# Adaption of XCS to 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

This paper investigates an approach of concurrent learners, where the credit assignment problem is specially addressed (How to divide team reward among the individuals?). Moreover, the investigated approach is focussed on the dynamics of learning (How to cope with the problem of coadaptation?) in a scenario as an instance of the homogeneous and (non-) communicating predator/prey example. To compensate the dynamic nature of such a scenario a new algorithm is developed ("SXCS") by modelling the reward function according to an heuristic with high performance and by using memory to record and analyze the movements and the history of function's past values.

## 1. INTRODUCTION

In this paper a predator/prey scenario [2] in the connection with a learning classifier system (LCS) is examined. Following the global task to observe one or up to $n$ (randomly) moving targets/preys, a number of rovers/predators has to achieve a common goal with local and distributed behavior, e. g., maximizing the global observation time or catching moving target(s). The scenario shows technical relevance to Organic Computing scenarios and it seems to be very flexible in its parameterization. Especially, it allows many variations: Depending on the playground the agents have to live in, implementations can vary the number of agents, the number of targets, the number and distribution of obstacles, the moving strategies, the learning behavior of the agents, the definition of local neighborhoods, the communication possibilities between the agents, and various forms of collaboration and collective learning could be analyzed. Here, the challenge of concurrent learning agents is addressed, where all agents are equipped with a single, independent LCS.

A current field of research in the field of LCSs are the so-called *eXtended Classifier Systems* (XCS) [3][5][18]. Basically a XCS is a LCS, i.e., a number of rules, so-called *classifiers*), each consisting of a condition/action pair. When the sensors of an agent match a condition then the action is executed. As wild-cards can be part of the condition a mechanism usually has to select from multiple classifiers. The classifiers are updated and adapted to the environment step by step using *reinforcement learning*. When an agent reaches a goal condition a reward is distributed among the classifiers. The way this is done usually depends on the type of scenario, in a single-step environment a reward is generated at each step while in a multi-step environment with local information the agent has to construct the global information using multiple repetitions and steps.

It has been ascertained that neither the single-step nor the multi-step approach of XCS, can be used to learn a dynamic observation task as described above. While there are implementations to handle such scenarios, they are either very simple with a goal object that can only be moved by agents, with only one or two agents and no obstacles [9] or use global communication and organization with a shared global classifier set [16].

Thus, a promising modified XCS approach has been investigated to overcome the drawbacks of the classical XCS algorithm. The proposed idea is mainly based on a local cooperative reward function and some kind of temporary memory, which stores past actions sets and their corresponding reward values. Thus, local payoffs can be delayed and the reward function reflects in a better way the local agent behavior. Cooperation (incorporated in the reward function) is more or less achieved through rejection and attraction. Predators reject each other, the prey attracts the predators. Thus, agents try to uniformly distribute on the grid and observation time of the prey seems to be maximized. In addition it is shown that the new XCS approach retains its ability to recognize local obstacle configurations in order to reach the goal object.

This paper is structured as follows. First learning classifier systems are introduced and their role in different scenarios discussed (see Section 2). A (in connection with XCS) new type of scenario (see Section 3) is then presented and classified. The conclusion is that a new reward function is needed, which is detailed in Section 4. Using this new reward function several experiments are executed and their results discussed (see Section 5). The paper concludes with Section 6 with the result that the new approach shows significant improvements over the standard implementation but still need further research.


*WOODSTOCK* '97 El Paso, Texas USA

## 2. LEARNING CLASSIFIER SYSTEMS

The field of LCSs, introduced in the 1970ies [6, 7, 8], is one of the most active and best-developed form of genetic based machine learning [11, 12, 14]. As mentioned above, much of learning classifier's theory is inherited from the reinforcement learning literature. The following section provides a brief overview what an LCS is.

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 [18]. 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, and a prediction. 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. For all distinct actions present in the match set the average prediction of all classifiers advocating that action is calculated. The action with the highest average prediction is selected for execution 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 predictions of all classifiers in the action set.

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 are inserted into the rule base by a process called *covering*. Furthermore, occasionally, a randomized *evolutionary component* 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. [11]), many are based on Wilson's XCS [18] 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 some action executed by the LCS is not instantaneous. Thus, further steps 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 like the predator/prey scenario additional measures are required. This will be the subject of the rest of this paper.

### 2.1 Single-Step vs. Multi-Step Problems

Literature about LCSs is 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 a 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* [18, 4].

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* [17] or *Wood2* [18]. 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 an environment with Markov property

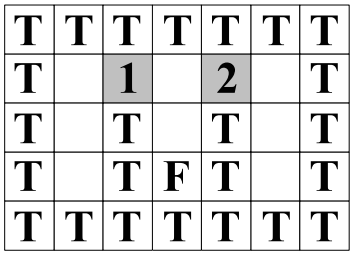
When the environment in question does have the Markov property, i.e., it is an environment where the sensory data for an agent differs for each position, then an agent can acquire global information by visiting all positions of the environment. With this information a LCS can find the optimal set of 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 rewarded positively. 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.

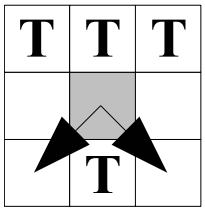
### 2.3 Non Markov environments

Some Maze problems offer perceptually identical situations that require different actions to reach the goal/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 food while trees 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 in position 2 the optimal action is to move to the bottom left. Thus XCS cannot evolve an optimal policy for Woods101.




(a) The Woods101 environment with aliasing positions marked with 1 and 2




(b) Sensory configuration of the two aliasing positions

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

Using records of past situations/actions by adding temporary memory is a widespread approach to cope with such environments, as investigated in [13, 15].

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 [16], this second non Markov property is defined as the *non observable Markov decision process (NOMDP)*.

Later in Section 3.3 it will be shown that the multi-agent scenario that is presented in the next Section 3 includes both non Markov properties (POMDP and NOMDP). Thus learning in such an environments is more complex than learning in single-agent environments and requires a different approach which will be discussed in Section 4.

# 3. THE PREDATOR/PREY EXAMPLE

As an example of a multi-agent approach, the predator/prey domain is an appropriate 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.

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. The general approach of the predator/prey example, the possibility to customize and adopt the scenario to manifold applications, or the widespread experience that is documented, not only in multi-agent literature, result in the assumption that the predator/prey example can be used as a valid testbed for OC scenarios.

Here a simpler implementation of the predator/prey scenario is examined where the predators are rewarded for just being very close to the prey and it is not necessary to surround and catch it. The overall scenario settings are described in Section 3.1 followed by a detailed description of the obstacle configurations that were tested (see Section 3.2). Putting all together the properties of the scenario are then compared to the properties of environments without the Markov property (see Section 3.3) and then classified according to the results of the discussion in Section 2.3.

## 3.1 Scenario settings

The algorithms are tested in a simple predator/prey scenario on a torus with discrete squares. The field is a quadratic, toroidal world consisting of 16x16 squares. The prey consists of a single moving goal object which the 8 agents (the predators) have not to catch but only to keep under surveillance. A cell of the two-dimensional grid-world can only be occupied by one agent. The quality of an algorithm that controls the agents is determined by the share of the time any agent has the goal object in surveillance range. It is calculated by averaging the qualities of 10 experiments consisting of 10 runs each. Each run consists of 2,000 steps after which the scenario is reset. The individual learning experiences of the agents are retained between each run but not between each experiment.

In each time step each agent can move only to one of the four neighboring fields while the goal object can move two fields. In addition there are obstacles on the field and any movement to an occupied field fails (without any further consequences). Both the goal object and the agents have 24 binary sensors that can sense their close environment but their lines of sight are blocked by obstacles or other agents. Half of the sensors can detect objects 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 (goal object, other agents and the obstacles).

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. As the goal object can move two fields it gathers new sensor data after the first step. If the goal object is surrounded by agents in all 4 directions it jumps to a random nearby free field which basically means a restart of the simulation. Experiments showed that this happened only very few times, so it does not affect the result.

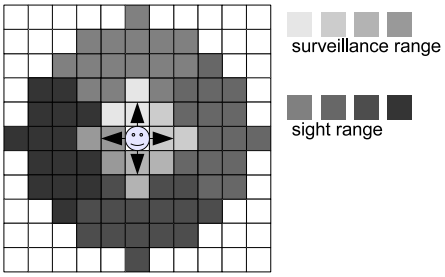



**Figure 2: Sight range (5.0, dark area) and surveillance range (2.0, bright area) of an agent / the goal object for each direction**

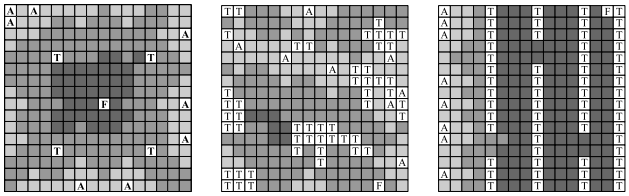
## 3.2 Scenario configurations

Three different configurations of obstacles and starting positions of the agents and the goal object were thoroughly tested.

In the *pillar scenario* (see Figure 3(a)) there were 4 obstacles in equal distance to each other, with the goal object starting in the center and the agents starting randomly along the borders. The idea was to minimize the number of blocked fields while still giving the agents some points of orientation.

In the second scenario, the *random scenario* (see Figure 3(b)), several obstacles were placed randomly on the field, with a certain tendency to build connected structures. In both scenarios a goal object with intelligent behavior (random walk, but move away from agents in sight) and a goal object with random walk is tested.

The third scenario, the *difficult scenario* (see Figure 3(c)), consisted of a cylinder and several walls with small openings. There, the goal was to find those openings that lead to the goal object which always stayed in the last section and did nothing but to move up two fields (or jump to a nearby free field if the path is blocked).



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

**Figure 3: These are sample start configurations for the three different types of scenarios. Trees are marked with T, food is marked with F, agents are marked with A and the surveillance and sight range is marked with light grey and grey color. Areas out of sight are marked with dark grey color.**

## 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 a predator/prey scenario are:

1. An agent has only access to local information,
2. the field usually consists of open areas with randomly placed obstacles,
3. each agent has an internal state unknown to other agents,
4. the scenario is dynamic,
5. the agents are sharing a global goal, and
6. the scenario runs continuously.

Clearly the scenarios in question constitute no MDPs because of limited sensory information (1, 3) and many different positions sharing the same sensory configuration (2). Because the introduction of memory would not restore the Markov property (4) it is not a POMDP either. This leaves the scenario as being a NOMDP.

In addition special care is needed when applying XCS as is no clear local goal (5) and the scenario is not restarted when reaching the global goal (6). All in all this means that a different approach is needed and a new XCS variant needs to be created that can handle these issues properly.

In the literature NOMDP environments are discussed in connection with XCS [10, 16]. There, the issue of non observability is either evaded by a very simple scenario with only two agents or solved by using complex communication and central organization where the agents share a classifier system. Neither approach can be used here as the scenario is rather complex with many agents and there is no communication between the agents allowed. Thus it is required to develop XCS further in order to be applicable to such a dynamic collaborative scenario. The most important step is to design the reward function which will be 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 heuristics 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 "goal position" in the context of a predator/prey scenario could be defined as "be in surveillance 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. Thus, this paper proposes another approach:

By looking at the global goal it is possible to examine reward functions by testing agents with static heuristics instead of XCS. Although heuristics do not have something like a reward function as they do not learn, they still evaluate situations and corresponding actions either as "good" ("move in that direction") or as "bad" ("don't move in that direction").

Of a set of simple implementations the best results (except for some special cases like very large environments and a relatively small number of agents) delivered the following heuristic:

- Cooperation: When the goal object is not in sight move randomly in a direction without other agents in sight.
- Egoism: Move in the direction of the goal object when it is in sight.

Modeling this heuristic would require multiple states. As the original implementation uses a binary reward function the following approximation of the reward function $r_b(s_a, s_g)$ with $s_a, s_g$ being indicators whether the goal object is in sight range ($s_g = true$) and whether any other agent is in sight range ($s_a = true$) is used:

$$r_b(s_a, s_g) = \begin{cases} 0 & s_a \wedge \overline{s_g} \\ 1 & \overline{s_a} \vee s_g \end{cases}$$

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 [5] 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 4).

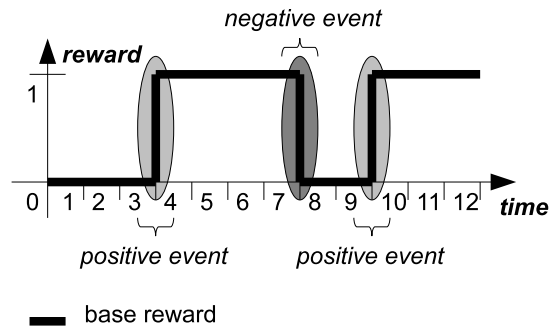



**Figure 4: Example of a series of base rewards that lead to positive and negative events.**

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). 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.

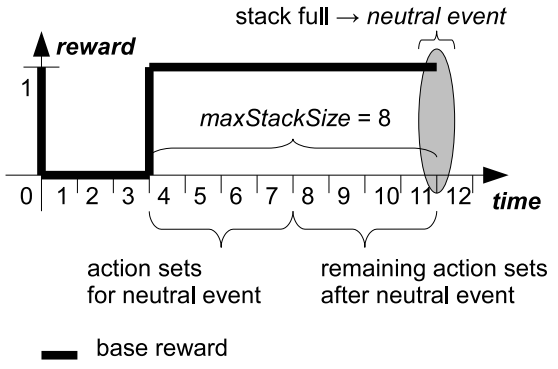



**Figure 5: Example of a series of base rewards that lead to neutral event (with $maxStackSize = 8$).**

## 4.3 Reward distribution

The standard implementation of XCS is based on the assumption that it works in an environment with the Markov property. 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 re-

ward to be transferred to all classifiers that contributed to the solution.

In the dynamic predator/prey 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) and an improvement in the performance is to be expected.

Thus, whenever (and only then) 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.

This is done with a quadratic function, i.e., with $r(a)$ being the *reward* for the *action set* with age $a$:

$$r(a) = \begin{cases} \frac{a^2}{\text{size}(action\ set)^2} & positive\ event \\ \frac{(1-a)^2}{\text{size}(action\ set)^2} & negative\ event \\ 1 & neutral\ event,\ base\ reward = 1 \\ 0 & neutral\ event,\ base\ reward = 0 \end{cases}$$

The idea behind using a quadratic function is that it loosely resembles the transfer of the reward in the original implementation. Figure 6 shows a distribution of the reward for an exemplary distribution of the base reward. For simplicity a linear distribution is displayed in the graphics.

More sophisticated approaches are possible, this is merely the most straightforward approach. Compared to a linear distribution tests showed no significant difference.

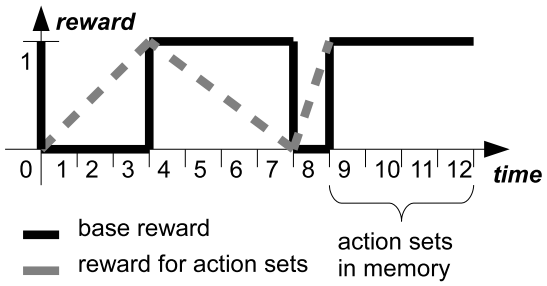



**Figure 6: Schematic presentation of the reward distribution to the action sets over time after several positive and negative events**

# 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 [5] ("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.

In the subsequent sections first the experiments with heuristics are examined (see Section 5.3) which is important to determine a good heuristic for the environmental reward function in Section 4.1. Then the three scenarios discussed in Section 3.2 are discussed: The pillar scenario (Section 5.4), the random scenario (Section 5.5) and the difficult scenario (Section 5.6).

## 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 environment reward function from Section 4.1 even reduced the performance. The best results showed a standard implementation of XCS with an environmental reward function that returns 1 when the goal object is in surveillance range and 0 at all other times.

Using the collaborative environment reward function in connection with events (see Section 4.2) and the reward distribution based on the generated events (see Section 4.3) results in the new XCS variant which will be called SXCS (*Supervising eXtended Classifier System*). This variant showed significant better performance than any of the other XCS variants. The results will be presented in the following sections.

## 5.2 XCS parameters

The parameter *maxStackSize* that was introduced in section 4.2 determines when the stack overflows and a neutral event occurs. In this paper a relatively good value was determined by several experiments but further research is needed in that direction. A compromise between several conflicting factors has to be made: Large values may lead to a delay between the rewarding situation and the actual reward and the reward of uninvolved rules. On the other hand, small values may lead to early stack overflows and a possible disregard of rules that are crucial to solve the problem. As Figure 7 shows the value does not have a large impact in the pillar or random scenario, only at around $maxStackSize = 8$ a difference can be seen. The difficult scenario on the other hand favors a larger value ($maxStackSize = 32$), so the optimal value depends on the scenario.

During the tests another important parameter was the learning rate $\beta$. In a similar type of scenario in [9] 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 8) showed an optimal value between 0.01 and 0.1 for SXCS, larger values seem to harm the learning process significantly. For XCS on the other hand the quality increases with larger values for $\beta$ in this scenario. This is still a subject of research, but be-

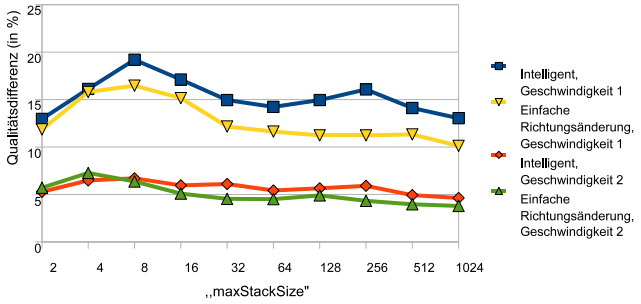



**Figure 7: Comparison of different values of *maxStackSize* in various scenarios**

cause larger values result in loss of comparability with other implementations of XCS the standard value of 0.2 was used.

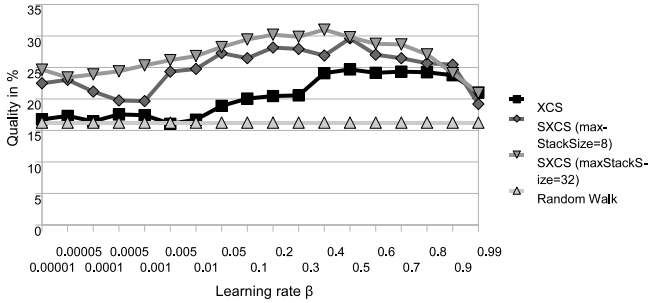



**Figure 8: Influence of the learning rate parameter $\beta$ on the performance (pillar scenario with XCS and SXCS agents and *best selection*)**

According to [5] 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 fulfills this criteria. Although random initializations are possible [3], it was chosen to start with empty classifier sets as tests have shown better performance.

The *GA threshold* parameter $\theta_{\text{GA}}$ 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* $\gamma$ 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, so it seems that while the reduction of the transfer of the reward is needed, the actual value is of little importance.

Table 1 show the special settings that were used.

## 5.3 Experiments with heuristics

The results displayed in Figure 9 show clearly that an agent with the collaborative heuristic is superior.

## 5.4 Comparison in the pillar scenario

TODO

See Figure 10

## 5.5 Comparison in the random scenario

TODO

**Table 1: Used parameter values and standard values**

| Parameter | Value | Standard (see [5]) |
|---|---|---|
| max population $N$ | **512** | [-] |
| subsumption threshold $\theta_{\text{sub}}$ | 20.0 | [20.0+] |
| GA threshold $\theta_{\text{GA}}$ | 25 | [25-50] |
| mutation probability $\mu$ | 0.05 | [0.01-0.05] |
| reward prediction discount $\gamma$ | 0.71 | [0.71] |
| learning rate $\beta$ | **0.01 - 0.2** | [0.1-0.2] |
| tournament factor | 0.84 | [-] |

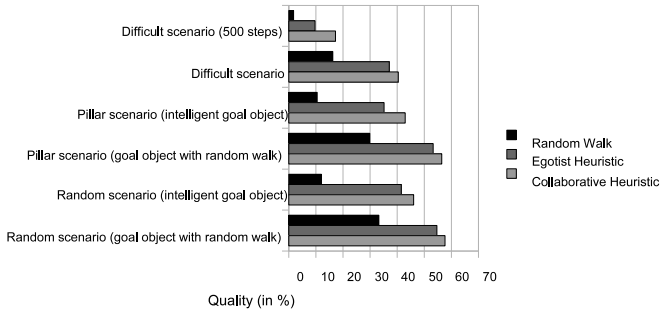



**Figure 9: Comparison of different static heuristics**

See Figure 11

## 5.6 Comparison in the difficult scenario

TODO

See Figure 12

# 6. CONCLUSIONS

In Section 5 it was shown that SXCS with the adapted reward function outperforms XCS.

TODO Text

All in all the results of this paper are:

TODO erst am SchluSS :)

# 7. REFERENCES


[1] A. J. Bagnall and Z. V. Zatuchna. On the classification of Maze problems. In L. Bull and T. Kovacs, editors, *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, July 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 generalisation and learning in XCS. *IEEE Transactions on Evolutionary Computation*, 8(1):28–46, February 2004.

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

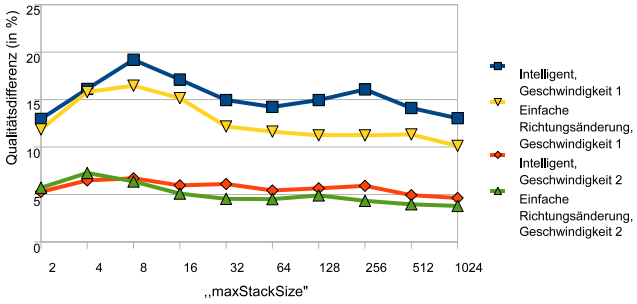



**Figure 10: Comparison of different XCS variants in the pillar scenario**

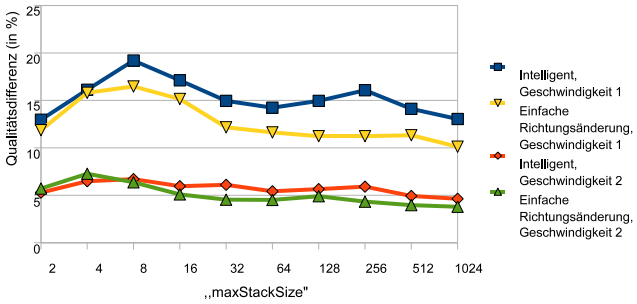



**Figure 11: Comparison of different XCS variants in the random scenario**


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

[7] J. H. Holland. Adaptation. In R. Rosen and F. M. Snell, editors, *Progress in Theoretical Biology*, volume 4, pages 263–293. Academic Press, New York, NY, United States of America, 1976.

[8] J. H. Holland and J. S. Reitman. Cognitive systems based on adaptive algorithms. In D. A. Waterman and F. Hayes-Roth, editors, *Pattern-Directed Inference Systems*, pages 313–329. Academic Press, 1978.

[9] 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.

[10] 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.

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

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

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

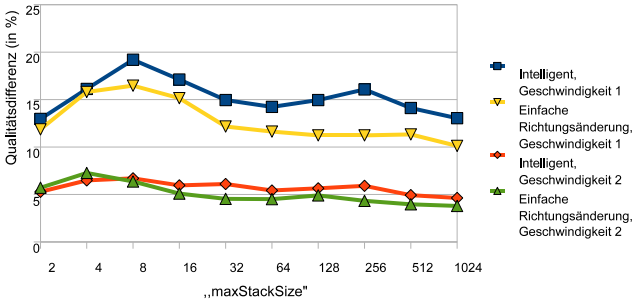




**Figure 12: Comparison of different XCS variants in the difficult scenario**


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

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

[16] K. Takadama, T. Terano, and K. Shimohara. Learning classifier systems meet multi-agent environments. In P. L. Lanzi, W. Stolzmann, and S. W. Wilson, editors, *Advances in Learning Classifier Systems*, volume 1996 of *LNAI*, pages 192–212. Springer, 2001.

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

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