Adaption of XCS to predator-prey scenarios

[Extended Abstract] *

Clemens Lode
Universität Karlsruhe (TH)
Institute AIFB
76128 Karlsruhe, Germany
clemens@lode.de

Urban Richter
Universität Karlsruhe (TH)
Institute AIFB
76128 Karlsruhe, Germany
urban.richter@kit.edu

ABSTRACT

This paper focuses on the application of XCS in predator-prey scenarios. Due to the dynamic nature of the problem common approaches cannot be used so a new algorithm ("‘SXCS"’) is developed by using studying heuristics in such scenarios and doing empirical studies in a computer simulation. Modelling the reward function after a well functioning heuristic and using memory to record previous steps significantly better results were produced. Still the implementation lacks proper cooperation between the agents and the algorithms themselves need a better theoretical foundation.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Delphi theory

Keywords

ACM proceedings, LaTeX, text tagging

1. INTRODUCTION

In this paper a predator-prey scenario [6] in the connection with a learning classifier system (LCS) is examined. Such a scenario consists of two groups of agents with sensors that move on a field with obstacles. Goal of the first group (predators) is to keep the prey (consisting of a single goal object which tries to evade the agents) in sight. Such a observation scenario seems to be of interest having parallels to real world problems.

A current field of research in the field of LCS are the so-called *eXtended Classifier Systems* (XCS) [7][2][3]. Basically a XCS is a LCS, i.e. a number of rules, so-called *classifiers*), each consisting of a condition and an action. When the sensors of an agent match to 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.

Most papers that discussed XCS so far concentrated either on static scenarios with a single agent or on more dynamic scenarios with global organization and communication. This paper on the other hand concentrates on developing an algorithm that solves such predator-prey scenarios better. Common methods are not applicable due to the dynamic nature and lack of communication.

Using empirical studies and by making a parameter study a new algorithm ("‘SXCS"’) was developed. It was then tested and compared to the standard implementation of XCS in several scenarios.

2. SCENARIO

The algorithms are tested in a simple predator-prey scenario on a torus with discrete squares. The field is a quadrat consisting of 16x16 squares. The prey consists of a single moving goal object which the predators (the agents) have not to catch but to keep under surveillance. 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 problems per experiment and 10 experiments. 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 12 binary sensors that can sense their close environment (up to two fields distance) while another 12 binary sensors can sense the environment further away (up to five fields distance), see Fig. 1. The 12 sensors are for the four directions and the three types of objects (goal object, other agents and the obstacles). Obstacles do block the lines of sight of a sensor.

*A full version of this paper is available as *Author's Guide to Preparing ACM SIG Proceedings Using $\LaTeX2_\epsilon$ and BibTeX* at www.acm.org/eaddress.htm

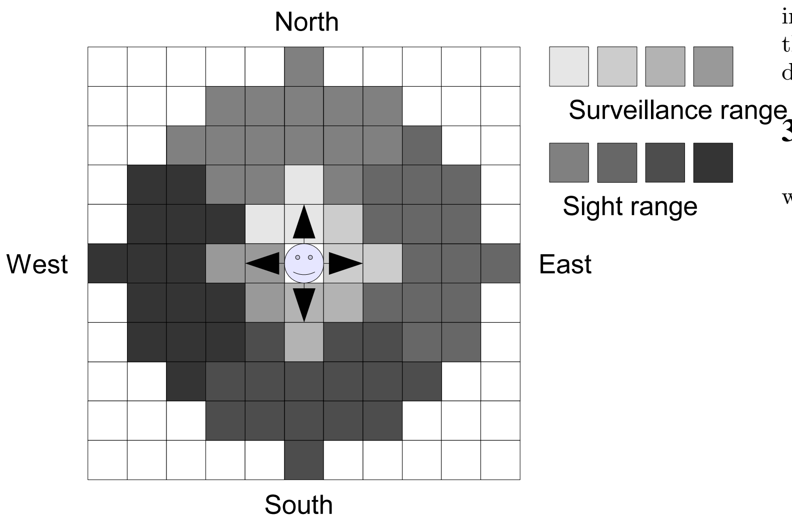


Figure 1: Sight range (5.0, dark area) and surveillance range (2.0, bright area) of an agent / the goal object for each direction

The configuration of the obstacles tested is mainly the pillar scenario. In this scenario there are obstacles each with 7 fields in between. The idea is to minimize the number of blocked fields while still giving the agents some points of orientation. An example start configuration is displayed in Fig. 2. Here the obstacles are red, the agents white and goal object is green.

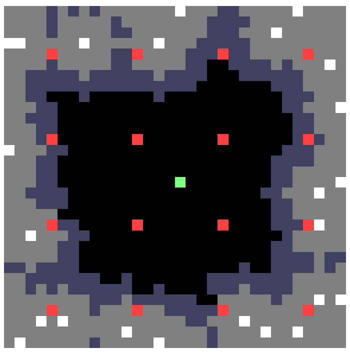


Figure 2: Example start configuration in the pillar scenario

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.

All in all these are characteristics of a dynamic collaborative scenario. The agents need to not only include their relation to their goal object in their decisions but also other agents. In addition the environment is constantly changing. For such types of scenarios there is hardly any existing literature so this paper will fill this gap. The first step is to look into the prerequisites of the algorithm of the usual XCS and then develop it further in order to be applicable to such a dynamic collaborative scenario.

3. PROBLEM DEFINITION

The main differences to standard scenarios that are tested with XCS are that

- the goal object does move,
- there are multiple agents on the field sharing a global goal,
- the goal of the agent is not to find the shortest way to the goal but to find a set of rules for optimal behavior and
- the quality is calculated online, there are no separate exploration phases.

Still the environment that comes closest to the scenario definition here is the multi step environment. There, a whole number of steps have to be executed to reach the goal and as opposed to single environments it lacks a global view on the problem.

The implementation of the XCS multistep method is similar as in [1]. The description of the algorithm can be found in [3] where there is also a discussion of the differences between the single step and multi step method.

In the simpler case of a single step environment the evaluation of new fitness values after each step while the more complex case of a multi step environment the evaluation is done not before the agent has reached its goal.

A classic example a the single step environment is the 6-multiplexer problem [2] where the XCS should emulate a multiplexer with 2 adress bits and 4 data bits. For multi step environments usually a variant of the *Maze N* problem is used where an agent has to find the shortest way through a labyrinth with the optimal distance of N steps. When the goal is reached the scenario is restarted and the last action is evaluated positvely. With each repetition actions that were executed before that last action that reached the goals are evaluated positively as well. After a large enough number of repetitions an optimal route is found if the problem can be represented by a Markov chain. If the Markov condition does not hold then XCS can only return a general behavior based on probabilities. TODO

4. XCS VARIANTS

Damit die Agenten lernen können, muss an einer Stelle eine Bewertung mit Hilfe einer sogenannten *reward* Funktion geschehen. XCS ist darauf ausgelegt, dass es eine komplette, genaue und möglichst allgemeine Darstellung einer solchen *reward* Funktion darstellt. Die *reward* Funktion läuft lokal auf jedem Agenten ab und die Bewertung, die der Agent berechnet, wird also auf Basis der eigenen Sensordaten gebildet. Die Bewertung wird im Folgenden als *base reward* Wert bezeichnet.
TODO

To determine the effectiveness of a new algorithm it is required to adapt the standard XCS to the predator prey scenario:

- There are no separate *exploit* phases to test the quality of an algorithm.

- The reward function has to be adjusted to the fact that there is no distinct goal to reach and that the quality of an algorithm is determined by constant good behaviour.
- Besides the usual goal condition "‘goal found"’ with a positive reward there will also be a goal condition "‘goal lost"’ with a reward of zero (see section 4.2).
- Some XCS parameter were adapted (see section 4.6).
- The problem always runs 2,000 time steps and is not resetted when some condition occurs as the scenario demands a continous observation of the goal object.
- The genetic operator uses two fixed locations for the *two point crossover.*

4.1 Reward function

In a single step environment 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. The main prerequisite for an environment that can be solved by the single step method is that the agent has global information. More complex problems that do not satisfy this prerequisite, i.e. where the agent only has local information, need to be solved by building up global information out of a sequence of local information bits. The prerequisite for this is that the agent can distinguish between all positions, i.e. the Markov property applies.

In the standard implementation of XCS there is very much time available and that by resetting the scenario and repeating the same steps in the same local environment it is possible to reach the goal.

In a dynamic environment the global information cannot be constructed as in the time the information is gathered it is no longer valid because other agents and the goal object may have moved. Also we cannot repeat the scenario and expect the same behaviour of the enviromnent. Thus it is important to speed up the learning and connect the learning steps directly instead of gradually pass the reward down.

The actual reward function in the multi step environment often consists of a simple binary value that is determined by the position of the agent compared to the goal position, i.e. whether it has reached the goal or not. In the predator/prey scenario the nature of the reward function isn't so obvious. Basically an agent has free choice about how this function should look like. 24 binary sensors result in up to 2^{24} situations an agent can recognize (actually 3^{12} due to some redundancy) and an equal number of reward functions if each situation is assigned an own value. This surpasses the available memory.

To answer the question one need to look at the global goal. One possibility is to choose a reward function by testing heuristics. Although heuristics do not work with a reward function as they do not learn it still evaluates 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 delievered the following heuristic:

- If the goal object is not in sight: Move randomly in a direction without other agents in sight
- Otherwise: Move in the direction of the goal object

In this implementation a binary reward function was used. This causes some problems when trying to model the heuristic as it is impossible to distinguish situation with e.g. one other agent and four other agent in sight. Probably a better implementation would be to count the number of agents and return it as a reward. This remains open for research.

An additional problem is that in scenarios with a relatively low number of agents the reward function nearly always returns 1 which could harm the learning process.TODO

As further adaptions to the reward is necessary the value the reward function returns will be called "‘base reward"’. Below (section 4.2) the actual reward for the rules of the XCS will be calculated.

4.2 Events

Compared to the usual implementation in [3] the main difference is that the environment does not restart when it returns a positive reward.

Assuming that the agent did something right when it comes 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.

4.3 XCS variant for surveillance scenarios (SXCS)

The hypothesis for this variant of XCS is basically the same as with the XCS multi step method itself, the combination of a number of actions lead to the goal. The difference is that in the multi step method the connection between different action sets is only created by repetition of the same scenario while in SXCS the past action sets will be recorded and rewarded directly. This is done because a repetition is not possible if the scenario continues to run and cannot be restarted. Even if it is restarted the dynamic and lack of the Markov property of the scenario would make it nearly impossible for an agent to encounter the identical situation multiple number of times, not to speak of a whole sequence of situations and actions.

While there are implementations for internal states (i.e. memory) in [5], these work only for static environments that lack the Markov property. For dynamic environments a different approach is needed.

While a memory does not restore the Markov property, so an agent can only learn an optimal local behaviour and cannot restore a global view on the problem. Still the idea is that this will improve the quality of the algorithm compared to the standard XCS. TODO

(*Supervising eXtended Classifier System*)

4.4 Implementation of SXCS

Whenever (and only then) an event occurs the reward is distributed among the entries of the action set. This is done with a quadratic function, i.e. with $r(a)$ being the *reward* for the *action set* with age a:

$$r(a) = \begin{cases} \frac{a^2}{\text{size}(\textit{action set})^2} & \text{falls } \textit{base reward} = 1 \\ \frac{(1-a)^2}{\text{size}(\textit{action set})^2} & \text{falls } \textit{base reward} = 0 \end{cases}$$

The idea behind this implementation is that it loosely resembles the transfer of the reward in the original implementation. Other implementations are possible, this is merely the simplest approach. In tests no significant differences were observed when comparing linear distribution with quadratic distribution.

This requires that not only the last but the whole history (up until the last event) is saved. As there can be cases where there is never an event for an agent the number of steps is limited to *maxStackSize*. If that number is reached a "'neutral event"' occurs.

TODO Zeitverzögerung, tiefere Analyse des Stacks

TODO Bild Bewertung!

In such a case

4.5 Size of the stack (*maxStackSize*)

An open question is the size of the stack. In this paper a good value was determined by tests, further research is needed in that direction. It is needed to find a compromise between three conflicting factors. Large values can lead to a delay between the rewarding situation and the actual reward and the reward of uninvolved rules. On the other hand, small values can lead to early stack overflows and a possible disregard of rules that are crucial to solve the problem.

As fig. 5 shows the value does not have a large impact, only at around *maxStackSize* $= 8$ a difference can be seen. Other tested scenarios favor larger values so the value is scenario dependant.

TODO evtl Code.

4.6 XCS Parameter

Most of the parameters correspond to the standard values given in [3] ("'Commonly Used Parameter Settings"'). Below special settings are discussed. While thousands of different combinations were tested, the parameter discussion is not necessarily complete. Of main importance was that any score below the algorithm where the agents move randomly 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.

According to [3] 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 (see [2]), it was chosen to start with empty classifier sets as tests have shown some advantages. The *GA threshold* parameter θ_{GA} was set to 25, larger values seemed to reduce the quality of the algorithm. The parameter *reward prediction discount* γ is only needed to compare XCS with SXCS, SXCS itself does use quadratic reward distribution. Tests showed inconclusive results 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. For the learning rate β in a similar type of scenario in [4] 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. In tests (see fig. 7) an optimal value between $0,01$ and $0,1$ for SXCS was determined, larger values seem to harm the learn process significantly. For XCS on the other hand the quality increases with larger values for β in this scenario. This is still a subject of research, but because larger values result in loss of comparability with other implementations of XCS the standard value of $0,2$ was used.

TODO pruefen ob notwendig

Table 1: Used parameter values and standard values

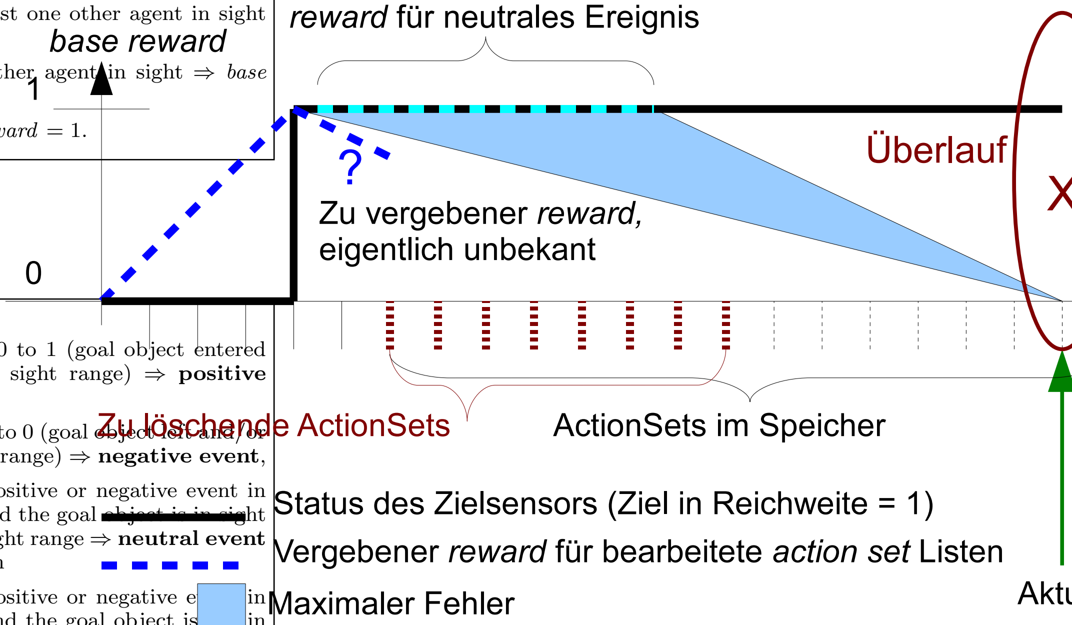
Parameter	Value	Standard value [3]
max population N	**512**	[*depends on scenario*]
subsumption threshold θ_{sub}	20,0	[20,0+]
GA threshold θ_{GA}	25	[25-50]
mutation probability μ	$0,05$	[0,01-0,05]
reward prediction discount γ	0,71	[0,71]
learning rate β	**0,01 - 0,2**	[0,1-0,2]
tournament factor	0,84	[-]

Model of the reward function:

- Goal object not in sight, at least one other agent in sight $\Rightarrow$ *base reward* $= 0$,
- Goal object not in sight, no other agent in sight $\Rightarrow$ *base reward* $= 1$ and
- Goal object in sight $\Rightarrow$ *base reward* $= 1$.

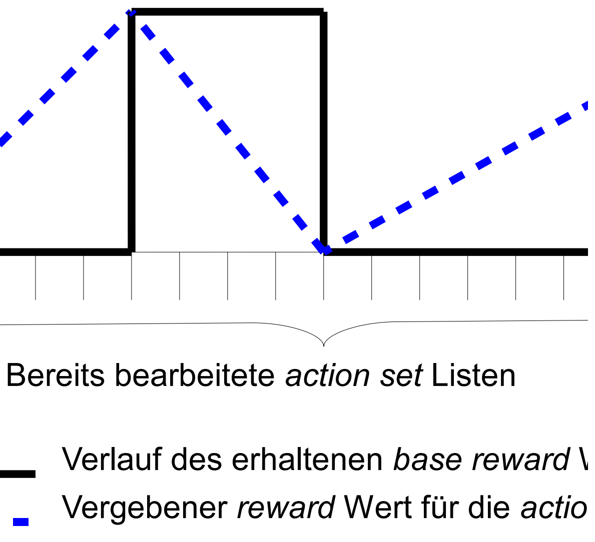
All in all an event occurs when...

- the *base reward* changes from 0 to 1 (goal object entered and/or the last agent left the sight range) $\Rightarrow$ **positive event**,
- the *base reward* changes from 1 to 0 (goal object left and/or the first agent entered the sight range) $\Rightarrow$ **negative event**,
- there is a stack overflow (no positive or negative event in the last *maxStackSize* steps) and the goal object is in sight range and/or no agents are in sight range $\Rightarrow$ **neutral event** (with *base reward* $= 1$) or when
- there is a stack overflow (no positive or negative event in the last *maxStackSize* steps) and the goal object is in sight range and/or an agent is in sight range $\Rightarrow$ **neutral event** (with *base reward* $= 0$).



reward für neutrales Ereignis
base reward
1
0
?
Überlauf
X
Zu vergebener *reward*, eigentlich unbekant
Zu löschende ActionSets
ActionSets im Speicher
Status des Zielsensors (Ziel in Reichweite = 1)
Vergebener *reward* für bearbeitete *action set* Listen
Maximaler Fehler
Aktu

Figure 4: Schematische Darstellung der Bewertung von *action set* Listen bei einem neutralen Ereignis (mit *base reward* = 1)



Bereits bearbeitete *action set* Listen
Verlauf des erhaltenen *base reward* \
Vergebener *reward* Wert für die *actio*

Figure 3: Schematische Darstellung der zeitlichen Verteilung des *reward* Werts an *action set* Listen nach mehreren positiven und negativen Ereignissen und der Speicherung der letzten *action set* Liste

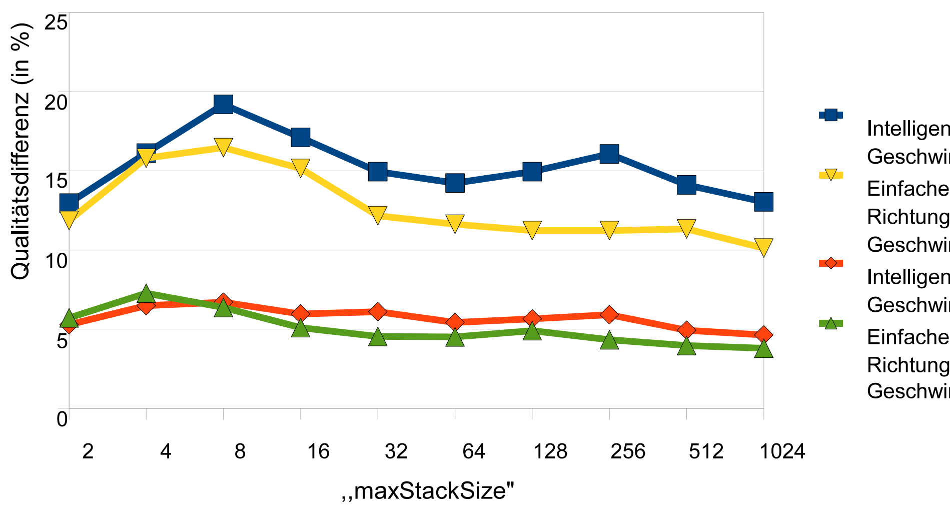


Figure 5: Vergleich verschiedener Werte für *maxStackSize* (Säulenszenario, SXCS Agenten, *tournament selection*, $p = 0,84$))

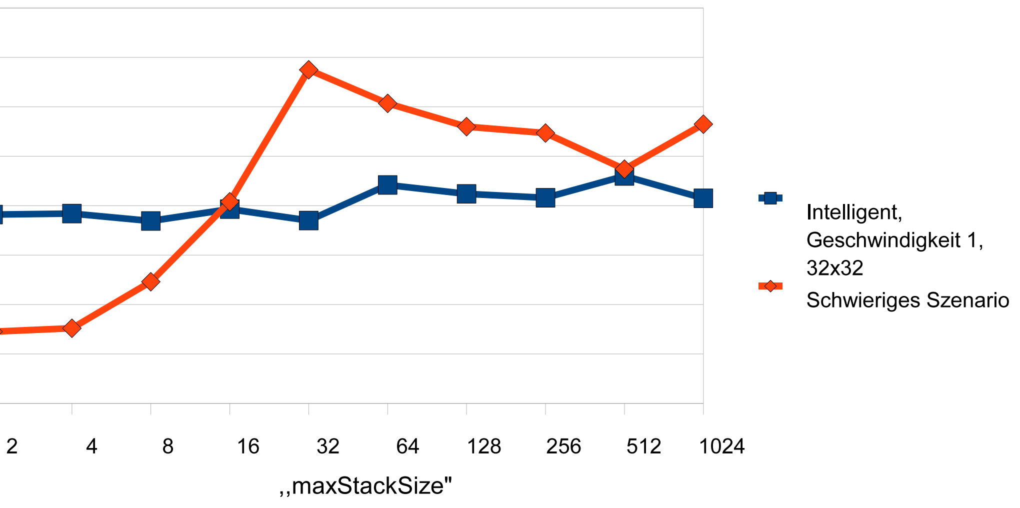


Figure 6: Vergleich verschiedener Werte für *maxStackSize* (spezielle Szenarien, SXCS Agenten, *tournament selection*, $p = 0,84$)

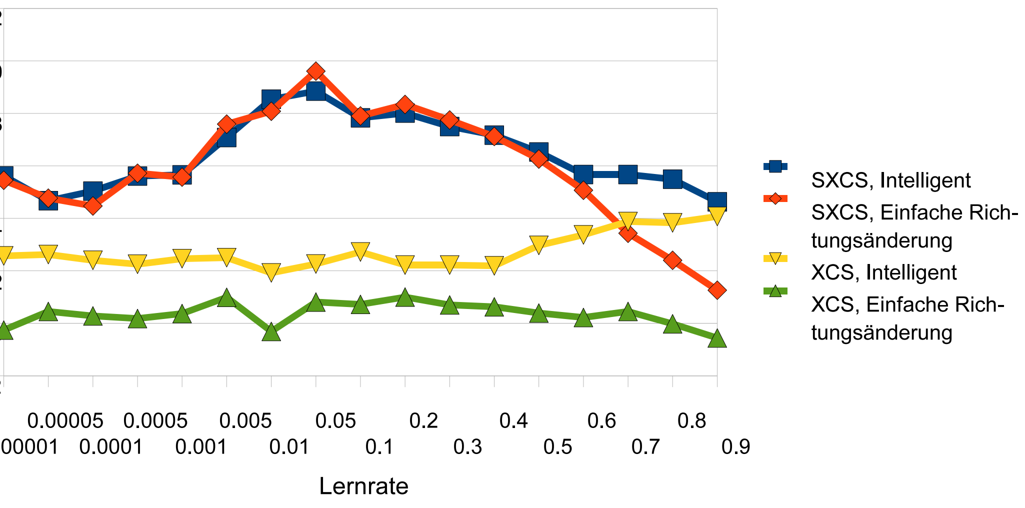


Figure 7: Auswirkung des Parameters Lernrate β auf die Qualität (Säulenszenario, Agenten mit XCS und SXCS Algorithmus, *best selection*)

5. REFERENCES

[1] M. V. Butz. Xcs classifier system in java, 2000.

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

[3] M. V. Butz and S. W. Wilson. An algorithmic description of XCS. *Lecture Notes in Computer Science*, 1996:253–272, 2001.

[4] 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.

[5] P. L. Lanzi and S. W. Wilson. Optimal classifier system performance in non-markovian environments. Technical Report 99.36, Milan, Italy, 1999.

[6] G. F. Miller and D. Cliff. Co-evolution of pursuit and evasion i: Biological and game-theoretic foundations. Technical Report CSRP311, August 1994.

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