations that achieved a lower performance than a simple *random walk* strategy are also omitted, since it is always difficult to argue, whether the increased performance of the predators' behavior is caused by an improved learning or by acting more equal to the *random walk* strategy. Finally, selected results are discussed in Section 5.2.

5.1 XCS Parameters

The parameter `maxStackSize` determines the stack overflow (and thus a *neutral event*), as introduced in Section 3.3. Similar to XCS' *prediction discount* parameter γ, the optimal value is a compromise between several conflicting factors: Using larger values results in an inclusion of older – maybe irrelevant – actions in the reward of positive or negative events. Using smaller values can reduce the delay between an event and the actual reward, but it may also lead to a possible disregard of actions that are important for achieving the current event. In scenarios with fewer obstacles and more open space (*random* and *pillar scenario*), a value between 4 and 8 seems optimal (see Figures 6(a) and 6(b)), while in more complex scenarios like the *difficult scenario*



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

Figure 6: Comparison of variants of *eventXCS* with different `maxStackSize` values in different scenarios

significant larger values up to 32 provide better results (see Figure 6(c)). Additional experiments have shown that the size of the grid is also relevant, which points to a correlation between the optimal value and the average distance to the prey. As a compromise to achieve a better comparison, a value of `maxStackSize` $= 8$ has been used in all tests.

Another important parameter is the learning rate β. In a similar scenario in [10], a value below the standard value is proposed ($\beta = 0.02$). The reason is that dynamic multi-agent systems can only be described by movement probabilities so that the learning process has to be slow and careful. Tests have shown that an optimal value ranges around 0.05 in the *pillar scenario* (see Figures 7(a) and 7(b)) and between 0.6 and 0.8 in the *random* and the *difficult scenario* (see Figures 7(c) and 7(d)). To maintain comparability between the scenarios and to other XCS implementations, a β value of 0.05 is used. According to [6], the maximum number of classifiers N should be chosen large enough so that *covering* only happens at the beginning. Here, tests have shown that a population size of 512 classifiers fulfills this criteria. The classifier sets are filled with random classifiers [3], but no significant difference could be seen compared to an empty initialization. Instead, sometimes a slower convergence has been observed, probably because the corresponding system has to unlearn irrelevant classifiers. The *GA threshold* parameter θ_{GA} is set to 25, larger values reduces the quality of the algorithm. As *eventXCS* itself makes use of a quadratic reward distribution, the parameter *reward prediction discount* γ is only needed to compare XCS to *eventXCS*. However, tests have been inconclusive so that the standard value of $\gamma = 0.71$ is used. Only $\gamma = 1.0$ has shown significant worse results in some cases, while the differences between the average qualities have been minimal for smaller values. Other parameters, like the subsumption threshold θ_{sub}, the GA threshold θ_{GA}, and the mutation probability μ, are initialized with default values (20.0, 25.0, and 0.05).

5.2 Discussion of Experimental Results

As shown in Figures 7 and 8, *XCS (sight)* is always inferior to *XCS (obs)*. Further tests on XCS plus different *environmental reward functions* have led to the conclusion that XCS should not be combined with a reward function different to the global goal. Moreover, *eventXCS* works well with strategies like *selfish behavior (sight range)*, which is shown by the higher average quality of *eventXCS* compared to *XCS (obs)* and an increased learning curve in the scenarios, depicted in Figures 8(a), 8(b), and 8(c). In the *random scenario* with an obstacle-evading prey (see Figure 8(c)), *XCS* even shows no learning at all. Not depicted is the *random scenario* with an predator-evading prey, the results are similar to the results of the *pillar scenario* with the same type of prey.

Looking further on the *difficult scenario*, it can be seen that *eventXCS* clearly fails, no matter which *learning rate* β is used (see Figure 7(d)). This problem can be solved by using one of the advanced variants, *eventXCS (ts)* or *eventXCS (coop, ts)*. While they do not reach the average quality level of *XCS (obs)*, they show a very stable learning, since *XCS (obs)* shows tendencies of over-learning or unlearning after 8 000 steps or 14 000 steps for $\beta = 0.2$ (see Figure 8(d)). In general, it seems not surprising that *XCS (obs)* reaches better results than *eventXCS*. *XCS (obs)* is designed to solve Maze scenarios, while *eventXCS* has been modified to solve more open scenarios with a non-blinded prey, as investigated on the *pillar* and the *random scenario*. Since *eventXCS* shares the same base algorithm with *XCS (obs)* and since it can solve the *difficult scenario* endowed with *tournament selection*, it points to a correctable flaw in the design of *eventXCS*. In summary, *eventXCS* is clearly superior to *XCS (obs)* in all four scenario configurations, it either reaches a better average quality or, with additional modifications, provides a similar quality with more stable learning.

6. CONCLUSION

A promising modified XCS approach has been investigated to overcome the drawbacks of the standard XCS algorithm concerning NOMDPs. The proposed idea is mainly based on expanding the local reward function and adding some kind of temporary memory, which stores past action sets and their corresponding reward values. Local payoffs can be delayed and analyzed later. Thus, the reward function reflects better the local agent's behavior. The experiments have shown that this new approach is superior to the standard XCS algorithm, mainly due to its ability to work well with advanced local reward functions, an ability, which XCS seemingly lacks. Besides improving the algorithm itself, future work will focus on an intelligent switching strategy between *explore* (when no prey is in sight) and *exploit phases* (when a predator is near by the prey). Further improvement is expected by an adaptive *learning rate* β and an adaptive `maxStackSize` to fit the current scenario's needs. Finally, it seems interesting to test the *eventXCS* approach on larger grids, on standard Maze scenarios, or other NOMDPs from literature.

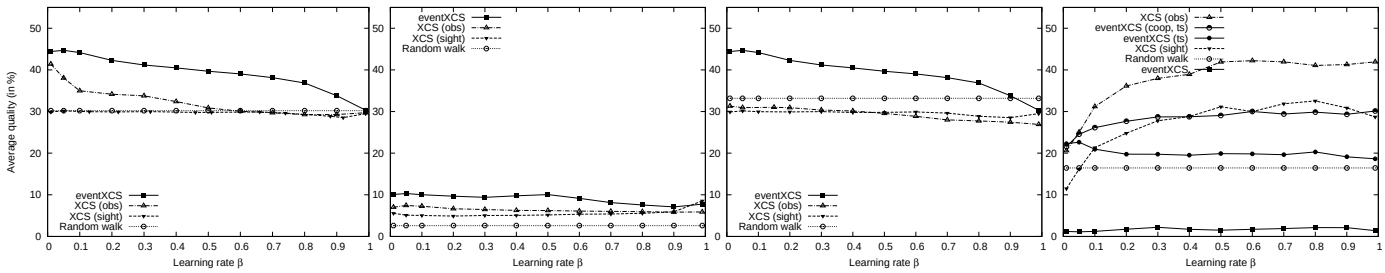
7. REFERENCES

[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, 1986.

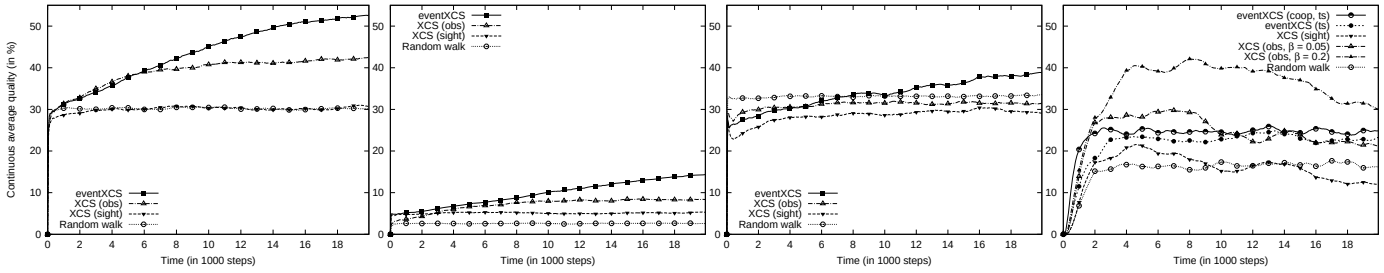
[3] M. V. Butz. *The XCS classifier system*, chapter 4, pages 51–64. Springer, 2006.

[4] M. V. Butz, T. Kovacs, P. L. Lanzi, and S. W. Wilson. Toward a theory of generalization and learning in



(a) *Pillar scenario* with an *obstacle-evading prey* (b) *Pillar scenario* with a *predator-evading prey* (c) *Random scenario* with an obstacle-evading prey (d) *Difficult scenario* with a *blinded prey*

Figure 7: Comparison of different values for the learning rate β for different XCS variants



(a) *Pillar scenario* with an *obstacle-evading prey* (b) *Pillar scenario* with a *predator-evading prey* (c) *Random scenario* with an obstacle-evading prey (d) *Difficult scenario* with a *blinded prey*

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

XCS. *IEEE Transactions on Evolutionary Computation*, 8(1):28–46, 2004.

[5] M. V. Butz, K. Sastry, and D. E. Goldberg. Tournament selection: Stable fitness pressure in XCS. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003)*, volume 2724 of *LNCS*, pages 1857–1869. Springer, 2003.

[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, 1975.

[8] J. H. Holland. Adaptation. In *Progress in Theoretical Biology*, volume 4, pages 263–293. Academic Press, 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 multi-agent environments. In *Proceedings of the 2005 Workshops on Genetic and Evolutionary Computation Conference (GECCO 2005)*, pages 109–111. ACM, 2005.

[11] S. K. K. M. K. Miyazaki, M. Yamamura and S. Kobayashi. On the rationality of profit sharing in multi-agent reinforcement learning. In *Proceedings of the 4th International Conference on Computational Intelligence and Multimedia Applications (ICCIMA 2001)*, pages 421–425. IEEE, 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 International Conference on Evolutionary Computation (CEC 1998)*, pages 609–614. IEEE, 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 *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.