Preliminary tests of a real-valued Anticipatory Classifier System

Norbert Kozlowski
Wroclaw University of Science and Technology
Department of Computer Engineering
Wroclaw, Poland
norbert.kozlowski@pwr.edu.pl

Olgierd Unold
Wroclaw University of Science and Technology
Department of Computer Engineering
Wroclaw, Poland
olgierd.unold@pwr.edu.pl

ABSTRACT

The paper describes the first attempts toward designing and evaluating anticipatory classifier systems working in a real-valued input domain using interval predicates representation. Promising results are obtained by testing two environments - real-valued multiplexer and checkerboard from the classical XCSR problem domain.

CCS CONCEPTS

• **Computing methodologies** → *Rule learning*; • **Software and its engineering**;

KEYWORDS

Anticipatory Learning Classifier Systems, OpenAI Gym

ACM Reference Format:
Norbert Kozlowski and Olgierd Unold. 2019. Preliminary tests of a real-valued Anticipatory Classifier System. In *Genetic and Evolutionary Computation Conference Companion (GECCO '19 Companion), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3319619.3326797

1 INTRODUCTION

Anticipations play an important role in our lives. They enable us to pursue both short- and long-term goals. By anticipating the consequences of our actions we are able to choose the best possible action to achieve them.

Anticipatory Classifier System (ACS) is a variant of Learning Classifier System (LCS) extending the classical human-interpretable, rule-based model with the psychological theory of anticipations. Every situation is accompanied by consequences after performing each possible behavior. ACS is tested in both single- and multi-step interaction processes like knowledge discovery [18] or controlling mobile robot's arm [15]. By design, all LCS-es are versatile towards the chosen alphabet. This work describes the first approach towards implementing interval predicates representation in ACS models. Traditionally, ACS use ternary alphabet to encode environmental state. Proposed modification enables to widen range of applicable problems by enabling adaptive interval ranges representation to model continuous-valued features.

Section 2 presents a general concept of LCS. ACS2 variant with all relevant components will be described. In section 3 a reminder of how real-valued inputs are handled in XCS (the most mature variant of LCS) is given with some remarks regarding the full workflow of the process. Different approaches towards representing a continuous value will be presented. Section 4 will discuss which components need to be tuned in order for the ACS to handle new input type, naming it tentatively rACS. Later on, in section 5 two environments (real-valued multiplexer and checkerboard) are used to evaluate the algorithm performance. Final conclusions and problems spotted so far are listed in section 6.

GECCO '19 Companion, July 13–17, 2019, Prague, Czech Republic

ACM ISBN 978-1-4503-6748-6/19/07.
https://doi.org/10.1145/3319619.3326797

2 LEARNING CLASSIFIER SYSTEMS (LCS)

LCS concept was developed by John Holland in the 1970s [9]. The term has been used to describe the family of machine learning (ML) algorithms that emerged from a founding concept designed to model complex adaptive systems [19].

LCS stores information using a vague concept of a *classifier*. A classifier is a rule (specifying the environmental conditions and corresponding action like IF-THEN clause) with some additional statistics. All knowledge derived from the environment is stored as a population of classifiers [P] and is refined with each interaction with the environment. New classifiers might be introduced into population [P] in two ways - by *covering* process or by *genetic algorithm*.

Covering creates classifiers precisely matching environmental perception and then randomly generalizing some attributes. From the other side genetic algorithms try to introduce new offspring into a population by mixing genotypes of most promising parents.

All rules existing in a population are being constantly refined with each interaction with the environment. Metrics describing the usefulness of the rule are responsible for deciding whether a given rule should be kept or be discarded.

The biggest advantage of LCS is in the possibility of modeling the output of the system (different domains) by using a set of understandable IF-THEN rules, that are covering only the portion of the input space. With this approach, the initially complex problem is broken into many simpler pieces. Other beneficial features are the ability to model complex patterns such as non-linear feature interaction (*epistasis*) or heterogeneous associations.

The drawbacks of the LCS-es such as (1) the usage of the naive *strength-based* fitness approach where it's value depends on the reward obtained from the environment or (2) the unrestricted process of GA application sometimes creating meaningless rules lead to creation of more robust variants (most popular Michigan-style are XCS, ZCS, UCS).

2.1 Anticipatory Learning Classifier Systems (ACS2)

In 1993 Hoffmann proposed a theory of *Anticipatory Behavioral Control* [7] that was further refined in [8]. It distinguishes between the following points (visualized in Figure 1):

(1) Any behavioral act or response (R) is accompanied by anticipation of its effects.
(2) The anticipations of the effects E_{ant} are compared with the real effects E_{real}.
(3) The bond between response and anticipation is strengthened when the anticipations were correct and weakened otherwise.
(4) The R - E_{ant} relations are further differentiated by behavioral relevant stimuli.

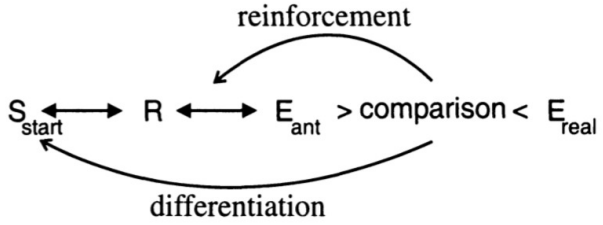

Figure 1: The theory of anticipatory behavioral control. Figure adapted from [3, p. 4].

That insight of presence and importance of anticipations in animals and man lead to the conclusion that it would be beneficial to represent and utilize them also in animats.

The first approach was undertaken by Stolzmann in 1997 [13]. He presented a system called ACS (*"Anticipatory Classifier System"*) enhancing the classifier structure with an anticipatory or effect part that anticipates the effects of an action in a given situation. To introduce new classifiers a dedicated component realizing Hoffmann's theory was introduced - *Anticipatory Learning Process*.

Later in 2002 Butz presented an extension called ACS2 [3]. Most importantly, he modified the original approach by an explicit representation of anticipations and by applying learning components across the whole action set [A]. The complete behavioral is presented on Figure 2 and the algorithm is described thoroughly in [5] [14].

In 2005 O'Hara and Bull took another approach designing anticipatory classifier system (X-NCS) by combining XCS system with ANN for predicting resulting states [11]. They replaced each traditional condition-action rule by a single, fully connected multilayer perceptron. Tests on Woods-1 and Maze-5 environments looks promising but no comparison to ACS2 system was made. However, the X-NCS system won't be discussed herein.

3 REAL-VALUED INTERVAL REPRESENTATION

In all further descriptions an assumption is made that single perception σ obtained from the environment is bounded to range $\sigma \in [0, 1]$.

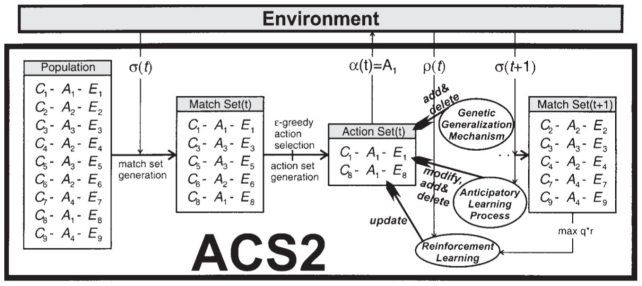

Figure 2: A behavioral act in ACS2 with reinforcement learning and anticipatory learning process application. Figure adapted from [3, p. 27].

If it's not true an additional pre-processing step should be applied beforehand.

3.1 Alphabet representation

As mentioned earlier LCS are versatile for various internal data representations (genotypes). Most popular representation for binary data uses *ternary representation*, where possible attributes values are encoded with three symbols - $\{0, 1, \#\}$.

For continuous input value still there are many approaches possible, that will be briefly explained.

The most simple case is to binary encode both interval ranges, expand the rule condition and effect parts twice and reuse the ternary alphabet without any algorithm modification. This technique is most primitive, because of the vast increase in problem complexity.

Unold and Mianowski [17] extended the binary alphabet for more states (partitions) where each one was mapped to different data range. The algorithm was evaluated on two real-valued multistep environments: the 1D linear corridor and the 2D continuous gridworld environments showing promising results. The drawback is, however, interval partitions are created upfront and cannot be changed in the process.

Dedicated alphabets were also created for XCS version of the algorithm (XCSR). In 1999 Wilson proposed a *center-spread representation* (CSR) [20], where interval was represented by two numbers - center of the range and the spread. Later in 2003 Stone and Bull analyzed two new representations *ordered-bounded* (OBR) and *unordered-bounded* (UBR) [16]. Both of them represent the range by using left x_1 and right x_2 bounds, but in OBR $x_1 < x_2$. By neglecting the ordering in UBR the crossover operator has greater chances of introducing better classifiers. In 2005 Dam and Abbass proposed also a *min-percentage* (MP) representation [6] trying to overcome some of the UBR drawbacks. In order to apply CSR, OBR, UBR and MP representations in XCS operators like covering, mutation and GA subsumption needed to be adjusted.

3.2 Binary encoding

Discretization of continuous value is done by binary encoding. Here the $[0, 1]$ range of available states is divided into 2^n states, where n is the number of bits used. Table 1 demonstrates the process.

Perception σ	1-bit	2-bit	3-bit	...	7-bits	...
0.0	0	0	0	...	0	...
0.1	0	0	0	...	12	...
0.2	0	0	1	...	25	...
0.3	0	1	2	...	38	...
0.4	0	1	3	...	51	...
0.5	1	2	4	...	64	...
0.6	1	2	4	...	76	...
0.7	1	2	5	...	89	...
0.8	1	3	6	...	102	...
0.9	1	3	7	...	115	...
1.0	1	3	7	...	127	...

Table 1: Examples of encoded values for different perceptions σ. Maximum resolution is calculated with 2^n, where n is the number of bits used. E.g. a range $[0.3; 0.6]$ encoded with 7 bits would be $[38; 76]$ for OBR/UBR or $[76, 38]$ for UBR encoding (order is irrelevant).

4 REAL-VALUED ACS (RACS)

Extending ACS system with real-valued alphabet facilitates the possibility of utilizing the algorithm on a more diverse set of problems, where different alphabets can be mixed, for example by using *attribute list knowledge representation* (ALKR) [1].

For the case of experiment, UBR was chosen for representing interval predicates. The original ACS2 algorithm cannot work "out-of-the-box" with that representation, so several changes were required. Most important are listed below.

- **Don't care symbol**. In rACS the feature attributes consists solely of interval ranges. The *"don't care"* and *"pass-through"* symbol is represented as a full-ranged interval (e.g. using 4 bit encoding - UBR(0, 15) or UBR(15, 0)).
- **Covering** The covering process introduces randomness when a new classifiers is added into population. A new parameter - *covering noise* ϵ_{cover} defines the maximum noise that can alter current perception. The noise ϵ is drawn from uniform random distribution $U[0, \epsilon_{cover}]$. When creating a new classifier each condition and effect attribute is spread UBR$(x_1 - \epsilon, x_2 + \epsilon)$ accordingly.
- **Mutation** Similarly, a new parameter - *mutation noise* $\epsilon_{mutation}$ is used for introducing slight disturbances. For each attribute of condition and effect perception string a noise ϵ is drawn from uniform distribution $U[-\epsilon_{mutation}, \epsilon_{mutation}]$ and added to the current value.
- **Subsumption** The mechanism was extended accordingly to analyze incorporating ranges.
- **Marking** Classifier's mark stores only single encoded exceptional perceptions (not intervals).

All changes are reflected in the source code that is publicly available.

5 EXPERIMENTS

rACS algorithm was implemented and evaluated in Python language [10][1]. Testing environments[2] were created in full compliance with the OpenAI Gym [2, 12] interface. Doing so facilitates other researchers with an easier comparison and benchmarking by using a common standard. Additionally, all experiments are fully reproducible by making use of tools such as Jupyter notebooks, which are included in the source code repository.

5.1 Environments

Experiments were performed on two single-step environments used for evaluating XCSR - real-multiplexer (rMPX) [20] and checkerboard [16]. In order observe anticipations a validation bit was appended to perception string and was activated when the agent proposed the correct action.

The Boolean n-bit multiplexer problem is conceptually based on the behavior of an electronic multiplexer (MUX), a device that takes multiple analog or digital input signals and switches them into a single output. Every trial a binary perception string of length n is randomly generated, which is divided into two parts - *address bits* and *register bits*. The target value is determined by the address bits and the value at the register bit they point to. An exemplary string generated by 6-bit multiplexer might look like `010110`. Here there are two address bits - `01` and four register bits - `0110`. The output value is located on the first register bit because $(01)_2 = (1)_{10}$, so the solution here is 1. For the rMPX the only difference between boolean multiplexer is that generated perception consists of real-values drawn from a uniform distribution. To validate the correct answer the additional variable - secret threshold $\theta = 0.5$ is used to map each allele into binary form. In order to solve rMPX a hyperplane decision boundary is sufficient (no interval-representation is needed). Also, the problem can be easily scaled in terms of complexity by changing the length of the generated perception vector.

The second testing environment is more complex. Checkerboard problem works by dividing up the n-dimensional solution space into equal-sized hypercubes. Each hypercube is assigned a white or black color (alternating in all dimensions), see Figure 3. The problem difficulty can be controlled by changing both the dimensionality n and the number of divisions in each dimension n_d. Keep in mind that in order to allow the colors to be alternating n_d must be an odd number. Finally, the environment presents a vector of length n_d, and the agent's goal is to guess the correct color.

5.2 Results

All experiments were executed in *explore-exploit* mode alternating in each trial. In exploring phase the agent was set to choose action fully randomly, in exploiting one the action from best-fitted classifier was chosen. Detailed metrics used for creating plots were collected every 5 trials. All experiments can be reproduced by running interactive notebooks localized inside the code repository.

5.2.1 Real Multiplexer. Parameters: $\beta = 0.05$, $\gamma = 0.95$, $\theta_r = 0.9$, $\theta_i = 0.2$, $\epsilon = 1.0$ $\theta_{GA} = 100$, $m_u = 0.1$, $\chi = 1.0$, $\epsilon_{cover} = 0$, $\epsilon_{mutation} = 0.25$.

[1] https://github.com/ParrotPrediction/pyalcs
[2] https://github.com/ParrotPrediction/openai-envs

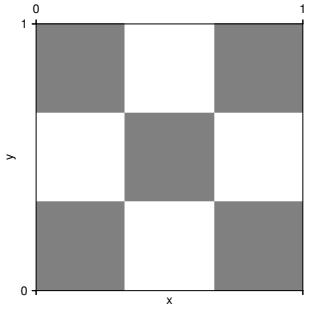

Figure 3: 2-dimensional checkerboard problem with $n_d = 3$.

The ability of modeling environment was examined on 3-bit rMPX. Preliminary tests are promising to show that the agent is able to capture the feature interaction. When using encoding with 1 or 2 bits (Figures 4, 5) the number of classifiers stabilizes after about 10000 trials. Introducing more accurate encoding is problematic. Figures 6, 7 (for 3 and 4 bits consecutively) shows that exploit phase is performing better, but an uncontrolled classifier growth is observed. All figures present the reward as the moving average calculated from the last 50 metrics.

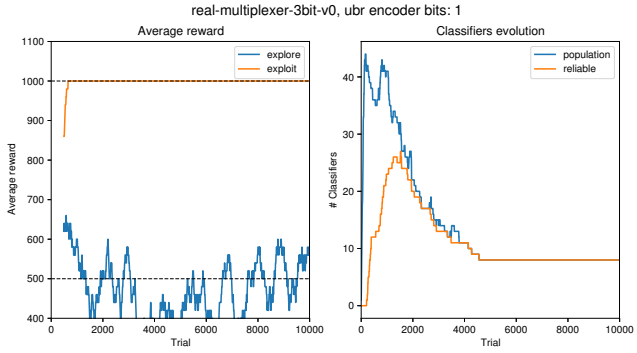

Figure 4: 3-bit rMPX. 1-bit continuous value encoding.

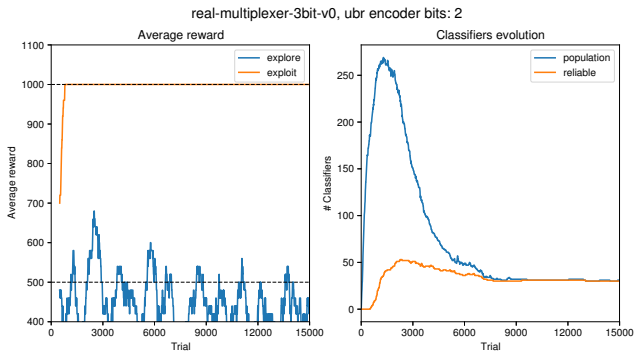

Figure 5: 3-bit rMPX. 2-bit continuous value encoding.

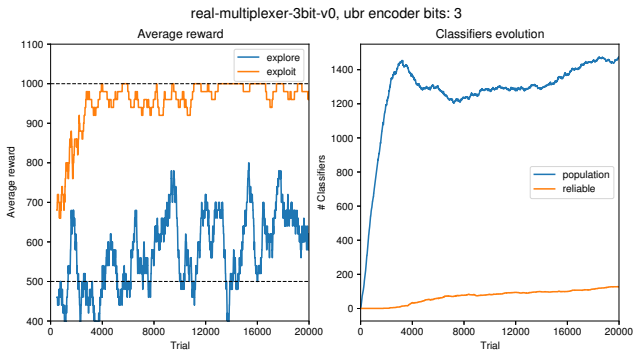

Figure 6: 3-bit rMPX. 3-bit continuous value encoding.

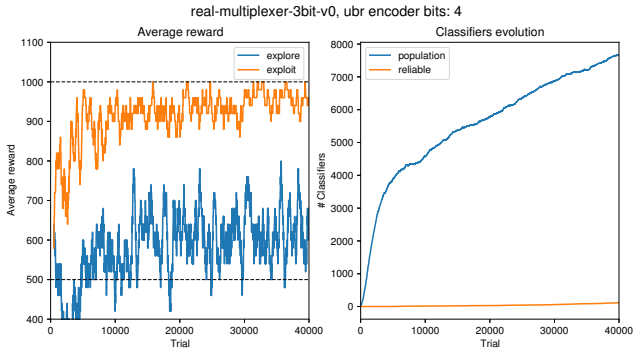

Figure 7: 3-bit rMPX. 4-bit continuous value encoding.

5.2.2 Checkerboard. Parameters: $\beta = 0.05$, $\gamma = 0.95$, $\theta_r = 0.9$, $\theta_i = 0.3$, $\epsilon = 0.9$, $\theta_{GA} = 100$, $m_u = 0.2$, $\chi = 0.6$, $\epsilon_{cover} = 0.1$, $\epsilon_{mutation} = 0.25$.

All checkerboard experiments were performed using 2-dimensional checkerboard with 3 splits in each dimension. Input perception was encoded with 4 bits. The reward for providing the correct answer was $\rho = 1$.

Figure 8 demonstrates that answers given in explore phase are near $\rho = 0.5$. This behavior is expected because in this phase agent is performing mostly random guessing. For the exploit phase, the average reward is noticeably better over time, meaning that the agent is able to learn the environment rules.

In order to visualize the evolution of condition perception strings Figure 9 presents the proportions of all *UBR* attributes. Interval regions were categorized into 4 regions:

- Region 1 $[p_i, q_i]$ - consists of specific intervals.
- Region 2 $[p_{min}, q_i)$ - interval bounded from the right side.
- Region 3 $[p_i, q_{max})$ - interval bounded from the left side.
- Region 4 $[p_{min}, q_{max})$ - general interval (*"don't care"*).

Figure 9 presents that the agent tends to favour creation of specialized intervals in the first region (about 80% of all attributes). However, in an ideal case for the test problems we would expect to obtain greater impact also of region 2 and region 3 (due to 3 splits of the checkerboard).

Finally, Figure 10 confirms the problem observed previously in rMPX environment regarding uncontrolled growth of classifier population. In this environment it's unable to obtain any reliable classifier (classifier cl is considered "reliable" when it's quality $cl.q > \theta_r$).

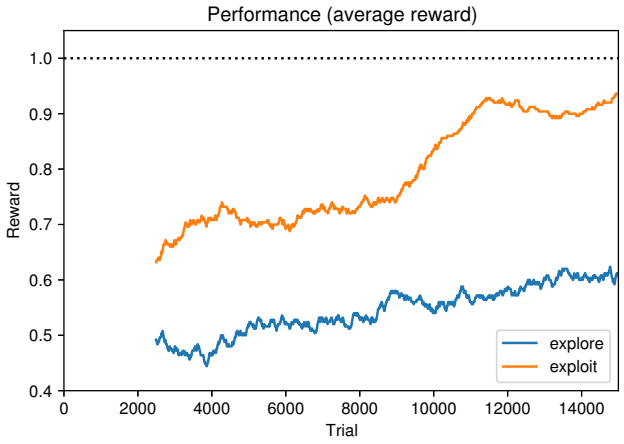

Figure 8: Average reward obtained in 2-d checkerboard problem with 4-bit binary encoding. Reward is averaged using 250 past metrics.

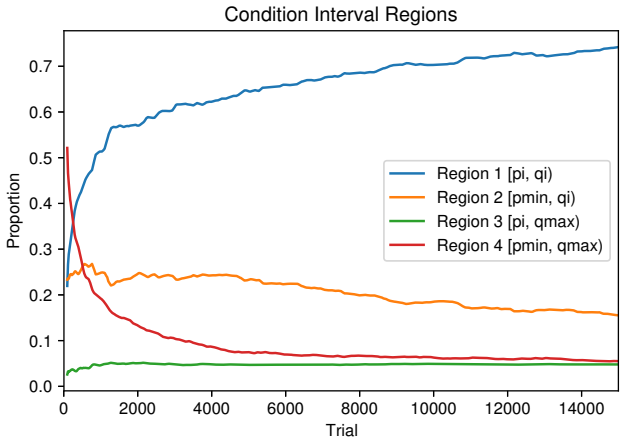

Figure 9: Proportions of UBR attributes in classifier condition parts.

6 CONCLUSIONS

The possibility of extending anticipatory classifier systems with the capability of representing intervals looks promising. Preliminary tests with two environments (from classical XCSR problems) revealed that the algorithm is able to exploit gathered knowledge substantially better than by random guessing.

However, there are some issues preventing the rACS from satisfying the assumptions established by Stolzmann - the created

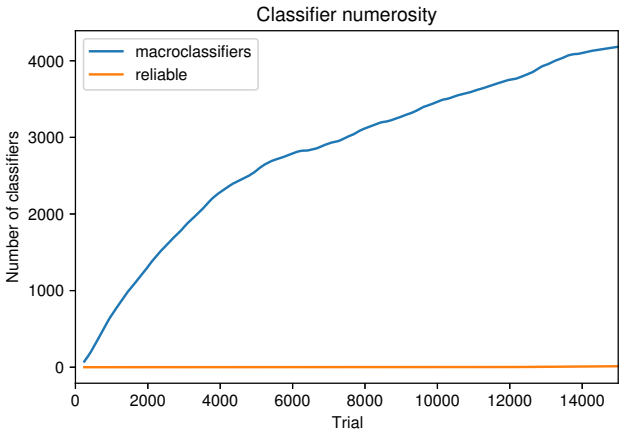

Figure 10: Classifiers created for 2D checkerboard environment.

classifiers should be accurate and maximally general. The most serious problem identified so far is related to the uncontrolled growth of the classifiers (see especially the Figure 10). The problem is caused by the fact that newly created classifiers are not compared to those already existing in the population [P]. The result is the creation of duplicated and overlapping off-springs covering the same niches. It's not the trivial issue because a new mechanism needs to be capable of subsuming and merging classifiers with similar ranges (condition and effect parts) meanwhile favoring more general classifiers.

Other aspects requiring further investigation:

(1) **impact of binary encoding**. Choosing an encoding scheme and a sufficient number of encoding bits. The problem must be carefully examined beforehand to select the best possible variant.
(2) **other forms of representing intervals**. Current experiments were using the most promising *UBR* representation. But applications of *hyper-alphabets* or *fuzzy logic* might also turn out to be beneficial.
(3) the **operators** used to create new classifiers working on interval ranges also need deeper investigation, e.g. - current implementation added noise in covering mechanism - which aids the system to better generalize but also introduces a chance of creating the wrong interval. This violates the original ALP principles, therefore the adapted and dedicated learning component should be devised from scratch,
(4) **lack of multi-step real-valued benchmarking environment** - in order to evaluate latent learning capabilities of rACS a new testing environment is required,
(5) impact of **generalized state values** for future multi-step problems [4].

7 ACKNOWLEDGMENTS

The work was supported by the statutory grant of the Wrocław University of Science and Technology, Poland.

REFERENCES

[1] Jaume Bacardit and Natalio Krasnogor. 2009. A mixed discrete-continuous attribute list representation for large scale classification domains. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. ACM, 1155–1162.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

[3] Martin V Butz. 2002. *Anticipatory learning classifier systems*. Vol. 4. Springer Science & Business Media.

[4] Martin V Butz and David E Goldberg. 2003. Generalized state values in an anticipatory learning classifier system. In *Anticipatory behavior in adaptive learning systems*. Springer, 282–301.

[5] Martin V. Butz and Wolfgang Stolzmann. 2002. An Algorithmic Description of ACS2. In *Advances in Learning Classifier Systems*, Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 211–229.

[6] Hai H Dam, Hussein A Abbass, and Chris Lokan. 2005. Be real! XCS with continuous-valued inputs. In *Proceedings of the 7th annual workshop on Genetic and evolutionary computation*. ACM, 85–87.

[7] Joachim Hoffmann. 2016. Vorhersage und erkenntnis. (2016).

[8] Joachim Hoffmann and Albrecht Sebald. 2000. Lernmechanismen zum Erwerb verhaltenssteuernden Wissens. *Psychologische Rundschau* (2000).

[9] John H Holland and Judith S Reitman. 1978. Cognitive systems based on adaptive algorithms. In *Pattern-directed inference systems*. Elsevier, 313–329.

[10] Norbert Kozlowski and Olgierd Unold. 2018. Integrating anticipatory classifier systems with OpenAI gym. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, 1410–1417.

[11] Toby O'Hara and Larry Bull. 2005. Building anticipations in an accuracy-based learning classifier system by use of an artificial neural network. In *2005 IEEE Congress on Evolutionary Computation*, Vol. 3. IEEE, 2046–2052.

[12] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. 2017. Mastering the game of Go without human knowledge. *Nature* 550 (Oct. 2017), 354–.

[13] Wolfgang Stolzmann. 1997. *Antizipative classifier systems*. Ph.D. Dissertation. Fachbereich Mathematik/Informatik, University of Osnabrück.

[14] Wolfgang Stolzmann. 2000. An Introduction to Anticipatory Classifier Systems. In *Learning Classifier Systems*, Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 175–194.

[15] Wolfgang Stolzmann and Martin Butz. 1999. Latent learning and action planning in robots with anticipatory classifier systems. In *International Workshop on Learning Classifier Systems*. Springer, 301–317.

[16] Christopher Stone and Larry Bull. 2003. For real! XCS with continuous-valued inputs. *Evolutionary Computation* 11, 3 (2003), 299–336.

[17] Olgierd Unold and Marcin Mianowski. 2016. Real-Valued ACS Classifier System: A Preliminary Study. In *Proceedings of the 9th International Conference on Computer Recognition Systems CORES 2015*, Robert Burduk, Konrad Jackowski, Marek Kurzyński, Michał Woźniak, and Andrzej Żołnierek (Eds.). Springer International Publishing, Cham, 203–211.

[18] Olgierd Unold and Krzysztof Tuszyński. 2008. Mining knowledge from data using anticipatory classifier system. *Knowledge-Based Systems* 21, 5 (2008), 363–370.

[19] Ryan J. Urbanowicz and Will N. Browne. 2017. *Introduction to Learning Classifier Systems* (1st ed.). Springer Publishing Company, Incorporated.

[20] Stewart W Wilson. 1999. Get real! XCS with continuous-valued inputs. In *International Workshop on Learning Classifier Systems*. Springer, 209–219.