Received March 13, 2022, accepted March 22, 2022, date of publication March 28, 2022, date of current version April 1, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3162925

Internalizing Knowledge for Anticipatory Classifier Systems in Discretized Real-Valued Environments

NORBERT KOZLOWSKI AND OLGIERD UNOLD
Department of Computer Engineering, Faculty of Computer Science and Telecommunications, Wrocław University of Science and Technology, 50-370 Wrocław, Poland

Corresponding author: Norbert Kozlowski (norbert.kozlowski@pwr.edu.pl)

ABSTRACT Real-valued environments are challenging for learning systems because of a significant increase in the input space size of the problem. This work demonstrates that Anticipatory Learning Classifier Systems (ALCS) can successfully build sets of conditional rules foreseeing the consequences of executed actions. Three major classes of Learning Classifier Systems - Anticipatory Classifier System (ACS), ACS2, Yet Another Classifier System (YACS), alongside the traditional Dyna-Q algorithm implementations were adapted to handle real-valued input signal discretization. Aspects like the ability to capture all possible interactions, model generalization capabilities, size of the solution of relative execution times were compared in four different problems using probabilistic modelling, providing unbiased judgments. Results proved that the examined ALCS are capable of solving selected problems. Despite increased input size, all possible environmental transitions were learned latently, without obtaining any explicit incentives. Such an internal representation provides a more compact solution representation and can optimize learning speed further by executing imaginary environmental interactions or performing action planning for a new set of potential problems.

INDEX TERMS Genetic algorithms, latent learning, learning classifier systems, reinforcement learning.

I. INTRODUCTION

The Reinforcement Learning (RL) framework considers adaptive agents involved in a sensori-motor loop interacting with the environment [1]. Here, the agent acts as the decision-maker and tries to influence the environment through actions. As a result, it obtains the scalar reward and observes a new environmental state. In this work, the agent is a *model-based* learner with a goal of maximizing the sum of environmental rewards by constructing a functional representation of an environment. Such an internal model comprise the transition function estimation and the reward signal. The transition function links a successive state s' to a state-action pair (s, a) and defines the probability of reaching that future state given state-action pair. The reward signal maps a transition tuple (s, a, s') to a scalar value ranking possible movements.

The associate editor coordinating the review of this manuscript and approving it for publication was Kathiravan Srinivasan.

An internal model of the environment can be developed by applying particular psychological findings. In contrast with behaviourist theories, Tolman postulated that animals develop a sort of cognitive map representing the surrounding world [2]. Later, Seward provided further empirical evidence by performing experiments with rats in mazes without incentives [3]. This concept of exploring the environment without any rewards or punishments is referred to as *latent learning*.

Latent learning capabilities are interesting because they are independent of the obtained reward and might improve the speed of building the agent's environmental model. When an agent interacts with the environment, the consequence of the action comprises both a possible reward as well as a new situation. Thus, the agent might be able to predict the sequence of the following states before executing an action, thus *internalizing knowledge* about the environment. This can be utilized to plan the next moves or speed up the RL process by simulating hypothetical situations [1].

In this paper, we examine the latent-learning capabilities of learning classifier systems (LCS) to solve RL problems.

Holland proposed LCS [4] showing their advantage over other RL systems, like Q-learning, by the generalisation ability [5], [6]. The framework has been successfully applied in multiple domains [7], like data mining [8]–[12], medical diagnosis or military operations [13]. By aggregating several situations into a single rule, LCS have the edge over tabular RL by effectively reducing both the model size and learning time.

A particular class of Michigan-style LCS, called Anticipatory Learning Classifier Systems (ALCS), capable of building rules in form condition-action-effect triples, is investigated herein. This structure makes it possible to anticipate a group of possible successive states related to executing the desired action in a particular condition. Stolzmann applied ALCS in robotics by performing two latent learning experiments [14]. In the first one, Khepera robots simulated rats in the maze; second, a robotic arm was coordinated with a camera in the hand-eye task. It was shown that it is possible to optimise the learning process by action planning and goal-directed learning mechanisms in both cases.

The main contribution includes adjusting ALCS implementations to real-valued perception and performing diverse tests across selected scenarios with ranging difficulties. Moreover, most of the selected benchmarking problems use a real-valued signal representation that better resembles real-world conditions and proves broader applicability. Successful adaptation to such input perception was performed for the XCS system with both tabular knowledge storage [15]–[18], and by using the piecewise-linear function approximation techniques [19], [20]. To our knowledge, such research including ALCS, has not yet been performed before. The examined algorithms were analysed in terms of their latent learning capabilities, thus revealing their merits and perils. Finally, all results were obtained using publicly available software, are easily reproducible and extensible.

Section II will explain ALCS systems like ACS, ACS2, YACS, alongside Dyna-Q. Particular attention will be placed on latent-learning capabilities and how rules can be created and utilised. Next, the "Testing Scenarios" section describes four benchmark problems with varying difficulties. The majority of them are discretised real-valued environments. Preliminary research performed in section III proved that ALCS can handle an increased input space caused by the real-valued input [21], [22]. Statistical comparison and conclusions are presented in sections IV and V accordingly. Finally, section VI proposes possible future directions.

II. MATERIALS AND METHODS

A. LATENT LEARNING IN LEARNING CLASSIFIER SYSTEMS

Learning Classifier Systems [4] are a family of flexible, evolutionary, rule-based machine learning systems involving a unique tandem of local learning and global evolutionary optimisation of the collective model localities. They describe a framework that consists of the discovery and learning components. Despite the misleading name, they should be instead viewed as a general, distributed optimisation technique.

The system stores the knowledge in an evolving population [P] of classifiers. All classifiers comprise an IF-THEN rule describing the environmental state, which could be applied alongside a few other metrics explaining their relevance. The discovery component might introduce new classifiers into the population by utilising two processes – covering and heuristics. The covering creates new classifiers with a rule matching exactly the environmental state. Then, some generalisation is introduced randomly. The heuristic (most often the genetic algorithms) attempts to mix rules of promising individuals from the population [P]. All rules are re-evaluated continuously by the learning component during the interaction with the environment, changing their metrics accordingly. Accurate classifiers are kept in population, while the others are discarded. The measure used to assess classifiers' performance was initially based on the obtained reward associated with the particular classifier and is termed as a *fitness value*.

In the most popular LCS modification – XCS [15], the classifier fitness is based on the accuracy of a classifier's payoff prediction instead of the prediction itself, and the learning component responsible for local optimisation follows the Q-learning pattern. In the next step, the classifier's predictions are updated using the immediate reward feedback. The main difference between XCS and Q-Learning is that, in XCS, it entails the prediction of a general rule that is updated, whereas, in Q-learning, the prediction is associated with an environmental *state-action* pair.

The latest LCS advancements focus mainly on problems related to knowledge visualization and rules compaction [23], [24], learning with incremental data [25], [26], classifying images using convolutional autoencoders [26]–[28], or dealing with perceptual aliasing environments [29].

However, in this paper, yet another family of LCS is considered – Anticipatory Learning Classifier Systems. They are differentiated from others so that a predictive schema model of the environment is learned rather than the reward prediction maps. In contrast to the usual classifier structure, classifiers in ALCS have a state prediction or an anticipatory part that forecasts the environmental changes caused when executing the specified action in the specified context. Knowing the consequences of the action classifiers naturally involves latent learning properties. Similarly, as in the XCS, ALCSs derive classifier fitness estimates from the accuracy of their predictions; however, the accuracy of anticipatory state predictions is considered rather than the reward prediction accuracy. The following sections describe the mechanism used for learning proper rules without environmental reward in three popular ALCS implementations – ACS, ACS2 and YACS alongside the Dyna-Q benchmark used as a reference.

B. ACS AND ACS2

Hoffmann proposed a theory of a psychological theory of anticipatory behavioural control [30], stating that conditional action-effects relations are learned latently using

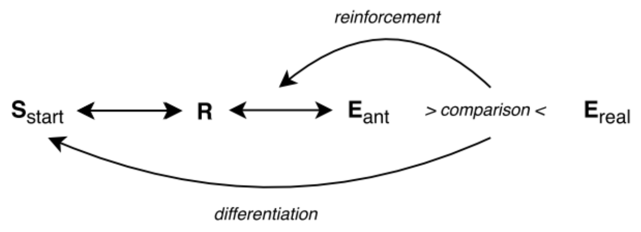


FIGURE 1. The theory of anticipatory behavioural control. Adapted from [31] p. 4.

anticipations, which he further refined in [31]. The following points (visualised in Figure 1) can be distinguished:

1) Any behavioural 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) When the anticipations were correct, the bond between response and anticipation is strengthened and weakened otherwise.
4) Behavioural stimuli further differentiate the $R - E_{ant}$ relations.

From this insight into the presence and importance of anticipations in animals and man, it can be inferred that representing and utilizing them in animats would be beneficial.

The first approach was undertaken in 1997 by Stolzmann [32]. He presented a system called ACS (“*Anticipatory Classifier System*”), enhancing the classifier structure with an effect part anticipating the consequences of an action in a given situation. A dedicated component realizing Hoffmann’s theory was introduced – *Anticipatory Learning Process* (ALP) to introduce new classifiers.

The ACS starts with a population $[P](t)$ of most general classifiers (’#’ in a condition and effect parts) for each action. To ensure the presence of a classifier in every consecutive situation, it is unable to delete those classifiers. During each behavioural act, the current perception of the environment $\sigma(t)$ is captured. Then, a match set $[M](t)$ is formed that comprises all classifiers from $[P](t)$ where the condition is found to match the perception $\sigma(t)$. Next, one classifier cl is drawn from $[M](t)$ using a certain strategy. Then the classifiers action $cl.a$ is executed in the environment, after which a new perception $\sigma(t+1)$ and reward $\rho(t+1)$ values are presented to the agent. Knowing the classifier anticipation and the current and next state, the ALP module can then adjust the condition and effect parts of the classifier cl. Certain cases might occur based on this comparison. In the *useless case*, no change in perception is perceived from the environment after taking a given action. If so, the quality $cl.q$ is decreased. In the *unexpected case*, the new state $\sigma(t+1)$ is not shown to match the prediction of $cl.E$. Then, a new classifier with a matching effect part is generated. The incorrect one is penalized as before. The last case is the *expected case* when the new state matches the classifier prediction, increasing its quality. After the ALP application, RL also updates attributes pertaining to obtained reward $\rho(t+1)$ based on the overall prediction’s correctness.

Later, in 2002 Butz [33] presented an extension called ACS2. The significant changes from the previous version include:

1) explicit representation of anticipations,
2) application of learning component across the whole action set $[A]$ (all classifiers from $[M]$ advocating selected action),
3) introduction of *Genetic Generalization* module for introducing new classifiers using promising offspring.

The complete behavioural act is presented in Figure 2, and the algorithm is described thoroughly in [14], [34]. The recent advancements for the ALCS family include the integration of the action planning mechanism [35], PEPACS extension where the concept of *Probability–Enhanced–Predictions* is used for handling non-deterministic environments [36] or BACS tackling the issue of perceptual aliasing by building *Behavioral Sequences* [37].

C. YACS

Gérard introduced a “*Yet Another Classifier System*” (YACS) [38] in 2002. It shares the same $C - A - E$ classifier structure as ACS and ACS2. The essential conceptual difference between those two is that YACS is designed to decorrelate the acquisition of relevant C and E parts. It builds them independently using a set of heuristics.

First, an E part representing the information about the anticipated effects occurring in the environment is built. Then, the second part of the process is in charge of discovering the C parts relevant to it. A C part must discriminate between situations, so that the E part discovered by the first mechanism is always correct when the classifier is applied.

YACS learns E parts by directly comparing the successive perceived situations. It remembers the last perceived situation and the last performed action. Knowing the current situation $\sigma(t)$ resulting from executing action A_{t-1} in situation $\sigma(t-1)$ at each time step, YACS computes the *desired effect* (DE). It refers to the ideal effect part of the classifier, which could have been fired at the preceding time step.

After forming the action set $[A]$, each classifier E part is compared to DE. If there are no matches, YACS creates a new classifier. Additionally, each classifier keeps track of whether recent matches were positive or negative. At a later stage, this property named *trace* is used for determining whether the classifier is accurate.

The C part of the classifier should be most general and as specific as possible, and this goal is achieved incrementally. Here neither mutation nor crossover operators, but the *mutspec* is used. It generates new classifiers by selecting one general feature of the C part and then creating one new classifier per possible value. In this way, each newly created classifier applies to less possible situations than the original one. The authors describe the process of a careful selection of a feature to specialize as the *expected improvement by specialization* [38].

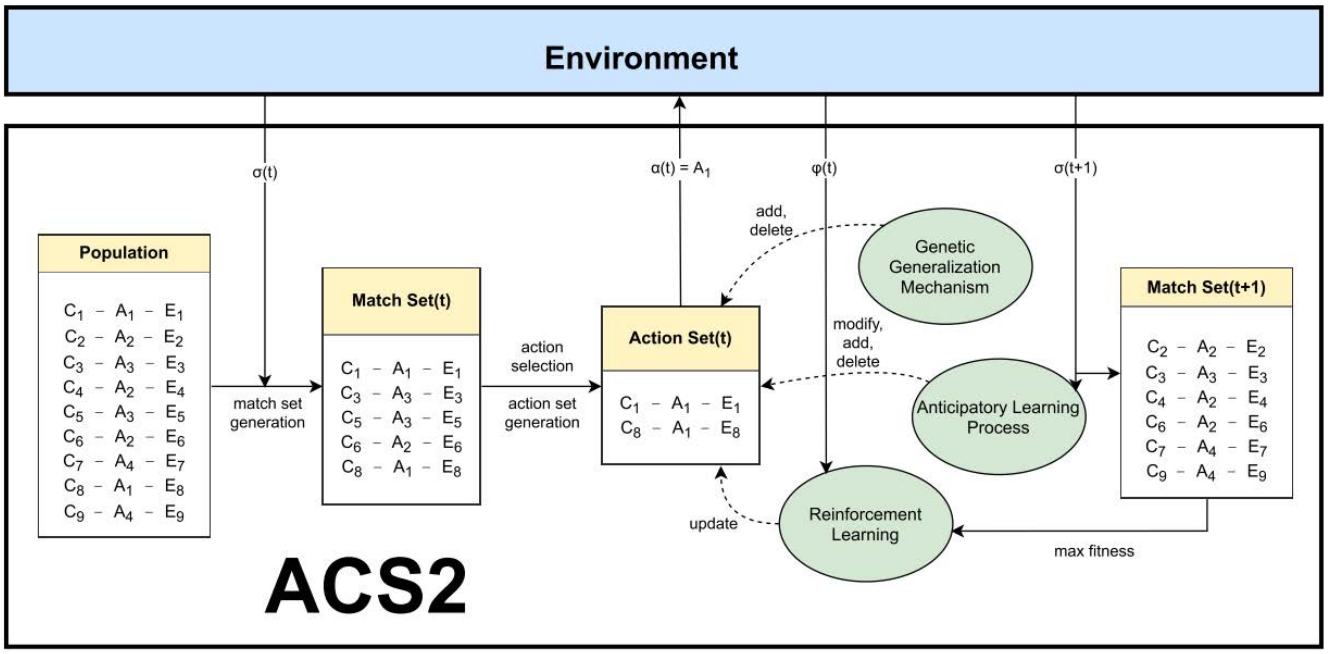


FIGURE 2. **A behavioural act in ACS2. Environmental model is stored as a set of rules as a population and later refined by GA, ALP and RL modules. Figure adapted from [31] p. 27.**

On the other side, ACS learns latently using the mechanism of "*specialization of changing components*". When the classifier anticipates badly, the ALP process may create a new one by specializing both C and E parts simultaneously and incrementally. Moreover, ACS2 relies on the genetic algorithm to correct over-specialization cases and reduce the population size.

D. DYNA-Q ARCHITECTURE

Basic RL algorithms like Q-Learning cannot perform "cognitive" operations, such as reasoning and planning (because they do not learn an internal model of the environment's dynamics). Dyna-Q architecture [39] is an online planning agent with an internal environmental model. In later experiments, it was used to compare and highlight the generalisation capabilities of LCS.

Each (S_t, A_t) tuple outputs a prediction of the resultant reward and next state (R_{t+1}, S_{t+1}). The environmental model is table-based and assumes the environment is deterministic. After each transition $(S_t, A_t) \rightarrow (R_{t+1}, S_{t+1})$ the model records in its table entry for (S_t, A_t) the prediction that (R_{t+1}, S_{t+1}) will deterministically follow.

Real experiences are augmented by performing learning steps using the internal model when interacting with the environment. The planning here is the random-sample one-step tabular Q-learning method that only uses previously experienced samples.

E. OTHER ANTICIPATORY CLASSIFIER SYSTEMS

There are other approaches in designing anticipatory classifier systems as well - MACS, AgentP and X-NCS. The Modular Anticipatory Classifier System (MACS) [40] introduces a new formalism where the classifiers capture changes only in limited perception attributes. Therefore, it can model certain regularities more efficiently. The AgentP model [41] was created for dealing with aliased states, and the X-NCS [42] combines the XCS with artificial neural networks, where two fully connected multilayer perceptrons replace a conventional condition-action rule.

Given that they are less popular and there are fewer reliable implementations and benchmarks available checking them reliably was challenging and not included in the comparison.

F. TESTING SCENARIOS

Four stationary toy problems were selected to compare the algorithm performance. Three of them present a vector of real values to the agent and are scalable. The last is a simple discrete multi-step benchmark problem used by the YACS authors to further highlight the latent-learning mechanism's capabilities. The real-value signal discretization sequence is presented in Figure 3. Since the overall goal is to learn the environmental model correctly, each problem presents different challenges. All environments were created accordingly to the OpenAI Gym [43] interface and are available publicly.[1]

1) CORRIDOR

The corridor is a 1D multi-step, linear environment introduced by Lanzi to evaluate the XCSF agent [44]. The system output is defined over a finite discrete interval $[0, n]$. On each trial, the agent is placed randomly on the path and can execute two possible actions - move left or right (which corresponds to moving one unit in a particular direction - see Figure 4).

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

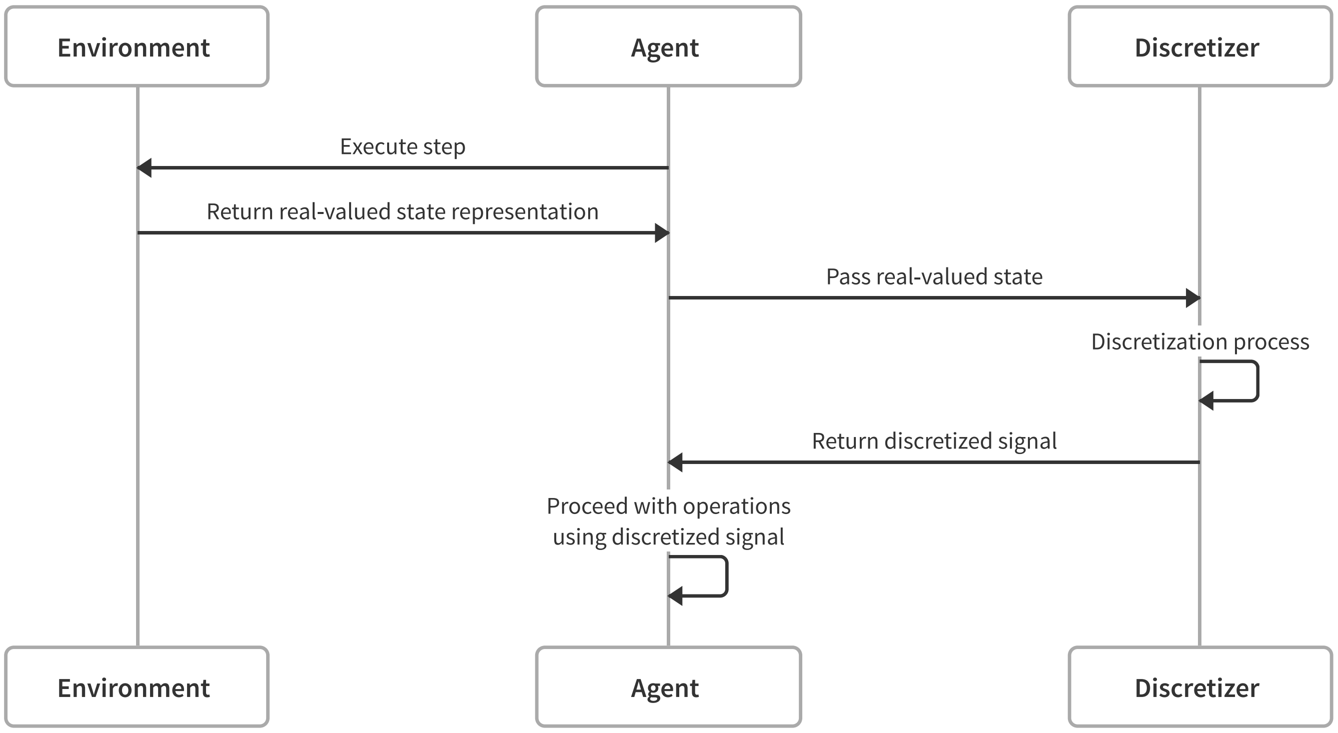

Environment
Agent
Discretizer
Execute step
Return real-valued state representation
Pass real-valued state
Discretization process
Return discretized signal
Proceed with operations using discretized signal
Environment
Agent
Discretizer

FIGURE 3. **Sequence flow for processing real-valued signal. The discretizer can be perceived as external component executing the desired strategy - i.e. creating *k* separate bins or using hashing functions. Finally, the agent is unaware of real-valued input and can work without any significant modifications.**

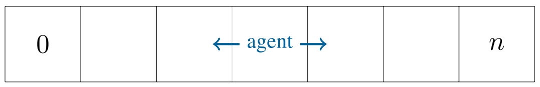


FIGURE 4. **The Corridor environment. The size of the input space is *n*. The agent perceives a one-element vector denoting its position.**

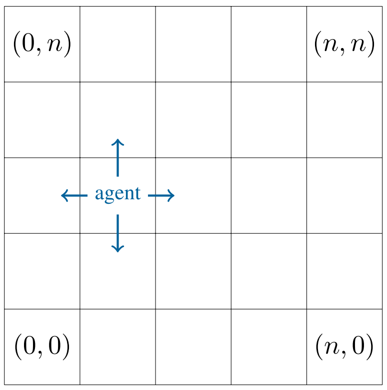


FIGURE 5. **The Grid environment. The size of the input space is n^2. The agent perceives a vector of two elements denoting its coordinates.**

The trial ends when it reaches the final state n (obtaining reward $r = 1000$) or when the maximum number of steps is exceeded.

Lanzi used a real-valued version of this environment where the agent location is denoted by a value between [0, 1]. Predefined step size was added to the current position when it executed an action, thus changing its value. The reward is paid out when the agent reaches the final state $s = 1.0$.

The environment examined herein signifies the state already in discretized form divided into 20 distinct states.

2) GRID

Grid refers to an extension of the Corridor environment [44] by adding a vertical dimension and two new actions (move up, move down). The raw agent perception is now identified as a pair of real numbers (s_0, s_1), where $s \in [0, 1]$. Similarly, the environment is presented to the agent in a discretized form. Each dimension is divided into $n - 1$ equally spaced buckets. The goal is to reach the reward located at position (n, n) - see Figure 5.

3) REAL-MULTIPLEXER (rMPX)

The modification to the traditional Boolean Multiplexer (MPX) was introduced by Wilson [16] to examine the performance in single-step environments using real-valued data. For the rMPX, the only difference between boolean multiplexer is that generated perception consists of real-value attributes 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. However, the standard version is still not suitable to be used with ALCS. Because an agent utilizes perceptual causality to form new classifiers, assuming that after executing an action, the state will change. The MPX does not have any possibility to send feedback about the correctness of the action.

Butz suggested two solutions to this problem [33]. In this paper, we assume that the state generated by the rMPX is extended by one extra bit, denoting whether the classification

TABLE 1. Example of discretizing real-valued input using 6 bins. A random perception signal of 6-bit rMPX extended with 1 bit (last column) was used. The size of the input space is $2 \cdot 6^6 = 93312$ unique states.

Original	0.86	0.29	0.56	0.89	0.33	0.49	0.0
Discretized	5	1	3	5	2	2	0

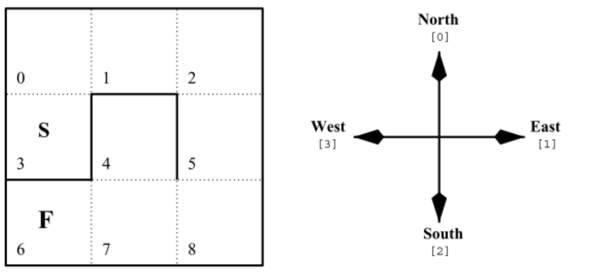


FIGURE 6. The discrete Simple Maze environment. The goal can be reached optimally in eight successive steps.

was successful. This bit is by default set to zero. The agent responds correctly after being switched, thus providing direct feedback. A detailed example can be found in [21].

A k bins discretizer is used to convert real numbers into integers. Its value is used to control the accuracy of generated rules – see Table 1. Thus, the input space of the environment can be calculated as $2k^n$, where k refers to the number of bins and n the length of the MPX signal. This environment is also particularly interesting because it possesses the properties of *epistasis* (non-linear feature interaction) and *heterogeneity*.

When the correct answer is given, the reward $r = 1000$ is obtained, otherwise $r = 0$.

4) SIMPLE MAZE

Gérard and Sigaud presented a *Simple Maze* environment in [45] to evaluate the properties of the YACS agent. It is a partially observable Markov problem where the agent is placed in the starting location (denoted as “S”), at the beginning of each new trial, and the goal is to reach the final state “F” by executing four possible actions – moving north, east, south or west – see Figure 6.

The agent perceives its surroundings (presence of path or walls) in four cardinal directions. The decision to move onto the wall does not affect the agent’s position - it remains in the same cell. After reaching the final location, the trial is finished, the agent is moved into starting position, and the reward of $r = 1$ is paid out.

III. RESULTS

ALCS algorithms were compared across four described environments to assess the latent learning capabilities. Four metrics – population size, model knowledge, generalisation and average trial time were captured.

- The population size represents how many rules are needed to model all possible transitions. The lower number means that the algorithm is more efficient in

TABLE 2. Classifier structure comparison in Grid environment for the (18, 19) state. The population was created after 25 explore trials. For each action, ACS2 manages to create a correct list of classifiers. The ACS is slower, and an initial default classifier accompanies each action. Finally, the YACS cannot create fully general and accurate classifiers at all.

ACS	**ACS2**	**YACS**
18 # ← 17 # \# # ← # #	18 # ← 17 #	18 19 ← 17 # 18 19 ← 3 #
18 # → 19 # \# # → # #	18 # → 19 #	18 19 → 19 # 18 19 → 7 #
\# # ↑ # #	\# 19 ↑ # #	\# 19 ↑ # #
\# 19 ↓ # 18 \# # ↓ # #	\# 19 ↓ # 18	18 19 ↓ # 18

rule compacting and performs latent learning more optimally.
- The knowledge metric checks every possible transition in the environment and determines if there is a classifier or rule inside the population for correctly representing such transition. The growth rate represents how quickly the internal model of the environment is built.
- The generalisation is the percentage of “*wildcards*” in the classifiers’ condition parts. A higher number means that model is generalising better (wildcard matches any value for a certain attribute). This metric does not apply to Dyna-Q. Ideally, it should be negatively correlated with population size.
- Moreover, a purely technical indicator is used for measuring the relative time needed to complete one learning trial. It allows comparing the effectiveness of each model in terms of execution speed. Because all examined ALCS in each trial perform exhaustive population look-ups, the execution time is estimated to increase with population size.

Additionally, the ACS2 algorithm is tested in its basic form alongside two extensions enabled - (1) the *genetic generalisation* and (2) the *Optimistic Initial Quality* (OIQ) [22].

1) The genetic generalisation is responsible for introducing new offspring classifiers by using operators such as *mutation* and *crossover*. ACS2 determines if this module should be fired by calculating its average time from the last application. Subsequently, two parent classifiers are selected from the action set $[A]$ using the roulette-wheel selection algorithm. Two offspring classifiers are created, to introduce wildcard attributes or perform two-point crossover randomly.
2) The OIQ ensures that all newly created classifiers are “optimistic” by increasing their initial quality metric. This modification was inspired by the *Optimistic Initial Values* introduced by Sutton in [1]. The value was increased from $cl.q = 0.5$ to $cl.q = 0.8$. The rationale for this modification is that classifiers’ fitness is partially based on quality. A greater value should cause faster convergence of an optimal number of classifiers in the population.

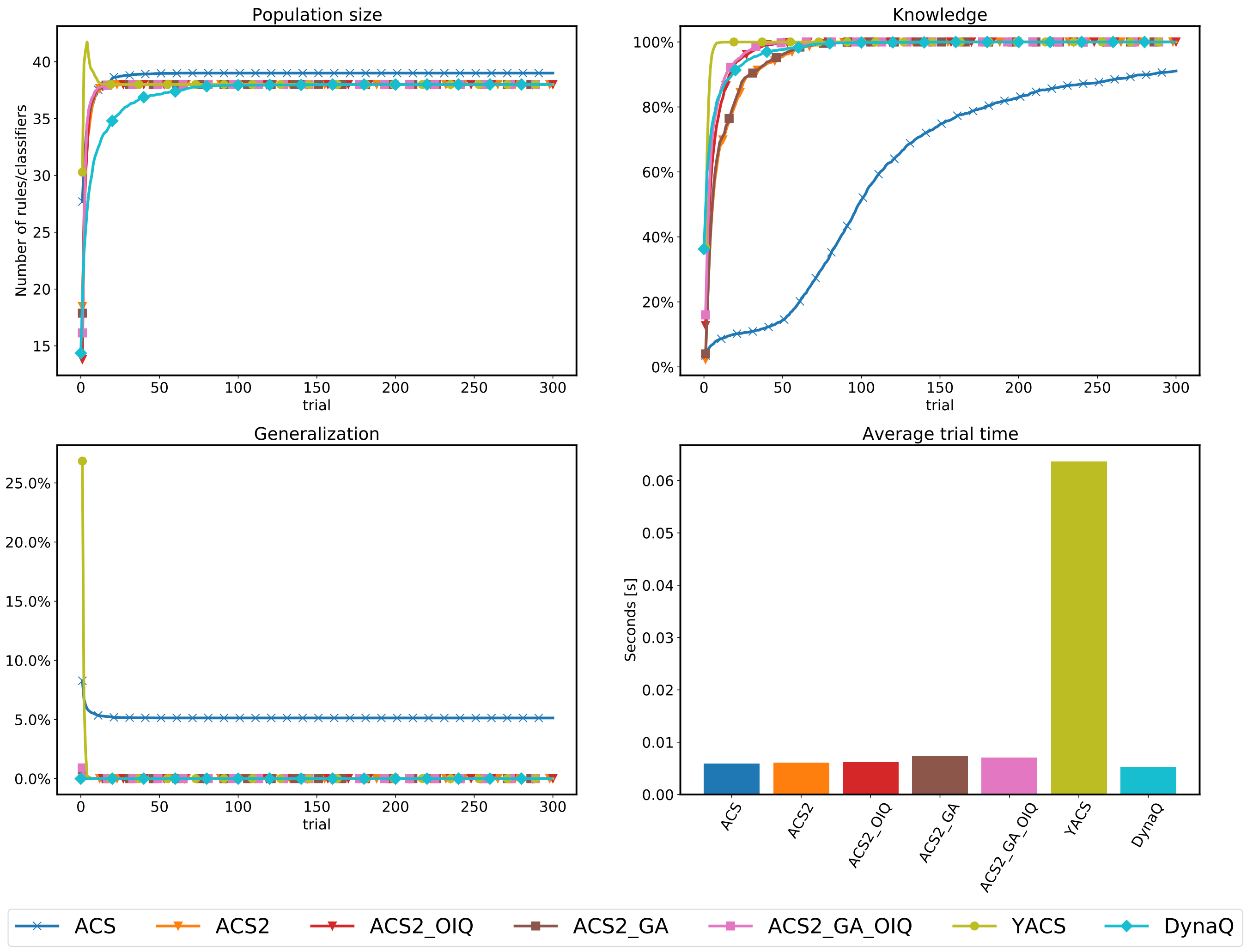


FIGURE 7. **Latent learning comparison in the Corridor environment discretized into 20 states.**

Experiments were performed in Python language using the open-sourced PyALCS[2] library [46]. All evaluated algorithms select an action using epsilon-greedy strategy. Random possible action is chosen when the threshold of $\epsilon = 0.5$ is greater than a random value drawn from a uniform distribution. Otherwise, the currently known best move is proposed. Additionally, all calculations were performed multiple times before being averaged to smooth the obtained results. Relevant environments were taken from the OpenAI Gym extension package.[3] The code for reproduction is delivered as an interactive Jupyter Notebook [4] and is publicly available.

Common parameters that were used across the experiments included the following: learning rate $\beta = 0.1$, exploration probability $\epsilon = 0.5$, discount factor $\gamma = 0.95$, inadequacy threshold $\theta_i = 0.1$, reliability threshold $\theta_r = 0.9$, YACS trace length 3, OIQ initial quality $q_{oiq} = 0.8$. The Dyna-Q algorithm performs five steps ahead simulation in each trial. Additionally, each experiment was executed 50 times and the results were averaged.

A. CORRIDOR

The first testing environment is the real-valued Corridor that is discretized into 20 distinct states. Figure 7 illustrates that almost all algorithms except ACS manage to stabilize the population size and ultimately learn the environment in about 100 trials. Although the environment does not expose any generalization capabilities, ACS created some invalid general classifiers. Therefore, the population size is larger than other algorithms, and knowledge acquisition is slower. Due to the aggressive classifier creation mechanism, the YACS learns the environment almost instantly, but the exhaustive heuristic needed for evolving classifiers results in a much slower execution time than other agents due to internal representation of all visited states.

B. GRID

The Grid environment increases the difficulty by adding one dimension to the Corridor. As was done before, it is

[2]https://github.com/ParrotPrediction/pyalcs/releases/tag/3.0.0
[3]https://github.com/ParrotPrediction/openai-envs/releases/tag/3.0.0
[4]https://zenodo.org/record/6347037

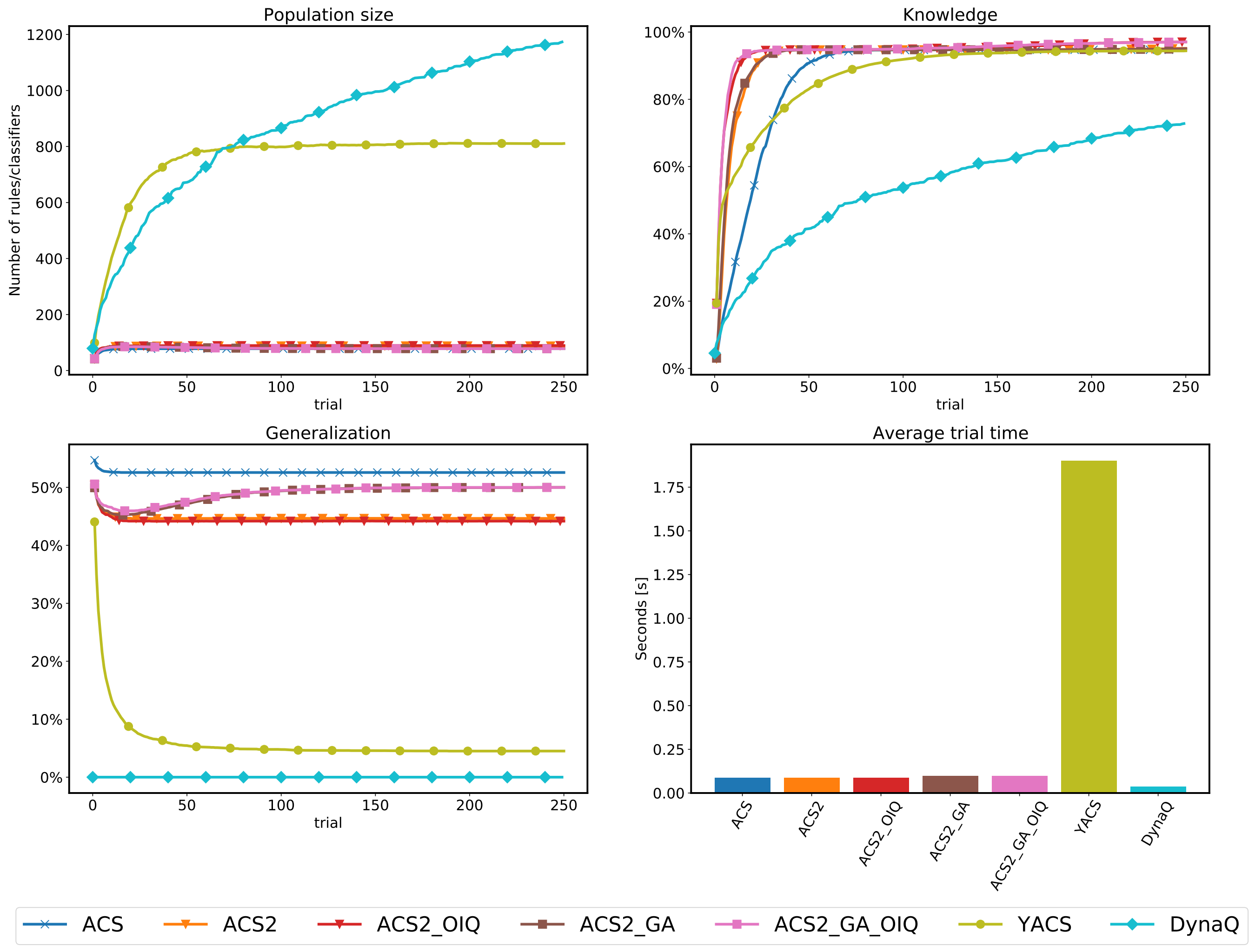


FIGURE 8. Latent learning comparison in the Grid environment discretized into 400 states.

discretized into $20 \cdot 20 = 400$ distinct, discrete states. The significant change is that now rule generalization is possible. In Figure 8 the population size shows that the Dyna-Q creates distinct rules for each state and action combination. Meanwhile, ACS and ACS2 algorithms model the environment with minimal rules. Classifiers from ACS2 GA reached a maximum of 50% generality. Although the ACS seems to have a greater generalization score, some classifiers are still over-general. As before, the YACS is the slowest implementation among all tested algorithms.

The population generated by YACS is non-deterministic; exemplary differences in the classifier structure is obtained after an arbitrary run is depicted in Table 2. It becomes evident that YACS lacks the generalization mechanism by creating over-specialized rules.

C. REAL-MULTIPLEXER

The rMPX is the only single-step environment considered. The examined version is a 3-bit, where one bit represents the location of the correct answer, and the remaining two bits hold arbitrary values. To determine whether the real-valued signal is either 0 or 1, the threshold of 0.5 was applied. In addition, each observation attribute was discretized into ten distinct values, which led to 2000 individual states. The environment is pretty easy to scale by changing the signal length and discretization factor, making it a good benchmark problem. Moreover, the signal needs to be estimated back from a discrete form into continuous to calculate the models' knowledge. With a higher discretization resolution, this may not be entirely feasible.

Algorithms' performance is presented in Figure 9. Interestingly, basic ACS did not manage to evolve any reliable classifiers through the experiments. Those that are created are over-general and did not learn any desired knowledge. It is also notable that the Dyna-Q explicitly learned all possible transitions. The rule acquisition is faster in earlier trials due to the stochastic nature of the environment. Classifiers created by YACS are the most specific from all ALCSs. It learns the environment dynamics very rapidly, but the created rules are oscillating (initial phases of gaining knowledge). The

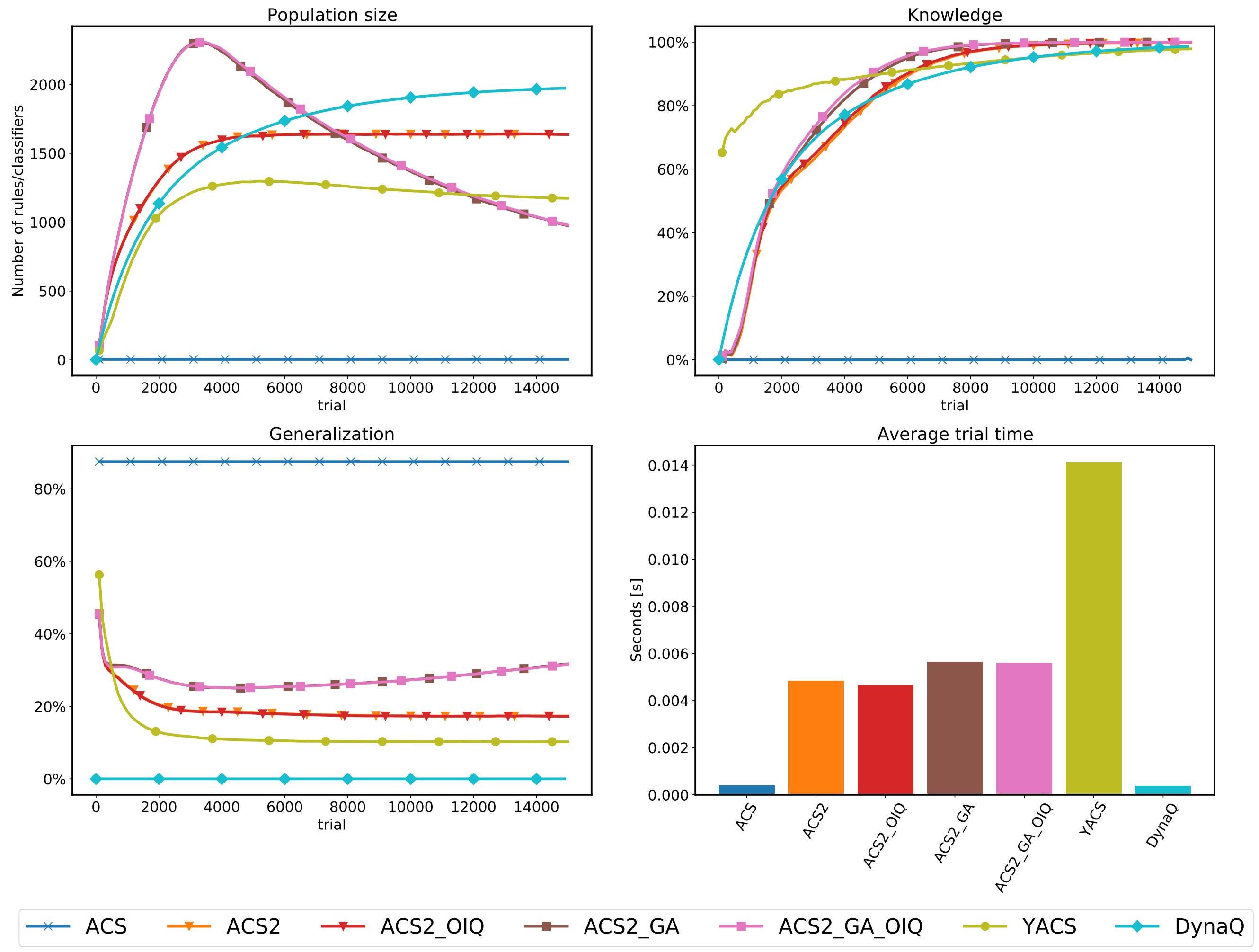


FIGURE 9. Latent learning comparison in the 3-bit rMPX environment discretized using 10 bins.

ACS2 obtained the best results with a genetic generalization mechanism. While an excess of classifiers has been generated in early trials, the population size is reduced over time while increasing generality. Only this algorithm among tested herein holds such capabilities.

D. SIMPLE MAZE

Lastly, to further highlight the potential of latent learning, the simple, discrete and deterministic environment was used. Figure 10 illustrates that the knowledge is reached by ACS2 (both variants) and Dyna-Q algorithms. Since no generalisation is possible, the optimal number of classifiers should converge to the number of Dyna-Q rules. Both ACS and YACS did not settle the optimal number of rules due to invalid generalisation. The ACS2 GA learned the environment in the early trials by generating many classifiers. Both its population and specificity decreases over time, resulting in the desired solution to the problem.

IV. BAYESIAN ANALYSIS

In order to statistically assess the significance and performance of obtained results, a Bayesian estimation [47], [48] (BEST) approach was used. We wanted to compare the performance of each agents' last trial using the Bayesian approach, which yields complete distributional information about the collected data, whereas the frequentist *Null Hypothesis Statistical Testing* (NHST) uses just a single point value. Having an explicit distribution of credible parameter values, inferences about null values can be made without even referring to *p values* as in NHST.

Each of the four experiments was executed 50 times and was independent. According to the Central Limit Theorem, this sample size of data is enough to consider approximating the normal distribution. [49], [50]. Every combination of collected *agent-metric* data samples x was modelled as Student-t distribution, which has the possibility of modelling a distribution with heavier tails. It was described with three parameters - μ_x (expected mean value), σ_x (standard deviation) and ν (degrees of freedom). The standard deviation σ parameter uniformly covers a vast possible parameter space. The degrees of freedom follows a shifted exponential distribution controlling the normality of the data. When $\nu > 30$, the Student-t distribution is close to a normal distribution. However, if ν is small, Student t-distributions have a heavy tail. Therefore, value of $\nu \sim \mathrm{Exp}(\frac{1}{29})$ allows the model to

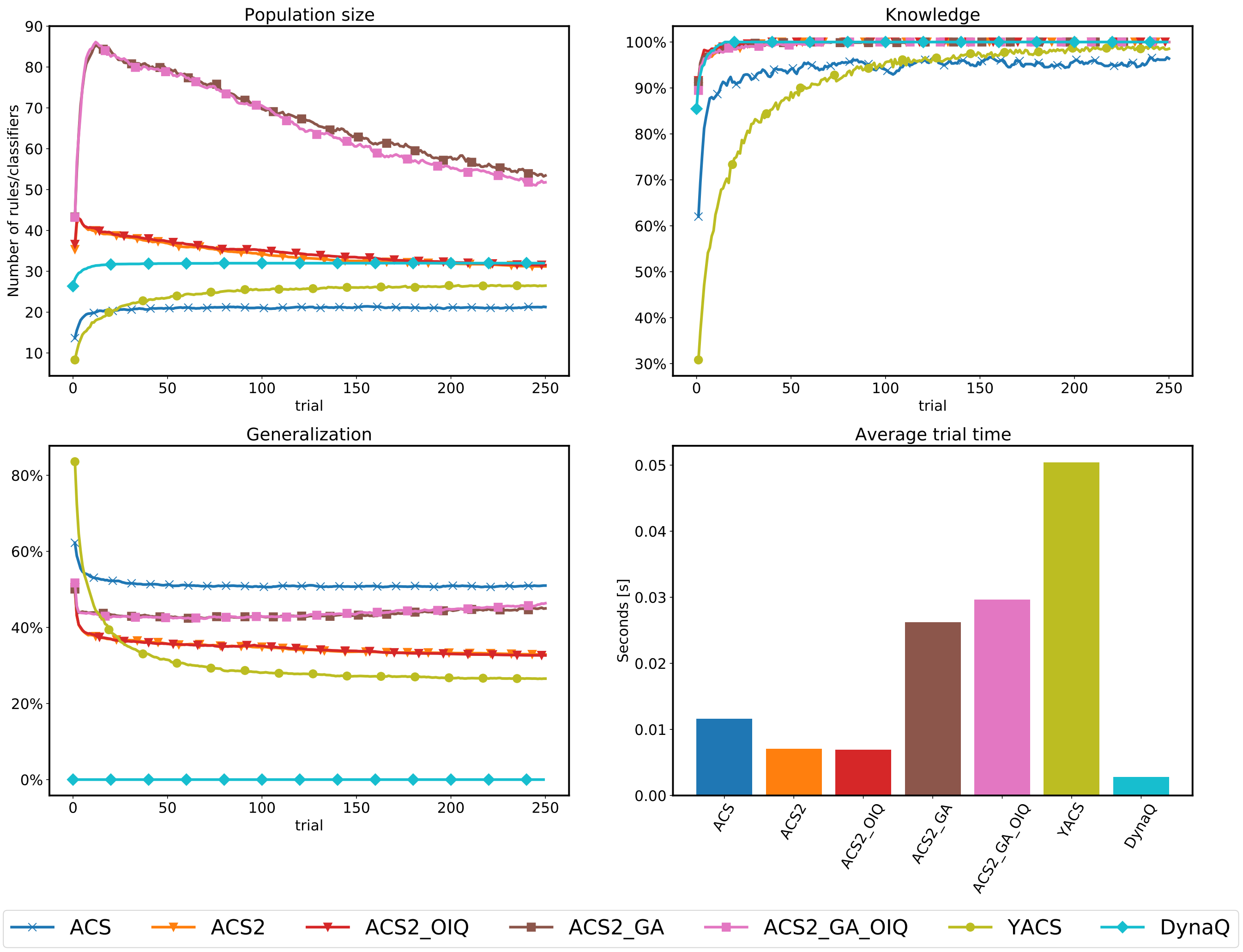


FIGURE 10. **Latent learning comparison in the Simple Maze environment.**

be more tolerant for potential outliers. Therefore, in order to apply the Monte Carlo sampling method, the following distributions were proposed [47]:

$$\mu \sim \mathrm{N}(\mu_x, \sigma_x^2) \tag{1}$$

$$\sigma \sim \mathrm{U}(\frac{1}{100}, 1000) \tag{2}$$

$$\nu \sim \mathrm{Exp}(\frac{1}{29}) \tag{3}$$

The random variables were drawn 100,000 times for each modelled data sample. Table 3 presents the means of the obtained distribution. Additionally, Figure 11 plots the relative differences across all methods. The following regularities can be observed:

1) in all cases the ACS agent produces the most general population (mostly not the most efficient one) and YACS agent is the slowest,
2) the performance of pairs ACS2 - ACS2 OIQ and ACS2 GA - ACS2 GA OIQ is very similar. The Genetic Algorithm component tends to increase the generalisation score, alongside population size and computation time.

V. DISCUSSION

Learning the environmental model latently from sensory input signals allows for a much richer source of information than just from a scalar reward signal. Having the internal representation of environmental dynamics facilitates optimizing learning speed (action planning, imaginary experiences) or faster adapting to new agent goals while having a world model remaining relatively intact.

This paper proves for the first time that the certain systems examined herein (ACS, ACS2, YACS) can successfully learn latently in real-valued environments by performing relevant, fully reproducible experiments when the input space is significantly increased. The modifications included discretizing the environmental perception to ensure compatibility with the inner agent's mechanism before applying its learning components. Minding the nature of ALCS, the usage of such nominal values for state representation would be the most straightforward approach. ALCS systems by design are not limited by *ternary alphabet*, therefore creating an arbitrary number of potential states is achievable. This possibility allows ALCS systems to be used in a much

TABLE 3. Means and standard deviations of Student-t distribution obtained by performing Bayesian estimation of metrics from the last trial runs.

	Knowledge	Generalisation	Population	Time
ACS	0.911 ± 0.009	0.051 ± 0.0	39.0 ± 0.0	0.006 ± 0.001
ACS2	1.000 ± 0.000	0.000 ± 0.0	38.0 ± 0.0	0.003 ± 0.001
ACS2 OIQ	1.000 ± 0.000	0.000 ± 0.0	38.0 ± 0.0	0.004 ± 0.001
ACS2 GA	1.000 ± 0.000	0.000 ± 0.0	38.0 ± 0.0	0.003 ± 0.001
ACS2 GA OIQ	1.000 ± 0.000	0.000 ± 0.0	38.0 ± 0.0	0.004 ± 0.001
YACS	1.000 ± 0.000	0.000 ± 0.0	38.0 ± 0.0	0.064 ± 0.005
Dyna-Q	1.000 ± 0.000	0.000 ± 0.0	38.0 ± 0.0	0.002 ± 0.001

(a) Corridor

	Knowledge	Generalisation	Population	Time
ACS	0.947 ± 0.000	0.526 ± 0.000	78.000 ± 0.000	0.081 ± 0.004
ACS2	0.951 ± 0.001	0.446 ± 0.005	87.889 ± 1.024	0.061 ± 0.002
ACS2 OIQ	0.971 ± 0.001	0.442 ± 0.006	89.103 ± 1.305	0.061 ± 0.002
ACS2 GA	0.951 ± 0.001	0.500 ± 0.000	78.000 ± 0.001	0.071 ± 0.002
ACS2 GA OIQ	0.971 ± 0.001	0.500 ± 0.001	78.000 ± 0.001	0.071 ± 0.002
YACS	0.944 ± 0.002	0.032 ± 0.005	804.550 ± 53.449	2.001 ± 0.134
Dyna-Q	0.727 ± 0.002	0.000 ± 0.000	1171.132 ± 3.131	0.037 ± 0.001

(b) Grid

	Knowledge	Generalisation	Population	Time
ACS	0.000 ± 0.000	0.875 ± 0.000	4.000 ± 0.000	0.000 ± 0.000
ACS2	0.998 ± 0.001	0.173 ± 0.001	1636.380 ± 6.589	0.004 ± 0.001
ACS2 OIQ	0.999 ± 0.001	0.172 ± 0.001	1634.535 ± 7.966	0.004 ± 0.001
ACS2 GA	1.000 ± 0.001	0.317 ± 0.001	973.439 ± 6.410	0.003 ± 0.001
ACS2 GA OIQ	1.000 ± 0.001	0.316 ± 0.002	979.342 ± 6.347	0.003 ± 0.001
YACS	0.979 ± 0.002	0.103 ± 0.005	1169.114 ± 9.550	0.014 ± 0.001
Dyna-Q	0.986 ± 0.001	0.000 ± 0.000	1972.311 ± 0.711	0.000 ± 0.000

(c) rMPX

	Knowledge	Generalisation	Population	Time
ACS	0.966 ± 0.007	0.510 ± 0.001	21.308 ± 0.137	0.005 ± 0.001
ACS2	1.000 ± 0.000	0.329 ± 0.003	30.875 ± 0.448	0.004 ± 0.001
ACS2 OIQ	1.000 ± 0.000	0.327 ± 0.003	31.409 ± 0.664	0.004 ± 0.001
ACS2 GA	1.000 ± 0.000	0.450 ± 0.004	53.349 ± 1.011	0.009 ± 0.002
ACS2 GA OIQ	1.000 ± 0.000	0.463 ± 0.004	51.717 ± 0.871	0.012 ± 0.002
YACS	1.000 ± 0.001	0.265 ± 0.005	26.473 ± 0.403	0.039 ± 0.004
Dyna-Q	1.000 ± 0.000	0.000 ± 0.000	32.000 ± 0.000	0.002 ± 0.001

(d) Simple Maze

broader range of possible environments without significant modifications.

The number of ways to discretize a continuous attribute is infinite. Kotsiantis in [51] surveys possible discretization techniques, but in this work, the preferred method is to divide the search space into *n* equally spaced intervals. This choice requires a careful selection of the number of buckets. Due to certain environmental regularities, an invalid setting might result in under-performing or creating an excessive model.

All examined ALCS created a more compact rules population than the straightforward Dyna-Q approach. The utilization of generalization capabilities is a trade-off between computation speed and readability. However, among all tested algorithms, Dyna-Q was the fastest and conceptually the easiest to understand.

Also, two families of ALCS were tested – the ACS alongside ACS2 and YACS. Due to experiments, ACS2 turned out to be the most mature and stable. The genetic generalization modification enables the population size to shrink, simplifying the internal model. However, the second modification - OIQ - improved the knowledge acquisition speed in the Corridor and Grid environments over the default ACS2.

YACS also showed unexpected behaviour when tested on environments other than discrete, multi-step mazes (initially suggested by Gérard). Due to its heuristics, certain problems were revealed like random population evolution (leading to different solutions in each trial) or non-optimal internal representation storage of all visited states. The latter is especially problematic in larger input-space environments where the slow performance will dampen the benefits of generalization capabilities.

To summarize, this work evaluated the capabilities of building internal environmental models in the context of learning classifier systems. ALCS was preferred over more popular XCS because their formalism naturally enables storing knowledge in a suitable form. Additionally, for the first time to our knowledge, ALCS were evaluated with real-valued environments that posed a new challenge. The experiments performed are fully reproducible using industry standards and encourage others to continue the research seamlessly.

VI. FUTURE ENHANCEMENTS

For future work, we recommend the following actions:

1) investigate the topic of rule-compaction dedicated to the C-A-E classifier structure. Being able to represent intervals in a single classifier would further reduce the population size [23], [24],

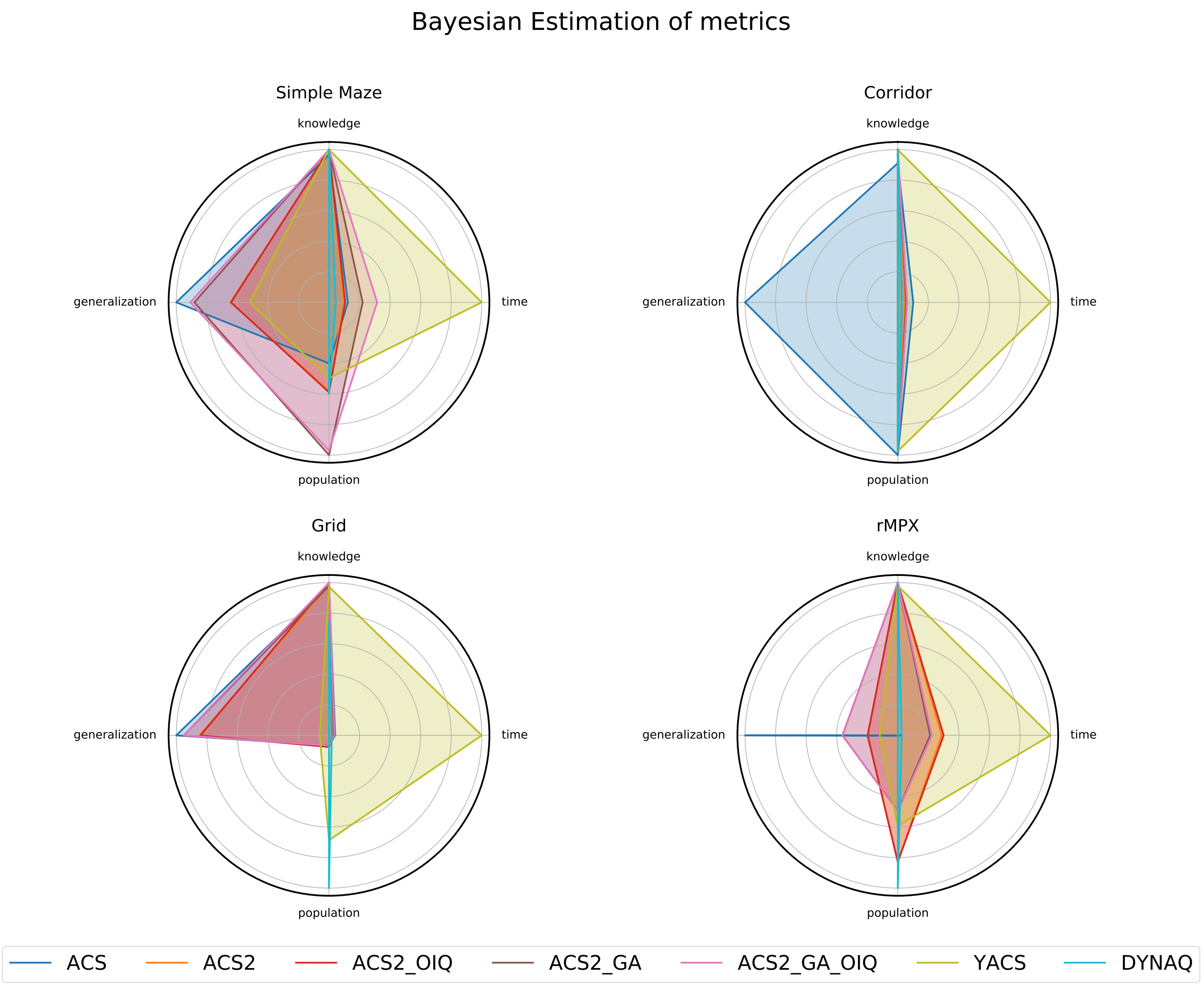


FIGURE 11. Means of Student-t distribution obtained by performing Bayesian estimation of metrics from the last trial runs. Values are normalized to the maximum value for each metric. There are N different classes of agents - ACS, ACS2 (with GA modification), YACS and Dyna-Q. The OIQ modification is highly correlated with the ACS2 variant used.

2) investigate the impact of utilizing internal knowledge in demanding environments (like real-valued ones) using mechanisms like action planning [35],
3) dimensionality reduction using an ensemble of LCS with deep-learning methods [26]–[28],
4) investigate the concept of performing *active latent learning* by enabling *curiosity* [52], [53]. The traditional environments used for evaluating LCS would need to be adjusted by enabling *cognitive maps* formation.

REFERENCES

[1] S. R. Sutton, *Introduction to Reinforcement Learning*, vol. 135. Cambridge, MA, USA: MIT Press, 1998.
[2] C. E. Tolman, *Purposive Behavior in Animals and Men*. Berkeley, CA, USA: Univ of California Press, 1932.
[3] P. J. Seward, "An experimental analysis of latent learning," *J. Exp. Psychol.*, vol. 39, no. 2, p. 177, 1949.
[4] H. J. Holland, "Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems," *Mach. Learn., Artif. Intell. Approach*, vol. 2, pp. 593–623, Feb. 1986.
[5] L. Pier Lanzi, "Learning classifier systems: A reinforcement learning perspective," in *Foundation Learning Classifier System*. Springer, 2005, pp. 267–284.
[6] P. L. Lanzi, "Learning classifier systems: Then and now," *Evol. Intell.*, vol. 1, no. 1, pp. 63–82, Mar. 2008.
[7] J. H. Holmes, D. R. Durbin, and F. K. Winston, "The learning classifier system: An evolutionary computation approach to knowledge discovery in epidemiologic surveillance," *Artif. Intell. Med.*, vol. 19, no. 1, pp. 53–74, May 2000.
[8] I. H. Witten, "Data mining: Practical machine learning tools and techniques with Java implementations," *Acm SIGMOD Rec.*, vol. 31, no. 1, pp. 76–77, 2016.
[9] E. Bernadó, X. Llora, and M. Josep Garrell, "XCS and GALE: A comparative study of two learning classifier systems on data mining," in *Proc. Int. Workshop Learn. Classifier Syst.* Springer, 2001, pp. 115–132.
[10] J. Bacardit and V. M. Butz, "Data mining in learning classifier systems: Comparing XCS with GAssist," in *Learning Classifier System*. Springer, 2003, pp. 282–290.

[11] A. H. Abbass, J. Bacardit, V. M. Butz, and X. Llora, "Online adaptation in learning classifier systems: Stream data mining," Illinois Genetic Algorithms Lab., Urbana, IL, USA, Tech. Rep. 2004031, 2004.

[12] H. H. Dam, C. Lokan, and A. H. Abbass, "Evolutionary online data mining: An investigation in a dynamic environment," in *Evolutionary Computation in Dynamic and Uncertain Environments*. Springer, 2007, pp. 153–178.

[13] R. Smith, A. El-Fallah, B. Ravichandran, R. Mehra, and B. Dike, *The Fighter Aircraft LCS: A Real-World, Machine Innovation Application*. Springer, 2004, pp. 113–142.

[14] W. Stolzmann and V. Martin Butz, "Latent learning and action planning in robots with anticipatory classifier systems," in *Proc. Int. Workshop Learn. Classifier Syst.* Springer, 1999, pp. 301–317.

[15] S. W. Wilson, "Classifier fitness based on accuracy," *Evol. Comput.*, vol. 3, no. 2, pp. 149–175, 1995.

[16] W. Stewart Wilson, "Get real! XCS with continuous-valued inputs," in *Proc. Int. Workshop Learn. Classifier Syst.* Springer, 1999, pp. 209–219.

[17] C. Stone and L. Bull, "For real! XCS with continuous-valued inputs," *Evol. Comput.*, vol. 11, no. 3, pp. 299–336, Sep. 2003.

[18] H. H. Dam, H. A. Abbass, and C. Lokan, "Be real! XCS with continuous-valued inputs," in *Proc. Workshops Genetic Evol. Comput.* 2005, pp. 85–87.

[19] L. Cielecki and O. Unold, "3D function approximation with rGCS classifier system," in *Proc. 8th Int. Conf. Intell. Syst. Design Appl.*, Nov. 2008, p. 974.

[20] S. W. Wilson, "Classifiers that approximate functions," *Natural Comput.*, vol. 1, no. 2, pp. 211–234, 2002.

[21] N. Kozlowski and O. Unold, "Preliminary tests of a real-valued anticipatory classifier system," in *Proc. Genetic Evol. Comput. Conf. Companion*, Jul. 2019, pp. 1289–1294.

[22] N. Kozlowski and O. Unold, "Investigating exploration techniques for ACS in discretized real-valued environments," in *Proc. Genetic Evol. Comput. Conf. Companion*, Jul. 2020, pp. 1765–1773.

[23] Y. Liu, W. N. Browne, and B. Xue, "A comparison of learning classifier Systems' rule compaction algorithms for knowledge visualization," *ACM Trans. Evol. Learn. Optim.*, vol. 1, no. 3, pp. 1–38, Sep. 2021.

[24] Y. Liu, W. N. Browne, and B. Xue, "Visualizations for rule-based machine learning," *Natural Comput.*, vol. 4, pp. 1–22, Jan. 2021.

[25] M. Irfan, Z. Jiangbin, M. Iqbal, Z. Masood, M. H. Arif, and S. R. U. Hassan, "Brain inspired lifelong learning model based on neural based learning classifier system for underwater data classification," *Expert Syst. Appl.*, vol. 186, Dec. 2021, Art. no. 115798.

[26] M. Irfan, Z. Jiangbin, M. Iqbal, Z. Masood, and M. H. Arif, "Knowledge extraction and retention based continual learning by using convolutional autoencoder-based learning classifier system," *Inf. Sci.*, vol. 591, pp. 287–305, Apr. 2022.

[27] M. Irfan, Z. Jiangbin, M. Iqbal, and M. H. Arif, "Enhancing learning classifier systems through convolutional autoencoder to classify underwater images," *Soft Comput.*, vol. 25, no. 15, pp. 10423–10440, Aug. 2021.

[28] R. J. Preen, S. W. Wilson, and L. Bull, "Autoencoding with a classifier system," *IEEE Trans. Evol. Comput.*, vol. 25, no. 6, pp. 1079–1090, Dec. 2021.

[29] A. Siddique, W. N. Browne, and G. M. Grimshaw, "Frames-of-reference-based learning: Overcoming perceptual aliasing in multistep decision-making tasks," *IEEE Trans. Evol. Comput.*, vol. 26, no. 1, pp. 174–187, Feb. 2022.

[30] J. Hoffmann, *Vorhersage Erkenntnis*. Göttingen, Germany: Hogrefe, 1993.

[31] J. Hoffmann and A. Sebald, "Lernmechanismen zum Erwerb verhaltenssteuernden Wissens," *Psychol. Rundschau*, Dec. 2000.

[32] W. Stolzmann, *Antizipative Classifier System*. Osnabrueck, Germany: Shaker Verlag 1997.

[33] V. Martin Butz, "*Anticipatory Learning Classifier System*, vol. 4. Springer, 2002.

[34] V. Martin Butz and W. Stolzmann, "An algorithmic description of ACS2," in *Proc. Int. Workshop Learn. Classifier Syst.* Springer, 2001, pp. 211–229.

[35] O. Unold, E. Rogula, and N. Kozlowski, "Introducing action planning to the anticipatory classifier system ACS2," in *Proc. Int. Conf. Comput. Recognit. Syst.* Springer, 2019, pp. 264–275.

[36] R. Orhand, A. Jeannin-Girardon, P. Parrend, and P. Collet, "PEPACS: Integrating probability-enhanced predictions to ACS2," in *Proc. Genetic Evol. Comput. Conf. Companion*, Jul. 2020, pp. 1774–1781.

[37] R. Orhand, A. Jeannin-Girardon, P. Parrend, and P. Collet, "BACS: A thorough study of using behavioral sequences in ACS2," in *Proc. Int. Conf. Parallel Problem Solving From Nature*. Springer, 2020, pp. 524–538.

[38] P. Gérard, W. Stolzmann, and O. Sigaud, "YACS: A new learning classifier system using anticipation," *Soft Comput. Fusion Found., Methodol. Appl.*, vol. 6, nos. 3–4, pp. 216–228, Jun. 2002.

[39] S. Richard Sutton, "Reinforcement learning architectures for animats," in *Proc. 1st Int. Conf. Simul. Adapt. Behav.*, 1991, pp. 288–296.

[40] P. Gérard, J.-A. Meyer, and O. Sigaud, "Combining latent learning with dynamic programming in the modular anticipatory classifier system," *Eur. J. Oper. Res.*, vol. 160, no. 3, pp. 614–637, Feb. 2005.

[41] V. Zhanna Zatuchna, "AgentP model: Learning classifier system with associative perception," in *Proc. Int. Conf. Parallel Problem Solving Nature*. Springer, 2004, pp. 1172–1181.

[42] T. O'Hara and L. Bull, "Building anticipations in an accuracy-based learning classifier system by use of an artificial neural network," in *Proc. Congr. Evol. Comput.*, vol. 3, Sep. 2005, pp. 2046–2052.

[43] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI gym," 2016, *arXiv:1606.01540*.

[44] P. L. Lanzi, D. Loiacono, S. W. Wilson, and D. E. Goldberg, "XCS with computed prediction in multistep environments," in *Proc. Conf. Genetic Evol. Comput.*, 2005, pp. 1859–1866.

[45] P. Gerard and O. Sigaud, "YACS: Combining dynamic programming with generalization in classifier systems," in *Proc. Int. Workshop Learn. Classifier Syst.* Springer, 2000, pp. 52–69.

[46] N. Kozlowski and O. Unold, "Integrating anticipatory classifier systems with OpenAI gym," in *Proc. Genetic Evol. Comput. Conf. Companion*, Jul. 2018, pp. 1410–1417.

[47] K. John Kruschke, "Bayesian estimation supersedes the T test," *J. Exp. Psychol., Gen.*, vol. 142, no. 2, p. 573, 2013.

[48] A. Benavoli, G. Corani, J. Demšar, and M. Zaffalon, "Time for a change: A tutorial for comparing multiple classifiers through Bayesian analysis," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 2653–2688, Jan. 2017.

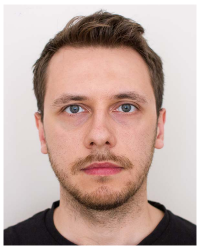
[49] S. G. Kwak and J. H. Kim, "Central limit theorem: The cornerstone of modern statistics," *Korean J. Anesthesiol.*, vol. 70, no. 2, p. 144, 2017.

[50] M. R. Islam, "Sample size and its role in central limit theorem (CLT)," *Comput. Appl. Math. J.*, vol. 4, no. 1, pp. 1–7, 2018.

[51] S. Kotsiantis and D. Kanellopoulos, "Discretization techniques: A recent survey," *Int. Trans. Comput. Sci. Eng.*, vol. 32, no. 1, pp. 47–58, 2006.

[52] M. Z. Wang and B. Y. Hayden, "Latent learning, cognitive maps, and curiosity," *Current Opinion Behav. Sci.*, vol. 38, pp. 1–7, Apr. 2021.

[53] R. Golman and G. Loewenstein, "Information gaps: A theory of preferences regarding the presence and absence of information," *Decision*, vol. 5, no. 3, p. 143, 2018.



NORBERT KOZLOWSKI was born in Poland, in 1990. He received the B.S. degree from the Department of Electronics, Wrocław University of Science and Technology, in 2014, and the M.S. degree in advanced informatics and control, in 2015. His research interest includes anticipatory learning classifier systems with a particular interest in the real-valued input signals.

OLGIERD UNOLD received the M.Sc. degree in automation systems, in 1989, the M.Sc. degree in information science, in 1991, and the Ph.D. and D.Sc. degrees in computer science, in 1994 and 2011, respectively. He is currently a Full Professor with the Department of Computer Engineering, Wrocław University of Science and Technology. His research interest includes adaptive machine learning methods.