WROCLAW UNIVERSITY OF SCIENCE AND TECHNOLOGY

Real-valued Anticipatory Classifier System

by

Norbert Kozłowski

A thesis submitted in partial fulfillment for the
degree of Doctor of Philosophy

in the
Faculty of Computer Science and Telecommunications
Department of Computer Engineering

April 10, 2022

Acknowledgements

First and foremost, I would like to express my most sincere gratitude to my supervisor, Prof. Olgierd Unold. It was a pleasure to pursue such a challenging goal while maintaining huge motivation.

Thanks to my wife Andżelika for her patience and generosity in giving me space for doing this.

I would also express gratitude to my parents for sparking an idea to pursue such scientific research. That would have never happened without it.

Last but not least, thanks to myself for the hard work and for not giving up in tough times.

Abbreviations

ACS	**A**nticipatory **C**lassifier **S**ystem
AACS	**A**veraged **A**nticipatory **C**lassifier **S**ystem
AD	**A**ction **D**elay (exploration technique)
ALCS	**A**nticipatory **L**earning **C**lassifier **S**ystem
ALP	**A**nticipatory **L**earning **P**rocess
BBA	**B**ucket **B**rigade **A**lgorithm
BEST	**B**ayesian **E**stimation **S**upersedes the **t**-test
CSR	**C**enter **S**pread **R**epresentation
E / E	**E**xplore / **E**xploit
EG	**E**psilon **G**reedy (exploration technique)
FSW	**F**inite **S**tate **W**orld
GA	**G**enetic **A**lgorithm
KA	**K**nowledge **A**rray (exploration technique)
KPI	**K**ey **P**erformance **I**ndicator
LCS	**L**earning **C**lassifier **S**ystem
MACS	**M**odular **A**nticipatory **C**lassifier **S**ystem
MDP	**M**arkov **D**ecision **P**roblem
ML	**M**achine **L**earning
MLP	**M**ulti **L**ayer **P**erceptron
MPX	**M**ulti**p**lex**e**r

`MCMC`	**M**arkov **C**hain **M**onte **C**arlo
`NHST`	**N**ull **H**ypothesis **S**tatistical **T**esting
`OBR`	**O**rdered **B**ounded **R**epresentation
`OIQ`	**O**ptimistic **I**nitial **Q**uality (exploration technique)
`rACS`	real **A**nticipatory **C**lassifier **S**ystem [126]
`rvACS`	real-valued **A**nticipatory **C**lassifier **S**ystem
`RL`	**R**einforcement **L**earning
`XAI`	e**X**plainable **A**rtificial **I**ntelligence
`XCS`	e**X**tended **C**lassifier **S**ystem
`UBR`	**U**nordered **B**ounded **R**epresentation
`YACS`	**Y**et **A**nother **C**lassifier **S**ystem
`ZCS`	**Z**eroth Level **C**lassifier **S**ystem

Symbols

β	learning rate
b_r	bid ratio
χ	crossover probability
γ	discount factor
ϵ	exploration probability
ϵ_{cover}	covering noise
$\epsilon_{mutation}$	mutation noise
μ	mutation rate
t_{alp}	time when the classifier underwent the last ALP update
t_{ga}	time when the classifier was part of $[A]$ in GA application
θ_i	inadequacy threshold
θ_{ga}	GA application threshold
θ_r	reliability threshold
σ	perception of environment's state
ρ	average reward
ζ	learning rate for the average reward

Abstract

The thesis focuses on the use of ACSs (*Anticipatory Classifier Systems*) with the input deliberately expressed as real-valued number. Other classifier system families thoroughly investigated such a research area, but no significant contribution was made to systems with explicit prediction capabilities. The aim was to present possible solutions to such integration using several benchmarking problems. Based on this assumption, the following thesis was formulated:

> *Anticipatory Classifiers Systems can build the correct internal model of the environment using the real-valued input.*

The thesis was substantiated by achieving the following goals:

Goal 1 - Propose modifications towards ACS system capable of handling real-valued input

This intention was achieved and validated using two independent approaches:

1. changing the rule's attribute representations to incorporate *interval predicates*, resulting in a proprietary system variation named rvACS (*real-valued Anticipatory Classifier System*),
2. implementing the prominent ACS algorithms with the intention of input discretization.

The first approach was based on the actual advancements made for the other systems. The rvACS managed to show promising results. Nevertheless, because of the much richer rule complexity and the cooperation of components, the obtained conclusion is that the nature of rvACS is not aligned with the overall ACS virtues.

The second approach maintains the characteristics and original intentions of three investigated algorithms by affecting only the agent-environment interface layer for discretizing

the input signal. This approach was considered superior to interval formation by obtaining promising results and better rule interpretability.

Goal 2 - Propose relevant benchmarking problems and metrics for evaluating ACS performance

The nature of the performance was carefully reviewed in six problems using the real-valued output as a description of an actual state. Most of them have been used as a toy-problems in LCS (*Learning Classifier Systems*) research, providing different problems of different natures, like:

- single and multiple steps,
- extensive mutual feature interaction,
- vast input space,
- the need for long-action chain building.

Moreover, a famous RL (Reinforcement Learning) *Cart Pole* benchmarking problem was investigated by the LCS for the first time, to our knowledge, obtaining encouraging results with compact knowledge representation.

To highlight performance, five key metrics were selected that investigate the state of the systems from multiple angles. Aspects like the quality of evolving solutions, size, and effective application are emphasized throughout the research.

Goal 3 - Propose relevant improvements towards neoteric changes

Preliminary tests revealed two possible limitations of the system that were further studied.

First, any form of real-valued representation increased the problem's search space, impeding the agent's learning speed. Four techniques probing for the most promising action selection were inspected, especially regarding knowledge acquisition rate. The novel approach, named *Optimistic Initial Quality* was also proposed herein.

The other limitation relates to multistep problems, where certain input discretization might demand the agent to perform a notably larger number of actions to receive the feedback signal. An approach replacing the credit assignment with the undiscounted incentive distribution version resulted in a system named AACS2 (*Average Anticipatory Classifier System 2*). The method showed performance improvements for specific problems.

Goal 4 - Conduct an experimental evaluation of intended adjustments

All the experiments were tried to illustrate the evolution of selected metrics throughout the agent's learning of selected problems.

Moreover, the Bayesian estimation approach emphasized the differences between investigated cases, drawing non-biased conclusions. Often, selected environments were scaled up, where agents' behaviour was challenged with increased complexity.

Goal 5 - Developing an open-sourced Python Machine Learning framework for evaluating various LCS algorithms

Significant effort was put to implement the most famous classifier systems algorithms (i.e. ACS, ACS2, YACS, MACS, X-NCS) using the tools used nowadays by data scientist communities. The original performance reported by authors is preserved while adding full support for the real-valued representation of perception. By integrating with the industry standards, all evaluated models share the same code base and operate with a commonly agreed interface enabling effortless benchmarking with other state-of-the-art systems.

Moreover, a conscious endeavour was put to simplify the usage of the developed framework to spark other researchers' interest.

Keywords

learning classifier systems; anticipatory classifier systems; reinforcement learning; openai gym, real-valued input

Streszczenie

Rozprawa doktorska koncentruje się na wykorzystaniu *antycypacyjnych uczących się systemów klasyfikujących* ACS (ang. *Anticipatory Classifier Systems*) w problemach, w których stan przedstawiany jest z wykorzystaniem liczb rzeczywistych. Wiele podobnych badań zostało dotychczas wykonanych dla innych uczących się systemów klasyfikujących LCS (ang. *Learning Classifier Systems*), jednak żaden z nich nie posiadał możliwości budowania wewnętrznej reprezentacji środowiska z użyciem mechanizmu antycypacji. Celem pracy jest przedstawienie możliwych realizacji powyższej integracji z użyciem wybranych testowych problemów. W oparciu o te założenia, sformułowana została hipoteza, zakładająca, że

> *Antycypacyjne uczące się systemy klasyfikujące są w stanie poprawnie zbudować wewnętrzny model rzeczywistoliczbowego środowiska.*

Hipoteza została uprawdopodobniona poprzez osiągnięcie poniższych celów:

Cel 1 - Zaproponowanie modyfikacji wybranych systemów ACS umożliwiającą obsługę danych wejściowych o wartościach rzeczywistych

Cel został osiągnięty i zweryfikowany na dwa różne sposoby:

1. użycie dozwolonych *przedziałów wartości* jako podstawowej jednostki służącej do reprezentacji danych - zapropowanie autorskiej wersji systemu rvACS (ang. *real-valued Anticipatory Classifier System)*,
2. dyskretyzację danych wejściowych - zaproponowanie modyfikacji wybranych algorytmów ACS.

Pierwszy sposób inspiruje się dotychczasowymi osiągnięciami uzyskanymi dla innych systemów uczących. Nowy algorytm rvACS osiąga zadowalające wyniki, jednak poziom złożoności utworzonego rozwiązania, jak i zawiłość interakcji jego wewnętrznych komponentów jest zbyt wysoki.

Drugi sposób pozwala na uniknięcie znaczących modyfikacji w sposobie działania trzech testowanych systemów. Transformacja sygnału wyjściowego środowiska zanim zostanie on przetworzony przez algorytmy uczące, umożliwia zachowanie ich charakterystycznych cech. Dodatkowo, otrzymane rezultaty badań są bardziej obiecujące niż w przypadku zastosowania reprezentacji z wykorzystaniem *przedziałów wartości*.

Cel 2 - Zaproponowanie środowisk testowych i metryk do oszacowania wydajności systemów ACS

Charakterystyka działania algorytmów antycypujących została poddana szczegółowej analizie wykorzystując sześć testowych problemów opisujących aktualny stan za pomocą wartości rzeczywistoliczbowych. Część z nich była już wykorzystywana wcześniej w literaturze podczas badań uczących się systemów klasyfikujących.

Każde z badanych środowisk wykazuje charakterystyczne własności, np:

- problem uczenia nadzorowanego (jednokrokowy) lub uczenia ze wzmocnieniem (wielokrokowy),
- wzajemna zależność poszczególnych atrybutów percepcji,
- rozmiar przestrzeni dostępnych stanów,
- wymagana liczba akcji w celu osiągnięcia nagrody.

Dodatkowo, po raz pierwszy w historii znany problem uczenia ze wzmocnieniem polegający na utrzymywaniu równowagi pręta umieszczonego na wózku został przetestowany przez systemy uczące. Otrzymane wyniki są szczególnie interesujące pod względem wydajności i minimalistycznej postaci otrzymanego rozwiązania.

Działania poszczególnych algorytmów zostały oceniane przez pięć metryk badających różne aspekty działania. Uwagę zwrócono na jakość generowanego rozwiązania, jego rozmiar oraz praktyczne wykorzystanie wiedzy w badanym środowisku.

Cel 3 - Zaproponowanie optymalizacji działania systemów ACS w kontekście przetwarzania wejścia rzeczywistoliczbowego

Wstępne badania wskazały na dwa obszary mające potencjał optymalizacji działania systemów antycypacyjnych w przypadku interakcji z rzeczywistoliczbowymi środowiskami.

Pierwszy, zauważa, że zastosowana reprezentacja wejścia znacząco zwiększa dostępną przestrzeń stanów, co ma znaczący wpływ na działanie każdego badanego algorytmu. Przetestowane zostały cztery strategie, skupiające się na inteligentym przeszukiwaniu

przestrzeni rozwiązań optymalizując tworzenie wewnętrznego modelu środowiska. Zaproponowane zostało także autorskie rozwiązanie, polegające na optymistycznym generowaniu nowych reguł.

Drugie ograniczenie dotyczy środowisk wielokrokowych, wymagających wykonanie wielu akcji w celu osiągnięcia finalnego stanu. Zaproponowano wymianę komponentu odpowiedzialnego za dystrybucję sygnału nagrody na uśrednioną wersję. Przedstawiono autorską wersję systemu - AACS2 (ang. *Average Anticipatory Classifier System 2)*, wykazującą wzrost wydajności w testowanych problemach.

Cel 4 - Eksperymentalna analiza przeprowadzonych badań

W celu zrozumiania natury przeprowadzanych eksperymentów, w każdym przypadku uwzględniany został aspekty czasowy ilustrujący sposób ewolucji wybranych metryk w całym procesie uczenia.

Dodatkowo, porównania pomiędzy poszczególnymi algorytmami bądź wersjami środowisk zostały wykonane z wykorzystaniem technik modelowania probabilistycznego.

Cel 5 - Stworzenie ogólnodostępnej biblioteki programistycznej

Efektem tezy jest ogólnodostępna biblioteka programistyczna w języku Python implementująca szereg systemów uczących (np. ACS, ACS2, YACS, MACS, X-NCS). Znaczącym osiągnięciem jest pełne odtworzenie dotychczasowych wyników badań wraz z dodaniem możliwości obsługi liczb rzeczywistych. Zaprojektowana ona została zgodnie z aktualnymi trendami, umożliwiając szybką integrację i porównanie się z innymi rodzinami algorytmów.

Słowa kluczowe

uczące się systemy klasyfikujące; antycypacyjne uczące się systemy klasyfikujące; uczenie ze wzmocnieniem; openai gym, wejście rzeczywistoliczbowe

Contents

Chapter 1

Introduction

With the advancement of understanding our world, sometimes the simple structures used for its description like linear models or decision trees become staggeringly insufficient. The innumerable systems that our world encompasses are composed of interconnected parts, exhibiting imperceptible properties that interact with each other non-linearly. By having the capacity to change over time and learn from experience, such systems become complex and adaptive.

More than three decades ago, Holland proposed a conceptual rule-based system[49] comprising a set of "IF condition THEN action" rules covering different situations - calling it the cognitive systems[54]. From the holistic perspective, it can be viewed as a group of collaborating "agents" represented by a collection of simple rules. The rules are formed when interacting with the external "environment" and might take, for example, the following forms:

- IF *no cars on the street* THEN *walk forward*,
- IF *stock share price is dropping for 3 days* THEN *buy*,
- IF *colour of the mushroom is red* THEN *do not pick*.

The general idea is that seeking a single, omnibus and complex rule is less desirable than evolving a population of them to model the environment behaviour collectively. Such an idea gave rise to the concept of LCSs [48], introducing a new abstract term - a *classifier*, which encompasses a rule itself width additional statistics (such as its quality). Despite their somewhat misleading name, LCSs are not only systems suitable for classification problems but maybe instead viewed as a very general, distributed optimization technique.

They fit into a trend of the XAI (Explainable Artificial Intelligence) [41] and can be used in a variety of fields [13] like data mining [1, 3, 7, 31, 143] (discovering patterns in data), supervised or RL tasks. Examples of such problems would include fighter aircraft manoeuvres [110], medical domains [56], robotic control [26, 58, 88, 122], game strategy [124], modelling complex time-dependent systems (e.g., stock market) [11, 79, 123] or design optimization (e.g., engineering applications) [96].

The desired outcome after running an LCS is a set of interpretable classifiers being able to model an intelligent decision-maker collectively. Two biological metaphors - *evolution* and *learning*[30] are employed to accomplish this intention. A pair of internal mechanisms - the *genetic algorithm* and the *learning mechanism* embody them respectively by actively interacting with the outside environment, which in this work is considered as an independent source of data for an LCS algorithm.

At this moment, plentiful different LCS variations exist [129], but according to Holmes [57], the following four major components are considered universal:

1. a finite population of classifiers representing current knowledge of the system,
2. a performance component regulating the interaction between the environment and classifier population,
3. a reinforcement (or credit assignment) component distributing the reward from the environment to particular classifiers,
4. a discovery component using various operators to discover better rules and improve existing ones.

However, this work is directed on a specific niche of ALCSs (Anticipatory Learning Classifier Systems), capable of learning a generalized predictive model [121] of the environment online. In contrast to the traditional IF-THEN rule structure, they also have a state prediction or *anticipatory* part that predicts the environmental changes caused when executing the specified action in the specified context. Forming such an internal structure might facilitate the agent's thought processes, such as planning or reflection, without any immediate behaviour. Thus, beliefs about the future control the decision-making process and behaviour in the present. This architecture also allows disambiguating perceptual aliasing problems, where the same observation is obtained in distinct states requiring different actions.

The capabilities of ALCS were exhaustively examined in environments with discrete and manageable *observation spaces* - such as navigating an agent in a maze or correctly

determining an answer in a binary multiplexer problem [18]. No comparative research focused on the problems where the observation space contains the attributes expressed as real-valued numbers - for example, a car's speed ranging from 0 to 200 km/h or a particular temperature range have not been performed yet.

This thesis demonstrates that certain families of ALCS can be adjusted to new problem domains. Despite the complicated and interconnected components hurdle, facilitating certain modifications allows them to be taking advantage of benefits associated with internal knowledge representation and prediction mechanisms. In his anticipatory systems book[102], Rosen goes one step further by putting the idea of anticipations in a mathematical framework and later identifying them as the essence of life[101].

> In anticipatory systems, as I have defined them, the present change of state depends on a future state, which is predicted from the present circumstances on the basis of some model. Anticipatory, model-based behavior provided one basis for what I later called complexity, and which I defined in "Life Itself" on the basis of non-computability or nonformalizability of models of a system of this kind.
>
> *Robert Rosen in[101]*

While this work does not pursue such a big claim, it proves that a greater realm of possible problems can be expressed by ALCS frameworks, therefore benefiting from more comprehensive, intrinsic representation.

1.1 Motivation and challenges

ALCS were designed and thoroughly tested in problems where the concept or a function sought is essentially "logical" - meaning that it can be expressed by a combination of logical operators applied to attribute values[138]. However, in many problems, the solution cannot be conveyed solely by `and`, `or`, `not` operators because its discrimination surface is oblique - neither parallel nor perpendicular to the attribute axes. If the problem's discrimination surface is oblique, the classifier rule can capture parts of space within

hyper-rectangles, which can only approximate the function shape using multiple distinct classifiers.

The most popular family of LCS and the ALCS predecessor - XCS (*Extended Classifier System*) [134], was comprehensively evaluated for problems with non-linear decision surfaces and real-valued input. In 1999 Lanzi took an approach to use *S-classifiers*[76], where the classifier's condition parts are Lisp S-expressions and can be based on arithmetic function primitives. In 2000 Wilson focused on continuous value inputs introducing XCSR implementation[136]. The condition attribute was an interval represented by a pair of numbers - the centre and the spread. He examined if the optimal decisive thresholds are found automatically in a modified Multiplexer benchmark problem. In the same year, he introduced an XCSI system[138], forming bounded intervals for integer inputs. One year later, in 2001, another implementation named XCSF[140] allowing piecewise-linear function approximation was proposed. For a function $y = f(x)$, the system assumed that x is the input, y the payoff, and after a sufficient amount of sampling, the input space XCSF should converge to a population of rules that, over their respective input ranges, predict payoff well. In 2003 Stone and Bull scrupulously addressed the limitations of the XCSR system[117] and introduced two new interval representations alongside a more versatile benchmark problem. The OBR (ordered-bounded) and UBR (unordered-bounded) representations describe the interval using left x_1 and right x_2 bounds but in OBR $x_1 < x_2$. By neglecting the explicit ordering, the system evolves better classifiers using the UBR representation. In 2005 Dam and Abbass also proposed a Min-Percentage approach[29], attempting to overcome some UBR drawbacks. Recent XCSR advancements aims to generate interpretable rules for high-dimensional data by employing encoder/decoder models for dimensionality reduction [86, 119]. Proposed modifications, by using deep-learning techniques, managed to obtain almost 99% accuracy on the MNIST database [33], where each data instance is represented by a vector of 784 values ranging $[0, 1]$.

In 2007, Cielecki [27, 125] proposed a real-valued modification towards GCS (*Grammar-based Classifier Systems*) in which the knowledge is represented by a Context-Free-Grammar in Chomsky normal form. The system evolves a population of rules where each one represents a single grammar. Preliminary tests were outperformed Stone's XCSR system by creating a perfect set of easy to interpret classifiers.

For the ALCS, only the preliminary studies transforming the real-valued input using discretization methods were performed by Unold in 2016 [126]. This work tries to bridge the gap between advancements from various LCS implementations mentioned earlier and the anticipatory systems by introducing features and modifications for handling input signals, especially discretization or interval encoding representations. Such changes would greatly increase the potential applicability of the algorithm. However, due to the

more complex rule structure of ALCS (additional prediction part) and the presence of other discovery components, certain limitations were observed and addressed herein. The biased exploration enhancement investigates the impact of optimizing the action selection method using specific strategies to stimulate knowledge formation [68]. On the other side, there is also a problem with long-action chains and diminishing reward that might occur when using fine-grained nominal representation for continuous data [69] - in this case, incorporating an average reward criterion resulted in a more distinct payoff landscape and faster knowledge acquisition.

1.2 Research hypothesis, its aims and goals

The research hypothesis is formulated as follows:

> *Anticipatory Classifiers Systems can build the correct internal model of the environment using the real-valued input.*

Aims and goals
In order to validate the expressed hypothesis, the following goals have been formulated:

1. Propose modifications towards the ACS system capable of handling real-valued input.
2. Propose relevant benchmarking problems and metrics for evaluating ACS performance.
3. Propose relevant improvements towards neoteric changes.
4. Conduct an empirical evaluation of intended adjustments.
5. Develop an open-sourced Python Machine Learning framework for evaluating various LCS algorithms.

1.3 Thesis structure

Chapter 2 introduces selected topics related to Learning Classifier Systems and handling of the real-valued input. The evolution of a system capable of predicting the consequences of executed actions is emphasized. Moreover, the experimentation process is described, including the statistical verification of obtained results and the description

of key performance indicators alongside benchmarking problems. Chapter 3 proposes two procedures for dealing with a real-valued input. The first one involves a novel system, representing the environmental state by the interval predicates, while the other transforms input signal before an actual algorithm perceives it. Both Chapter 4 and 5 focus on potential issues spotted when processing real-valued data - optimizations for a more efficient knowledge collection process and the problems when obtaining the reward feedback requires multiple steps. Finally, Chapter 6 concludes the thesis and presents potential future research directions.

Chapter 2

Selected topics of LCSs

2.1 Road towards ALCSs

The concept of LCS was introduced by John Holland in 1975 [48] to model the idea of cognition based on adaptive mechanisms. From the early days, they consist of a set of rules named *classifiers* combined with mechanisms in charge of evolving them. Initially, the goal was to handle problems related to online interaction with external environments, as described by Wilson in [132].

To accomplish this, the emphasis was put on parallelism in the architecture and evolutionary mechanisms allowing the agent to adapt to potentially changing environments [40]. This approach was referenced as *"escaping brittleness"* [51] due to the problems related to the lack of robustness of the current artificial intelligence systems.

The naming convention used to refer to the LCS algorithm also changed since its infancy. Holland initially called it a classifier system, abbreviated either as (CS) or (CFS) [100]. From that time, it was also referred to as adaptive agents [53, 54], cognitive systems [53] and genetic-based ML (machine learning) [30, 39]. The current name of LCS was not adopted until the late 80s [98] after extending the architecture with a credit assignment component [51, 52].

This section provides a synopsis of crucial LCS concepts alongside the most popular variants and their contributions to the field.

2.1.1 Rules and Classifiers

LCS utilizes *rules* as a fundamental block of modelling knowledge in a general form. It comprises a *condition* (i.e. specified feature states) and an *action* (also referred to

as the class). They can be interpreted using the "IF condition THEN action" logical expression. The generalization property using the "do not care" symbol - # is possible when the condition part is expressed with either boolean or nominal representation. Two input situations are considered equivalent for a given classifier if the specified (non-#) values in the condition match the corresponding attributes of the two situations.

In addition to condition and action, a rule typically has many algorithm-related parameter values associated with it (like its performance or expected reward). The term *classifier* is used to describe a rule and its associated parameters.

It is essential to realize that LCS comprise a population of single rules that collaboratively seek to cover the problem space. The number of classifiers needed to solve the particular problem depends on factors like the problem complexity or rule representation used.

2.1.2 Driving Mechanisms

There are two fundamental components behind every LCS algorithm - discovery and learning (credit assignment). Both of them have generated respective fields of study, but in the context of LCS, we wish to understand their function and purpose.

Discovery component
The discovery component is responsible for exploring the search space of the target problem to uncover new rules. A vast majority of LCS algorithms employ some form of evolutionary computation, most often a GA (genetic algorithm), employing the Neo-Darwinist theory of natural selection [14, 39]. The evolution of rules is modelled after the evolution of organisms using the following biological analogies:

1. a code is used to represent the genotype/genome (condition),
2. a solution (phenotype) representation is associated with that genome (action),
3. a phenotype selection process (survival of the fittest) - the fittest organism (rule) has the greatest chances of reproducing and passing parts of its genome to offspring,
4. certain genetic genome operators are utilized in order to chase after fitter organisms (rules) [55, 108].

Two genetic operators are typically used to alter genome (rule) - mutation and crossover (recombination). The first one randomly modifies an element in an individual's genotype (rule), while the latter recombine parts of two favourable genotypes (rules), creating a new one. The selection pressure driving better organisms (rules) to reproduce more

frequently depends on the fitness function. The fitness function quantifies the optimality of a given rule, allowing it to be ranked across the entire population.

GA implementation varies in particular LCS, but the overall scheme involves evaluating all available rules, selecting the most promising offsprings (according to fitness value), applying genetic operators, introducing new offspring back to the population set and finally removing surplus or under-performing individuals.

Learning component
As mentioned, each classifier is accompanied by specific parameter values. The iterative update of them drives the process of LCS reinforcement by distributing any incoming reward signal to the classifiers that are accountable for it. This process serves two purposes:

1. identification of classifiers responsible for obtaining large future rewards,
2. encourage the discoverability of new rules (by directly affecting the fitness value).

The learning strategy depends on the nature of the problem and is realized differently in LCS implementations. In all cases, however, the process is conducted through trial-and-error interactions with the environment, where the occasional immediate reward is used to generate the policy (state-action mapping of agent-environment interactions), maximizing long-term reward [84, 97, 118].

2.1.3 Functional cycle

The agent interacts with the environment in consecutive trials. Each trial consists of sequential steps usually executed as follows:

1. Filter population $[P]$ and select classifiers where condition matches environmental perception forming a *match-set* $[M]$.
2. Determine the action that will be executed (depending on the strategy).
3. Narrow down the match-set by selecting only classifiers advocating proposed action - create the *action-set* $[A]$.
4. Execute the action in the environment, obtaining a new state.
5. Refine classifiers by executing discovery and learning components.

Algorithm 1 LCS experiment workflow

```
[P] ← initialize population
while termination criteria are met do
    env: reset state
    RUN TRIAL(P)
end while
```

Algorithm 2 Proposition of single LCS trial

```
procedure RUN TRIAL(P)
    t ← 0
    σ ← env: perceive situation
    while trial is finished do
        if [A]_{-1} is not empty then
            APPLY DISCOVERY COMPONENT considering σ, σ_{-1}, a
            APPLY LEARNING COMPONENT in [A]_{-1} considering ρ
        GENERATE MATCH SET [M] out of [P] using σ
        a ← CHOOSE ACTION using [M]
        GENERATE ACTION SET [A] according to [M] using a
        σ_{-1} ← σ
        (σ, ρ) ← env: execute action a          ▷ Obtain next state and reward
        t ← t + 1
        [A]_{-1} ← [A]
    end while
    APPLY DISCOVERY COMPONENT considering σ, σ_{-1}, a
    APPLY LEARNING COMPONENT in [A] considering ρ
end procedure
```

Algorithms 2, 1 present an overall course of an experiment and an exemplary trial interaction for niche-based LCS implementation respectively.

The action selection and classifiers evolution phases are implemented individually in different LCS, but the main objective of reaching the ideal generalization level is crucial to all of them. The system should find a population that covers the search space as compactly as possible without being detrimental to the optimality of behaviour.

2.1.4 Pittsburgh vs Michigan approach

One of the most fundamental distinctions in LSC research is storing knowledge by using two different approaches - *Michigan-style* or *Pittsburgh-style*. The first ones were proposed by Holland [54] while the latter one by Kenneth DeJong and his student [111, 112].

The fundamental discrepancy is the structure of an individual. In Michigan systems, each individual is a classifier; in Pittsburgh, each individual is a set of classifiers - see Figure 2.1.

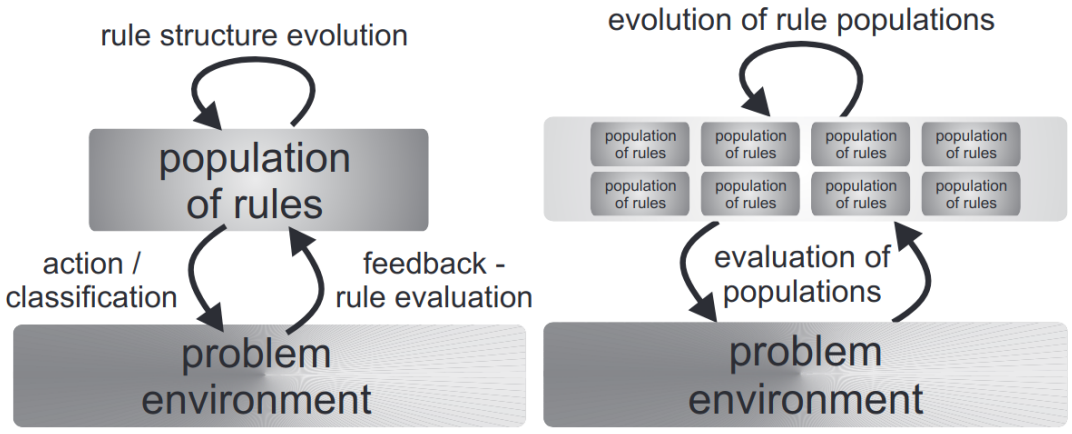

Figure 2.1: *Differences between representing knowledge in both Michigan-style and Pittsburgh-style LCS. Figure taken from [2].*

Thus, the classifiers in Michigan-style LCS are being continuously evaluated and evolved, while in Pittsburgh-style LCS, the same process is much more complicated because the whole population needs to be assessed. Therefore, Michigan-style systems are typically applied in interactive, online learning problems, while Pittsburgh ones are rather suitable for offline learning [2].

This work focuses solely on the first class of LCS.

2.1.5 Representative LCS

This section describes Michigan-style LCS implementations contributing primarily to the current advancements.

CS-1

Two years after presenting the theoretical model of CS, Holland and Reitman proposed its implementation of CS-1 [54]. The system realized the Darwinian principle of the *survival of the fittest* [50, 105] and was generating adaptive behaviour by maximizing reinforcement using the BBA (bucket brigade algorithm) [50, 105].

Some internal mechanisms like the usage of an internal message list (handling all input and output communications between the system and the environment and providing a makeshift memory) or interactions between evolving both single classifier and entire population were considered difficult [39, 142]. Moreover, the obtained results were inconsistent, signifying the need for improvements.

ZCS

The ZCS (*Zeroth Level Classifier*) introduced by Wilson in 1994 [133] encompasses all

LCS components while simplifying the CS-1, increasing its understandability and performance. The significant change was the removal of the internal message list (determining the rule's format entirely by the system interface) alongside with rule-bidding credit assignment (replacing it with BBA/Q-learning algorithm [131]).

Moreover, the classifier fitness was based on the accumulated reward that the agent can get from firing the classifier, giving rise to the *"strength-based"* family of LCS. As a result, the discovery component eliminates classifiers providing less reward than others from the population.

ZCS achieved similar performance to CS-1, demonstrating that Holland's idea could work even in a straightforward framework. However, the premature converge onto suboptimal rules before search space can be explored appropriately and a stable population formed led Wilson to consider other ways to achieve this.

XCS

In 1995 Wilson introduce yet another, groundbreaking modification called the XCS (*eXtended Classifier System*) [24, 134, 135, 137] noted for being able to reach optimal performance while evolving accurate and maximally general classifiers.

The essential changes include:

- rule fitness is based on the accuracy of predictions (forming the *"accuracy-based"* LCS family),
- replacement of panmictically acting GA with a *niche-GA* [10] (applied only in action set $[A]$ instead of globally $[P]$),
- explicit generalization mechanism (*subsumption*),
- an adaptation of Q-Learning credit assignment.

The XCS design drives it to form an all-inclusive and accurate representation of the problem space rather than focusing on higher payoff niches. Auspicious performance results showed that RL [108] and LCS are not only linked but inherently overlapping, redefining LCSs as RL endowed with generalization capabilities [74, 75]. As a result, it becomes the most popular LCS implementation, guiding other system implementations heavily inspired by its architecture.

Modern LCS

The latest LCS advancements focus mainly on problems related to:

- Knowledge visualization and rules compaction [80, 81] - new visualization techniques, termed as *Feature Importance Map*, *Action-based Feature Importance Map* and *Action-based Feature's Average value Map* successfully produce human-discernable results for the investigated complex Boolean problems. Domains, where the pattern consists of 6435 different cooperating rules, were translated into concise graphs, facilitating tracking the overall training progress.

- Learning with incremental data [60, 61] - a solution for extracting knowledge and utilizing it in further experiments using various deep convolutional blocks. The proposed method obtained better accuracy than other state-of-the-art algorithms using the investigated image datasets.

- Classifying images using convolutional autoencoders [59, 60, 93] - high-dimensional problems are investigated by an ensemble of LCS with deep-learning methods. The input is compressed using the autoencoder and later processed by the LCS algorithm. Promising applications involve designing an intrusion detection system [12] or classifying MNIST images [86].

- Dealing with perceptual aliasing environments [107] by utilizing a feature of vertebrate intelligence allowing multiple simultaneous representations of an environment at different levels of abstraction. Considering states at a constituent level enables the system to place them appropriately in holistic-level policies for multistep problems.

2.1.6 Anticipatory Learning Classifier Systems

The field of cognitive psychology initially guided the investigation about the presence and importance of anticipations. It was proved that "higher" animals form and exploit anticipations while adopting their behaviour in distinct tasks. Wilson noted that when simulating adaptive behaviour, the animats should be endowed with anticipations [87]. In traditional RL, the first approaches manifested in Dyna architecture [118], but due to the lack of generalization capabilities, its utilization was limited. The ALCS aims to bridge the gap for exploiting the generalization capabilities while including the explicit notion of anticipations.

CFCS2

Riolo laid the foundations towards including predictions in 1991 by introducing a modification called CFCS2 [99]. It addressed the task of performing "latent learning" or "lookahead planning" where *"actions are based on predictions of future states of the world, using both current information and past experience as embodied in the agent's internal models of the world"* [78].

CFCS2 used "tags" to specify if a current action posted to a message list refers to actual action or anticipation, claiming a reduction in the learning time for general sequential decision tasks. Riolo was able to show the possibilities of latent learning and lookahead planning, but the implicit formation of anticipations appeared to be misleading. Additionally, the CFCS2 did not achieve any generalization capabilities.

ACS

The first system competent of evolving a maximally generalized, accurate and complete mapping of all possible situation-effect-action triples observable in the environment - the ACS (Anticipatory Classifier System) was introduced by Stolzmann in 1997 [113, 114].

The rule structure was enhanced with an *anticipatory* or *effect* part that explicitly anticipates the effects of an action in a given situation. The effect part could determine which attributes change after executing an action by using a *"pass-through"* symbol - #, which significantly enhances the rule's interpretability. In order to perform latent learning, forming and refining classifiers, a discovery component - the ALP (Anticipatory Learning Process) was introduced.

ALP realizes the psychological Hoffmann's anticipatory behavioral control theory [44, 45] stating that conditional action-effects relations are learned latently using anticipations, which he further refined in [46]. The following points (visualised in Figure 2.2 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 are correct, the bond between response and anticipation is strengthened and weakened otherwise.
4. Behavioural stimuli further differentiate the $R - E_{ant}$ relations.

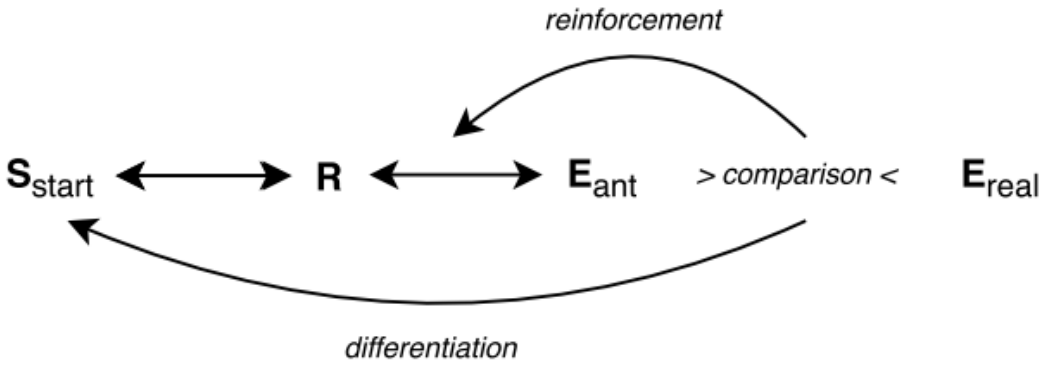

Figure 2.2: *The theory of anticipatory behavioural control. Adapted from [46].*

The implementation of this component compares the obtained state σ_{t+1} with the classifier's anticipation σ_{t+1}^{ant}. Then, the *quality* parameter of the involved classifier is updated according to four possible scenarios:

1. *Useless case.* After performing an action, no change in perception is perceived from the environment. The classifier's quality decreases.
2. *Expected case.* When the newly observed state matches the classifier prediction. Classifiers' quality is increased.
3. *Correctable case.* When new state $\sigma(t+1)$ does not match the anticipation of $cl.E$. A new classifier with a matching effect part is generated.
4. *Not correctable case.* When new state $\sigma(t+1)$ does not match the anticipation of σ_{t+1}^{ant}, and it is not possible to correct the classifier, then its quality is penalized like in the useless case.

The classifiers are removed from the population if they became *inadequate* - the quality metrics fall below a threshold denoted by the θ_i parameter.

For the reward distribution, the proposed architecture is described by Algorithm 3 and use the BBA algorithm) [50]. The performance of both components result in a system capable of solving multistep problems, planning, speeding up learning or disambiguating perceptual aliasing.

ACS2

Later, in 2002 Martin Butz [18, 23] extended ACS by introducing a system named ACS2. The significant changes from the previous version include:

Algorithm 3 ACS Trial

procedure RUN ACS TRIAL(P)
  $t \leftarrow 0$
  $\sigma \leftarrow$ *env:* perceive situation
  while trial is finished **do**
    if classifier c_{-1} is present **then**
      `APPLY ALP` considering σ, σ_{-1}, c_{-1}
      `APPLY BBA` in c_{-1} considering ρ
    `GENERATE MATCH SET` $[M]$ out of $[P]$ using σ
    `SELECT CLASSIFIER` c from $[M]$
    $\sigma_{-1} \leftarrow \sigma$
    $(\sigma, \rho) \leftarrow$ *env*: execute classifier's c action ▷ Obtain next state and reward
    $t \leftarrow t + 1$
    $c_{-1} \leftarrow c$
  end while
  `APPLY ALP` considering σ, σ_{-1}, c
  `APPLY BBA` in c considering ρ
end procedure

- application of learning component across the whole action set $[A]$ (all classifiers from $[M]$ advocating selected action),
- introduction of GA (genetic algorithm) component,
- refinement of learning component utilizing the Q-learning [118] algorithm,
- condition-action-effect triples that anticipate no change in the environment are explicitly encompassed in population,
- add *mark* property to classifier, recording the properties in which it did not perform correctly before.

The ALP process was refined due to its delicate nature. The first improvement relates to the process of improving new classifier generation by covering all possible condition-action-effect triples. The process called `COVERING` is called when there is no classifier in the action set $[A]$ representing the encountered situation. A newly created classifier specifies all changes from the previous situation σ_{t-1} to situation σ in condition and effect part. The idea of covering is presented by Algorithm 4.

Algorithm 4 Generating new classifier by covering condition-action-effect triple

```
procedure COVERING(σ_{-1}, act, σ)
    child ← generate empty classifier with action act
    for all positions i in σ do
        if σ_{-1} ≠ σ_t then
            child.C[i] ← σ_{-1}[i]
            child.E[i] ← σ[i]
    end for
    child.q ← 0.5
    return child
end procedure
```

Second, because the offspring might be introduced by ALP and GA, the insertion process must be cautious. A *subsumption* operation checks if one classifier is not covered by the other, more general one. For a classifier cl_{sub} to subsume another classifier cl_{tos}, the subsumer needs to be experienced, reliable and not marked. Moreover, the subsumer's condition part needs to be syntactically more general, and the effect part needs to be identical.

The GA algorithm is responsible for generalizing the condition parts. The method starts by determining if the GA should actually take place, controlled by the t_{ga} time stamp and the actual time t. If a GA takes place, preferable accurate, over-specified classifiers are selected, mutated and crossed, see Algorithm 5 and [23] for a detailed explanation.

Algorithm 5 Genetic Generalization process

procedure APPLY GENETIC ALGORITHM($[A], t$)
  if $t - \frac{\sum_{cl \in [A]} cl.t_{ga} cl.num}{\sum_{cl \in [A]} cl.num} > \theta_{GA}$ **then**
    for each classifier cl in $[A]$ **do**
      $cl.t_{ga} \leftarrow$ actual time t
    end for
    $parent_1 \leftarrow$ `SELECT OFFSPRING` in $[A]$
    $parent_2 \leftarrow$ `SELECT OFFSPRING` in $[A]$
    $child_1 \leftarrow$ copy classifier $parent_1$
    $child_2 \leftarrow$ copy classifier $parent_2$
    $child_1.num \leftarrow child_2.num \leftarrow 1$
    $child_1.exp \leftarrow child_2.exp \leftarrow 1$
    `APPLY GENERALIZING MUTATION` on $child_1$
    `APPLY GENERALIZING MUTATION` on $child_2$
    if RandomNumber[0, 1) $< \chi$ **then**
      `APPLY CROSSOVER` on $child_1$ and $child_2$
      $child_1.r \leftarrow child_2.r \leftarrow \frac{parent_1.r + parent_2.r}{2}$
      $child_1.q \leftarrow child_2.q \leftarrow \frac{parent_1.q + parent_2.q}{2}$
    $child_1.q \leftarrow child_1.q/2$
    $child_2.q \leftarrow child_2.q/2$
    `DELETE CLASSIFIERS` in $[A], [P]$ to allow the insertion of 2 children
  for each *child* **do**
    if $child.C$ is all general **then**
      next *child*
    else
      `ADD GA CLASSIFIER` *child* to $[P]$ and $[A]$
  end for
end procedure

Algorithm 6, which will be used in further examples, presents the ACS2 trial workflow with the possibility of explicitly determining whether the GA mechanism is applied. Experiments showed that the genetic generalization process in ACS2 decreased the population size and consequently speed up the computational process.

Figure 2.3 demonstrates an example of RL policy generated by a population of 322 classifiers trained over 5000 trials operating in popular Maze5 benchmarking problem. The model knows the consequences of each available action in every possible state; therefore, estimating the fitness value of each classifier can suggest the most promising action. The

Algorithm 6 ACS2 Trial

procedure RUN ACS2 TRIAL(P)
  $t \leftarrow 0$
  $\sigma \leftarrow$ *env:* perceive situation
  while trial is finished **do**
    if $[A]_{-1}$ is not empty **then**
      `APPLY ALP` in $[A]_{-1}$ considering σ, σ_{-1}, t and $[P]$
      `APPLY REINFORCEMENT LEARNING` in $[A]_{-1}$ considering ρ
      if genetic generalization is enabled **then**
        `APPLY GENETIC ALGORITHM` in $[A]_{-1}$ considering t
    `GENERATE MATCH SET` $[M]$ out of $[P]$ using σ
    $a \leftarrow$ `CHOOSE ACTION` using $[M]$
    `GENERATE ACTION SET` $[A]$ according to $[M]$ using a
    $\sigma_{-1} \leftarrow \sigma$
    $(\sigma, \rho) \leftarrow$ *env*: execute action a ▷ Obtain next state and reward
    $t \leftarrow t + 1$
    $[A]_{-1} \leftarrow [A]$
  end while
  `APPLY ALP` in $[A]$ considering σ, σ_{-1}, t and $[P]$
  `APPLY REINFORCEMENT LEARNING` in $[A]$ considering ρ
  if genetic generalization is enabled **then**
    `APPLY GENETIC ALGORITHM` in $[A]$ considering t
end procedure

learning can be still optimized by using cognitive mechanisms like lookahead planning [78].

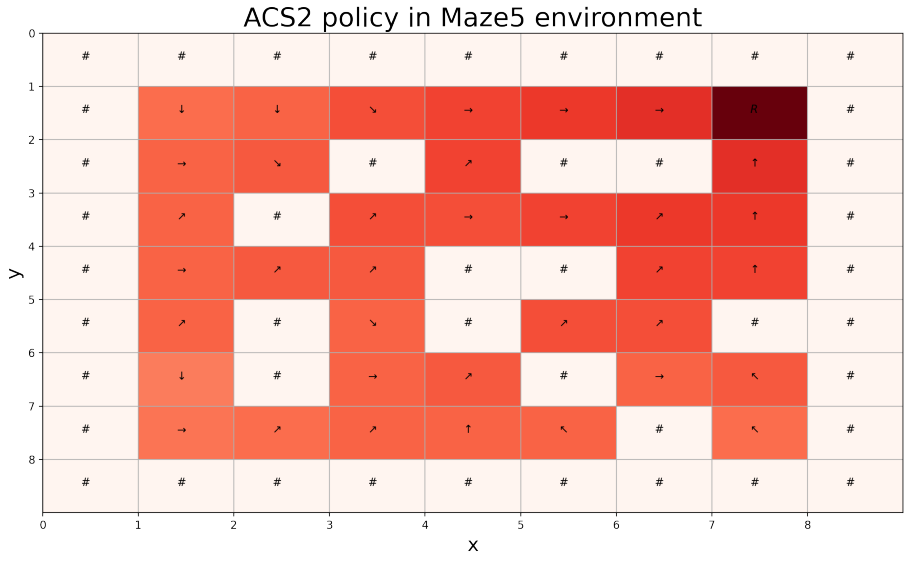

Figure 2.3: *Policy generated by ACS2 in Maze5 environment. The saturation of the red colour reflects the best classifier fitness value. The agent is inserted randomly on the red field in each trial to reach the reward state "R" by executing one of the eight possible actions.*

Algorithm 7 YACS Trial

procedure RUN YACS TRIAL(P)
  $t \leftarrow 0$
  $dv \leftarrow$ initialize *desirability values* table
  $\sigma \leftarrow$ *env:* perceive situation
  while trial is finished **do**
    `MEMORIZE PERCEPTION` σ
    $a \leftarrow$ `CHOOSE ACTION` using defined strategy
    $\sigma_{-1} \leftarrow \sigma$
    $(\sigma, \rho) \leftarrow$ *env*: execute action a ▷ Obtain next state and reward
    `GENERATE MATCH SET` $[M]$ out of $[P]$ using σ
    `GENERATE ACTION SET` $[A]$ according to $[M]$ using a
    if $[A]$ is not empty **then**
      $c \leftarrow$ `APPLY CLASSIFIER COVERING` considering $[P], \sigma_{-1}, \sigma, a$
      Append c to $[P]$
    `APPLY EFFECT COVERING` considering $[P], \sigma_{-1}, \sigma, a$
    `APPLY SPECIALIZATION OF CONDITIONS` considering $[P]$
    `APPLY SELECTION OF ACCURATE CLASSIFIERS` considering $[P]$
    Update *desirability values* using $[P], \sigma, dv, a, \rho$
    $t \leftarrow t + 1$
  end while
end procedure

The recent advancements include the integration of the action planning mechanism [127], PEPACS extension where the concept of *Probability–Enhanced–Predictions* is used for handling non-deterministic environments [92] or BACS tackling the issue of perceptual aliasing by building *Behavioral Sequences* [91].

YACS

In the same year, 2002, Gérard introduced a YACS (*Yet Another Classifier System*) [38]. It shares the same C-A-E classifier structure as ACS and ACS2, but the essential conceptual difference is that YACS is designed to decorrelate the acquisition of relevant C and E parts by building them independently using a set of heuristics. The latent learning process is designed to set the E parts to perceived changes in the environment and then discover relevant C parts. Moreover, it does not take advantage of genetic generalization mechanisms relying on the discovery process to determine significant attributes correctly. In order to compute the optimal policy YACS memorizes internally every encountered state and uses a simplified variety of value iteration. The complete trial workflow is depicted in - Algorithm 7

The evaluation was performed only in simple multistep maze environments reaching *near optimal* solutions. The specialization process in YACS leads to less over-specialization than the corresponding process in ACS but still suffers from the lack of a dedicated

Algorithm 8 MACS Trial

procedure RUN MACS TRIAL(P)
  $t \leftarrow 0$
  $dv \leftarrow$ initialize *desirability values* table
  $\sigma \leftarrow$ *env:* perceive situation
  while trial is finished **do**
    `MEMORIZE PERCEPTION` σ
    $a \leftarrow$ `CHOOSE ACTION` using defined strategy
    $\sigma_{-1} \leftarrow \sigma$
    $(\sigma, \rho) \leftarrow$ *env*: execute action a ▷ Obtain next state and reward
    $\sigma_{seen} \leftarrow$ extract observed states using dv
    `APPLY CONDITION GENERALIZATION` considering $[P], \sigma_{-1}, a, \sigma, \sigma_{seen}$
    `APPLY SPECIALIZATION OF CONDITIONS` considering $[P], \sigma_{seen}$
    `APPLY TRANSITION COVERING` considering $[P], \sigma_{-1}, a, \sigma$
    `APPLY POPULATION EVALUATION` considering $[P], \sigma_{-1}, a, \sigma$
    `APPLY ACCURATE RULES SELECTION` considering $[P]$
    $t \leftarrow t + 1$
  end while
end procedure

generalization mechanism. Additionally, the YACS cannot deal with uncertainty and operate only in Markov and deterministic environments.

MACS

Knowing certain limitations of ACS and YACS in 2005, Gérard proposed a MACS (*Modular Anticipatory Classifier System*) [36]. He realised that specific problems would be better approached with a different representation of the rule formalism. While in ACS and YACS, generalisation and selective attention are afforded by the joint use of *don't care* and *pass-through* symbols, it can detect if a particular attribute is changing or not. The efficient representation of regularities across different attributes is impossible because each situation is considered an unsecable whole.

The particularity of such regularity is that the new value of an attribute depends on the previous value of another one. To overcome this problem, the *pass-through* symbol (#) in the expected situation E was replaced with the *don't know* symbol (?). This formalism allows decorrelating the attributes from E, where previously, the new value of an attribute may only depend upon the previous value of the same attribute.

This modification resulted in specific rule representation and modified architecture for latent learning. Each time a match-set $[M]$ is built a rule for each attribute change is picked, and then the final anticipation is combined, Algorithm 8 demonstrates the sequence of behavioural act.

The example of differences between rules is illustrated using the Maze228 problem [36]. The agent's goal is to obtain the final reward. It perceives its surroundings (wall - `1`,

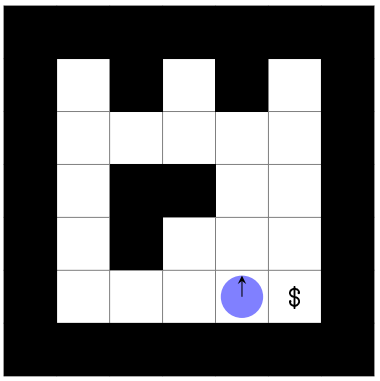

Figure 2.4: *Agent located on the left side and facing north of the Maze228's reward. Two steps - rotate right and step ahead are needed to reach it.*

	Step Ahead	**Rotate Left**	**Rotate Right**
Classifiers	0#9###### 0 0????????		
	#0###10## 0 ?0????????	######0## 1 0????????	##9###### 2 9????????
	0#9##1### 0 ?0????????	#######0# 1 ?0???????	###1##### 2 ?1???????
	#0######## 0 ??0??????	0######## 1 ??0??????	####1#### 2 ??1??????
	0#9###### 0 ???9?????	#0######## 1 ???0?????	#####1### 2 ???1?????
	0######## 0 ????0????	##9###### 1 ????9????	######0## 2 ????0????
	0#####0## 0 ?????0???	###1##### 1 ?????1???	#######0# 2 ?????0???
	#######0# 0 ??????0??	####1#### 1 ??????1??	0######## 2 ??????0??
	##9###### 0 ??????0??	#####1### 1 ???????1?	#0######## 2 ???????0?
	##91#10## 0 ???????1?	######### 1 ????????0	######### 2 ????????0
	0######## 0 ????????0		
Anticipation	000900010	000091110	911100000

Table 2.1: *MACS population of classifiers for the perception of 009111000 in Maze228 environment - see Figure 2.4. Notice how atomic classifiers create final anticipation by using the dont-know symbol.*

path - 0 or reward - 9) using the clockwise vector of nine attributes and can execute three actions - step ahead - 0, rotate left - 1 or rotate right - 2. Assume that the current situation is depicted in Figure 2.4, where the agent is located on the left of the reward, and its actual perception is 009111000.

After training, MACS represents complete knowledge about this environment using 211 classifiers. The final anticipation is obtained after combining particular classifiers selected for each action, which is visualized in Table 2.1.

On the other side, the ACS2 creates a much larger population of 402 reliable classifiers. There is a single classifier describing the change of multiple attributes for each action. The anticipation in both cases is the same, but due to the specific nature of this problem,

	Step Ahead	**Rotate Left**	**Rotate Right**
Classifiers	##9111#0# 0 ##0900#1#	##911#00# 1 ##009#11#	009#11### 2 911#00###
Anticipation	000900010	000091110	911100000

Table 2.2: *ACS2 population of classifiers for the perception of 009111000. A single classifier is used to represent each action. Despite this, the population size is still much larger than in MACS.*

MACS results in an overall more compact internal model of the environment. The difference is depicted in Table 2.2.

2.2 Real-valued input challenge

Most real-world environments or datasets are represented using exclusively continue-valued inputs or alongside discrete attributes. Unfortunately, the described ALCS implementations were not initially designed for such representations. The fundamental modification required to overcome the limitation above is the adaptation of rule representation within the system. Despite the variety of possible approaches listed in this chapter, each comes with its own merits and perils.

Two significant concepts involving the discretization and interval encoding using dedicated alphabets will be pursued later, while the hindmost ones involving the neural network and fuzzy logic will be only mentioned. There are two main reasons for this decision:

1. drastic changes in rule representation requires significant modifications in existing and tightly coupled components. In some cases, some of them are not usable in their proposed form - like the ALP component in ACS systems needs a complete redefinition for other alphabet encodings,
2. rule interpretability is relevantly blemished.

Therefore, even if some approaches are more appealing for the problem, their usage violates principal ALCS virtue of transparency (creating a compact set of human-readable rules) or does not formally specify the behaviour of related inner mechanisms fulfilling the evolution and assessing each classifier. However, more detailed research is highly advised but will not be pursued herein.

2.2.1 Discretization

One of the approaches for handling continuous attributes is to partition them into a number of sub-ranges and treat each sub-range as a category. This process is known as *discretization* and is used across many families of ML algorithms in order to create better models [35]. In the XCS family, the modification of XCSI [139] adopted the algorithm for the integer domain. The modification was evaluated on the *Wisconsin Breast Cancer* dataset and turned out to be competitive with other ML algorithms. Minding the nature of ALCS, the usage of such nominal values would also be the most straightforward approach. ALCS systems by design are not limited by *ternary representation*, therefore creating an arbitrary number of potential states is achievable. The first such implementation called rACS (real Anticipatory Classifier System) was proposed by Unold and Mianowski in 2016 [126] and tested successfully on *Corridor* and *Grid* environments. Both have a regular structure, meaning that the intervals are evenly distributed throughout the investigated range.

Unfortunately, the number of ways to discretize a continuous attribute is infinite. Kotsiantis in [65] takes a survey of possible discretization techniques, but in this work, the preferred method is to divide the search-space into n equally spaced intervals, referred later as *bins* or *buckets*. When the n is large, system precision is increased, resulting obviously in the growth of the classifier population. Such a population can be further optimized by compacting the classifiers operating in the neighbouring space regions [139, 144]. On the contrary, when the n is low, the system under-performs due to the inability of creating accurate classifiers.

The process of assigning a discrete value for each consecutive interval region within $[0, 1]$ range is depicted on Figure 2.5.

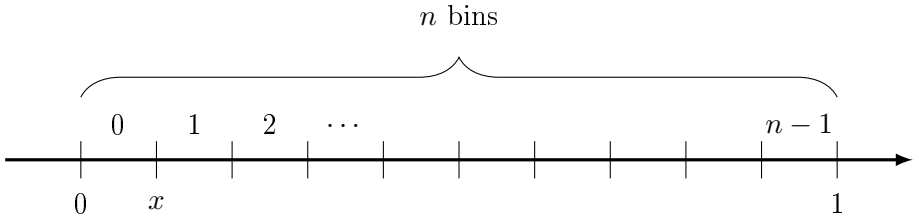

Figure 2.5: *Representation of allele as a natural number of the partition. The size of the input space is n.*

Such a solution for real-value representation in a simple scenario does not require significant changes to any components and retain the human readability and interpretability of created rules. However, when an arbitrary number of bins for each observation attribute

is used, certain modifications might be necessary (for example, additional restrictions for the GA cross-over operator).

2.2.2 Interval predicates

The discretization approach assumed that a distinct value of the condition attribute represents a fixed-length interval part of an input range. Another approach is to encode the input using custom hyper-rectangular boundaries described by half-open interval $[p_i, q_i)$, which matches the environment signal x_i if $p_i \leq x_i < q_i$.

Such approach facilitates creation of more general and compact population size, due to arbitrarily interval ranges. In 1999 Wilson took the approach to adapt the XCS to automatically search for optimally decisive thresholds [136]. His approach representation named CSR (*center-spread representation*) used to represent an interval tuple (c_i, s_i) where $c_i, s_i \in \mathbb{R}$ where c_i is the center of the interval, and s_i its spread. The interval is therefore described in Equation 2.1:

$$\begin{aligned} p_i &= \min(p_{min}, c_i - s_i) \\ q_i &= \max(q_{max}, c_i + s_i) \end{aligned} \tag{2.1}$$

His system, called XCSR, differs only at the input interface, mutation and covering operators. Preliminary tests revealed the weakness of the crude mutation operator. Moreover, the testing environment, which is very regular and therefore did not challenge the agent with interesting problems like noise or data contradiction.

In 2003 Stone and Bull thoroughly addressed those problems in [117] by introducing two new representations - OBR (ordered-bounded) and UBR (unordered-bounded). The OBR is an extension of Wilson XCSI [138] but enhanced with real-valued interval tuple (l_i, u_i) where $l_i, u_i \in \mathbb{R}$. Here l_i and u_i relates to lower and upper bounds, respectively. The encoding imposes $l_i < u_i$ ordering; therefore, all system components that might inadvertently change must be recognized and taken care of. This requirement was relaxed in UBR [140], thus the same interval can be encoded by both (p_i, q_i) and (q_i, p_i). The advantage is that no operator constraints are needed when the ordering restriction is violated, which constitutes a form of epistasis between l_i and u_i, as their values are mutually dependent. The authors showed that CSR and OBR are biased in interval generation in bounded solution spaces. The UBR obviating limitations of OBR was assumed to yield better results in one of the testing problems and therefore considered superior to the other ones.

Finally, in 2005 Dam and Abbass [29] recognized that UBR changes the semantics of the chromosome by alternating between min and max genes. This discrepancy is challenging for the XCS because it disturbs the genetic process evolving the population of classifiers [48] [39]. They present the Min-Percentage representation of (m_i, k_i) where $m_i, k_i \in \mathbb{R}$ tuple, where the interval is determined by the Equation 2.2, which compared with UBR did not provide any substantial improvements.

$$\begin{aligned} p_i &= m_i \\ s_i &= k_i \cdot (k_{max} - p_i) \\ q_i &= m_i + s_i \end{aligned} \tag{2.2}$$

It is also worth mentioning an important, subtly different, family of learning classifier systems handling real-valued input natively by *approximating functions. The most popular implementation of XCSF* [42, 140] computes the payoff value locally instead of learning its prediction through a gradient-descent type update. The classifier's condition covers the continuous input by an OBR interval. Enhanced with additional weights vector, updated by regression techniques, the output prediction can be calculated. The classifier structure was further simplified by eliminating the proposed action. Because of the lack of explicit state anticipation capabilities, the function approximation learning classifiers systems are not considered in this work.

2.2.3 Neural networks

O'Hara and Bull experimented with representing the rule structure of an XCS classifier with two artificial neural networks, introducing the system named X-NCS [89, 90]. While the first one, called the *action-network*, determines the application condition of the classifiers replacing the condition-action part, the latter one - *anticipation network* - forms the description of the predicted next state.

Both networks are fully connected MLP (multi-layered perceptrons) with the same nodes in their hidden layer. The input layer in both cases matches the observation state vector provided by the environment. In the action network, the size of an output layer is equal to the number of possible actions incremented with an extra node signifying a non-matching situation. Hence, the anticipation network is responsible for predicting the next state; the size of its output layer is equal to the input one. Figures 2.6 and 2.7 visualize both topologies.

The system starts with an initial random population, containing the maximum number of classifiers considered in a particular experiment, as opposed to standard XCS [134]. The network weights are randomly initialized in the range $[-1.0, 1.0]$ and are updated throughout the experiment using two search techniques - the local search performed by backpropagation complemented by the global sampling performed by GA.

In order to assess the general system error, all of the internal mechanisms remained unchanged except for the calculation of the rule's absolute error, which is defined as the sum of prediction and lookahead error (measuring the correctness of the prediction).

The critical difference between the classic ternary representation is the lack of explicit wildcard symbol and hence no explicit pass-through of input to anticipations. A concept of a reliable classifier cannot be applied in X-NCS - the anticipation accuracy is based on a percentage of accurate anticipations per presentation.

The authors tested the extension in various configurations showing promising results, especially on discrete multistep problems. Due to novel rule representation, the X-NCS system is suited for real-valued data representation, but the conceptual differences make it difficult to compare it with other systems. Aspects like vague generalization metric, constant population size or the anticipation accuracy computation would require dedicated research solely on this implementation.

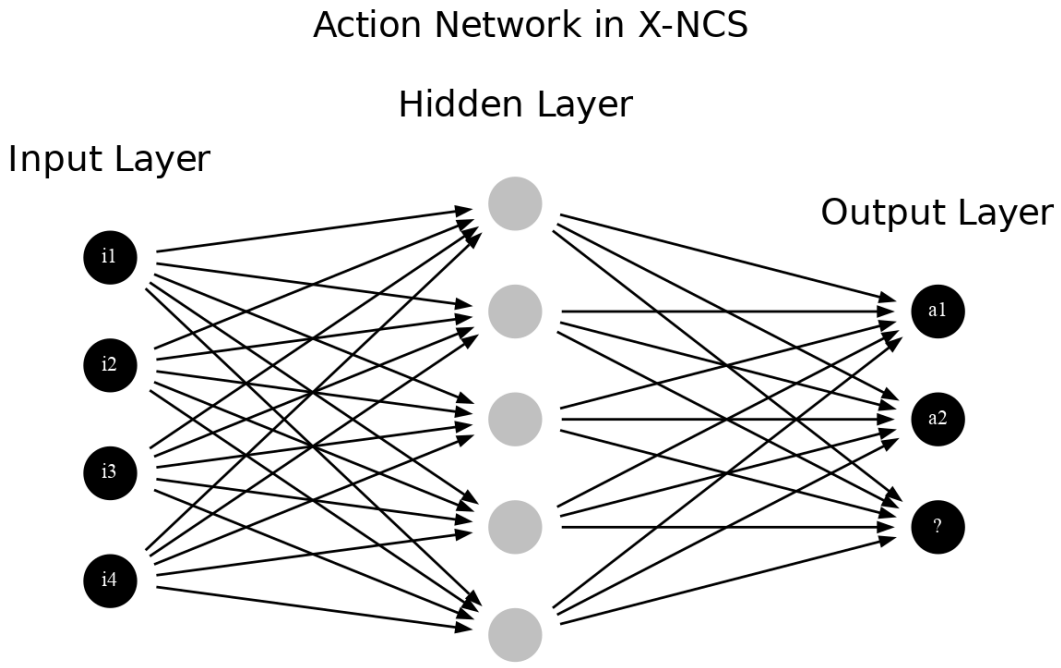

Figure 2.6: *The topology of the fully connected MLP of the network determines the agent's action based on the observed state (input layer). The number of output nodes equals the number of possible actions with an extra state representing a non-matching case. Figure adapted from [90].*

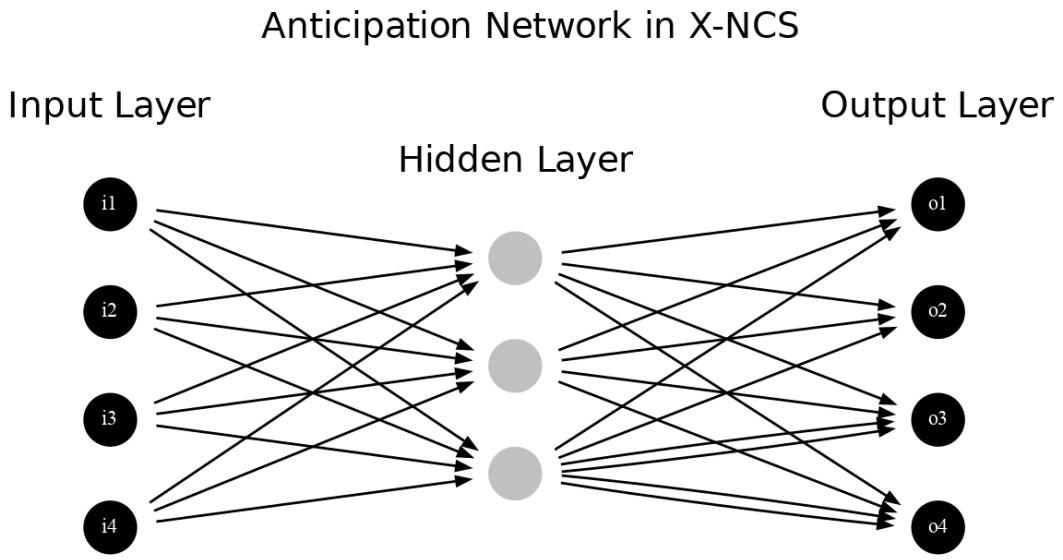

Figure 2.7: *The topology of fully connected MLP of the network determines the anticipated states using the observed environmental state (input layer). Specific output nodes refer to particular input nodes. Figure adapted from [90].*

Interestingly, two years later, a similar system evolved. The authors take advantage of the idea of both the function mapping [141] and the neural anticipation [90]. The classifier structure was extended with a parametrized anticipatory function an_f, which is trained using supervised learning based on the current state σ_t and the next state σ_{t+1}. The proposed XCSAM system (XCS with anticipation mappings [15]) does not use the lookahead error and focuses on the value of actions. The authors claim that the anticipatory capabilities are obtained as the "side effects". Because of the lack of any dedicated prediction components, the anticipatory burden falls on the function an_f. Preliminary experiments showed that both X-NCS and XCSAM obtain comparable results on discrete multistep environments, however both of the systems can be generalized to problems involving continuous inputs [77].

2.2.4 Fuzzy representation

The Michigan-style genetic fuzzy rule-based system [28] is a machine learning system that employs linguistic rules and fuzzy sets in its representation and an evolutionary algorithm for rule discovery. Casillas proposed relevant modifications to the XCS and introduced a new modification called Fuzzy-XCS [26]. The newly created system was capable of dealing with continuous states and actions while maintaining maximal generalization. A similar solution was also proposed later by Bonarini in [9].

Kondziela undertook an approach to create a fuzzy variant of ACS in 2021 [64]. The presented idea modified the system comprising four major elements [103]:

- *fuzzification* - assigns set membership of current environmental perception,
- *inference* - aggregates the results by comparing the membership functions with collected knowledge,
- *knowledge-store* - stores population of classifiers where representing rules using the *Fuzzy Inference System* (Mamandi) of `IF..AND..THEN` statements [146],
- *defuzzification* - provides a single value response obtained from the inference phase determining the final action value.

As the first step the vector of environment signal determines set memberships using predefined functions [9, 25]. Then using the rule representation described by Equation 2.3 the match set is formed. Each input value X_i is equal to a linguistic set of $\tilde{A}_i = \{A_{i1} \vee \cdots \vee A_{il}\}$ meaning that classifier internally consists of a rule described with $\{0, 1, \#\}$ symbols (ternary alphabet).

$$\textbf{IF } X_1 \text{ is } \tilde{A}_1 \text{ and } \ldots \; X_n \text{ is } \tilde{A}_n \textbf{ THEN } Y \text{ is } B \tag{2.3}$$

In the next step, the action set $[A]$ is formed in the same way as in traditional ACS, but the final action selection procedure differs - it is proposed by taking advantage of each rule's membership function values, and the *Center of Gravity* method for defuzzification [64].

Preliminary tests made on multistep, discrete maze environments showed that fuzzy ACS implementation successfully predicted more than 75% of encountered situations and maintained a limited number of reliable classifiers (although both metrics were highly oscillating). The author did not report any other operational performance indicators.

The usage of fuzzy logic enables the system to handle the real-valued input naturally. The apparent impediment is the requirement to specify membership functions for each environmental perception upfront. The selection of optimal values is complicated [8], and further increases the number of overall system tunable parameters. The other identified flaw is the GA phase which is not suited for new representation. Both the mutation and cross-over operators should be reviewed accordingly.

2.3 Key Performance Indicators

The ALCS comprise complex mechanisms such as ALP or GA, which might be hard to comprehend and analyse. Therefore, specific metrics reflecting their nature were chosen to visualise system behaviour over time and assess its performance. This section presents several KPIs (key performance indicators) used across further experimental evaluations. Each of them can be collected at a particular interval of time (for example, every ten trials) and can access the properties of both the environment and the agent within the single simulation.

2.3.1 Classifier population size

Each LCS creates an internal population of classifiers. Knowing its total size $\|P\|$ represents the agent's ability to express the internal model of the environment. When maintaining high performance, having a smaller population also means that the agent is commendable at building a more compact and compressed representation of knowledge [18].

Besides knowing the total population size, it's subset of reliable classifiers size $\|P_{reliable}\|$ might also be easily obtained - Equation 2.4. The difference $\|P\| - \|P_$reliable$\|$ estimates the size of currently evolving classifiers, where part of them gets either accepted or dismissed. Eventually, the total population and reliable population size should converge throughout an experiment to a single value.

$$P_{\text{reliable}} = \{cl \in P : cl.q \geq \theta_r\} \tag{2.4}$$

2.3.2 Generalization

The generalization metric [16, 18, 21, 22] reflect the agent's generalization abilities. It is calculated by analyzing all classifiers in a population $[P]$. Equation 2.5 shows the proportion of condition attributes containing the wildcard symbol.

$$\text{Generalization}(P) = \frac{|\bigcup \Big(cond.att \| cond \in \big(cl.cond \| cl \in P\big) \wedge cond.att = \# \Big)|}{|\bigcup \Big(cond.att \| cond \in \big(cl.cond \| cl \in P\big) \Big)|} \tag{2.5}$$

The greater generalization score means that the algorithm creates classifiers covering a larger portion of the observation space, creating a more compact solution of the problem.

2.3.3 Knowledge

The knowledge KPI was proposed, which allows analyzing all possible consequences of actions in all available states of a particular environment, giving a comprehensive picture of the state of the agent's internal environment model [114]. The metric is a ratio of reliable classifiers that can predict each *state-action-state'* movement and the total number of possible transitions.

Due to the possible complexity of increased observation space, not all environments can calculate this metric. Moreover, the agent itself obviously does not have access to the internal state of the environment.

2.3.4 Trial time

The trial time is simply a time needed to complete one agent's trial phase - either exploration or exploitation. This metric is used to infer the computation complexity of the examined algorithms and environments.

The computations are performed in a controlled environment on a single machine in all cases. Therefore, the obtained results can be relatively compared to each other.

2.3.5 Exploitation performance

The exploitation performance is a measure of utilizing the agent's internal model of the environment in order to obtain the highest possible payoff [118]. It is most often gathered in the exploit phase, where the agent focuses solely on selecting the best possible classifiers for each situation.

Depending on the type of environment, this metric might differ - for example, in a multistep *Corridor*, the interest is in reaching the final state as fast as possible. Therefore, the metric is simply the number of steps in each trial that should be close to the optimal value. For a single-step environment, such as *rMPX*, a reward is paid out after giving a correct answer in each trial. In this case, a good measure is to keep track of latest obtained rewards and average them.

The metric itself depends on significant components (learning and credit assignment) involved in the agent's algorithm to estimate its practical usefulness.

2.4 Statistical verification of results

In order to assess the significance and performance of obtained results, two statistical approaches were used. The majority of metrics were compared using the BEST (Bayesian estimation) method [72], while the other straightforward metrics were just averaged.

2.4.1 Bayesian analysis

The Bayesian approach towards comparing data from multiple groups was used instead of traditional methods of NHST (*null hypothesis significance testing*). They are more intuitive than the calculation and interpretation of *p-value* scores, provides complete information about credible parameter values and allow more coherent inferences from data [34].

Benavoli depicted the perils of the frequentist NHST approach when comparing machine learning classifiers in [6], which is particularly suited for this work. He points out the following reasons against using the NHST methods:

- it does not estimate the probability of hypotheses,
- point-wise null hypotheses are practically always false,
- the *p-value* does not separate between the effect size and the sample size,
- it ignores magnitude and uncertainty,
- it yields no information about the null hypothesis,
- there is no principled way to decide the α level.

Additionally, in 2016 the *American Statistical Association* made a statement against *p-values* [130] which might be a motivation for other disciplines to pursue the Bayesian approach.

In this work, we focus on establishing a descriptive mathematical model of the data D using the Bayes' theorem deriving the posterior probability as a consequence of two antecedents: a prior probability and a "likelihood function" derived from a statistical model for the observed data (Equation 2.6).

$$\underbrace{p(\mu, \sigma, \nu \mid D)}_{\text{posterior}} = \underbrace{p(D \mid \mu, \sigma, \nu)}_{\text{likelihood}} \times \underbrace{p(\mu, \sigma, \nu)}_{\text{prior}} \Big/ \underbrace{p(D)}_{\text{evidence}} \tag{2.6}$$

Each experiment is performed 50 times, generating independent samples, which according to the Central Limit Theorem, should be enough to consider it is approximating the normal distribution [62, 73]. To further provide a robust solution towards dealing with potential outliers, the *Student t-distribution* is chosen. The prior distribution, described with three parameters - μ (expected mean value), σ (standard deviation) and ν (degrees of freedom) is presented using the Equation 2.7. 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 $\nu \sim \text{Exp}(\frac{1}{29})$ allows the model to be more tolerant for potential outliers.

$$\begin{aligned} \mu &\sim \text{N}(\mu_D, \sigma_D^2) \\ \sigma &\sim \text{U}(\frac{1}{100}, 1000) \\ \nu &\sim \text{Exp}(\frac{1}{29}) \end{aligned} \tag{2.7}$$

The posterior distribution is approximated arbitrarily high accuracy by generating a large representative sample using MCMC (*Markov chain Monte Carlo*) methods. Its sample provides thousands of combinations of parameter values $< \mu, \sigma, \nu >$. Each such combination of values is representative of credible parameter values that simultaneously accommodate the observed data and the prior distribution. From the MCMC sample, one can infer credible parameter values like the mean or standard deviation.

2.4.2 Example

To show an example of this technique, the performance of an ACS2 algorithm operating in a multistep, toy-problem - Simple Maze environment [37] (see Figure 2.8) will be discussed in terms of Hypothesis H_0 using three methods - the summary statistics, frequentist approach and the Bayesian estimation.

Hypothesis H_0 (Null hypothesis)**:** The classifier population sizes obtained by ACS2 and ACS2 GA in the last trial x are equal.

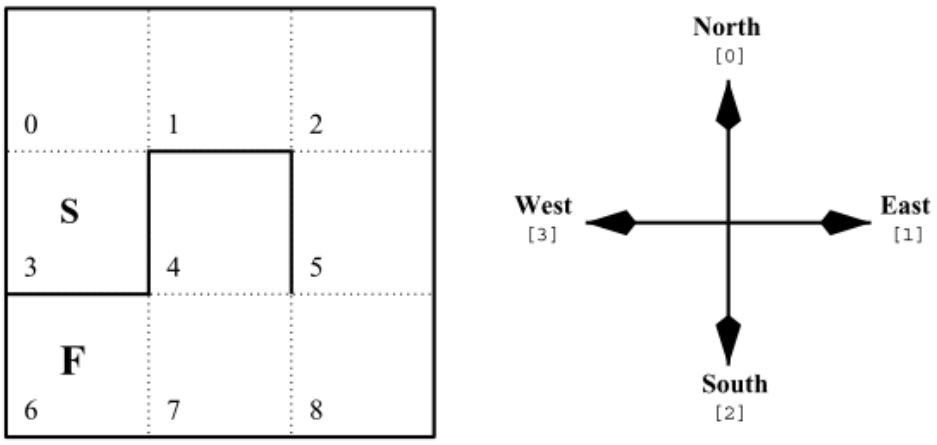

Figure 2.8: *The Simple Maze environment is a Partially Observable Markov Decision Problem where the agent is placed in the starting location (denoted as "S"), at the beginning of each new trial, and the goal is to reach the final state "F" by executing four possible actions – moving north, east, south or west. The bolded lines represent the walls, and the goal can be reached optimally in seven successive steps.*

The ACS2 agent will be tested in two variants - with and without the genetic algorithm modification. All other settings are identical. Both agents in each experiment will execute 500 trials, each time randomly selecting an action. Finally, each such experiment will be independently repeated 50 times.

2.4.2.1 Descriptive statistics

The first option is to understand the data and extract basic summary statistics. Figure 2.9 presents a whisker plot showing basic data aggregations like the minimum, first quartile, median, third quartile, maximum values alongside the histogram visualization. While the values look similar, it is still unclear whether they can be considered the same.

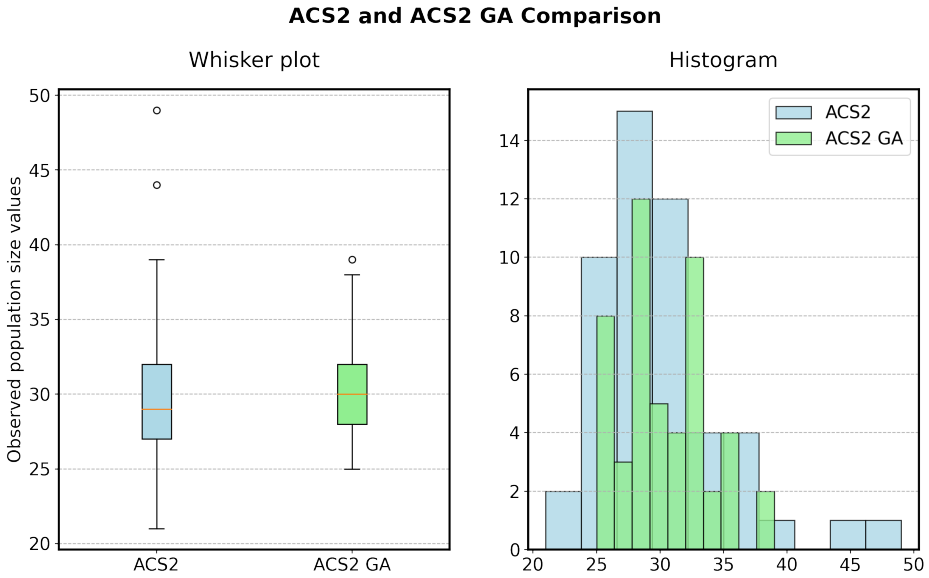

Figure 2.9: *Descriptive statistics depicting population size obtained after executing two versions of the ACS2 agent in the Simple Maze environment.*

2.4.2.2 Frequentist approach

For the frequentist approach first two hypotheses about data distribution are formed:

Hypothesis H_0: The classifier population sizes obtained by ACS2 and ACS2 GA are equal.

Hypothesis H_1: The classifier population sizes obtained by ACS2 and ACS2 GA are different.

In traditional NHST workflow, the first step would be to apply normality tests to verify whether the data is normally distributed. However, looking at the histograms from the Figure 2.9 we might assume that the data follow the Gaussian distribution and skip it.

If the H_0 hypothesis is rejected, there is no significant difference between the means. To do so, a *p-value* will be calculated and compared with a certain threshold $\alpha \leq 0.05$. If the *p-value* falls below the threshold, it means that the null hypothesis H_0 can be rejected and *there is 95% confidence that both means are significantly different.*

In this case, after running the two-sided t-Test, the calculated *p-value* is 0.68, which indicates strong evidence for the H_0 hypothesis, meaning that it is retained. Remember that NHST does not accept the null hypothesis; it can be only rejected or failed to be rejected. Stating that it is correct implies 100% certainty, which is not valid in this methodology.

2.4.2.3 Bayesian estimation

Throughout this work a PyMC3[1] open-sourced Python framework is used for the probabilistic programming. Figure 2.10 depicts two separate Student-t posterior distribution hyper-parameters estimated by using the MCMC method.

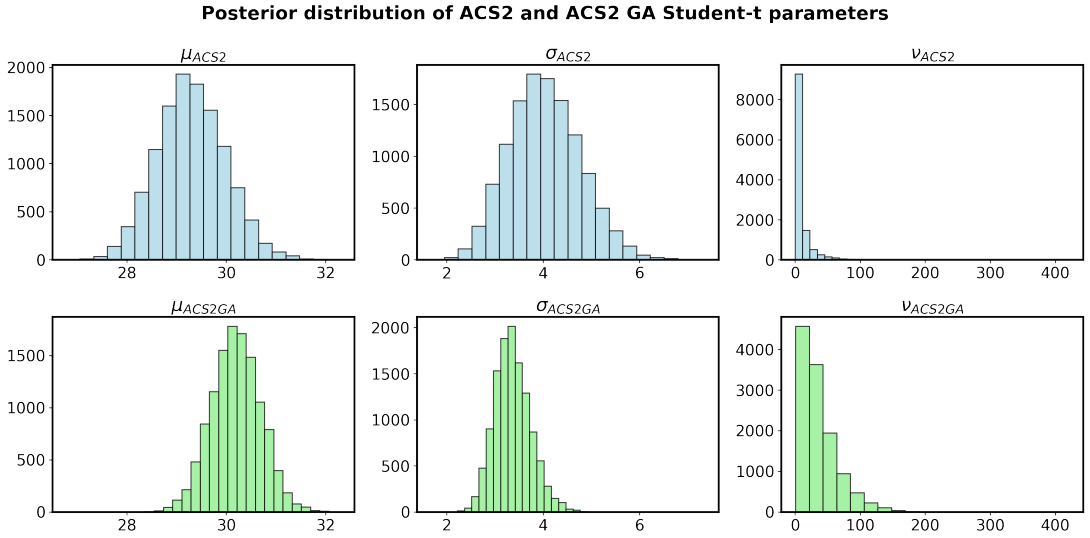

Figure 2.10: *Parameter distributions of Student-t estimation of population size data for two agents.*

The posterior distribution allows assessing the null value more credibly. Contrary to NHST, it can also accept H_0. To do so, a researcher defines a ROPE (*region of practical equivalence*) [71], enclosing the values of the parameter that are deemed to be negligibly different from the null values for practical purposes. When nearly all the credible values fall within the ROPE, the null value is said to be accepted for practical purposes.

Figure 2.11 shows the graphical example of verifying the null hypothesis using the difference between two posterior distributions of population means $\mu_{ACS2} - \mu_{ACS2GA}$. The ROPE region (red sector) was assumed to be $[-1, 1]$, which means that the difference of one classifier is acceptable to consider two populations equal. Blue lines represent the area of credible values of the previously calculated difference. Since the credible value region does not fall into the ROPE region, the H_0 is rejected (while the NHST approach retained it).

[1] https://docs.pymc.io/en/v3

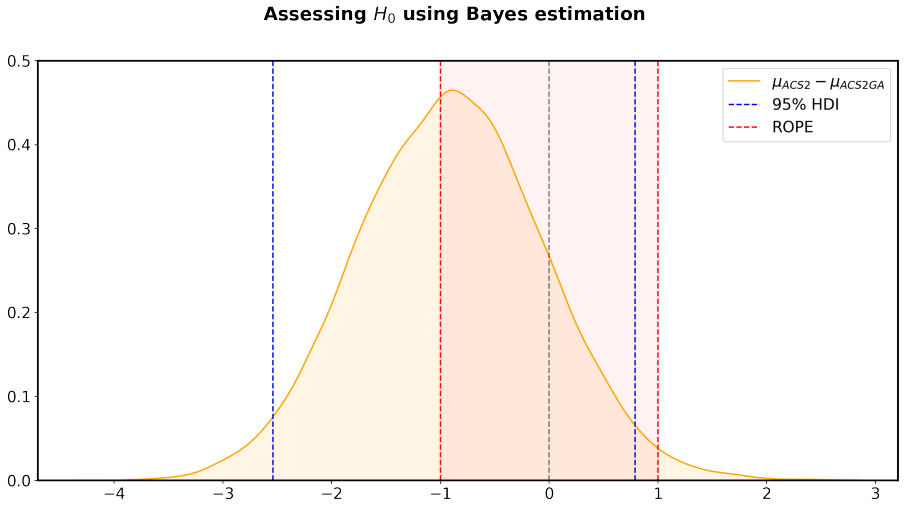

Figure 2.11: *Assessing H_0 using the probabilistic density function of μ parameter difference of two examined populations. The ROPE interval was chosen to $[-1, 1]$ meaning that a difference of one classifier can be neglected in terms of population equality. Even though the mean difference is less than one, the H_0 is rejected because the HDI region is not contained within the ROPE.*

2.5 Overview of the selected environments

The range of problem domains to which ALCS can be broadly divided into two categories: classification problems and RL problems [20]. Classification problems seek to find a compact set of rules that classify all problem instances with maximal accuracy. They frequently rely on supervised learning, where feedback is provided instantly. The RL problems seek to find an optimal behavioural policy represented by a compact set of rules. These problems are typically distinguished by inconsistent environmental rewards, requiring multiple actions before a reward is obtained. They can be further discriminated by having Markov properties [19, 104, 108].

This section describes six single- and multi-steps environments used in further experiments. All of them are stationary, Markov toy-problems exhibiting real-valued properties facilitating the evaluation of modifications of anticipatory systems. Most of the environments were used as benchmarks in other works related to measuring the performance of the LCS; however, the Cart Pole environment was never used in this class of algorithms before.

All the mentioned environments pose the standardized interface for enabling the agent to interact with it in a consistent manner [66]. After executing an action, the agent is presented with the current observation (that is not equal to its internal state), possible

reward from previously executed action, and whether the interaction is finished. In most cases, a specific reward state acts as an incentive, motivating the forager to reach it.

2.5.1 Corridor

The corridor is a 1D multi-step, linear environment introduced by Lanzi to evaluate the XCSF agent [77]. 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 2.12.

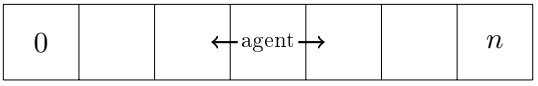

Figure 2.12: *The Corridor environment. The size of the input space is n.*

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

The environment examined herein signifies the state already in discretized form, available in three preconfigured sizes with increasing difficulty. The main challenge for the agent here is mainly to learn the reward distribution in possibly long action chains successfully.

Reward scheme: The trial ends when it reaches the final state n (obtaining reward $r = 1000$) or when the maximum number of 200 steps in each episode is exceeded. Otherwise, the reward after each step is $r = 0$.

2.5.2 Grid

Grid refers to an extension of the Corridor environment [77]. A vertical dimension and two new actions (move up, move down) are added. 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 - see Figure 2.13.

Reward scheme: The trial ends with reward $r = 1000$ when the final state (n, n) is reached. Otherwise, the reward after each step is $r = 0$. Additionally, the episode is terminated after exceeding the maximum number of 2000 steps.

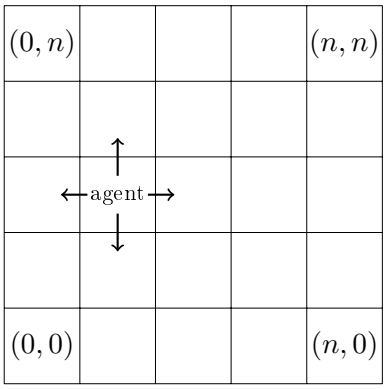

Figure 2.13: *The Grid environment. The size of the input space is n^2. The agent can change its position by moving in four different directions.*

2.5.3 Real Multiplexer

Wilson modified the traditional MPX (Boolean Multiplexer) [136], called rMPX (Real Multiplexer), to examine the performance in single-step environments using real-valued data.

The Boolean *n–bit* multiplexer defines a set of single-step supervised learning problems conceptually based on an electronic device taking multiple inputs and switching them to the single output. A random, fixed binary string of predefined length is generated in each trial. It comprises two parts - the address and register segments. The first one points to a specific register address that is considered to be a truth value - Figure 2.14 describes the process. This environment is also exciting because it possesses the properties of *epistasis* (describes non-linear interaction effect between multiple features) and *heterogeneity* (for different sets of instances, a distinct subset of features will determine the class value), which are often present within real-world problems such as bioinformatics, finance or behaviour modelling.

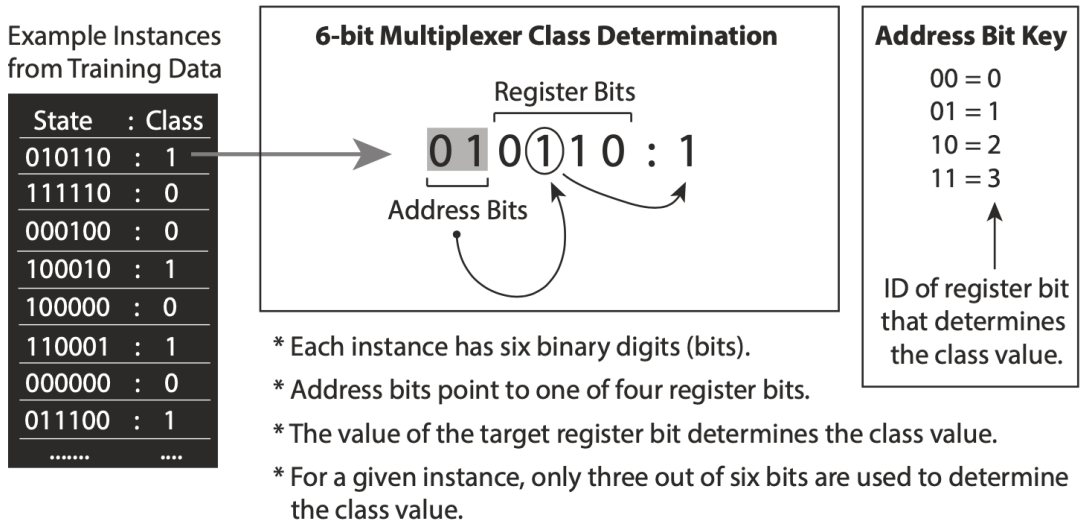

Figure 2.14: *Visualization of determining the output value of 6-bit multiplexer. The address bit points to the first value of the register array that is considered the true output. Diagram taken from [128].*

For the rMPX, the only difference between a 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. See Table 2.3 for the example of such mapping.

rMPX output	0.94	0.07	0.15	0.11	0.33	0.77	0.0
MPX output	1	0	0	0	0	1	0

Table 2.3: *Example of mapping a random 6-bit rMPX output into MPX problem. A threshold of* $\theta = 0.5$ *was used. The last bit is set to* 0 *as an initial value that will be changed to* 1 *after performing the correct action.*

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 [18]. In this work, the assumption is that the state generated by the rMPX is extended by one extra bit, denoting whether the classification was successful. This bit is by default set to zero. When the agent responds correctly, it is being switched, thus providing direct feedback. A detailed example can be found in [67].

Reward scheme: If the correct answer is given, the reward $r = 1000$ is paid out, otherwise $r = 0$.

2.5.4 Checkerboard

The Checkerboard is a single-step environment introduced by Stone in [117]. It was proposed to circumvent certain limitations of the rMPX when using the interval predicates approach. Because the rMPX problem can be solved by using just a hyperplane decision surface, it is not considered to represent arbitrary intervals in solution space. In order to solve the Checkerboard problem, a hyper-rectangle decision surface is needed for modelling certain interval regions.

This environment works by dividing the n-dimensional solution space into equal-sized hypercubes. Each hypercube is assigned a white or black color (alternating in all dimensions), see Figure 2.15. The problem difficulty can be controlled by changing the dimensionality n and the number of divisions in each dimension n_d. In order to allow the colours to be alternating, n_d must be an odd number.

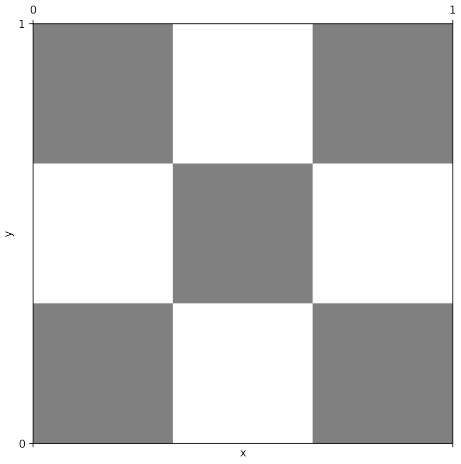

Figure 2.15: *2-dimensional Checkerboard problem with $n_d = 3$. Precise interval predicates are needed for representing selected regions.*

In each trial, the environment presents a vector of length n_d or real numbers in the interval $[0, 1)$, representing a point in the solution space. The agent's goal is to guess the correct colour by performing one of two actions depending on a pointed hypercube's colour (black or white).

Reward scheme: When the correct answer is given, the reward $r = 1$ is paid out, otherwise $r = 0$.

2.5.5 Cart Pole

Barto introduced the *Cart Pole*[2] environment as a RL control problem [5]. The task is to balance a pole that is hinged to a movable cart by applying forces (move left or move right) to the cart's base (Figure 2.16). The system starts upright, and its goal is to prevent the stick from falling over.

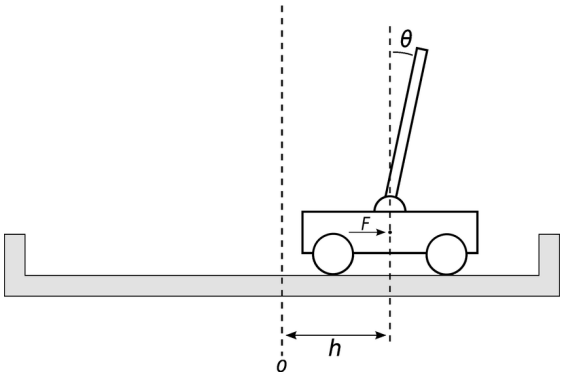

Figure 2.16: *The Cart Pole environment. The goal is to maintain the pole upright for a maximum number of trials. Figure taken from [85].*

The observation returned by the environment is a vector of four elements - presented in Table 2.4. The challenge for the agent is that each attribute has a specific range of possible values, and two of them additionally span the whole search space.

Attribute	**Observation**	**Min**	**Max**
σ_0	Cart Position	-2.4	2.4
σ_1	Cart Velocity	$-\infty$	∞
σ_2	Pole Angle	∼-41.8°	∼41.8°
σ_3	Pole Velocity at Tip	$-\infty$	∞

Table 2.4: *Agent's observation of the Cart Pole environment.*

The environment is considered solved if the average reward is greater than or equal to 195 over the last 100 trials.

Reward scheme: After each step, a reward of +1 is provided. The episode ends when the pole is more than 15 degrees from vertical or the cart moves more than 2.4 units from the centre.

[2]https://gym.openai.com/envs/CartPole-v0

2.5.6 Finite State World

Barry introduced the FSW (*Finite State World*) [4] environment to investigate the limits of XCS performance in long multi-step environments with a delayed reward. It consists of *nodes* and directed *edges* joining the nodes. Each node represents a distinct environmental state and is labelled with a unique state identifier. Each edge represents a possible transition path from one node to another and is also labelled with the action(s) that will cause the movement. An edge can also lead back to the same node. The graph layout used in the experiments is presented in Figure 2.17.

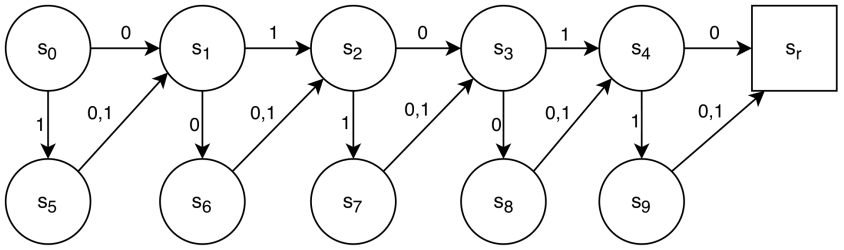

Figure 2.17: *A Finite State World of length 5 (FSW-5). The environment is especially suited for measuring reward propagation for long action chains. Each trial always starts in state s_0, and the agent's goal is to reach the final state s_r.*

Although the environment does not expose any real-valued state, it can be treated as a further extension of a discretized Corridor. Most importantly, the challenge is that the agent is presented with the sub-optimal route at every step, which slows for the pursuit of the reward. Additionally, the environment is easily scalable - changing the number of nodes will change the total action chain length.

Reward scheme: A of reward $r = 100$ is provided when agent reaches final state s_r, otherwise $r = 0$.

Chapter 3

Internalizing knowledge with increased input-space

3.1 Interval-based representation

There are several possible ways of representing intervals as described in Section 2.2. This chapter deliberately selects the UBR approach introduced by Stone and Bull for XCS [117], since it supersedes both CSR and OBR encodings and tries to apply it within the ALCS algorithm.

However, any agent cannot be adapted to UBR just by changing the communication layer with the environment. Therefore, the ACS2 algorithm was elected as a representative candidate of the ALCS family due to its maturity and the considerable number of tightly interacting components.

Moreover, to limit the flourishing population size, a hybrid approach was taken to represent intervals, where each boundary is represented using a nominal integer value within a certain range. It is assumed that the input attribute perception σ_i is defined as $\sigma_i \in [0, 1]$. Table 3.1 demonstrates an example of encoding real value input using particular encoding resolutions. Obviously, low encoding values introduce ambiguity, and this parameter must be chosen carefully.

The proprietary variation of the ACS2 system was named rvACS [67] and can be distinguished from the native implementation by:

- **Don't care symbol** - In rvACS the feature attributes consist solely of interval ranges. The *"don't care"* and *"pass-through"* wildcard symbols are represented as a full-ranged interval (e.g. using 4 bit encoding - UBR$(0, 15)$ or UBR$(15, 0)$).

Perception	1-bit	2-bit	3-bit	4-bit	5-bit	6-bit	7-bit
0.0	0	0	0	0	0	0	0
0.1	0	0	0	1	3	6	12
0.2	0	0	1	3	6	12	25
0.3	0	1	2	4	9	19	38
0.4	0	1	3	6	12	25	51
0.5	1	2	4	8	16	32	64
0.6	1	2	4	9	19	38	76
0.7	1	2	5	11	22	44	89
0.8	1	3	6	12	25	51	102
0.9	1	3	7	14	28	57	115
1.0	1	3	7	15	31	63	127

Table 3.1: *Examples of encoded values for different perceptions σ values. 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).*

- **Covering** - The covering process introduces randomness when a new classifier is added to the 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 $\text{UBR}(x_1 - \epsilon, x_2 + \epsilon)$ 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).

The general workflow of a trial behavioral act is exactly like in the ACS2 (see Algorithm 6). However, several refined procedures are listed below.

Perception encoding - Algorithm 9

Every time a real-valued environmental perception $\sigma \in (0, 1)$ is perceived, it is being optionally disturbed and discretized uniformly using selected resolution.

Algorithm 9 Perception encoding in rvACS

```
function ENCODE(σ, noise)
    step ← 1/2^bits
    σ ← σ + noise                          ▷ Disturb perception signal
    if σ < 0 ∨ σ > 1 then
        Trim σ to range (0, 1)
    for i = 0 to 2^bits do
        x1 ← i · step
        x2 ← (i + 1) · step
        if σ ∈ (x1, x2) then
            return i
    end for
end function
```

Perception matching operator - Algorithm 10

The perception matching investigates whether conditional UBR intervals incorporate relevant encoded perception attributes.

Algorithm 10 Condition matching in rvACS

```
function DOES MATCH(condition, σ)
    σ' ← ENCODE(σ, 0.0)
    for i = 0 to ‖condition‖ do
        (x1, x2) ← condition[i]
        if σ'[i] ∉ (x1, x2) then
            return False
    end for
    return True
end function
```

Classifier specialization - Algorithm 11

The ALP process utilizes previous σ_{-1} and current perception σ_{-1} for adjusting condition and effect parts of the classifier examined. To ensure robustness, a random noise signal with maximal value of ϵ_{cover}, might be added.

Algorithm 11 ALP classifier specialization rvACS

function SPECIALIZE(cl, σ_{-1}, σ)
  $\sigma' \leftarrow$ `ENCODE`$(\sigma, 0.0)$
  $\sigma'_{-1} \leftarrow$ `ENCODE`$(\sigma_{-1}, 0.0)$
  for $i = 0$ to $\|$cl.condition$\|$ **do**
    if $\sigma'[i] \neq \sigma'_{-1}[i]$ **then**
      $\epsilon \leftarrow \mathrm{U}(0, \epsilon_{cover})$
      cl.condition$[i] \leftarrow$ `UBR`(`ENCODE`$(\sigma_{-1}, -\epsilon)$, `ENCODE`(σ_{-1}, ϵ))
      cl.effect$[i] \leftarrow$ `UBR`(`ENCODE`$(\sigma, -\epsilon)$, `ENCODE`(σ, ϵ))
  end for
end function

Assure anticipation correctness - Algorithm 12

The `DOES ANTICIPATE CORRECTLY` function checks if the classifier correctly anticipated the consequences of performed action.

Algorithm 12 Classifier anticipation correctness in rvACS

function DOES ANTICIPATE CORRECTLY(cl, σ_{-1}, σ)
  $\sigma' \leftarrow$ `ENCODE`$(\sigma, 0.0)$
  $\sigma'_{-1} \leftarrow$ `ENCODE`$(\sigma_{-1}, 0.0)$
  for $i = 0$ to $\|$cl.effect$\|$ **do**
    $e_{ubr} \leftarrow$ cl.effect$[i]$
    if $e_{ubr} =$ WILDCARD **then**
      if $\sigma'[i] \neq \sigma'_{-1}[i]$ **then**
        return False
    else
      if $\sigma'[i] = \sigma'_{-1}[i]$ **then**
        return False
      if $\sigma'[i] \notin e_{ubr}$ **then**
        return False
  end for
  return True
end function

Subsumption - Algorithm 13

Two condition parts of the examined classifiers are compared to distinguish whether the second one is incorporated within the first.

Algorithm 13 Condition subsumption in rvACS

function SUBSUMES($cond_1, cond_2$)
  for $i = 0$ to $\|cond_1\|$ **do**
    if $cond_1[i] \not\subset cond_2[i]$ **then return** False
  end for
  return True
end function

Classifier mark in rvACS

In traditional ACS2 implementation, the classifier's *mark* property stores a set of attributes for which the classifier anticipated wrongly. This information was further used to enhance the new classifier during the creation process. In the rvACS, where each conditional attribute is represented using the interval predicate, the *mark* still uses the same representation but maintains a list of specific nominal values within each interval that provided wrong predictions. This representation is sufficient to indicate problems with a particular interval.

Mutation in rvACS - Algorithms 14, 15

The mutation operator tries to alter the classifier's applicability by affecting the interval ranges. Each non-general interval occurring in both condition and effect parts have μ chances of being changed. The `DISTURB` process randomly creates a noise using range defined by the $\epsilon_{mutation}$ parameter and adds it to decoded perception. Both $\epsilon_{mutation}$ and encoding resolution must be chosen carefully because improper values might diminish the effects of this operator - further encoding will not capture the difference introduced by the noise.

Algorithm 14 Mutation operator in rvACS

function MUTATE($cl, \mu, \epsilon_{mutation}$)
  for $i = 0$ to $\|cl.condition\|$ **do**
    $c_{ubr} \leftarrow cl.condition[i]$
    $e_{ubr} \leftarrow cl.effect[i]$
    if $c_{ubr} \neq$ WILDCARD $\wedge e_{ubr} \neq$ WILDCARD **then**
      `DISTURB` ($c_{ubr}, \mu, \epsilon_{mutation}$)
      `DISTURB` ($e_{ubr}, \mu, \epsilon_{mutation}$)
  end for
end function

Algorithm 15 Interval attribute generalization in rvACS

function DISTURB($i_{ubr}, \mu, \epsilon_{mutation}$)
  if Random Number $< \mu$ **then**
    noise $\leftarrow$ U($-\epsilon_{mutation}, \epsilon_{mutation}$)
    $x_0 \leftarrow$ Decode first UBR boundary into real-valued number using encoder
    $i_{ubr}[0] \leftarrow$ `ENCODE`(x_0, noise)
  if Random Number $< \mu$ **then**
    noise $\leftarrow$ U($-\epsilon_{mutation}, \epsilon_{mutation}$)
    $x_1 \leftarrow$ Decode second UBR boundary into real-valued number using encoder
    $i_{ubr}[1] \leftarrow$ `ENCODE`(x_1, noise)
end function

3.1.1 Research questions

The conducted research aims to answer the following question regarding the rvACS algorithm and the usage of interval-based representation

Q1. Can the rvACS algorithm form the internal model of the environment and exploit it successfully?

3.1.2 Experimental evaluation

This section presents setup of the performed experiments and their results.

Goals of the experiments

Experiment 1 - Encoding precision

The impact of using different numbers of bits for creating UBR ranges will be contrasted with the ability to exploit the single-step rMPX environment. Due to the environment's increasing complexity, a simple 3-bit variant is sufficient to demonstrate the potential pitfalls of the proposed interval-based representation

Experiment 2 - Nature of the intervals

The main goal of the experiment is to investigate the nature and the evolution of condition intervals. An experiment using the Checkerboard environment will highlight significant differences between conditional attributes, but also the overall performance in this environment will be analyzed.

3.1.2.1 Experiment 1 - Encoding precision

The impact of setting the encoder bits value when forming interval predicates was validated on the single-step rMPX environment. Due to potential computational complexity issues, the problem was scaled down to 3bit rMPX, where the perception vector is represented by four attributes (the first three bits are both the address and register, and the last one indicates whether the given answer was correct).

The impact of four different encoder bits values was examined by collecting metrics, including representative metrics of the average reward in each execution phase and the size of the evolved population.

In each trial of the experiment, the agent alternates between explore and exploit phases for the total of 20000 trials, which allows a discerning shift in performance over time. Moreover, to present coherent results and draw statistical inferences, each experiment is repeated 50 times and those independent runs are averaged.

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

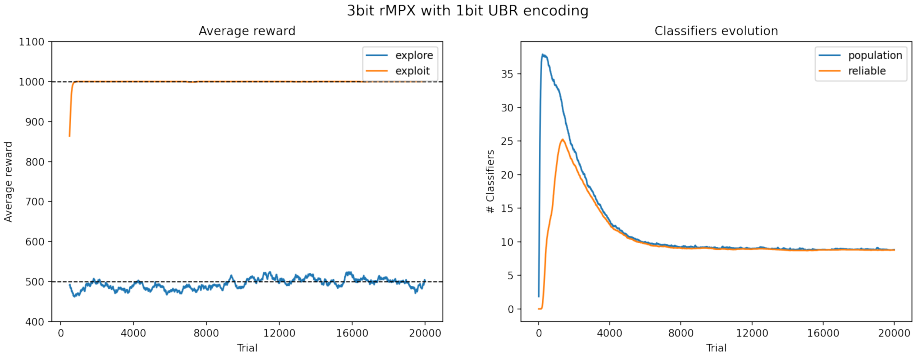

Figure 3.1: *Performance in 3bit rMPX UBR with 1 bit. The reward is averaged across 50 last runs.*

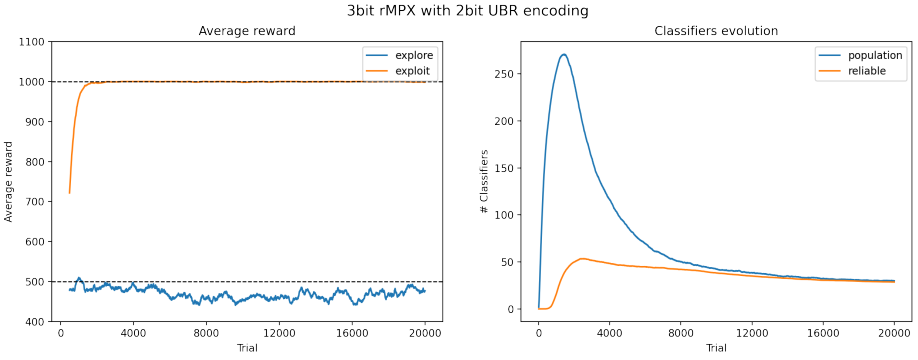

Figure 3.2: *Performance in 3bit rMPX UBR with 2 bits. The reward is averaged across 50 last runs.*

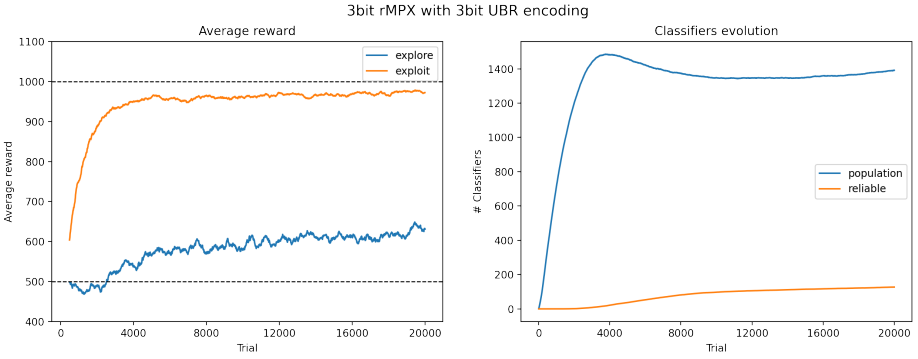

Figure 3.3: *Performance in 3bit rMPX UBR with 3 bits. The reward is averaged across 50 last runs.*

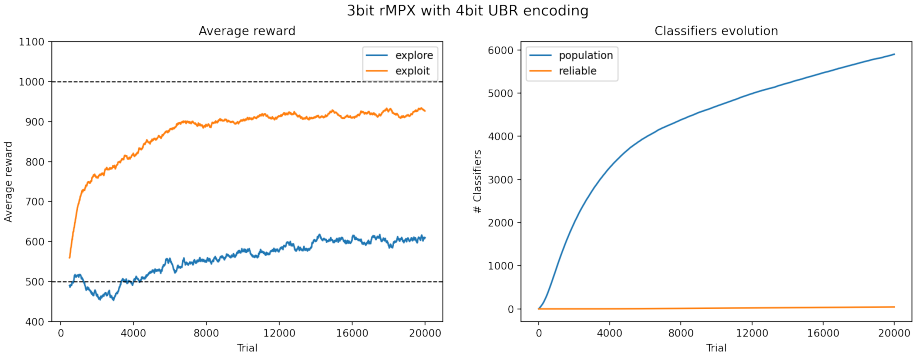

Figure 3.4: *Performance in 3bit rMPX UBR with 4 bits. The reward is averaged across 50 last runs.*

Statistical verification

To statistically assess the population size, the posterior data distribution was modelled using 50 metric values collected in the last trial and then sampled with 100,000 draws. - Table 3.2. The average value from the last 100 exploit trials is considered a representative state of algorithm performance for the obtained reward.

	1 bit	**2 bit**	**3 bit**	**4 bit**
population of classifiers	8.8 ± 0.13	29.58 ± 0.71	1392.25 ± 25.68	5892.82 ± 86.8
reliable classifiers	8.75 ± 0.12	28.47 ± 0.54	127.08 ± 2.33	44.35 ± 1.6
reward from last 100 exploit runs	1000.0	998.6	974.6	927.6

Table 3.2: *Descriptive statistic metrics obtained using Bayesian estimation of obtained results for rMPX problem.*

Observations

The rvACS algorithm implementation proved to be capable of solving 3bit rMPX problem with the hyper-plane decision surface. The performance of four distinct values of encoding resolutions are depicted in Figures 3.1, 3.2, 3.3 and 3.4

For this particular environment, the threshold mapping each real-value attribute into binary representation was set to 0.5; therefore, satisfying performance was obtained with very rough encoding values. The agent consistently exploited the environment with a single encoding bit by choosing the correct answer and converging the population size.

Raising the encoding bits should lead to similar results but slower. Due to the expansion of possible interval combinations, the learning component needs more time to evolve proper rules. This progression is observed in Figure 3.2 where 2 encoding bits are used. Similar performance is observed, but the final population size is almost three times larger due to the lack of rule compaction ability.

The problems are emphasized when using 3 and 4 encoding bits. The average reward in the exploit phase is still significantly greater than in explore one, meaning that the agent can pick up correct decisions most of the time, but the convergence of population size is becoming elusive. The algorithm is struggling with increasing population size, which, due to its nature, is a significant computational bottleneck.

3.1.2.2 Experiment 2 - Nature of the intervals

In order to provide the correct answer to the checkerboard problem, the agent must be able to correctly partition the hyper-rectangular solution space. Four categories are used to determine the nature of the evolved condition intervals:

- 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"*).

The two-dimensional checkerboard divided by three splits in each direction is used. The experiment uses the rvACS agent and evaluates its performance with different encoding values. Because of the splits, the system response is dependent on precise boundaries estimations. Figure 3.5 shows possible ambiguities near the split lines, where the same nominal value is applicable in both regions.

As in the previous section, in each trial of the experiment, the agent alternates between explore and exploit phases for the total of 15000 trials. Each independent pass is averaged across 50 times. For the collected metrics, besides the average performance and the population size, the proportion of condition interval regions is collected for each trial.

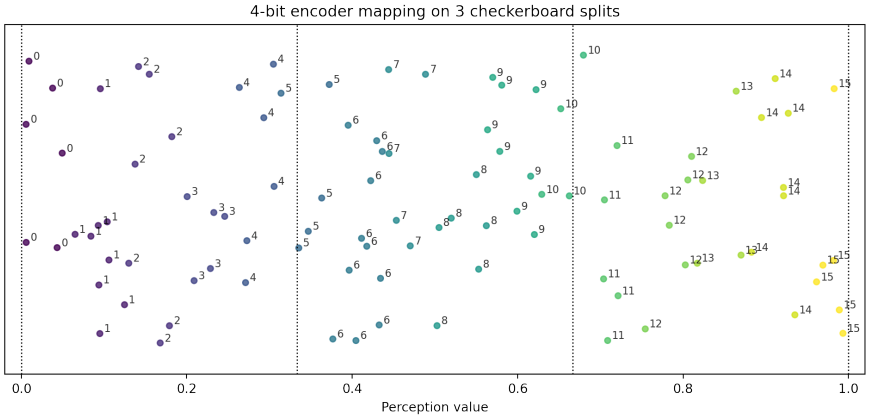

Figure 3.5: *Example of dividing the space into three equal splits. When using low encoding resolution potential ambiguity is visible near the splitting lines.*

Figures 3.6, 3.7 and 3.8 illustrate the metrics progression on the 3x3 Checkerboard problem using 4 bit interval encoding. Due to brevity the plots for other encoding values were not presented, but final values are outlined using statistical estimation.

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

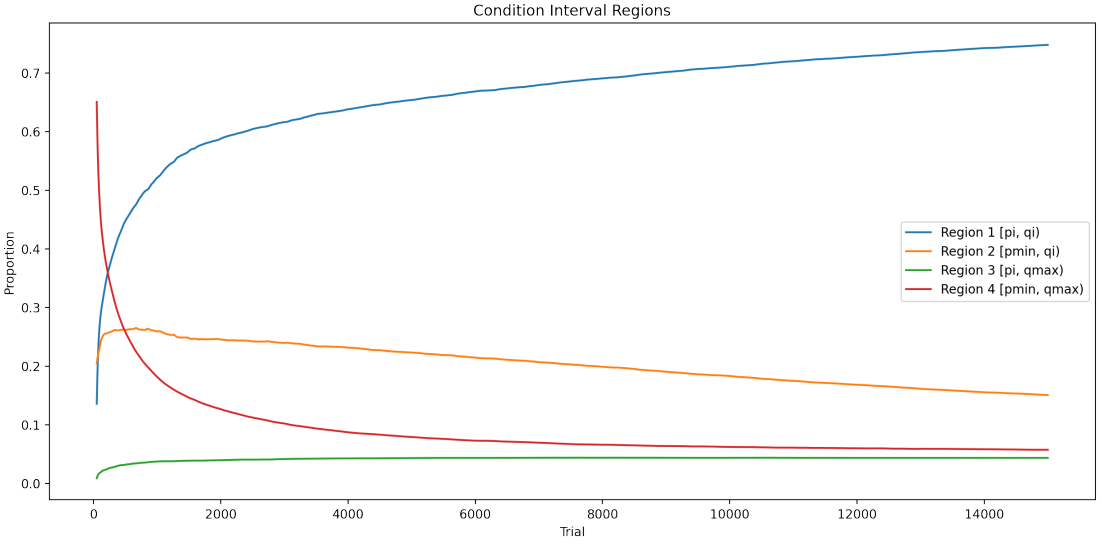

Figure 3.6: *Evolution of condition interval regions in 3x3 Checkerboard environment encoded with 4 bits.*

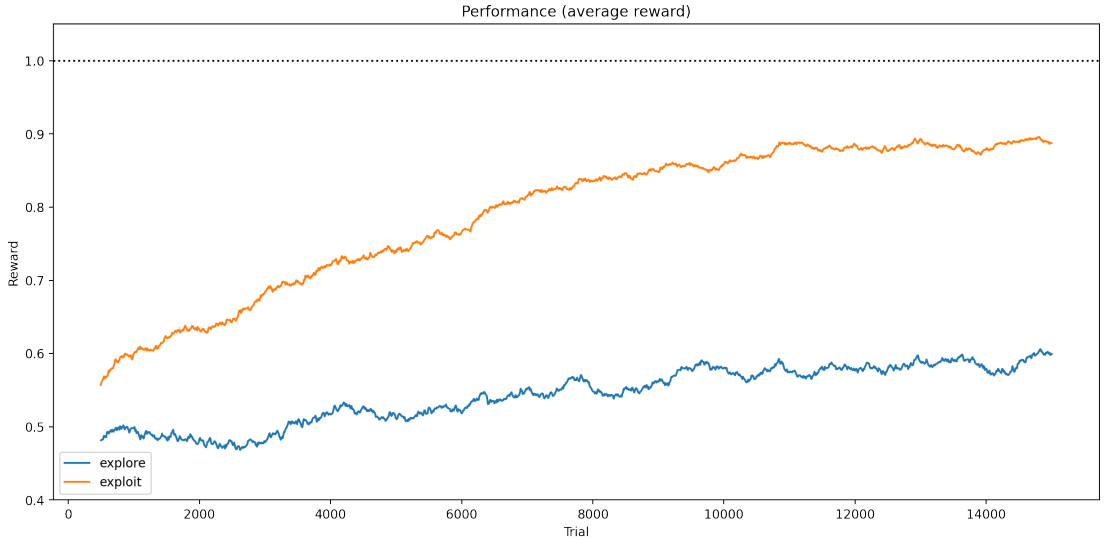

Figure 3.7: *Average reward obtained in 3x3 Checkerboard environment encoded with 4 bits.*

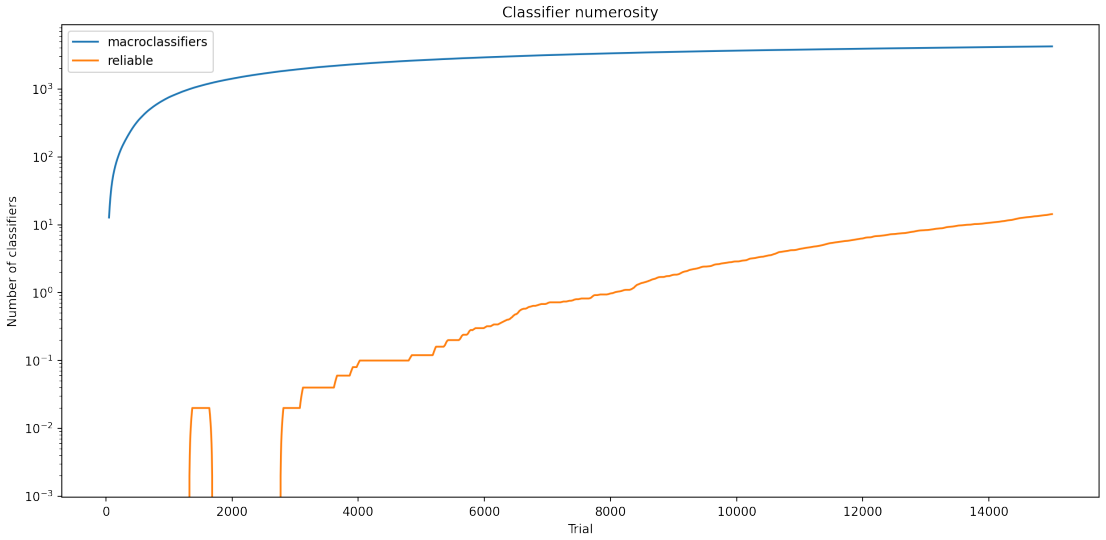

Figure 3.8: *Population size of 3x3 Checkerboard environment encoded with 4 bits (notice the logarithmic scalling of y-axis).*

Statistical verification

To statistically assess the population size and region ratios, the posterior data distribution was modelled using 50 metric values collected in the last trial and then sampled with 100,000 draws. - Table 3.3. For the obtained reward, the mean value from the last 100 exploit trials is considered as a representative state of algorithm performance.

	1 bit	2 bit	3 bit	4 bit	5 bit
Region 1	0.0 ± 0.0	0.32 ± 0.0	0.69 ± 0.0	0.75 ± 0.0	0.71 ± 0.0
Region 2	0.46 ± 0.0	0.35 ± 0.0	0.18 ± 0.0	0.15 ± 0.0	0.2 ± 0.0
Region 3	0.26 ± 0.01	0.14 ± 0.0	0.08 ± 0.0	0.04 ± 0.0	0.02 ± 0.0
Region 4	0.27 ± 0.01	0.19 ± 0.0	0.05 ± 0.0	0.06 ± 0.0	0.08 ± 0.0
population of classifiers	20.22 ± 0.38	52.63 ± 0.57	735.64 ± 8.67	4244.18 ± 35.75	10358.61 ± 47.0
reliable classifiers	-0.0 ± 0.0	8.0 ± 0.0	89.51 ± 1.17	14.42 ± 0.6	0.64 ± 0.16
reward from last 100 exploit runs	0.52	0.71	0.81	0.89	0.78

Table 3.3: *Descriptive statistic metrics obtained using Bayesian estimation of obtained results for 3x3 Checkerboard problem.*

Observations

In most experiments, the rvACS agent builds the population consisting primarily of Region 1 interval predicates. The amount of attributes represented as Region 3 and 4, spanning to the maximum value from the right side, tends to diminish. However, the results are correlated with the number of trials that were kept the same in all cases. More precise boundary representation naturally would require more trials to converge. However, for the first four bits, there is the following trend can be noticed when intensifying encoding resolution:

- Ratio of Region 1 attributes increases,
- Ratio of Region 2, 3, 4 attributes decreases.

This is caused by the lack of online rule compaction or consolidation mechanism. The only possibility for the agent to create a more general attribute is due to the mutation algorithm controlled by the $\epsilon_{mutation}$ parameter. However, this value must be set carefully because limitations of selected encoding resolution can inadvertently ignore its effect.

The other metrics also show the hypothesis about the need for more trials. The size of the overall population is correlated with the number of encoding bits, but it became more difficult for the agent to discriminate between reliable classifiers. For example, when using 5-bits encoding after 150000 trials, there is no single reliable classifier despite having a population with more than 10 thousand individuals.

The experiment with 1-bit encoding also confirms the situation when it is impossible to learn the environment with hyper-plane decision boundary successfully. Such representation is insufficient to handle regularities, resulting in unreliable classifiers and random average rewards from exploit runs.

3.1.3 Research summary

The answers to the previously formulated research questions are as follows:

Q1: Can the rvACS algorithm form the internal model of the environment and exploit it successfully?

The performed experiments involved two single-step problems with varying difficulties. The rvACS implementation showed promising results but revealed certain limitations were revealed in both of them.

Rule compaction

Due to the nature of LCS systems, most of the components interact sequentially, frequently iterating over the population of classifiers. rvACS agents showed to be biased towards creating rules capturing specific small niches of the solution space, therefore significantly increasing population size and making it more indecipherable. The enhancement merging rules affecting neighbouring areas of solution space would have a heavy impact on the solution's computational performance and compactness.

Encoding resolution

The proposed approach represented interval boundaries as integer values. While this decision has a positive impact mainly on the simplification of internal operators, there are certain drawbacks. First and foremost, there is a tradeoff between selected resolution and the size of the population. In all performed experiments, situations with high resolution yield the most significant internal models containing very specific classifiers. Second, knowledge about environment internal regularities (like the rRMPX threshold θ described in Section 2.5.3) is beneficial for determining optimal encoding value because some of the regularities might not be captured due to perceptional ambiguity (see Table 3.1).

Effect part

The majority of rvACS modifications focus on building the correct *condition* part, ignoring the evolution of the *effect* side. The ALP component responsible for driving the agent's learning mechanism was not intended for real-valued representation. Moreover, the problem with the growing population is also related to the creation of duplicated and overlapping classifiers where the aforementioned processes cannot perform subsumption and merging of classifiers with similar rule structure while favouring more general classifiers. Therefore, in rvACS, its usage was simplified, indicating a need for a dedicated solution.

3.2 Discretizing input signal

The previous chapter proved that ALCS could operate in a real-valued realm using UBR interval representation. Due to the underlying foundations, certain limitations were revealed, requiring major rethinking to hold virtues like balancing compact population size while maintaining maximum accuracy and generalization.

Nevertheless, ALCS are not limited to ternary alphabets - they can be arbitrarily extended to any discrete number of symbols. This provides an opportunity to split the range of perception value into k fixed-sized intervals (*buckets*). Each consecutive symbol represents a successive interval by performing such discretization. The most significant advantage is that no modifications to internal components are needed for any of the investigated systems - just an additional layer between the environment-agent interface. The interface is capable of executing any predefined input transformation, like assigning each observation attribute using the same logic or having dedicated rules for each of them (in a case when a range of values varies significantly). The difference between approach suggested by Unold and Mianowski [126] is the distinct separation of discretization component that can be applied independently in any of the ALCS algorithms.

The algorithm 16 presents the extended trial flow with discretization process. Each experiment uses an external *discretizer* d, capable of transforming raw environmental signal $\sigma \rightarrow \Sigma$ using desired attribute resolution. All investigated ALCS algorithms share a similar behavioural act cycle; therefore, it is possible to use a shared implementation of an external discretizer in each of the investigated systems in a standardized approach.

Algorithm 16 Proposition of single LCS trial using external discretizer

procedure RUN TRIAL(P, d)
  $t \leftarrow 0$
  $\sigma \leftarrow$ *env:* perceive situation
  $\Sigma \leftarrow$ `DISCRETIZE`(σ, d)
  while trial is finished **do**
    if $[A]_{-1}$ is not empty **then**
      `APPLY DISCOVERY COMPONENT` considering Σ, Σ_{-1}, a
      `APPLY LEARNING COMPONENT` in $[A]_{-1}$ considering ρ
    `GENERATE MATCH SET` $[M]$ out of $[P]$ using Σ
    $a \leftarrow$ `CHOOSE ACTION` using $[M]$
    `GENERATE ACTION SET` $[A]$ according to $[M]$ using a
    $\Sigma_{-1} \leftarrow \Sigma$
    $(\sigma, \rho) \leftarrow$ *env*: execute action a ▷ Obtain next state and reward
    $\Sigma \leftarrow$ `DISCRETIZE`(σ, d)
    $t \leftarrow t + 1$
    $[A]_{-1} \leftarrow [A]$
  end while
  `APPLY DISCOVERY COMPONENT` considering Σ, Σ_{-1}, a
  `APPLY LEARNING COMPONENT` in $[A]$ considering ρ
end procedure

This chapter aims to contrast the latent learning capabilities of ACS, ACS2, YACS systems in both single- and multistep environments. Furthermore, they were compared to basic RL algorithm competent of building internal environmental model and lookahead planning - Dyna-Q [118].

Dyna architecture

RL algorithms like Q-Learning cannot perform operations considered "cognitive", such as reasoning and planning (because they do not learn an internal model of the environment's dynamics). Dyna architecture is an online planning agent with an internal environmental model.

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.

3.2.1 Research questions

The conducted research aims to answer the following questions about bucketing discretization:

Q1. Can popular ALCS systems build the internal model of the environment when discretizing the real-valued input into fixed-width buckets?

Q2. Which system creates the most compact and general population of classifiers?

Q3. What is the relative trial execution time for each evaluated system?

Q4. How selected systems relate to the benchmark Dyna-Q algorithm?

3.2.2 Experimental evaluation

This section presents setup of the performed experiments and their results.

Goals of the experiments

Experiment 3 - Single-step environment performance
This experiments aim to determine the agents' latent learning competencies using the rMPX discretized into a fixed amount of buckets.

Experiment 4 - Multiple-step environments performance
Corresponding comparison is performed for two multistep problems with similar regularities - Corridor and Grid.

3.2.2.1 Experiment 3 - Single-step environment performance

The potential of evolving the internal model of the environment is explored by three ALCS implementations - ACS, ACS2 and YACS. Moreover, two additional enhancements are also explicitly investigated - ACS2 with enabled GA component and traditional Dyna-Q implementation. All the algorithms use similar rule structure and their behaviour can be compared using a set of common metrics of population size, knowledge, generalization and trial time.

The chosen environment is a single-step, 3-bit rMPX. The investigation was not carried out for greater perception vectors due to the increased computational complexity required for handling a greater amount of interacting classifiers and calculating the metrics, however with sufficient resources, similar results can also be obtained for larger problem instances.

The agent perceives an observation vector of four real-valued attributes. Then, a k bins discretizer is used to convert them into integers, where the k value controls the accuracy of generated rules. Thus, the input space of the environment can be calculated as $2k^n$, where k refers to the number of bins and n is the length of the multiplexer output.

Because the rMPX threshold was set to the value of 0.5, a setting of $k = 2$ bins should be sufficient to model the hyper-plane decision surface. The tests were, however, performed by discretizing each attribute into 10 fixed-sized buckets. The following setting provides excessive-resolution and substantiates the capabilities of real-valued input handling.

In each trial of the experiment, the agent executes the exploration phase for the total of 15000 trials solely. Moreover, to present coherent results and draw statistical inferences, each experiment is repeated 50 times - see Figure 3.9.

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. The Dyna-Q algorithm performs five steps ahead simulation in each trial.

	Knowledge	Generalization	Population	Trial time
ACS	0.0 ± 0.0	0.875 ± 0.001	4.0 ± 0.001	0.001 ± 0.001
ACS2	0.998 ± 0.001	0.174 ± 0.001	1622.405 ± 7.313	0.007 ± 0.001
ACS2 GA	1.0 ± 0.001	0.318 ± 0.001	977.517 ± 5.966	0.006 ± 0.001
YACS	0.978 ± 0.001	0.106 ± 0.005	1160.178 ± 6.192	0.03 ± 0.001
Dyna-Q	0.998 ± 0.001	0.0 ± 0.0	2000.0 ± 0.001	0.0 ± 0.0

Table 3.4: *Descriptive statistic metrics obtained using Bayesian estimation of obtained results for discretized rMPX problem.*

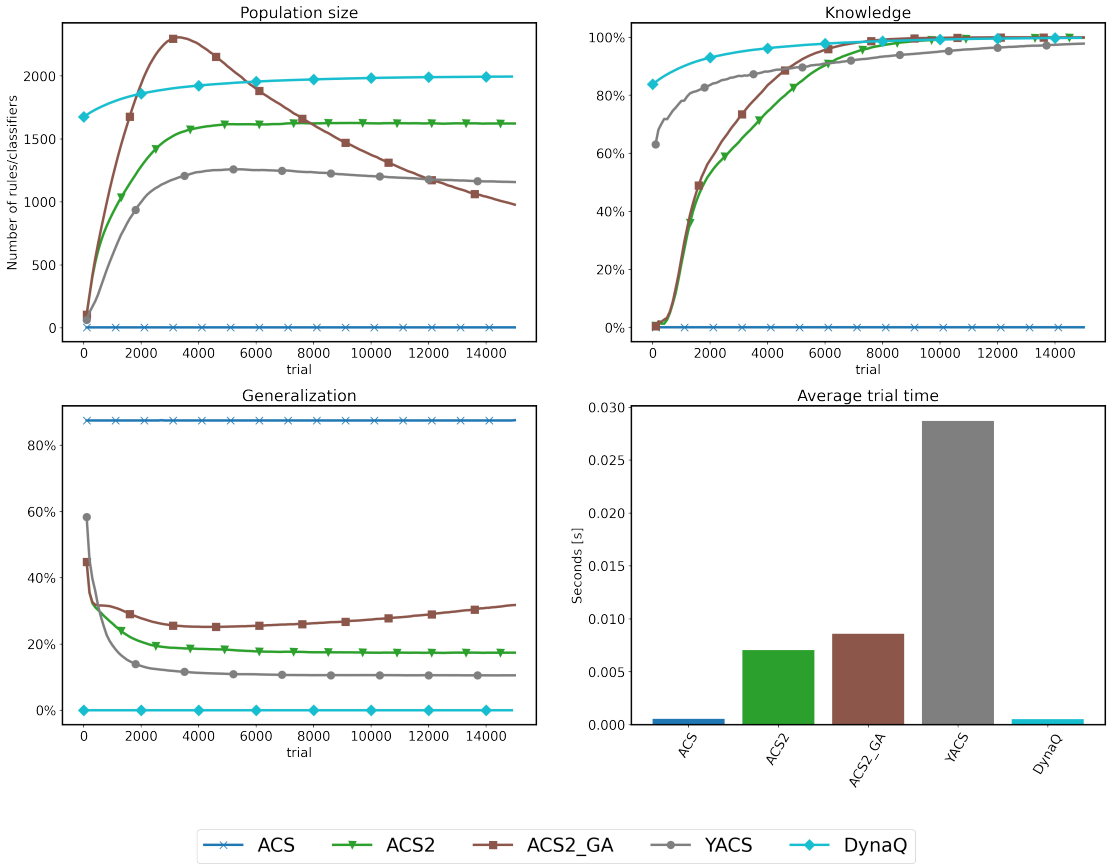

Figure 3.9: *Performance of 3bit rMPX discretized with 10 bins. Metric collected every 100 trials and averaged across 50 independent runs.*

Statistical verification

The posterior data distribution was modelled using 50 metric values collected in the last trial and then sampled with 100,000 draws. The obtained results were presented in Table 3.4 and using the radar plot in Figure 3.10. The radar plot axes were scaled accordingly, highlighting the relative differences between algorithms.

Figure 3.10: *Normalized metrics presented on the radar plot for the 3bit rMPX environment.*

Observations

The examined problem input space size is $2 \cdot 10^3 = 2000$ individual states. Even though using the excessive-resolution size, almost every algorithm built the internal model of the environment in an imposed number of trials.

ACS

Only the ACS agent could not go beyond four semi-general classifiers in the whole experiment, being incapable of learning any of the transitions. Therefore, the generalization and trial-time metrics cannot be directly compared to other agents due to the wrong population size.

ACS2

The successor of ACS - ACS2 ultimately learned the environment using significantly fewer classifiers than the problem input size. The GA extension has a positive effect on compacting the population size and increasing generalization.

YACS

The heuristic nature of YACS allows it to gain knowledge at the beginning of the experiment rapidly. However, the lack of generalization capabilities provides the worst generalization compared to both ACS2 variants. Moreover, the trial execution time is significantly larger than in other systems because all visited states are represented and processed inside the agent's memory.

Dyna-Q

By design, the Dyna-Q algorithm does not expose any generalization capabilities - therefore is unable to learn latently with more compacted knowledge representation. Due to the lookahead capabilities, the system rapidly learned all available transitions while operating faster than competitors.

3.2.2.2 Experiment 4 - Multiple-step environments performance

The multistep learning performance was also examined by a set of five algorithms - ACS, ACS2, ACS2 with GA, YACS and Dyna-Q using the same metrics as in the single-step experiment case.

On the contrary, the problems investigated herein does not provide immediate feedback to the agent about the potential outcomes of the selected action. Therefore, a chain of correct decisions needs to be formed to locate the incentive. The Corridor is a one-dimensional grid discretized into 20 states, and Grid provides an extension by adding another dimension of the same length alongside two more possible actions.

In each trial of the experiment, the agent executes the exploration phase for the total of 300 trials solely. Moreover, to present coherent results and draw statistical inferences, each experiment is repeated 50 times.

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. The Dyna-Q algorithm performs five steps ahead simulation in each trial. Figures 3.11 and 3.12 present the metrics evolution over time for Corridor and Grid environments respectively.

Statistical verification

The posterior data distribution was modelled using 50 metric values collected in the last

trial and then sampled with 100,000 draws. Obtained results were presented in Tables 3.5 and 3.6 and using the radar plots scaled accordingly (Figures 3.13, 3.14), highlighting the relative differences between algorithms.

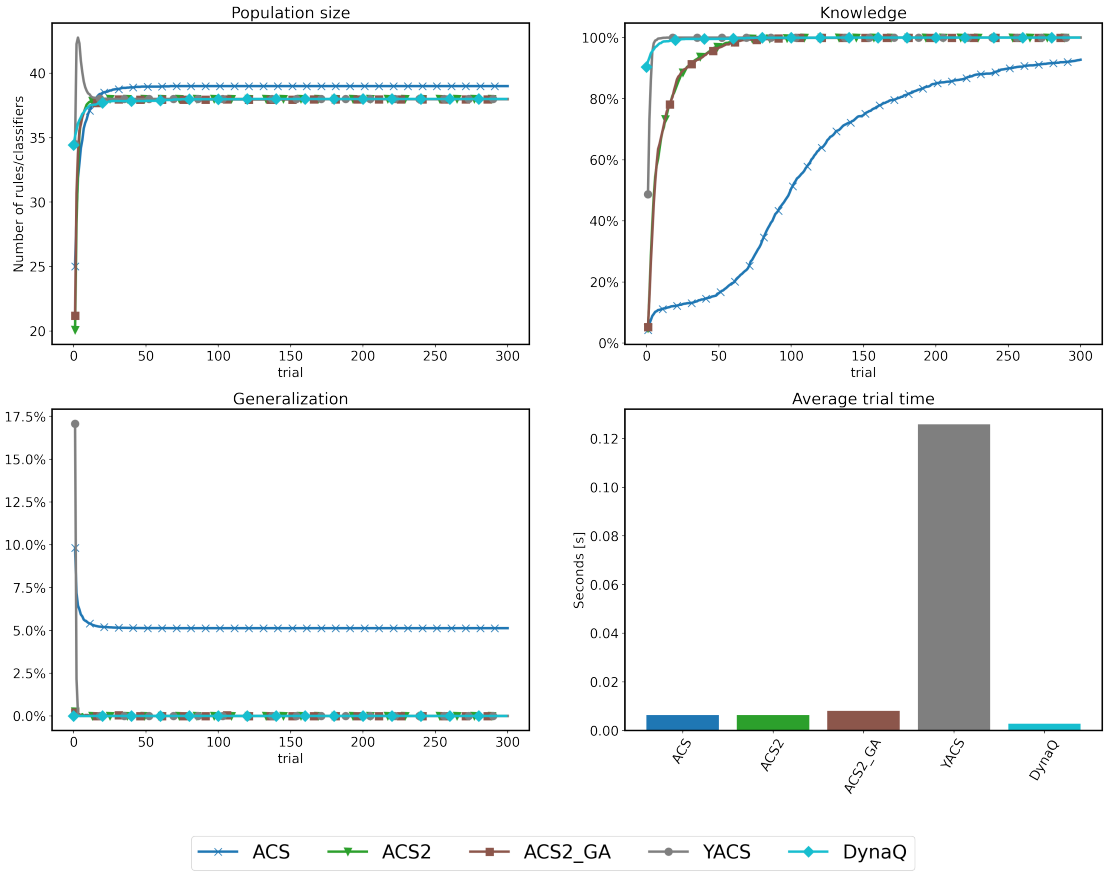

Figure 3.11: *Performance of discretized Corridor-20 environment. Experiments were executed 50 times and averaged.*

	Knowledge	Generalization	Population	Trial time
ACS	0.93 ± 0.009	0.051 ± 0.0	39.0 ± 0.0	0.007 ± 0.001
ACS2	1.0 ± 0.0	0.0 ± 0.0	38.0 ± 0.0	0.006 ± 0.001
ACS2 GA	1.0 ± 0.0	0.0 ± 0.0	38.0 ± 0.001	0.008 ± 0.001
YACS	1.0 ± 0.0	0.0 ± 0.0	38.0 ± 0.0	0.104 ± 0.012
Dyna-Q	1.0 ± 0.0	0.0 ± 0.0	38.0 ± 0.0	0.002 ± 0.001

Table 3.5: *Descriptive statistic metrics obtained using Bayesian estimation of obtained results for discretized Corridor-20 problem.*

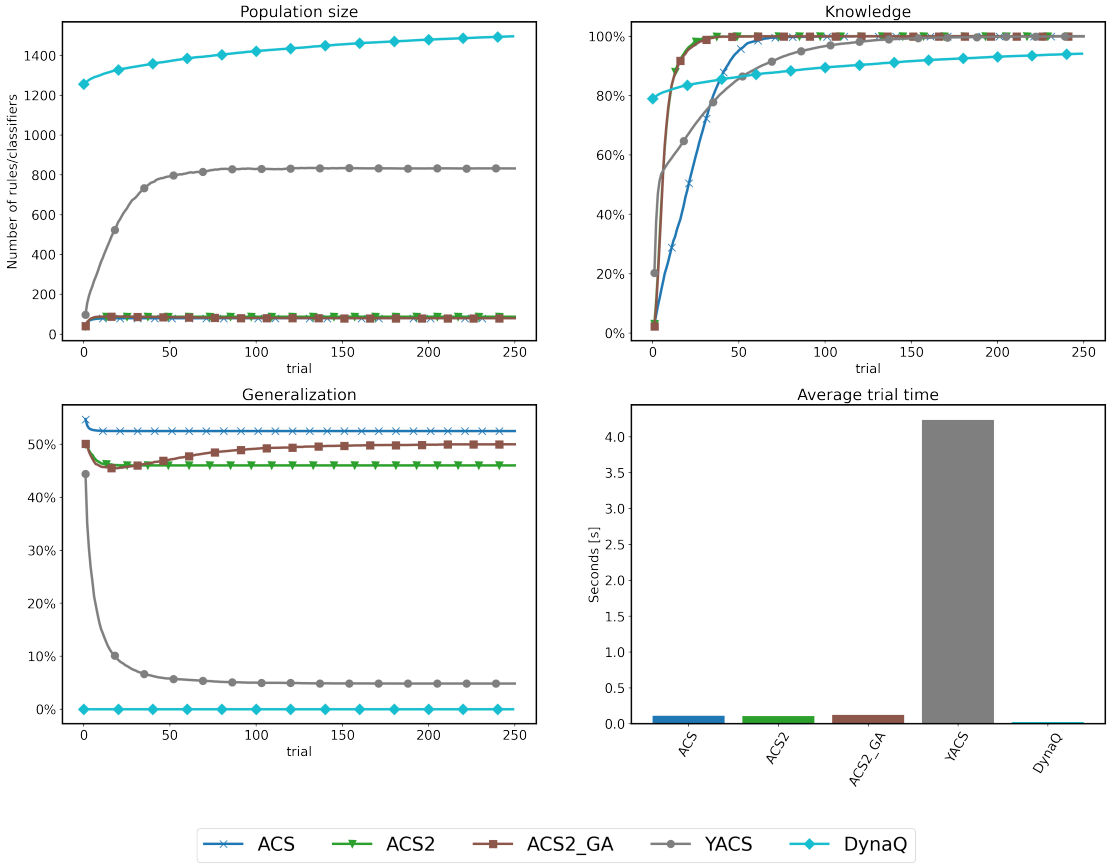

Figure 3.12: *Performance of discretized Grid-20 environment with 400 distinct states. Experiments were executed 50 times and averaged.*

	Knowledge	Generalization	Population	Trial time
ACS	1.0 ± 0.0	0.525 ± 0.0	80.0 ± 0.0	0.117 ± 0.002
ACS2	1.0 ± 0.0	0.461 ± 0.005	87.049 ± 0.954	0.115 ± 0.004
ACS2 GA	1.0 ± 0.0	0.5 ± 0.0	80.0 ± 0.0	0.126 ± 0.005
YACS	1.0 ± 0.001	0.022 ± 0.002	830.655 ± 50.379	4.299 ± 0.33
Dyna-Q	0.978 ± 0.008	0.0 ± 0.0	1551.849 ± 16.427	0.025 ± 0.003

Table 3.6: *Descriptive statistic metrics obtained using Bayesian estimation of obtained results for discretized Grid-20 problem.*

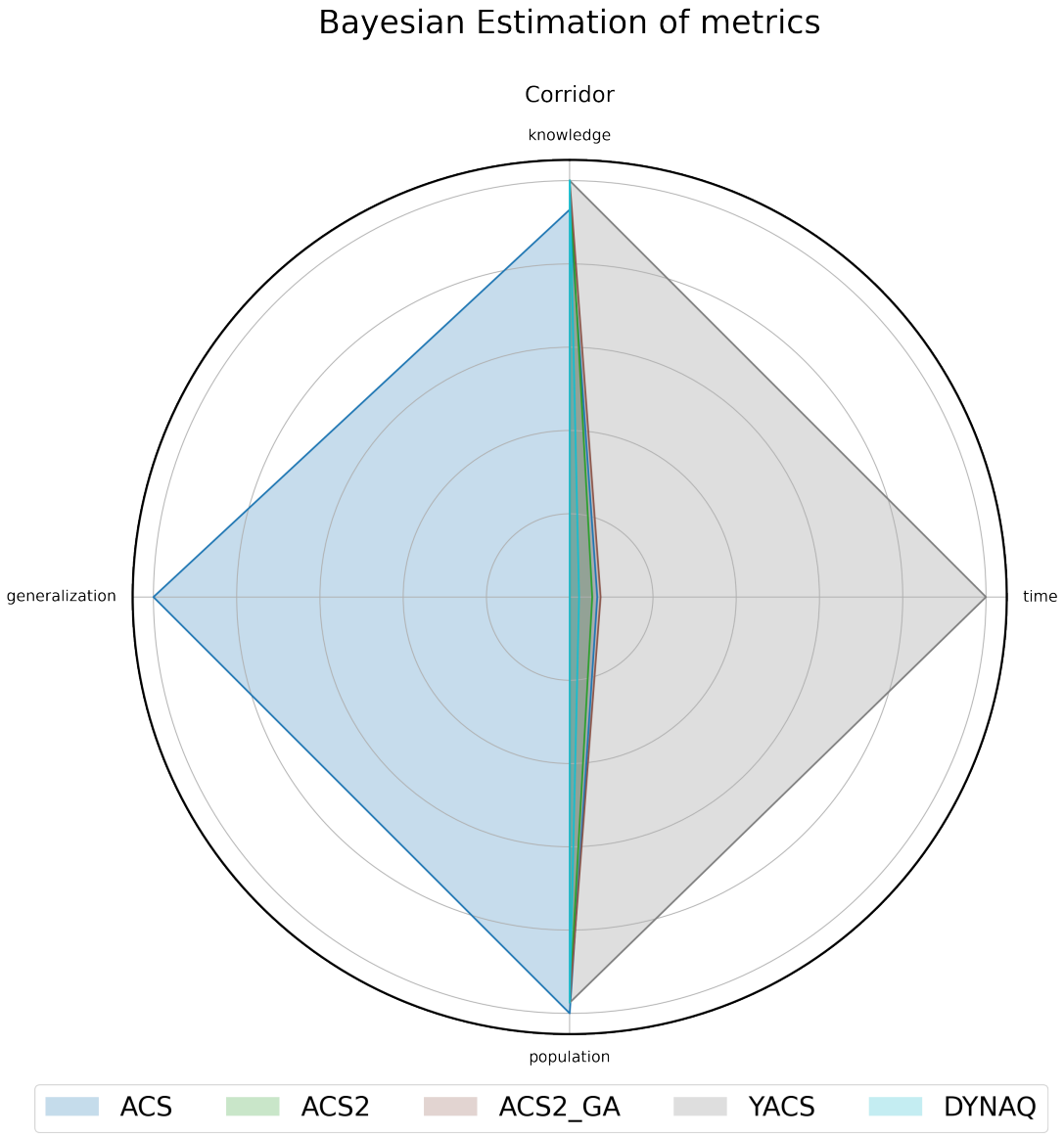

Figure 3.13: *Normalized metrics presented on the radar plot for Corridor-20 environment.*

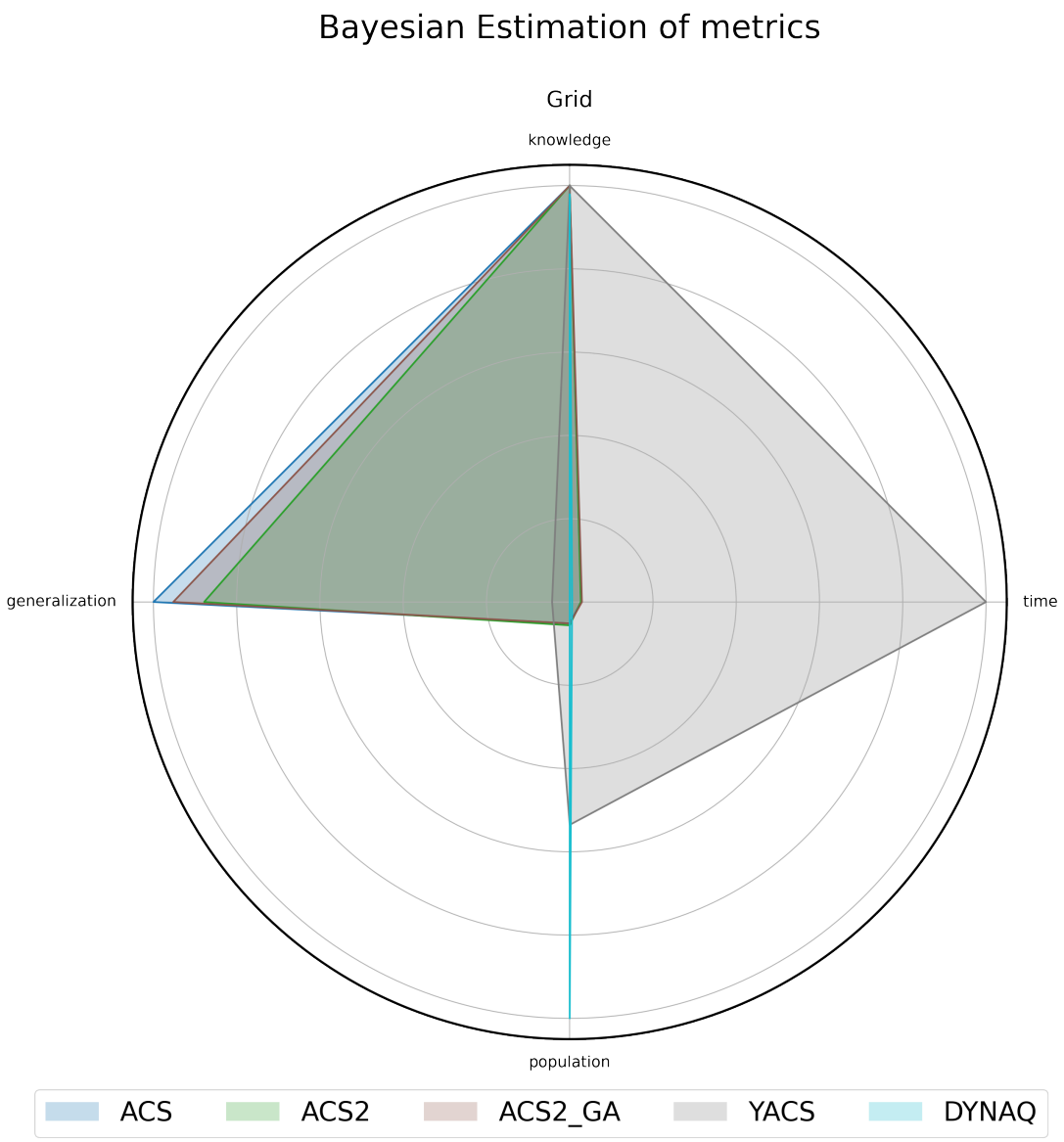

Figure 3.14: *Normalized metrics presented on the radar plot for Grid-20 environment.*

Observations

All investigated algorithms converge towards obtaining complete knowledge of selected problems. Interesting behavioural patterns are revealed despite the relatively small number of input-space.

ACS

For the simple Corridor environment, the ACS maintains a stable population with only one irrelevant classifier, therefore wrongly suggesting generalization capabilities. The agent has the slowest learning rate.

Surprisingly, the Grid environment managed to have the smallest population of classifiers with the highest generalization score outperforming other agents.

ACS	**ACS2**	**YACS**
$\#\# \leftarrow \#\#$ $18\# \leftarrow 17\#$	$18\# \leftarrow 17\#$	$18\# \leftarrow 17\#$
$\#\# \rightarrow \#\#$ $18\# \rightarrow 19\#$	$18\# \rightarrow 19\#$	$1819 \rightarrow \#\#$ $1819 \rightarrow 19\#$
$\#\# \uparrow \#\#$	$\#19 \uparrow \#\#$	$\#19 \uparrow \#\#$
$\#\# \downarrow \#\#$ $\#19 \downarrow \#18$	$\#19 \downarrow \#18$	$18\# \downarrow \#5$ $18\# \downarrow \#4$

Table 3.7: *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 is unable to create fully general and accurate classifiers at all.*

ACS2

Since the lack of generalization capabilities in the Corridor problem, the performance of ACS2 and ACS2 GA is identical in this environment. They modelled the environment internally using a minimal possible number of rules. In the Grid environment, the GA addition further reduced the population size by extending the applicability of rules covering greater environmental niches.

YACS

The YACS, on average, took the longest amount of time to compute each trial. The Corridor case started very rapidly by generating an overpopulation of classifiers (being the fastest of learning the whole environment) and eventually settled into optimal values. However, it could not form the correct population size for the Grid problem. While knowing the consequences of all actions, the under-performing generalization resulted in an excessive number of classifiers.

The visual comparison of rules created for certain environmental perception in the Grid environment is shown in Table 3.7.

Dyna-Q

Dyna-Q stands out in computation time, being the fastest investigated algorithm. However, complete representation of all possible state-action transitions needs to explicitly process each environmental interaction, which might be difficult with potential noise or other disturbances. Therefore, the knowledge accumulation process was significantly slower compared to other algorithms.

3.2.3 Research summary

The answers to the previously formulated research questions are as follows:

Q1: Can popular ALCS systems build the internal model of the environment when discretizing the real-valued input into fixed-width buckets?
All investigated ALCS systems with common rule structure (ACS, ACS2, YACS) managed to deal with real-valued input represented as a vector of nominal values. The corresponding changes affected only the interface layer, leaving agent mechanisms intact. Moreover, the nature of the *don't care* and *pass-through* symbol was fully preserved which additionally increased model interpretability.

Q2: Which system creates the most compact and general population of classifiers?
ACS2 with GA enhancement proved to evolve the most general and compact population of classifiers diligently. However, specific problems were discovered in ACS, which was unable to progress by creating novel classifiers or YACS that suffers from insufficient heuristics and lacks a dedicated generalization mechanism.

Q3: What is the relative trial execution time for each evaluated system?
Due to its simplicity, Dyna-Q was the fastest algorithm in all comparisons. However, for the ALCS family, the ACS and ACS2 are much faster than YACS, mainly due to YACS processes where each visited state is stored internally despite generalization. Those states are processed in each trial, which for large problem spaces might squander computational resources.

Q4: How selected systems relates to the benchmark Dyna-Q algorithm?
The Dyna-Q is a traditional benchmarking RL problem capable of representing the consequences of certain actions in particular states. The most obvious difference is the lack of generalization capabilities, which forces the Dyna-Q to model all encountered transitions explicitly. This leads to a larger internal model size and potentially slower formation of an optimal policy. However, on the other side, a lack of sophisticated, interacting components results in more transparent workflow and swifter execution.

Chapter 4

Biased exploration

LCS are being considered as self-adaptive and autonomous learners. However, in order to reach full autonomy, the credit assignment part must decide on their own when to exploit the existing knowledge by taking the most promising action and when to deliberately select an action that is not apparent best to gain additional knowledge potentially.

This decision is commonly referred to as the E/E (explore/exploit) dilemma, since obtaining new knowledge through exploration incurs a short-term performance loss, while too much exploitation risks staying on an unnecessarily low level of performance in the long term [118]. In a typical exploration phase, the action is selected randomly with the intention of an unbiased exploration.

Such exploration might be inefficient for real-valued environments where the search space is presumably large. Certain regions of space might be explored multiple times, while the other ones remain unknown. Therefore, an approach towards optimizing the process of acquiring knowledge by suggesting more valuable actions is considered herein.

At first, model learning capabilities in ALCS was enhanced by Stolzmann and Butz by using behavioural capabilities (internal reinforcement learning or lookahead action selection) in [116] and by introducing the *action planning* mechanism [115, 127].

Later, Butz suggested that by using computationally inexpensive methods by biasing exploration towards specially chosen regions of search-space, the model can be learned locally [17].

This chapter aims to narrow the existing gap in research by experimentally comparing four biased exploration strategies. Any ALCS agent operating a large realm of observation space might take advantage of the possibility of improving the time needed to form an internal model. First, a baseline method - *epsilon-greedy*, which is a default option

for LCS, will be described. Later two methods were introduced by Butz - *action-delay* and *knowledge-array* bias. They examine the match set $[M]$ searching for indications of which action might result in the highest information gain. Eventually, the latter approach is inspired by an *"Optimistic Initial Values"* approach described by Sutton in [118]. This strategy turned out to be very effective for Multi-armed bandit problems, where the main objective is to select the most promising action and was never examined in any LCS domain.

Interestingly, Hansmeier and Platzner made an effort to compare four strategies optimizing the time for alternating E/E phases using the XCS algorithm [43]. It turned out that despite automized parameter tuning that none of the strategies is superior to the other. On the other side, they noticed that specific multi-step environments with scarce reward signals become challenging due to setting the classifier's accuracy value too aggressively. Such problem with scarce reward and long action-chains is further discussed in the following chapter.

Best action

The best action selection relates to the exploitation mode. The algorithm selects most promising classifier capable of handling certain situation using attributes like the fitness score from the desired population $[P]$.

Algorithm 17 Best action selection

```
function BEST ACTION(P)
    cl ← determine most promising classifier from [P]
    return cl.action
end function
```

Epsilon-Greedy (EG)

In the epsilon-greedy approach, the agent equally discovers all regions from the input-space, not favouring any specific behaviour. In each step, random action is executed with p_{explr} or ϵ probability. Then it is chosen uniform randomly from classifiers composing a match set $[M]$. In the case of $1 - p_{explr}$, action from the most fitted classifier is executed. By doing so, the agent can occasionally perform the move he thinks is the best at a given time, reinforcing its beliefs about the consequences.

Algorithm 18 Epsilon-Greedy action selection

function EPSILON GREEDY(P, ϵ)
  if random number drawn from U[0, 1] distribution $< \epsilon$ **then**
    return random possible action
  else
    return `BEST ACTION`(P)
end function

Action-Delay (AD)

This bias is based on the *recency-based* principle assuming that the action executed a long time ago might introduce new situation-action-effect triples. To determine which action was executed most long ago at the current time t the t_{alp} field of all classifiers in a match set $[M](t)$ is analyzed. For situation $\sigma(t)$ the action of classifier cl with the lowest value of t_{alp} is selected.

In case there exists an action not represented by any classifier in $[M](t)$ that it is assumed to be experienced most long ago (if ever) and therefore chosen for execution.

Algorithm 19 Action-Delay biased action selection

function ACTION DELAY(P)
  $m \leftarrow$ map the number of classifiers from $[P]$ proposing to each available action
  $cl \leftarrow$ determine last executed classifier from $[P]$ using the t_{alp} property
  if there is any action a with no classifiers, $m[a] = 0$ **then**
    return a
  if cl is present **then**
    return $cl.action$
  else
    return random possible action
end function

Knowledge Array (KA)

This method, on the contrary, is based on the *error-based* principle. A classifier quality q metric denoting the accuracy of predictions for each individual can be used to measure it.

The bias generates the knowledge array KA from classifiers in a match set $[M]$ in which each entry specifies the averaged quality for the anticipation for each possible action - see Equation 4.1.

$$KA[a] = \frac{\sum_{cl \in [M] \wedge cl.A=a} cl.q \cdot cl.num}{\sum_{cl \in [M] \wedge cl.A=a} cl.num} \tag{4.1}$$

An action with the lowest value in the knowledge array is selected for execution. Similarly, as in the action delay bias, if any classifiers do not express actions, they are chosen first.

Algorithm 20 Knowledge-array biased action selection

function KNOWLEDGE ARRAY(P)
  $ka[a] \leftarrow$ initialize knowledge to 0 for each possible action
  for each available action a **do**
    $C \leftarrow$ select classifiers advocating action a from $[P]$
    $q \leftarrow \sum_{cl \in C} cl.q \cdot cl.num$
    $num \leftarrow \sum_{cl \in C} cl.num$
    $ka[a] \leftarrow \frac{q}{num}$
  end for
  return action a with the lowest knowledge using ka
end function

Optimistic Initial Quality (OIQ)

By default, newly created classifiers cl have the initial quality set to $cl.q = 0.5$. It can be said that they are biased towards their initial quality. In practice, it should not be a problem because this value will converge to optimal ones over a set of trials. Changing this parameter provides an easy way of supplying the agent with the confidence of the generated classifiers.

In this method, the agent's behaviour was parametrized by an extra parameter - *initial quality* - q_0. Every time a new classifier is generated, its quality is set to new value $cl.q = q_0$.

In all the experiments, there was fixed $q_0 = 0.8$, expecting the agent to build an internal model of knowledge representation faster (especially in the early stages). For the action selection strategy, the default epsilon-greedy method was chosen.

Moreover, the AD and KA biased action selection methods have ϵ_{biased} chances of being executed when the agent is exploring the environment - see Algorithm 21.

Algorithm 21 Biased exploration execution

```
function RUN BIASED EXPLORATION(P, ε, ε_biased)
    if random number drawn from U[0, 1] distribution < ε then
        if random number drawn from U[0, 1] distribution < ε_biased then
            return action elected by biased exploration strategy
        else
            return random possible action
    else
        return BEST ACTION(P)
end function
```

4.1 Research questions

The conducted research aims to answer the following question regarding the biased exploration strategies

Q1. Does the biased exploration methods (AD, KA, OIQ) significantly accelerate the agent's learning speed?

Q2. Can the OIQ method improve the performance of ingesting knowledge or reducing classifier population size?

4.2 Experimental evaluation

This section presents setup of the performed experiments and their results.

Goals of the experiments

Experiment 1 - Single-step problem performance

Similarly, as above but in this case, a single step 6-bit rMPX environment is used, introducing much larger possible states space. Since calculating precise model knowledge is infeasible, the key performance indicators chosen are the average obtained reward and model size.

Experiment 2 - Multiple-step problems performance

Use two basic multistep environments (Corridor and Grid) to determine the differences between the rate of gaining knowledge, the ability to build an internal pool of classifiers and operating in the environments.

Experiment 3 - Balancing the pole
The methods will be evaluated on the Cart Pole problem of balancing a pole on a cart. This is a novel problem for the LCS due to specific observation space (where two attributes span to infinity) and a specific reward scheme based on how long the pole is kept upright.

4.2.1 Experiment 1 - Single-step problem performance

The effect of biasing exploration was first evaluated on the single-step 6-bit rMPX environment. The input was discretized using the following buckets of sizes alongside the respective size of the input space:

1. 5 bits - $2 \cdot 5^6 = 31250$ possible states,
2. 6 bits - $2 \cdot 6^6 = 93312$ possible states,
3. 7 bits - $2 \cdot 7^6 = 235298$ possible states.

Each experiment was performed by alternating explore and exploit phases for 30000 trials. The cross-over capabilities were deliberately disabled because the last bit in perception σ was denoting the prediction result, which might inadvertently cause invalid rule evolution.

The efficiency of aforementioned biased exploration strategies is measured by studying the trajectory of the average reward and classifiers population size plots over time. Figures 4.1, 4.2 and 4.3 presents the average aggregation of results along with statistical inference calculated over 50 independent runs using the following parameters $\beta = 0.2$, $\gamma = 0.95$, $\theta_r = 0.9$, $\theta_i = 0.1$, $\epsilon = 0.8$ $\theta_{GA} = 50$, $\theta_{AS} = 20$, $\theta_{exp} = 50$, $m_u = 0.03$, $u_{max} = 4$, $\chi = 0.0$.

Statistical verification
To statistically assess the population size, the posterior data distribution was modelled using 50 metric values collected in the last trial and then sampled with 100,000 draws. For the obtained reward, the average value from the last 100 exploit trials is considered a representative state of algorithm performance. Tables 4.1 and 4.2 present computed results.

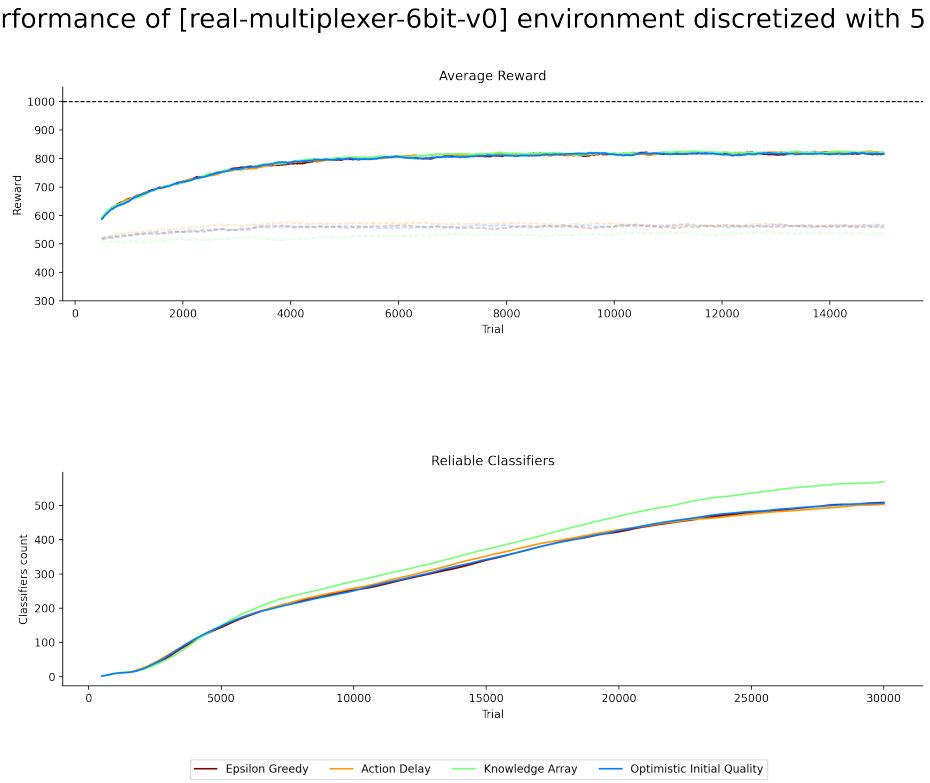

Figure 4.1: *6-bit rMPX discretized with 5 bins. Faded lines shows performance of explore mode, solid ones for exploit.*

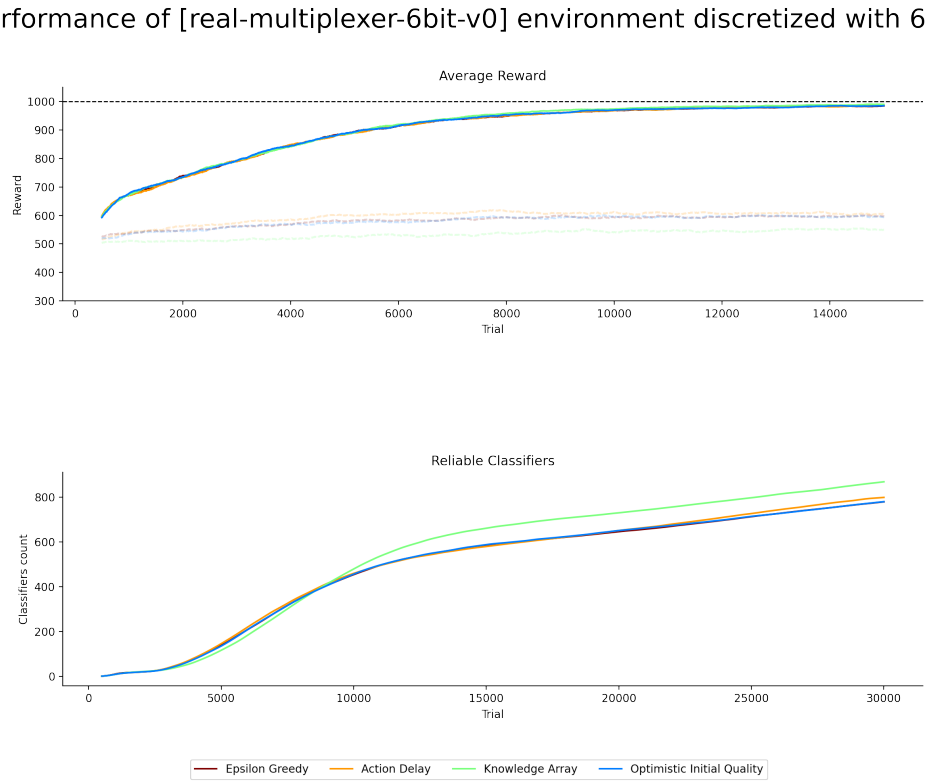

Figure 4.2: *6-bit rMPX discretized with 6 bins. Faded lines shows performance of explore mode, solid ones for exploit.*

	Epsilon Greedy	Action Delay	Knowledge Array	Optimistic Initial Quality
5 bins	819.40	818.20	824.80	815.60
6 bins	985.40	986.20	987.40	984.80
7 bins	839.40	832.00	848.60	835.00

Table 4.1: *Descriptive statistic metrics obtained using Bayesian estimation of the average reward for the investigated 6-bit rMPX problem.*

	Epsilon Greedy	Action Delay	Knowledge Array	Optimistic Initial Quality
5 bins	509.18 ± 3.67	505.28 ± 3.92	570.16 ± 4.08	508.31 ± 3.39
6 bins	781.81 ± 4.09	801.03 ± 4.03	872.07 ± 3.83	783.06 ± 3.89
7 bins	695.18 ± 3.72	698.2 ± 2.86	785.8 ± 3.22	698.58 ± 3.03

Table 4.2: *Descriptive statistic metrics obtained using Bayesian estimation of the population size for the investigated 6-bit rMPX problem.*

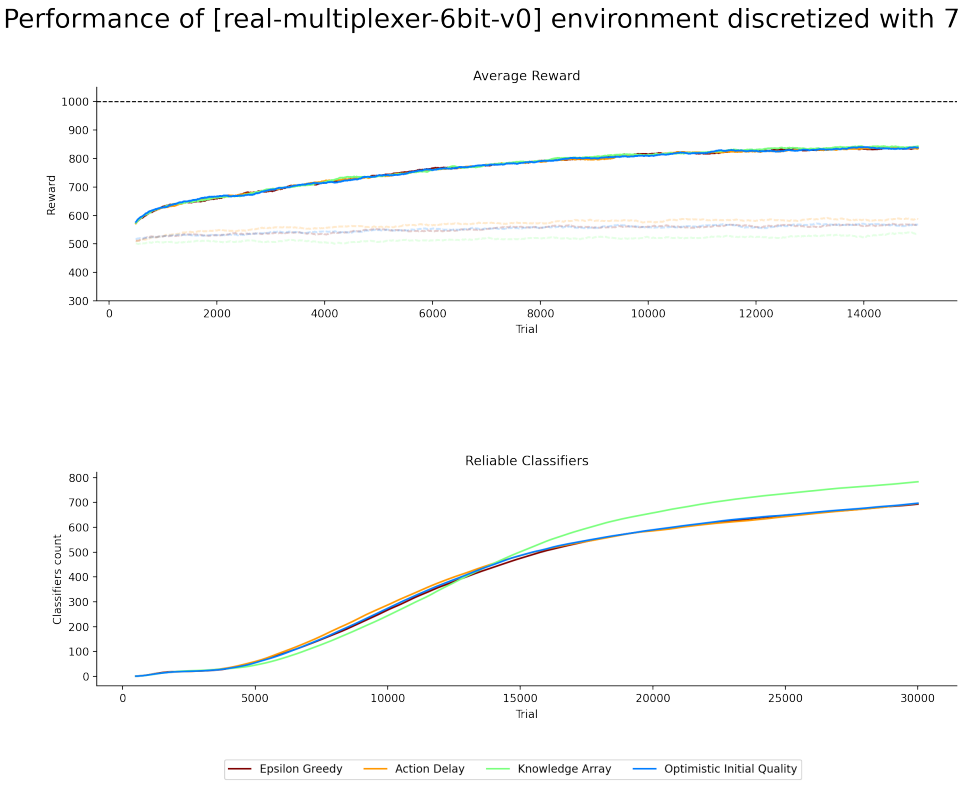

Figure 4.3: *6-bit rMPX discretized with 7 bins. Faded lines shows performance of explore mode, solid ones for exploit.*

Observations

Despite the large size of the problem's search space, the investigated biased exploration strategies have no impact on the average obtained reward. None of the methods can be distinguished by introducing peculiar rules into the population, imprinting on the

performance. However, the differences between discretization resolution are depicted. The ACS2 agent selects valid action when six bins are used to divide the perception input range most of the time. The exploit performance is significantly worse for the other odd values but still better than random guessing.

The rate of reliable classifier acquisition begins equally for all the strategies and resolutions. After a few thousand trials, the KA method tends to drive the exploration process into the unknown regions of the space, therefore forming more and better novel rules. The performance of other approaches of EG, AD and OIQ is not distinguishable.

4.2.2 Experiment 2 - Multiple-step problems performance

Both Corridor and Grid multiple-step environments were used for verifying the biased exploration strategies. In each case the ACS2 agent starts by performing 60 explore trials with selected strategy, followed by 20 where the evolved population is validated.

The following metrics are considered:

- knowledge - depicting the process of building an internal model,
- population size - demonstrate the total number of classifiers,
- steps in a trial - both in explore and exploit phase.

Figures 4.4 and 4.5 present the metric evolution for the basic versions of Corridor and Grid problems, containing 20 and 400 distinct states respectively, but the overall metrics look similar for larger instances. Additionally, for the Corridor, the cross-over capability of the agent was switched off because of the unit length of the perception vector σ.

To amplify the agent's motivation for exploring possible options, each problem was additionally increased to the sizes of $n = 40$ and $n = 100$. Last trial statistical inferences were collected in all cases to estimate overall performance.

Corridor ACS2 parameters: $\beta = 0.2$, $\gamma = 0.95$, $\theta_r = 0.9$, $\theta_i = 0.1$, $\epsilon = 0.8$ $\theta_{GA} = 50$, $\theta_{AS} = 20$, $\theta_{exp} = 50$, $m_u = 0.03$, $\chi = 0$.

Grid ACS2 parameters: $\beta = 0.2$, $\gamma = 0.95$, $\theta_r = 0.9$, $\theta_i = 0.1$, $\epsilon = 0.8$ $\theta_{GA} = 50$, $\theta_{AS} = 20$, $\theta_{exp} = 50$, $m_u = 0.03$, $u_{max} = 1$, $\chi = 0.8$.

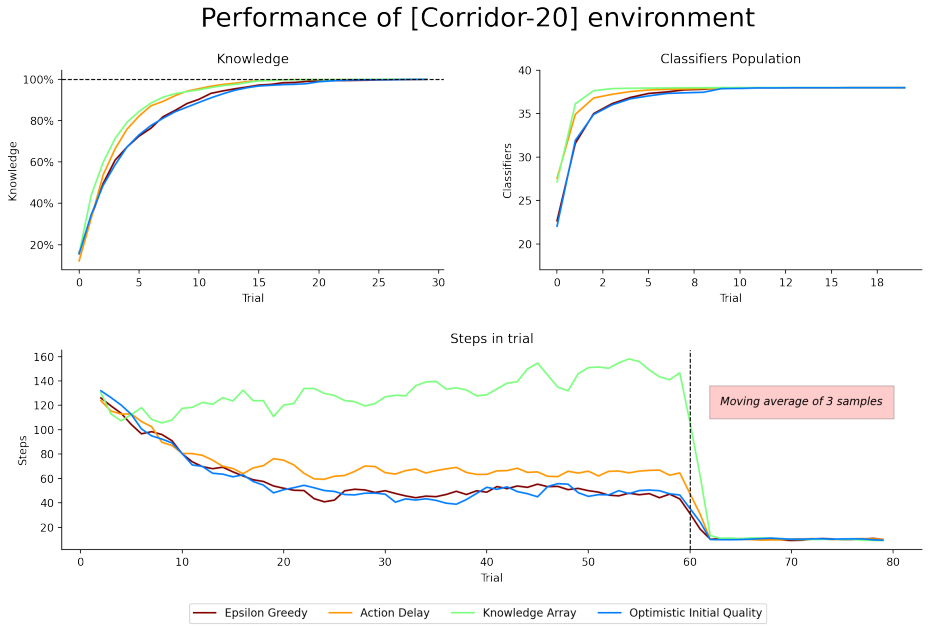

Figure 4.4: *Performance of Corridor-20 environment. 60 exploration and 20 exploitation trials averaged over 50 runs. Steps in a trial was plotted with a moving average of 3 last steps for clarity. No explicit discretizer was needed. The maximum number of steps in a trial is 200. The dotted vertical line indicates the execution of explore and exploit phases.*

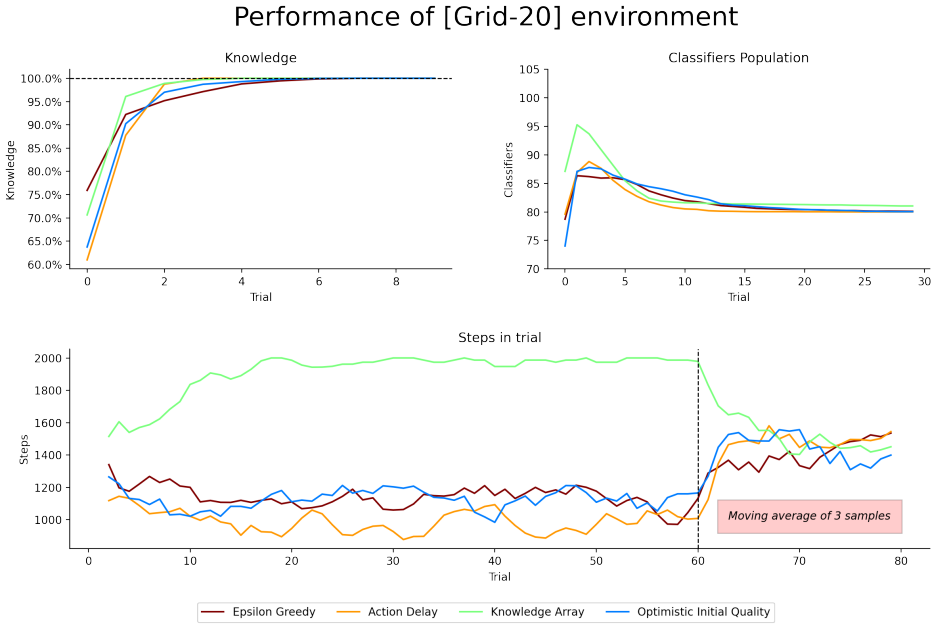

Figure 4.5: *Performance of Grid-20 environment. 60 exploration and 20 exploitation trials averaged over 50 runs. Steps in a trial was plotted with a moving average of 3 last steps for clarity. No explicit discretizer was needed. The maximum number of steps in a trial is 2000. The dotted vertical line indicates the execution of explore and exploit phases.*

Statistical verification

To statistically assess the population size, the posterior data distribution was modelled

	Epsilon Greedy	Action Delay	Knowledge Array	Optimistic Initial Quality
population of classifiers	38.0 ± 0.0	38.0 ± 0.0	38.0 ± 0.0	38.0 ± 0.0
reliable classifiers	38.0 ± 0.0	38.0 ± 0.0	38.0 ± 0.0	38.0 ± 0.0
knowledge	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0
trial execution time	0.02 ± 0.0	0.03 ± 0.0	0.05 ± 0.0	0.02 ± 0.0
average exploit reward	1000.0	1000.0	1000.0	1000.0

Table 4.3: *Descriptive statistic metrics obtained using Bayesian estimation of Corridor-20 performance.*

	Epsilon Greedy	Action Delay	Knowledge Array	Optimistic Initial Quality
population of classifiers	78.0 ± 0.0	78.0 ± 0.0	78.0 ± 0.0	78.0 ± 0.0
reliable classifiers	78.0 ± 0.0	78.0 ± 0.0	78.0 ± 0.0	78.0 ± 0.0
knowledge	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0
trial execution time	0.06 ± 0.0	0.07 ± 0.0	0.09 ± 0.0	0.06 ± 0.0
average exploit reward	949.0	998.0	1000.0	962.0

Table 4.4: *Descriptive statistic metrics obtained using Bayesian estimation of Corridor-40 performance.*

	Epsilon Greedy	Action Delay	Knowledge Array	Optimistic Initial Quality
population of classifiers	198.0 ± 0.0	198.0 ± 0.0	198.0 ± 0.0	198.0 ± 0.0
reliable classifiers	195.25 ± 0.53	195.78 ± 0.38	196.0 ± 0.0	195.67 ± 0.39
knowledge	98.63 ± 0.27	98.88 ± 0.2	98.98 ± 0.0	98.83 ± 0.21
trial execution time	0.15 ± 0.0	0.16 ± 0.0	0.17 ± 0.0	0.16 ± 0.0
average exploit reward	228.0	222.0	329.0	217.0

Table 4.5: *Descriptive statistic metrics obtained using Bayesian estimation of Corridor-100 performance.*

	Epsilon Greedy	Action Delay	Knowledge Array	Optimistic Initial Quality
population of classifiers	80.0 ± 0.0	80.0 ± 0.0	80.0 ± 0.0	80.0 ± 0.0
reliable classifiers	80.0 ± 0.0	80.0 ± 0.0	80.0 ± 0.0	80.0 ± 0.0
knowledge	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0
trial execution time	0.73 ± 0.01	0.69 ± 0.01	1.32 ± 0.0	0.72 ± 0.01
average exploit reward	459.0	440.0	377.0	448.0

Table 4.6: *Descriptive statistic metrics obtained using Bayesian estimation of Grid-20 performance.*

using 50 metric values collected in the last trial and then sampled with 100,000 draws. For the obtained reward, the average value from exploit trials is considered a representative state of algorithm performance.

Distinct computations were performed 3 sizes of each environment. Tables 4.3, 4.4, 4.5 present results for Corridor environments, while 4.6, 4.7 and 4.8 for Grid ones.

	Epsilon Greedy	Action Delay	Knowledge Array	Optimistic Initial Quality
population of classifiers	160.0 ± 0.0	160.0 ± 0.0	161.67 ± 0.34	160.0 ± 0.0
reliable classifiers	160.0 ± 0.0	160.0 ± 0.0	161.24 ± 0.24	160.0 ± 0.0
knowledge	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0
trial execution time	1.65 ± 0.02	1.63 ± 0.01	1.74 ± 0.01	1.67 ± 0.02
average exploit reward	197.0	191.0	141.0	216.0

Table 4.7: *Descriptive statistic metrics obtained using Bayesian estimation of Grid-40 performance.*

	Epsilon Greedy	Action Delay	Knowledge Array	Optimistic Initial Quality
population of classifiers	404.67 ± 0.54	401.98 ± 0.3	409.1 ± 0.87	403.7 ± 0.57
reliable classifiers	400.0 ± 0.0	400.0 ± 0.0	401.45 ± 0.25	400.0 ± 0.0
knowledge	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0
trial execution time	3.24 ± 0.02	3.33 ± 0.02	3.35 ± 0.02	3.23 ± 0.02
average exploit reward	47.0	45.0	24.0	17.0

Table 4.8: *Descriptive statistic metrics obtained using Bayesian estimation of Grid-100 performance.*

Observations

Corridor

Based on the Figure 4.4 all methods converge to optimal population size, and after switching to the exploitation mode, can utilize gained knowledge fully. Regardless of the exploration technique chosen, the agent can obtain complete knowledge of the environment in about 20 trials. AD and KA techniques seem to reach this point faster than the baseline EG and with OIQ modification.

The AD and KA methods accelerate the process of investigating the search-space resulting in earlier classifier creation.

Finally, the agent can fully exploit the environment after switching to "exploit" mode performing a minimal number of steps to reach the goal in each trial. Its also worth mentioning the constant effect of the KA method in explore phase, not taking the optimal actions most of the time, by continually updating the assumptions about all possibilities.

However, when the problem size increased twice ($n = 40$), while all strategies obtained full knowledge of the environment, only the KA method managed to exploit it totally unerringly. This difference is highlighted more for $n = 100$, where only about a third of all exploit trials were successful by KA, and other strategies performed significantly worse.

Grid

Performance plot using Grid of size $n = 20$ in Figure 4.5 shows that regardless of exploration technique chosen, the agent is still able to obtain full knowledge of the environment (even faster than in Corridor) and converge with the number of optimal classifiers (KA method here also creates much more classifiers at the beginning of the experimentation).

Interestingly, the KA obtains the worst average reward despite having the largest amount of reliable classifiers for problems where $n = 20$ and $n = 40$.

Moreover, what is interesting is that the agent cannot exploit the environment even though it knows the exact consequences of each action (non-optimal number of steps in the exploitation phase). After investigation, it was found that most classifiers have a very similar $cl.r$ value, representing the expected future reward. The agent in the current form is unable to differentiate between aliasing states, resulting in an inability to form an optimal policy. This finding emphasizes a need for a universal metric for quantifying the agent's performance. The current definition of knowledge, modelling only encountered

transitions, is inaccurate when the estimated reward is not distributed correctly amongst participating classifiers.

4.2.3 Experiment 3 - Balancing the pole

The challenging part about the Cart Pole problem is that attributes from the perception vector are described with different scales. Moreover, two of them range to infinity. This situation might occur when applying the ALCS agent to the real-world domain.

Splitting each attribute into a fixed amount of buckets is infeasible. Proposed solution involved assigning maximum, experienced values for both the cart σ_1 and pole σ_3 velocity. In this case:

- cart velocity $\sigma_1 \in [-0.5, 0.5]$,
- pole velocity at tip $\sigma_3 \in [-3500, 3500]$.

Additionally, a specific discretizer was used to divide each attribute into a predefined number of bins. This procedure implies precautions when performing the cross-over operation; therefore, it was disabled.

The experiment analyzes both the impact of selecting the granularity of the discretization scheme and the biased exploration technique. The ACS2 agent is first executing 500 explore trials using a specific method and then tries to use gained knowledge by selecting best action in further 500 exploit trials.

Five different discretization schemes chosen arbitrarily, defining a number of bins per attribute are listed below:

- `1, 1, 6, 6`,
- `4, 4, 4, 4`,
- `2, 2, 6, 6`,
- `1, 2, 4, 4`,
- `1, 1, 8, 8`.

The metrics of reliable population size and actual performance were both depicted in Figure 4.6 and estimated probabilistically for the above-mentioned schemes using the

	Epsilon Greedy	Action Delay	Knowledge Array	Optimistic Initial Quality
1,1,6,6	178.40	138.69	175.72	171.20
4,4,4,4	18.85	19.14	20.34	18.56
2,2,6,6	59.73	44.58	95.68	60.72
1,2,4,4	133.93	150.62	128.70	132.36
1,1,8,8	181.61	154.09	172.75	176.42

Table 4.9: *Descriptive statistic metrics obtained using Bayesian estimation of number of steps for Cart Pole environment.*

following ACS2 parameters: $\beta = 0.01$, $\gamma = 0.995$, $\theta_r = 0.9$, $\theta_i = 0.1$, $\epsilon = 0.9$ $\theta_{GA} = 50$, $\theta_{AS} = 20$, $\theta_{exp} = 50$, $m_u = 0.03$, $u_{max} = 4$, $\chi = 0.0$.

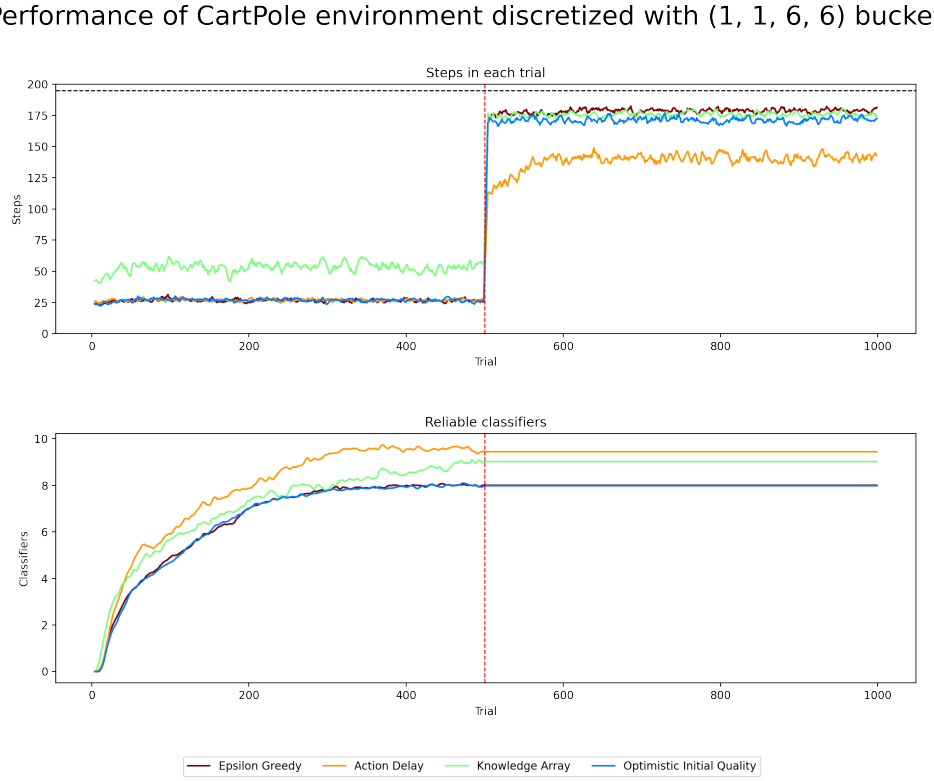

Figure 4.6: *Performance in CartPole environment. 500 exploration and 500 exploitation trials averaged over 50 runs. Moving average of 5 last trials applied for clarity. The dotted vertical line indicates the execution of explore and exploit phases. The environment is considered solved if the average reward is greater than or equal to 195 over the last 100 trials.*

Statistical verification

To statistically assess the population size(Table 4.10), the posterior data distribution was modelled using 50 metric values collected in the last trial and then sampled with 100,000 draws. For the obtained reward (Table 4.9), the average value from exploit trials is considered a representative state of algorithm performance.

	Epsilon Greedy	Action Delay	Knowledge Array	Optimistic Initial Quality
1,1,6,6	8.0 ± 0.0	9.39 ± 0.18	9.01 ± 0.23	8.0 ± 0.0
4,4,4,4	6.33 ± 0.26	4.38 ± 0.19	5.65 ± 0.23	6.06 ± 0.23
2,2,6,6	6.99 ± 0.25	7.29 ± 0.25	8.27 ± 0.33	7.48 ± 0.31
1,2,4,4	11.23 ± 0.18	9.83 ± 0.18	10.01 ± 0.18	10.86 ± 0.22
1,1,8,8	9.14 ± 0.12	8.92 ± 0.12	10.42 ± 0.21	9.18 ± 0.14

Table 4.10: *Descriptive statistic metrics obtained using Bayesian estimation of reliable classifiers for Cart Pole environment.*

Rule	Mark	Quality	Reward	Numerosity
##23 0 ####	00##	0.96	3.34	1
##32 1 ####	00##	0.96	3.24	1
##22 1 ####	00##	0.98	2.78	1
##33 0 ###	00##	0.95	2.23	3
##12 0 ####	00##	0.96	1.44	1
##12 1 ####		1.00	1.36	20
##43 1 ####	00##	0.97	1.32	6
##43 0 ####		1.00	1.22	20

Table 4.11: *Example of top reliable classifiers for Cart Pole problem with `1, 1, 6, 6` discretization.*

Observations

Surprisingly, when using the discretization of `1, 1, 6, 6`, the agent can keep the pole upright for about 175 steps in each trial after performing just 500 learning trials. This score was possible for every method except the AD. On the other side, AD created more reliable classifiers quicker than other methods.

The experiment's performance turned out to be very sensitive to the discretization bins chosen. For example, a slightly larger amount of bins for pole angle and velocity (eight bins in both cases) increased the number of upright steps. In official terms, the environment is still not solved. However, it turned out that the number of reliable classifiers required to obtain such a score is less than 10. That allows a very compact and human-readable form of storing knowledge - see Table 4.11 example. It can be seen that the majority of reliable classifiers are marked on the first two attributes, meaning that they too sweep and therefore should be more distinguishable (for example, by increasing discretization). In order to set them properly, a dedicated hyper-parameter optimization process is advised.

Despite fragility, the obtained result is auspicious, showing that ALCS methods can be compared to other highly sophisticated black-box approaches and maintain a highly verbose problem model.

4.3 Research summary

The answers to the previously formulated research questions are as follows:

Q1: Does the biased exploration methods (AD, KA, OIQ) significantly accelerate the agent's learning speed?

Conducted research showed that biased exploration methods like AD and KA positively impact the knowledge evolution process. The OIQ performance was correlated with the basic EG method and did not yield any competitive performance.

Q2: Can the OIQ method improve the performance of ingesting knowledge or reducing classifier population size?

The impact of initializing classifier quality with a higher (positive) value resulted in having a neglectable effect on all evaluated metrics. The OIQ, in theory, can perform better when forming a population of reliable classifiers, but for the investigated problems, the performance was highly correlated with a default EG method.

Chapter 5

Optimizing reward distribution through long action chains

Previous chapters showed that applying real-valued input handling either by using the interval predicates or discretization significantly increases the input space. This magnitude of possible states implies the growth of classifier population size; therefore, the traditional discounted sum of reward distribution might not be an appropriate option. The desired intention is for the case when the rewards received in all decision instances are equally important.

The alternative criterion applied in this situation is called *the average reward criterion* and was introduced by Puterman [95]. He stated that the decision-maker might prefer it when the decisions are made frequently (so that the discount rate is very close to 1), or other terms cannot easily describe the performance criterion. Possible areas of an application might include situations where system performance is assessed based on the throughput rate (like making frequent decisions when controlling the flow of communication networks).

The averaged reward criterion was first implemented in XCS by Tharakunnel and Goldberg [120]. They called their modification AXCS and showed that it performed similarly to the standard XCS in the Woods2 environment. Later, Zang et al. [147] formally introduced the R-learning [106, 109] technique to XCS and called it XCSAR. They compared it with XCSG (where the prediction parameters are modified by applying the idea of gradient descent) and ACXS (maximizing the average of successive rewards) in large multistep problems (Woods1, Maze6, and Woods14).

This chapter replaces the ACS2 credit assignment component, optimizing the performance in the infinite horizon (discounted reward) with an averaged version. The introduced variant is AACS2 (Averaged Anticipatory Classifier System) and is implemented in two slightly different variants - AACS2-v1 and AACS2-v2. The performance is validated using two scalable and discretized environments requiring multiple steps to pursue reward.

5.1 Reinforcement Learning and Reward Criterion

RL is a formal framework in which the agent can influence the environment by executing specific actions and receive corresponding feedback (reward) afterwards. Usually, it takes multiple steps to reach the goal, which makes the process much more complicated. In the general form, RL consists of:

- a discrete set of environment states S,
- a discrete set of available actions A,
- mapping R between a particular state $s \in S$ and action $a \in A$. The environmental payoff $r \in R$ describes the expected reward obtained after executing an action in a given state.

In each trial, the agent perceives the environmental state s. Next, it evaluates all possible actions from A and executes action a in the environment. The environment returns a reward r and next state s' as intermediate feedback.

The agent's task is to represent the knowledge, using the policy π mapping states to actions, therefore optimizing a long-run measure of reinforcement. There are two popular optimality criteria used in MDPs (Markov Decision Problems) - a *discounted reward* and an *average reward* [63, 83].

5.1.0.1 Discounted Reward Criterion

In discounted RL, the future rewards are geometrically discounted according to a discount factor γ, where $0 \leq \gamma < 1$. The performance is usually optimized in the infinite horizon [118]:

$$\lim_{N\to\infty} E^{\pi}\left(\sum_{t=0}^{N-1}\gamma^{t} r_t(s)\right) \tag{5.1}$$

The E expresses the expected value, N is the number of time steps, and $r_t(s)$ is the reward received at time t starting from state s under the policy.

5.1.0.2 Undiscounted (Averaged) Reward Criterion

The *averaged reward criterion* [106], which is the undiscounted RL, is where the agent selects actions maximizing its long-run average reward per step $\rho(s)$:

$$\rho^{\pi}(s) = \lim_{N \to \infty} \frac{E^{\pi}\left(\sum_{t=0}^{N-1} r_t(s)\right)}{N} \tag{5.2}$$

If a policy maximizes the average reward over all states, it is a *gain optimal policy*. Usually, average reward $\rho(s)$ can be denoted as ρ, which is state-independent [82], formulated as $\rho^{\pi}(x) = \rho^{\pi}(y) = \rho^{\pi}, \forall x, y \in S$ when the resulting Markov chain with policy π is ergodic (aperiodic and positive recurrent) [94].

To solve an average reward MDP problem, a stationary policy π maximizing the average reward ρ must be determined. To do so, the *average adjusted sum* of rewards earned following a policy π is defined as:

$$V^{\pi}(s) = E^{\pi}\left(\sum_{t=0}^{N \to \infty} (r_t - \rho^{\pi})\right) \tag{5.3}$$

The $V^{\pi}(s)$ can also be called a *bias* or *relative value*. Therefore, the optimal relative value for a state–action pair (s, a) can be written as:

$$V(s,a) = r^{a}(s,s') - \rho + \max_{b} V(s',b) \forall s \in S \text{ and } \forall a \in A \tag{5.4}$$

where $r^{a}(s,s')$ denotes the immediate reward of action a in state s when the next state is s', ρ is the average reward, and $\max_b V(s',b)$ is the maximum relative value in state s' among all possible actions b. Equation 5.4 is also known as the Bellman equation for an average reward MDP [94].

5.2 Integrating Reward Criterions in ACS2

Despite the ACS's *latent-learning* capabilities, the RL is realized using two classifier metrics - reward $cl.r$ and immediate reward $cl.ir$. The latter stores the immediate reward

predicted to be received after acting in a particular situation and is used mainly for model exploitation where the reinforcement might be propagated internally. The reward parameter $cl.r$ stores the reward predicted to be obtained in the long run.

For the first version of ACS, Stolzmann proposed a *bucket-brigade* algorithm to update the classifier's reward r_c [47, 114]. Let c_t be the active classifier at time t and c_{t+1} the active classifier at time $t+1$:

$$r_{c_t}(t+1) = \begin{cases} (1-b_r) \cdot r_{c_t}(t) + b_r \cdot r(t+1), & \text{if } r(t+1) \neq 0 \\ (1-b_r) \cdot r_{c_t}(t) + b_r \cdot r_{c_{t+1}}(t), & \text{if } r(t+1) = 0 \end{cases} \tag{5.5}$$

where $b_r \in [0,1]$ is the *bid-ratio*. The idea is that if there is no environmental reward at time $t+1$, then the currently active classifier c_{t+1} gives a payment of $b_r \cdot r_{c_{t+1}}(t)$ to the previous active classifier c_t. If there is an environmental reward $r(t+1)$, then $b_r \cdot r(t+1)$ is given to the previous active classifier c_t.

Later, Butz adopted the Q-learning idea in ACS2 alongside other modifications [18]. For the agent to learn the optimal behavioural policy, both the reward $cl.r$ and intermediate reward $cl.ir$ are continuously updated. To assure maximal Q-value, the quality of a classifier is also considered assuming that the reward converges in common with the anticipation's accuracy. The following updates are applied to each classifier cl in action set $[A]$ during every trial:

$$\begin{aligned} cl.r &= cl.r + \beta \left(\phi(t) + \gamma \max_{cl' \in [M](t+1) \wedge cl'.E \neq \{\#\}^L} (cl'.q \cdot cl'.r) - cl.r \right) \\ cl.ir &= cl.ir + \beta \left(\phi(t) - cl.ir \right) \end{aligned} \tag{5.6}$$

The parameter $\beta \in [0,1]$ denotes the learning rate and $\gamma \in [0,1)$ is the discount factor. With a higher β value, the algorithm takes less care of past encountered cases. On the other hand, γ determines to what extent the reward prediction measure depends on future reward.

Thus, in the original ACS2, the calculation of the discounted reward estimation at a specific time t is described as $Q(t)$, which is part of Equation 5.6:

$$Q(t) \leftarrow \phi(t) + \gamma \max_{cl' \in [M](t+1) \wedge cl'.E \neq \{\#\}^L} (cl'.q \cdot cl'.r) \tag{5.7}$$

The modified ACS2 implementation replacing the discounted reward with the averaged version with the formula $R(t)$ is defined below - Equation 5.8:

$$R(t) = \phi(t) - \rho + \max_{cl' \in [M](t+1) \wedge cl'.E \neq \{\#\}^L} (cl'.q \cdot cl'.r) \tag{5.8}$$

The definition above requires an estimate of the average reward ρ. Equation 5.4 showed that the maximization of the average reward is achieved by maximizing the relative value. The following sections will propose two variants of using the average reward criterion for internal reward distribution. The altered version is named AACS2, which stands for *Averaged ACS2*.

As the next operation in both cases, the reward parameter of all classifiers in the current action set $[A]$ is updated using the following formula:

$$cl.r \leftarrow cl.r + \beta(R - cl.r) \tag{5.9}$$

where β is the learning rate and R was defined in Equation 5.8.

5.2.1 AACS2-v1

The first variant of the AACS2 represents the ρ parameter as the ratio of the total reward received along the path to reward and the average number of steps needed. It is initialized as $\rho = 0$, and its update is executed as the first operation in RL using the Widrow-Hoff delta rule - Equation 5.10. The update is also restricted to be executed only when the agent chooses the action greedily during the explore phase:

$$\rho \leftarrow \rho + \zeta[\phi - \rho] \tag{5.10}$$

The ζ parameter denotes the learning rate for the average reward and is typically set at a very low value. This ensures a nearly constant value of average reward for the update of the reward, which is necessary for the convergence of average reward RL algorithms [32].

5.2.2 AACS2-v2

The second version is based on the XCSAR proposition by Zang [147]. The only difference from the AACS2-v1 is that the estimate is also dependent on the maximum classifier fitness calculated from the previous and current match set:

$$\rho \leftarrow \rho + \zeta[\phi + \max_{cl \in [M](t) \wedge cl.E \neq \{\#\}^L}(cl.q \cdot cl.r) - \max_{cl \in [M](t+1) \wedge cl.E \neq \{\#\}^L}(cl.q \cdot cl.r) - \rho] \quad (5.11)$$

5.3 Research questions

The conducted research aims to answer the following questions:

Q1. Can the undiscounted reward criterion be used in discretized multistep environments?

Q2. Does the undiscounted reward criterion result in better environment exploitation?

5.4 Experimental evaluation

This section presents setup of the performed experiments and their results.

Goals of the experiments

Experiment 1 - Straight Corridor

The Corridor environment provides a simple, real-valued and scalable benchmark perfectly suited for the problem of long action chains. The agent must repeat the same action a certain amount of times unless reaching the final reward state.

Experiment 2 - Deceptive Corridor

The FSW environment is an extension to the Corridor. In each state, actions alternate - one will bring the agent closer to the reward, and the other is languishing. The forager will have to distinguish every other state and distribute rewards properly.

5.4.1 Experiment 1 - Straight Corridor

The following section describes the differences observed between using the ACS2 with standard discounted reward distribution and two proposed modifications. In all cases, the experiments were performed in an explore-exploit manner for the total number of 10000

trials, where the mode was alternating in each trial. Additionally, for better reference and benchmarking purposes, basic implementations of Q-Learning and R-Learning algorithms were also introduced and used with the same parameter settings as ACS2 and AACS2.

The most important thing was to distinguish whether the new reward distribution proposition still allows the agent to successfully update the classifier's parameter to allow the exploitation of the environment. To illustrate this, figures presenting the number of steps to the final location, estimated average change during learning, and the reward payoff landscape across all possible state-action pairs were plotted for the Corridor of size $n = 20$ - Figure 5.2.

To assure that the modification worked as expected, the statistical inference of obtained result was performed on a scaled version of the problem. Each experiment is averaged over 50 independent runs and run with the following parameters: $\beta = 0.8$, $\gamma = 0.95$, $\epsilon = 0.2$, $\theta_r = 0.9$, $\theta_i = 0.1$, $m_u = 0$, $\chi = 0$, $\zeta = 0.0001$.

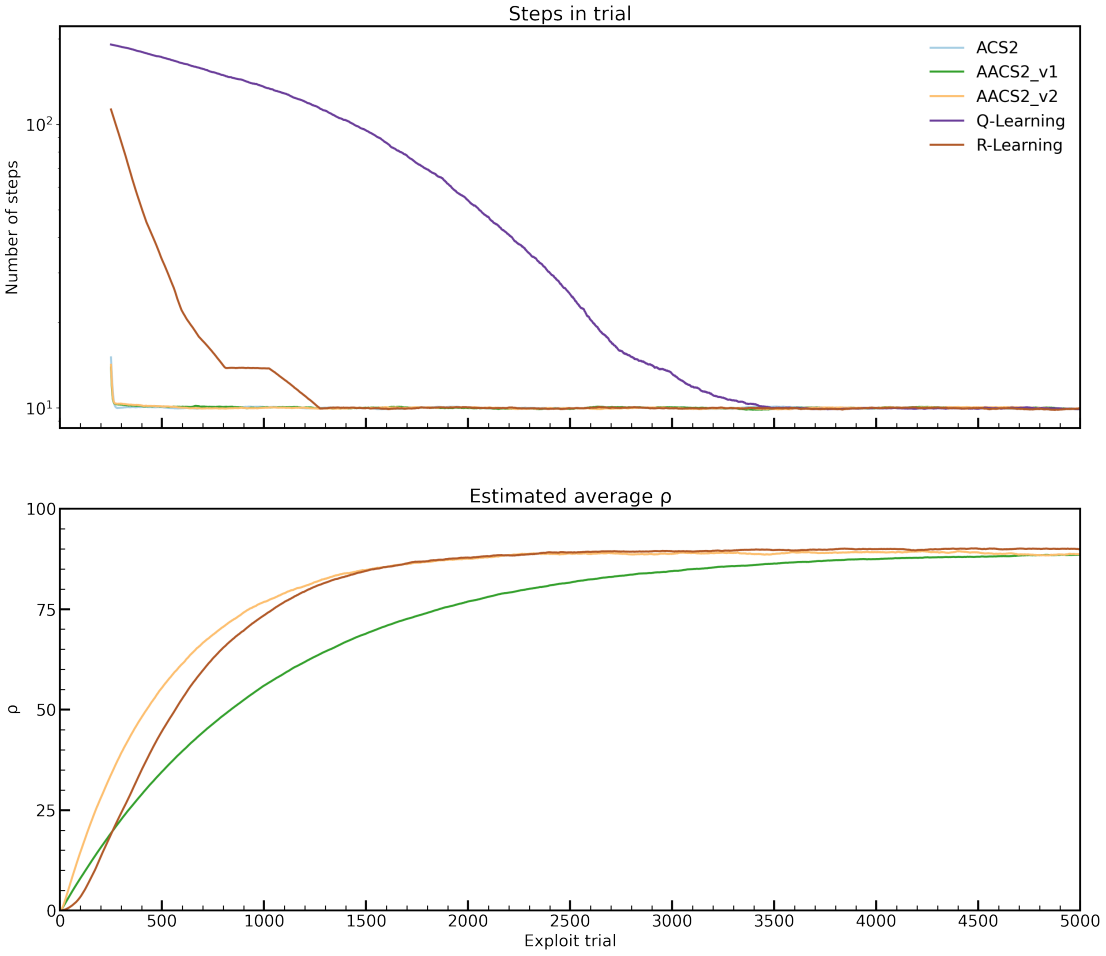

Figure 5.1: *Performance in Corridor-20 environment. Plots averaged over 50 independent runs. Number of steps in exploit trials is averaged over 250 last data points.*

	ACS2	AACS2v1	AACS2v2	Q-Learning	R-Learning
steps in last trial	9.61 ± 0.84	9.54 ± 0.84	9.0 ± 0.82	7.91 ± 0.79	9.31 ± 0.9
average reward per step	-	88.52 ± 0.16	88.65 ± 0.29	-	89.9 ± 0.19

Table 5.1: *Descriptive statistic metrics obtained using Bayesian estimation of obtained results for Corridor-20 problem.*

	ACS2	AACS2v1	AACS2v2	Q-Learning	R-Learning
steps in last trial	19.42 ± 1.65	19.43 ± 1.7	19.47 ± 1.46	124.96 ± 14.21	17.64 ± 1.67
average reward per step	-	44.7 ± 0.14	44.7 ± 0.19	-	44.93 ± 0.12

Table 5.2: *Descriptive statistic metrics obtained using Bayesian estimation of obtained results for Corridor-40 problem.*

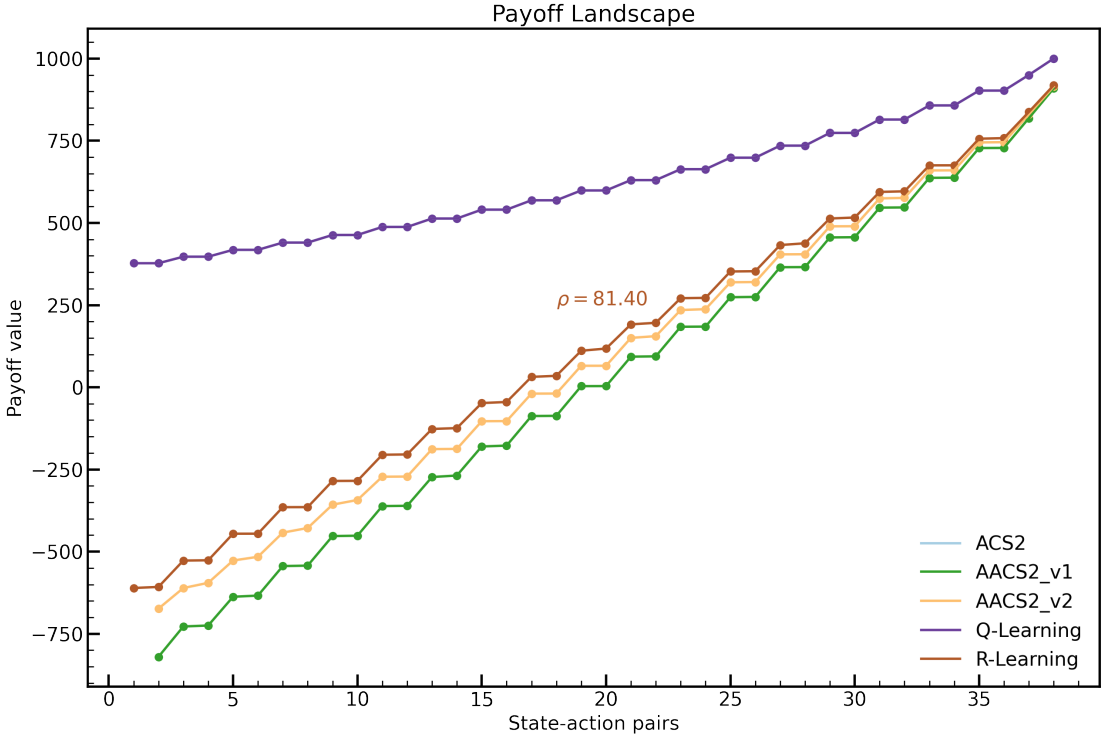

Figure 5.2: *Payoff Landscape for Corridor-20 environment. Payoff values were obtained after 10000 trials. For the Q-Learning and R-Learning, the same learning parameters were applied. The ACS2 and Q-Learning generate exactly the same payoffs for each state-action pair.*

Statistical verification

To statistically assess the population size, the posterior data distribution was modelled using 50 metric values collected in the last trial and then sampled with 100,000 draws. Tables 5.1, 5.2, 5.3 presents data for three versions of the problem.

Observations

	ACS2	AACS2v1	AACS2v2	Q-Learning	R-Learning
steps in last trial	55.95 ± 4.13	51.25 ± 4.19	49.83 ± 4.48	200.0 ± 0.0	48.65 ± 4.37
average reward per step	-	17.84 ± 0.09	17.75 ± 0.13	-	17.85 ± 0.07

Table 5.3: *Descriptive statistic metrics obtained using Bayesian estimation of obtained results for Corridor-100 problem.*

The average number of steps can be calculated $\frac{\sum_0^n n}{n-1}$, where n is the number of distinct Corridor states. For the tested environment it gives the approximate value of 11.05, therefore the average reward per step estimation should be close to $1000/11.05 = 90.49$, which corresponds to the Figure 5.1.

The same Figure demonstrates that all investigated agents learned the environments. The anticipatory classifier systems obtained an optimal number of steps after the same number of exploit trials, which is about 200. In addition, the AACS2-v2 updates the ρ value more aggressively in earlier phases, but the estimate converges near the optimal reward per step.

For the payoff-landscape in Figure 5.2, all allowed state–action pairs were identified in the environment (38 in this case). The final population of learning classifiers was established after 100 trials and was the same size. Both Q-table and R-learning tables were populated using the same parameters and number of trials.

The relative distance between adjacent state-action pairs can be divided into three groups. The first one relates to the discounted reward agents (ACS2, Q-Learning). Both generate almost a similar reward payoff for each state–action. Later, there is the R-Learning algorithm, which estimates the ρ value and separates states evenly. Furthermore, two AACS2 agents are performing very similarly. The ρ value calculated by the R-Learning algorithm is lower than the average estimation by the AACS2 algorithm.

Scaled problem instances revealed interesting properties:

- the Q-Learning algorithm was not capable of executing the optimal number of steps in environments with $n = 40$, $n = 100$,
- for the most challenging problem of $n = 1000$, the AACS2 modification yield better performance than ACS2,
- all algorithms with undiscounted reward critera managed to calculate the average reward ρ.

5.4.2 Experiment 2 - Deceptive Corridor

FSW differentiate from the Corridor by the presence of futile movement available in every step. Moreover, this action alternates between the states, demanding the agent to discriminate between relevant perceptions.

Agent's behaviour is expected to properly distribute reward across all valuable states, meanwhile ignoring the rest of them. The analysis was performed by executing consecutive 10000 trials alternating between explore and exploit phases. Similarly, as in the previous experiment, there is only one state-observed; thus, the ACS2 genetic generalization mechanism remains turned off.

Parameters: $\beta = 0.5$, $\gamma = 0.95$, $\epsilon = 0.1$, $\theta_r = 0.9$, $\theta_i = 0.1$, $m_u = 0$, $\chi = 0$, $\zeta = 0.0001$.

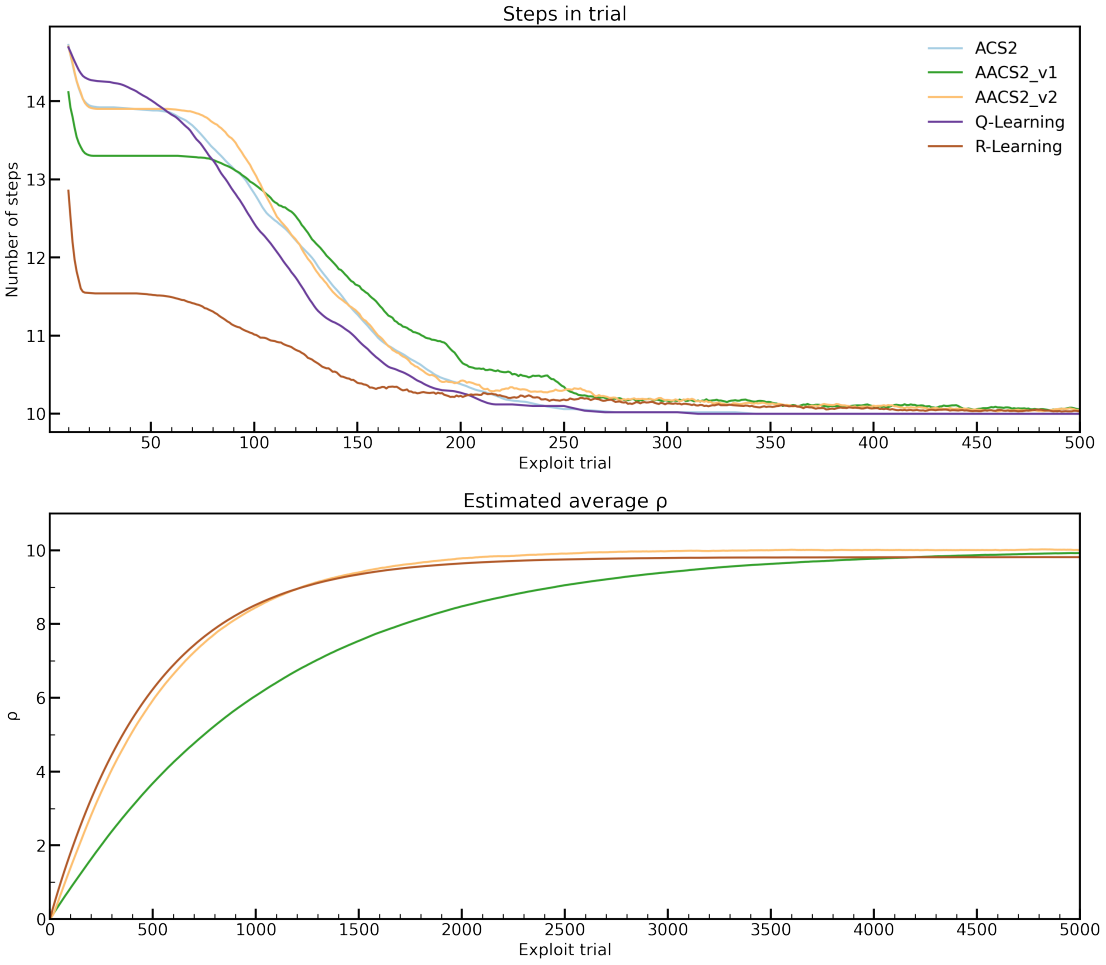

Figure 5.3: *Performance in FSW-10 environment. Plots are averaged over 50 experiments. A moving average with window 25 was applied for the number of steps. Notice that the abscissa on both plots is scaled differently.*

	ACS2	AACS2v1	AACS2v2	Q-Learning	R-Learning
steps in last trial	10.0 ± 0.0	10.0 ± 0.0	10.0 ± 0.0	10.0 ± 0.0	10.0 ± 0.0
average reward per step	-	9.93 ± 0.01	10.01 ± 0.01	-	9.81 ± 0.0

Table 5.4: *Descriptive statistic metrics obtained using Bayesian estimation of obtained results for FSW-10 problem.*

	ACS2	AACS2v1	AACS2v2	Q-Learning	R-Learning
steps in last trial	20.0 ± 0.0	20.0 ± 0.0	20.0 ± 0.0	20.0 ± 0.0	20.0 ± 0.0
average reward per step	-	4.99 ± 0.01	5.0 ± 0.01	-	4.9 ± 0.0

Table 5.5: *Descriptive statistic metrics obtained using Bayesian estimation of obtained results for FSW-20 problem.*

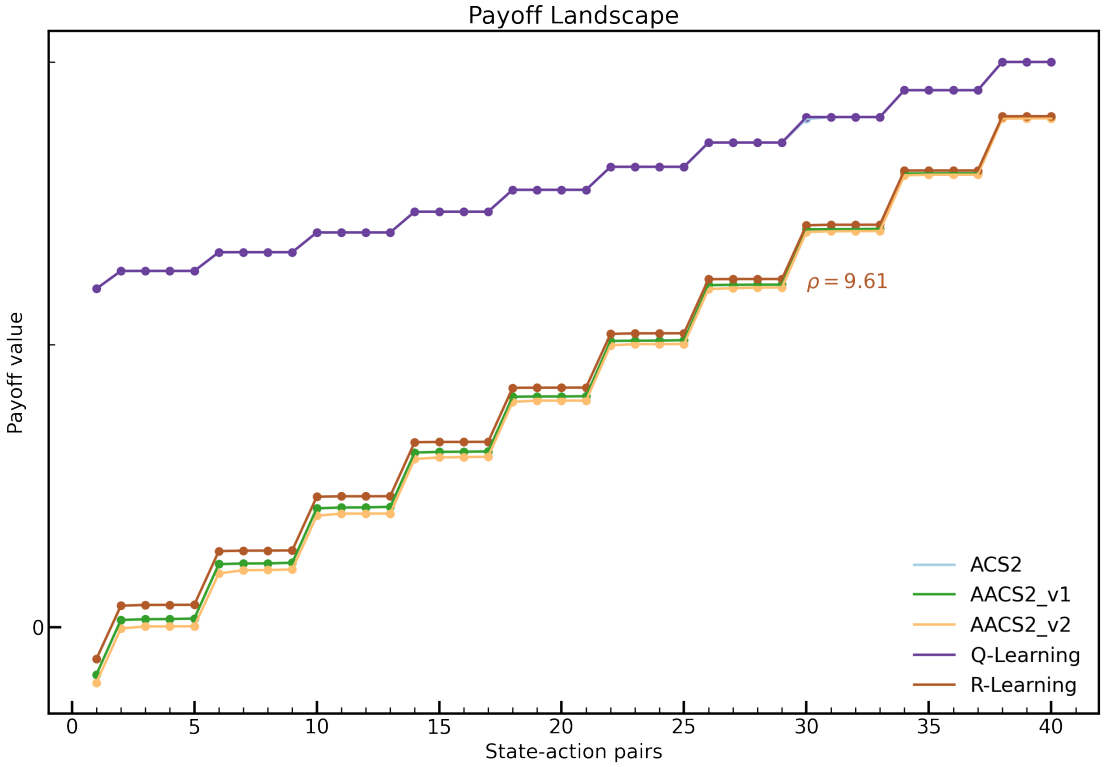

Figure 5.4: *Payoff Landscape for the FSW-10 environment. Payoff values were obtained after 10000 trials. For the Q-Learning and R-Learning, the same learning parameters were applied.*

Statistical verification

To statistically assess the population size, the posterior data distribution was modelled using 50 metric values collected in the last trial and then sampled with 100,000 draws. Tables 5.4, 5.5, 5.6 presents data for three versions of the problem.

Observations

	ACS2	AACS2v1	AACS2v2	Q-Learning	R-Learning
steps in last trial	40.0 ± 0.0	40.0 ± 0.0	40.0 ± 0.0	40.0 ± 0.0	40.0 ± 0.0
average reward per step	-	2.51 ± 0.0	2.51 ± 0.01	-	2.45 ± 0.0

Table 5.6: *Descriptive statistic metrics obtained using Bayesian estimation of obtained results for FSW-40 problem.*

The size of the environment used on Figure 5.3 was chosen to be $n = 10$, resulting in $2n + 1 = 21$ distinct states. Statistical inference proves that the undiscounted versions of rewarding manage to properly discriminate between worthwhile states without fluctuations for all problem sizes.

The same Figure depicts that agents can learn in a more challenging environment without problems. It takes about 250 trials to perform an optimal number of steps to reach the reward state. The ρ parameter converges with the same dynamics as the Corridor environment from the previous section.

The payoff-landscape Figure 5.4 shows that the average value estimate is very close to the one calculated by the R-learning algorithm. The difference is primarily visible in the state-action pairs located afar from the final state. The discounted versions of the algorithms performed precisely the same.

5.5 Research summary

The answers to the previously formulated research questions are as follows:

Q1: Can the undiscounted reward criterion be used in discretized multistep environments?

Concluded experiments have shown that the undiscounted reward criterion can be successfully applied to a problem requiring long-action chains such as the Corridor of FSW. The new system AACS2 varies only in a way the classifier reward $cl.r$ parameter is calculated. The clear difference between the discounted criterion is visible in the payoff landscapes generated from the testing environments. The AACS2 can produce a distinct payoff landscape with uniformly spaced payoff levels, similar to the one generated by the R-learning algorithm. When taking a closer look, all algorithms generate step-like payoff-landscape plots, but each particular state-action pairs are more distinguishable when the reward-criterion is used. The explanation of why the agent moves toward the goal at all can be found in Equation 5.8 - it can find the following best action by using the best classifiers' fitness from the next match-set.

In addition, the rate at which the average estimate value ρ converges is different for AACS2-v1 and AACS2-v2. Figures 5.1 and 5.3 demonstrate that the AACS2-v2 settles to the final value faster, but also has greater fluctuations. That is caused by the fact that both match sets' maximum fitness is considered when updating the values. Zang also observed this and proposed that the learning rate ζ in Equation 5.10 could decay over time [147]:

$$\zeta \leftarrow \zeta - \frac{\zeta^{max} - \zeta^{min}}{NumOfTrials} \tag{5.12}$$

where ζ^{max} is the initial value of ζ, and ζ^{min} is the minimum learning rate required. The update should take place at the beginning of each exploration trial.

Q2: Does the undiscounted reward criterion result in better environment exploitation?

Number of steps during exploitation trials depicted on Figures 5.1 and 5.3 indicate that for the investigated problems the agent is able to pick up better actions when the undiscounted criterion is used.

In addition, the fact that the ρ converged to a slightly suboptimal value might be caused by the exploration strategy adopted, which was set to the ϵ-greedy. Because the estimated average reward is updated only when the greedy action is executed, the number of greedy actions to be performed during the exploration trial is uncertain. Moreover, the probability distribution when the agent observes the rewarding state might be too low in order to enable the estimated average reward to reach optimal value. This was observed during the experimentation - the ρ value was correlated with the ϵ parameter.

Chapter 6

Summary

6.1 Conclusions

This thesis focused on integrating anticipatory classifier systems with problems described by real-valued output. While this area was researched extensively in other LCS families, no major contribution was still made for the ALCS.

The stated hypothesis - *"Anticipatory Learning Classifiers Systems are capable of building the correct internal model of the environment using the real-valued input"* is considered as valid by achieving the following goals:

1. Propose modifications towards ALCS system capable of handling real-valued input

This intention was achieved and validated using two independent approaches:

1. changing the rule's attribute representations to incorporate *interval predicates*, resulting in a proprietary system variation named rvACS,
2. implementing the prominent ALCS algorithms with the intention of input discretization.

The first approach was based on the advancements made for the XCS systems. The rvACS managed to show promising results. Nevertheless, because of the much richer rule complexity and the cooperation of components precisely forming the condition and effect parts, the obtained conclusion is that the nature of rvACS is not aligned with the overall ALCS idea. The favoured implementation of ACS2, which is considered the most mature algorithm, is built upon the psychological theory of behavioural control, which

was not investigated for this kind of problem. The algorithm performance, while being valid, is not elegant for the virtues of ALCS for creating the most general, compact and accurate rules.

The second approach maintains the characteristics and original intentions of investigated algorithms by affecting only the agent-environment interface layer. Hopefully, the internal representation is not limited to the ternary alphabet (binary values and generalization symbol), so any nominal value can also be used. This enables the possibility of performing input discretization *before* an agent processes it. While promising results and better transparency, this approach was considered superior to interval formation (see Figures 3.4, 3.9 using the same testing problem) and was pursued in consecutive experiments.

However, the tradeoff between interpretability and efficiency in ALCS is evident in both cases. The lack of dedicated compaction mechanisms results in a population of fine-grained rules that quickly becomes a bottleneck in algorithm performance. The potential performance optimisations are also challenging due to complex components interactions and the sequential nature of the considered algorithms.

The proposition of the first approach was published in [67], while the latent learning experiments in discretized environments in [70].

2. Propose relevant benchmarking problems and metrics for evaluating ALCS performance

The nature of the performance was carefully reviewed in six problems using the real-valued input as a description of an actual state. Most of them have been used as a toy-problems in LCS research, providing different problems of different natures, like:

- single and multiple steps,
- extensive mutual feature interaction,
- vast input space,
- the need for long-action chain building.

Moreover, a famous RL *Cart Pole* benchmarking problem was investigated by the LCS for the first time, to our knowledge, obtaining encouraging results with compact knowledge representation.

To highlight performance, five key metrics were selected that investigate the state of the systems from multiple angles. Aspects like the quality of evolving solutions, size, and effective application are emphasized throughout the research.

3. Propose relevant improvements towards neoteric changes

This work's primary effort is to investigate the ALCS behaviour with either interval-based rule representation (rvACS) or by the use of discretization. The rvACS adaptation requires redefining certain internal processes and introducing new system parameters. Preliminary tests revealed two possible limitations of the system that were further studied.

First, any form of continuous-valued representation increased the problem's search space, impeding the agent's learning speed. Four techniques probing for the most promising action selection were inspected, especially regarding knowledge acquisition rate. The novel approach, based on the promising approach of *Optimistic Initial Values* in multi-armed bandits, named *Optimistic Initial Quality* was proposed herein but did not provide substantial performance gains.

The other limitation relates to multistep problems, where certain input discretization might demand the agent to perform a notably larger number of actions to receive the feedback signal. The default reward assignment component might distribute the signal incorrectly amongst participating classifiers when such long-action chains are experienced. An approach replacing the credit assignment with the undiscounted incentive distribution version resulted in a system named AACS2. The method showed performance improvements for specific problems.

The biased exploration research was published in [68], and the performance of long-action chaining in [69].

4. Conduct an experimental evaluation of intended adjustments

All the experiments were tried to illustrate the evolution of selected metrics throughout the agent's learning of selected problems. Such characteristics of investigated strategies of ALCS implementations were further smoothed by averaging multiple independent experiments.

Moreover, the Bayesian estimation approach emphasized the differences between investigated cases, allowing to draw non-biased conclusions. Often, selected environments were scaled up, where agents' behaviour was challenged with increased complexity.

5. Developing an open-sourced Python Machine Learning framework for evaluating various LCS algorithms

LCS algorithms are still not recognized by many ML practitioners and researchers. Significant effort was put to implement the most famous ALCS algorithms using the tools used nowadays by data scientist communities. Despite a limited number of reference

resources, the historical experiments were reproduced, drastically simplifying newcomers learning curve. The refreshed version of code adds adds several abstraction layers, facilitating the introduction of new features like handling real-valued input.

By integrating with the industry standards, all algorithms share the same code base and operate with a commonly agreed interface enabling effortless benchmarking with other state-of-the-art systems. The incentives like open-source availability, a vast amount of usage examples, possibility of interactive execution are intended to shed more light on this promising yet underappreciated field of ML.

The developed *PyALCS*[1] framework was described in [66] research paper.

6.2 Future Work

The ideas presented in this thesis may be potentially developed in the following directions:

- Research on compacting the created population [P] [80]. The current solution evolves a set of distinct classifiers representing fine-grained niches of search space. An online process capable of merging neighbouring rules would positively impact the overall performance and transparency.
- Examine the impact of choosing different discretization strategies for specific problems. Example methods might include supervised learners (using class information like *entropy* or *error* to determine cut-points) or hierarchical, where the ranges are built incrementally [145].
- Investigation of agent's behaviour for non-stationary and non-deterministic environments. Problems like noise perception, of *concept drifting* where the agent has to refine population of classifiers over time.
- Utilization of specific ALCS mechanisms like *action planning* to optimize the process of building an internal model of the real-valued environment.
- Efficient way of tuning system parameters. ALCS comprises numerous hyper-parameters; therefore, employing a heuristic process would contribute to better performance.

[1] https://github.com/ParrotPrediction/pyalcs

6.3 Publications

The selected parts of the thesis have been already published in:

- Norbert Kozlowski and Olgierd Unold. *Integrating anticipatory classifier systems with OpenAI Gym.* In Proceedings of the Genetic and Evolutionary Computation Conference Companion, 1410–1417. ACM, 2018 (**CORE**: A, **MNiSW**: 140).
- Norbert Kozlowski and Olgierd Unold. *Preliminary tests of a real-valued anticipatory classifier system.* In Proceedings of the Genetic and Evolutionary Computation Conference Companion, 1289–1294. 2019 (**CORE**: A, **MNiSW**: 140).
- Norbert Kozlowski and Olgierd Unold. *Investigating exploration techniques for ACS in discretized real-valued environments.* In Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion, 1765–1773, 2020 (**CORE**: A, **MNiSW**: 140).
- Norbert Kozłowski and Olgierd Unold. *Anticipatory classifier system with average reward criterion in discretized multi-step environments.* Applied Sciences, 11(3):1098, 2021 (**IF**: 02.679, **MNiSW**: 100).
- Norbert Kozłowski and Olgierd Unold - *Internalizing Knowledge for Anticipatory Classifier Systems in Discretized Real-Valued Environments.* IEEE Access (10), 33816-33828, 2022 (**IF**: 03.557, **MNiSW**: 100)

During the work on the thesis I have also coauthored other research:

- Olgierd Unold, Edyta Rogula, and Norbert Kozłowski. *Introducing action planning to the anticipatory classifier system ACS2.* In International Conference on Computer Recognition Systems, 264–275. Springer, 2019 (**CORE**: National, **MNiSW**: 20),
- Olgierd Unold, Norbert Kozłowski, Łukasz Śmierzchała. *Preliminary tests of an Anticipatory Classifier System with Experience Replay*, Gecco 2022 (in review).

Selected research advancements were presented at the following conferences:

- Kraków, Poland, 2018, QIPLSIGML - *PyALCS and OpenAI Gym*,
- Kyoto, Japan, 2018, GECCO - *Integrating anticipatory classifier systems with OpenAI Gym*,

- Wrocław, Poland, 2019, PP-RAI - *Preliminary tests of a real-valued Anticipatory Classifier System*,
- Prague, Czech Republic, 2019, GECCO - *Preliminary tests of a real-valued Anticipatory Classifier System*,
- Cancún, Mexico, 2020, GECCO - *Investigating Exploration Techniques for Anticipatory Classifier System in Real-Valued Environments*,
- Gdynia, Poland, 2022, PP-RAI - *Internalizing Knowledge for Anticipatory Classifier Systems in Discretized Real-Valued Environments.*

The interactive [2] and reproducible [3] version of this thesis is available online.

[2] https://khozzy.github.io/phd
[3] https://zenodo.org/record/6427066

Bibliography

[1] Hussein A. Abbass, Jaume Bacardit, Martin V. Butz, and Xavier Llora. Online adaptation in learning classifier systems: stream data mining. *Illinois Genetic Algorithms Laboratory*, (2004031), 2004.

[2] Jaume Bacardit, Ester Bernadó-Mansilla, and Martin V. Butz. Learning classifier systems: looking back and glimpsing ahead. In *Learning Classifier Systems*, pages 1–21. Springer, 2006.

[3] Jaume Bacardit and Martin V. Butz. Data mining in learning classifier systems: comparing XCS with GAssist. In *Learning Classifier Systems*, pages 282–290. Springer, 2003.

[4] Alwyn Barry. *XCS Performance and Population Structure in Multi-Step Environments*. PhD thesis, Citeseer, 2000.

[5] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5):834–846, 1983.

[6] Alessio Benavoli, Giorgio Corani, Janez Demšar, and Marco Zaffalon. Time for a change: a tutorial for comparing multiple classifiers through bayesian analysis. *The Journal of Machine Learning Research*, 18(1):2653–2688, 2017.

[7] Ester Bernadó, Xavier Llora, and Josep M. Garrell. XCS and GALE: A comparative study of two learning classifier systems on data mining. In *International Workshop on Learning Classifier Systems*, pages 115–132. Springer, 2001.

[8] Andrea Bonarini, Claudio Bonacina, and Matteo Matteucci. Fuzzy and crisp representations of real-valued input for learning classifier systems. In *International Workshop on Learning Classifier Systems*, pages 107–124. Springer, 1999.

[9] Andrea Bonarini and Matteo Matteucci. Fixcs: A fuzzy implementation of XCS. In *2007 IEEE International Fuzzy Systems Conference*, pages 1–6. IEEE, 2007.

[10] Lashon B. Booker. Improving the performance of genetic algorithms in classifier systems. In *ICGA*, pages 80–92, 1985.

[11] William N. Browne. The development of an industrial learning classifier system for data-mining in a steel hop strip mill. In *Applications of learning classifier systems*, pages 223–259. Springer, 2004.

[12] Seok-Jun Bu and Sung-Bae Cho. A convolutional neural-based learning classifier system for detecting database intrusion via insider attack. *Information Sciences*, 512:123–136, 2020.

[13] Larry Bull. *Applications of learning classifier systems*, volume 150. Springer Science & Business Media, 2004.

[14] Larry Bull and Tim Kovacs. Foundations of learning classifier systems: An introduction. In *Foundations of Learning Classifier Systems*, pages 1–17. Springer, 2005.

[15] Larry Bull, Pier Luca Lanzi, and Toby O'hara. Anticipation mappings for learning classifier systems. In *2007 IEEE Congress on Evolutionary Computation*, pages 2133–2140. IEEE, 2007.

[16] A Martin V Butz, B David E Goldberg, and C Wolfgang Stolzmann. The anticipatory classifier system and genetic generalization. *Natural Computing*, 1(4):427–467, 2002.

[17] Martin V. Butz. Biasing exploration in an anticipatory learning classifier system. In *International Workshop on Learning Classifier Systems*, pages 3–22. Springer, 2001.

[18] Martin V. Butz. *Anticipatory learning classifier systems*, volume 4. Springer Science & Business Media, 2002.

[19] Martin V. Butz. *Rule-based evolutionary online learning systems*, volume 259. Springer, 2006.

[20] Martin V. Butz. Combining gradient-based with evolutionary online learning: an introduction to learning classifier systems. In *7th International Conference on Hybrid Intelligent Systems (HIS 2007)*, pages 12–17. IEEE, 2007.

[21] Martin V Butz, David E Goldberg, and Wolfgang Stolzmann. Introducing a genetic generalization pressure to the anticipatory classifier system-part 1: Theoretical approach. In *GECCO*, pages 42–49. Citeseer, 2000.

[22] Martin V Butz, David E Goldberg, and Wolfgang Stolzmann. Introducing a genetic generalization pressure to the anticipatory classifier system-part 2: Performance analysis. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pages 42–49. Citeseer, 2000.

[23] Martin V. Butz and Wolfgang Stolzmann. An algorithmic description of ACS2. In *International Workshop on Learning Classifier Systems*, pages 211–229. Springer, 2001.

[24] Martin V. Butz and Stewart W. Wilson. An algorithmic description of XCS. In *International Workshop on Learning Classifier Systems*, pages 253–272. Springer, 2000.

[25] Jorge Casillas, Brian Carse, and Larry Bull. Fuzzy-XCS: A michigan genetic fuzzy system. *IEEE Transactions on Fuzzy Systems*, 15(4):536–550, 2007.

[26] Jorge Casillas, Brian Carse, Larry Bull, and Brian Carse. Fuzzy XCS: an accuracy-based fuzzy classifier system. In *Proceedings of the XII Congreso Espanol sobre Tecnologia y Logica Fuzzy (ESTYLF 2004)*, pages 369–376, 2004.

[27] Łukasz Cielecki and Olgierd Unold. Real-valued GCS classifier system. *International Journal of Applied Mathematics and Computer Science*, 17(4):539–547, 2007.

[28] Oscar Cord et al. *Genetic fuzzy systems: evolutionary tuning and learning of fuzzy knowledge bases*, volume 19. World Scientific, 2001.

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

[30] Hai H. Dam, Hussein A. Abbass, Chris Lokan, and Xin Yao. Neural-based learning classifier systems. *IEEE Transactions on Knowledge and Data Engineering*, 20(1):26–39, 2007.

[31] Hai H. Dam, Chris Lokan, and Hussein A. Abbass. Evolutionary online data mining: An investigation in a dynamic environment. In *Evolutionary computation in dynamic and uncertain environments*, pages 153–178. Springer, 2007.

[32] Tapas K. Das, Abhijit Gosavi, Sridhar Mahadevan, and Nicholas Marchalleck. Solving semi-markov decision problems using average reward reinforcement learning. *Management Science*, 45(4):560–574, Apr 1999.

[33] Li Deng. The MNIST database of handwritten digit images for machine learning research. *IEEE signal processing magazine*, 29(6):141–142, 2012.

[34] Zoltan Dienes. Bayesian versus orthodox statistics: Which side are you on? *Perspectives on Psychological Science*, 6(3):274–290, 2011.

[35] Tapio Elomaa and Juho Rousu. Efficient multisplitting revisited: Optima-preserving elimination of partition candidates. *Data Mining and Knowledge Discovery*, 8(2):97–126, 2004.

[36] Pierre Gérard, Jean-Arcady Meyer, and Olivier Sigaud. Combining latent learning with dynamic programming in the modular anticipatory classifier system. *European Journal of Operational Research*, 160(3):614–637, 2005.

[37] Pierre Gerard and Olivier Sigaud. YACS: Combining dynamic programming with generalization in classifier systems. In *International Workshop on Learning Classifier Systems*, pages 52–69. Springer, 2000.

[38] Pierre Gerard, Wolfgang Stolzmann, and Olivier Sigaud. YACS: a new learning classifier system using anticipation. *Soft Computing*, 6(3-4):216–228, 2002.

[39] David E. Goldberg. Genetic algorithms in search, optimization, and machine learning, 1989.

[40] David E. Goldberg and Holland John H. Genetic algorithms and machine learning. *Machine Learning*, 3(2):95–99, 1988.

[41] David Gunning, Mark Stefik, Jaesik Choi, Timothy Miller, Simone Stumpf, and Guang-Zhong Yang. XAI — explainable artificial intelligence. *Science Robotics*, 4(37):eaay7120, 2019.

[42] Ali Hamzeh and Adel Rahmani. An evolutionary function approximation approach to compute prediction in XCSF. In *European Conference on Machine Learning*, pages 584–592. Springer, 2005.

[43] Tim Hansmeier and Marco Platzner. An experimental comparison of explore/exploit strategies for the learning classifier system XCS. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1639–1647, 2021.

[44] Johann Friedrich Herbart. *Psychologie als Wissenschaft: neu gegründet auf Erfahrung, Metaphysik und Mathematik*, volume 2. AW Unzer, 1825.

[45] Joachim Hoffmann. *Vorhersage und erkenntnis*. Hogrefe, 1993.

[46] Joachim Hoffmann and Albrecht Sebald. Lernmechanismen zum erwerb verhaltenssteuernden wissens. *Psychologische Rundschau*, 2000.

[47] John H. Holladn. Properties of the bucket brigade. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 1–7, 1985.

[48] John H. Holland. Adaptation in natural and artificial systems. *Ann Arbor*, 1975.

[49] John H. Holland. Progress in theoretical biology. *Adaptation*, 4:264–293, 1976.

[50] John H. Holland. Properties of the bucket brigade. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 1–7, 1985.

[51] John H. Holland. Escaping brittleness: the possibilities of general purpose learning algorithms applied to parallel rule-based system. *Machine learning*, pages 593–623, 1986.

[52] John H. Holland. A mathematical framework for studying learning in classifier systems. *Physica D: Nonlinear Phenomena*, 22(1-3):307–317, 1986.

[53] John H. Holland. *Hidden order: How adaptation builds complexity*. Addison Wesley Longman Publishing Co., Inc., 1996.

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

[55] John H. Holmes. Learning classifier systems applied to knowledge discovery in clinical research databases. In *International Workshop on Learning Classifier Systems*, pages 243–261. Springer, 1999.

[56] John H. Holmes, Dennis R. Durbin, and Flaura K. Winston. The learning classifier system: an evolutionary computation approach to knowledge discovery in epidemiologic surveillance. *Artificial Intelligence in Medicine*, 19(1):53–74, 2000.

[57] John H. Holmes, Pier L. Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson. Learning classifier systems: New models, successful applications. *Information Processing Letters*, 82(1):23–30, 2002.

[58] Jacob Hurst, Larry Bull, and Chris Melhuish. TCS learning classifier system controller on a real robot. In *International Conference on Parallel Problem Solving from Nature*, pages 588–597. Springer, 2002.

[59] Muhammad Irfan, Zheng Jiangbin, Muhammad Iqbal, and Muhammad Hassan Arif. Enhancing learning classifier systems through convolutional autoencoder to classify underwater images. *Soft Computing*, 25(15):10423–10440, 2021.

[60] Muhammad Irfan, Zheng Jiangbin, Muhammad Iqbal, Zafar Masood, and Muhammad Hassan Arif. Knowledge extraction and retention based continual learning by using convolutional autoencoder-based learning classifier system. *Information Sciences*, 591:287–305, 2022.

[61] Muhammad Irfan, Zheng Jiangbin, Muhammad Iqbal, Zafar Masood, Muhammad Hassan Arif, and Syed Rauf ul Hassan. Brain inspired lifelong learning model based on neural based learning classifier system for underwater data classification. *Expert Systems with Applications*, 186:115798, 2021.

[62] Mohammad R. Islam. Sample size and its role in Central Limit Theorem (CLT). *Computational and Applied Mathematics Journal*, 4(1):1–7, 2018.

[63] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[64] Dawid Kondziela. *Design and implementation of a fuzzy anticipatory classifier system.* Bachelor's thesis, Wroclaw Univesity of Science and Technology, 2021.

[65] Sotiris Kotsiantis and Dimitris Kanellopoulos. Discretization techniques: A recent survey. *GESTS International Transactions on Computer Science and Engineering*, 32(1):47–58, 2006.

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

[67] Norbert Kozłowski and Olgierd Unold. Preliminary tests of a real-valued anticipatory classifier system. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1289–1294, 2019.

[68] Norbert Kozłowski and Olgierd Unold. Investigating exploration techniques for ACS in discretized real-valued environments. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, pages 1765–1773, 2020.

[69] Norbert Kozłowski and Olgierd Unold. Anticipatory classifier system with average reward criterion in discretized multi-step environments. *Applied Sciences*, 11(3):1098, 2021.

[70] Norbert Kozłowski and Olgierd Unold. Internalizing knowledge for anticipatory classifier systems in discretized real-valued environments. *IEEE Access*, 2022. (in review).

[71] John K. Kruschke. Bayesian assessment of null values via parameter estimation and model comparison. *Perspectives on Psychological Science*, 6(3):299–312, 2011.

[72] John K. Kruschke. Bayesian Estimation Supersedes the t Test. *Journal of Experimental Psychology: General*, 142(2):573, 2013.

[73] Sang Gyu Kwak and Jong Hae Kim. Central limit theorem: the cornerstone of modern statistics. *Korean journal of anesthesiology*, 70(2):144, 2017.

[74] Pier L. Lanzi. Learning classifier systems: A reinforcement learning perspective. In *Foundations of Learning Classifier Systems*, pages 267–284. Springer, 2005.

[75] Pier L. Lanzi. Learning classifier systems: then and now. *Evolutionary Intelligence*, 1(1):63–82, 2008.

[76] Pier L. Lanzi et al. Extending the representation of classifier conditions part i: from binary to messy coding. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 99)*, pages 337–344. Morgan Kaufmann, 1999.

[77] Pier L. Lanzi, Daniele Loiacono, Stewart W. Wilson, and David E. Goldberg. XCS with computed prediction in multistep environments. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 1859–1866, 2005.

[78] Pier L. Lanzi and Rick L. Riolo. Recent trends in learning classifier systems research. In *Advances in Evolutionary Computing*, pages 955–988. Springer, 2003.

[79] Ju Yin Lin, Chi Pin Cheng, Wen Chih Tsai, and An-Pin Chen. Using learning classifier system for making investment strategies based on institutional analysis. In *Artificial Intelligence and Applications*, pages 765–769, 2004.

[80] Yi Liu, Will N Browne, and Bing Xue. A comparison of learning classifier systems' rule compaction algorithms for knowledge visualization. *ACM Transactions on Evolutionary Learning and Optimization*, 1(3):1–38, 2021.

[81] Yi Liu, Will N Browne, and Bing Xue. Visualizations for rule-based machine learning. *Natural Computing*, pages 1–22, 2021.

[82] Sridhar Mahadevan. Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine learning*, 22(1-3):159–195, 1996.

[83] Sridhar Mahadevan. Sensitive discount optimality: Unifying discounted and average reward reinforcement learning. In *ICML*, pages 328–336. Citeseer, 1996.

[84] Harmon E. Mance and Stephanie S. Harmon. Reinforcement learning: A tutorial, 1996.

[85] Charles E Martin and Heiko Hoffmann. Fast re-learning of a controller from sparse data. In *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 973–978. IEEE, 2014.

[86] Kazuma Matsumoto, Ryo Takano, Takato Tatsumi, Hiroyuki Sato, Tim Kovacs, and Keiki Takadama. XCSR based on compressed input by deep neural network for high dimensional data. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1418–1425, 2018.

[87] Jean-Arcady Meyer and Stewart W. Wilson. The animat path to ai. In *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 15–21. MIT Press, 1991.

[88] Petr Musilek, Sa Li, and Loren Wyard-Scott. Enhanced learning classifier system for robot navigation. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3390–3395. IEEE, 2005.

[89] Toby O'Hara and Larry Bull. Prediction calculation in accuracy-based neural learning classifier systems. *Tech report UWELCSG04-004*, 2004.

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

[91] Romain Orhand, Anne Jeannin-Girardon, Pierre Parrend, and Pierre Collet. BACS: A thorough study of using behavioral sequences in ACS2. In *International Conference on Parallel Problem Solving from Nature*, pages 524–538. Springer, 2020.

[92] Romain Orhand, Anne Jeannin-Girardon, Pierre Parrend, and Pierre Collet. PEPACS: integrating probability-enhanced predictions to ACS2. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*, pages 1774–1781, 2020.

[93] Richard J Preen, Stewart W Wilson, and Larry Bull. Autoencoding with a classifier system. *IEEE Transactions on Evolutionary Computation*, 25(6):1079–1090, 2021.

[94] Martin L. Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

[95] Martin L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[96] Robert A Richards and Sheri D Sheppard. Three-dimensional shape optimization utilizing a learning classifier system. In *Proceedings of the 1st annual conference on genetic programming*, pages 539–546, 1996.

[97] Michael M. Richter. Reinforcement learning: a brief overview. *Adaptivity and Learning*, pages 243–264, 2003.

[98] Rick L. Riolo. *Empirical studies of default hierarchies and sequences of rules in learning classifier systems.* PhD thesis, University of Michigan, 1988.

[99] Rick L. Riolo. Lookahead planning and latent learning in a classifier system. In *Proceedings of the First International Conference on Simulation of Adaptive Behavior on From Animals to Animats*, page 316–326, Cambridge, MA, USA, 1991. MIT Press.

[100] George G. Robertson and Rick L. Riolo. A tale of two classifier systems. *Machine Learning*, 3(2):139–159, 1988.

[101] R Rosen. Life itself, complexity in ecological systems series, 1991.

[102] Robert Rosen. *Anticipatory Systems.* Elsevier, 1985.

[103] Katarzyna Rudnik. *Koncepcja i implementacja systemu wnioskującego z probabilistyczno-rozmytą bazą wiedzy.* PhD thesis, Politechnika Opolska, 2011.

[104] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 3 edition, 2010.

[105] Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 44(1.2):206–226, 2000.

[106] Anton Schwartz. A reinforcement learning method for maximizing undiscounted rewards. In *Proceedings of the tenth international conference on machine learning*, volume 298, pages 298–305, 1993.

[107] Abubakar Siddique, Will N Browne, and Gina M Grimshaw. Frames-of-reference based learning: Overcoming perceptual aliasing in multi-step decision making tasks. *IEEE Transactions on Evolutionary Computation*, 2021.

[108] Olivier Sigaud and Stewart W. Wilson. Learning classifier systems: a survey. *Soft Computing*, 11(11):1065–1078, 2007.

[109] Satinder P. Singh. Reinforcement learning algorithms for average-payoff markovian decision processes. In *AAAI*, volume 94, pages 700–705, 1994.

[110] Robert Smith, A. El-Fallah, B. Ravichandran, Raman Mehra, and Bruce Dike. *The Fighter Aircraft LCS: A Real-World, Machine Innovation Application*, pages 113–142. Springer, 01 2004.

[111] Stephen Smith. *A learning system based on genetic algorithms.* PhD thesis, University of Pittsburgh, 1980.

[112] Stephen F. Smith. Flexible learning of problem solving heuristics through adaptive search. In *IJCAI*, volume 83, pages 422–425, 1983.

[113] Wolfgang Stolzmann. *Antizipative classifier systems*. Shaker, 1997.

[114] Wolfgang Stolzmann. An introduction to anticipatory classifier systems. In *International Workshop on Learning Classifier Systems*, pages 175–194. Springer, 1999.

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

[116] Wolfgang Stolzmann, Martin V. Butz, and David E. Goldberg. First cognitive capabilities in the anticipatory classifer system. *et al.[228]*, pages 287–296, 2000.

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

[118] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[119] Masakazu Tadokoro, Satoshi Hasegawa, Takato Tatsumi, Hiroyuki Sato, and Keiki Takadama. Knowledge extraction from XCSR based on dimensionality reduction and deep generative models. In *2019 IEEE Congress on Evolutionary Computation (CEC)*, pages 1883–1890. IEEE, 2019.

[120] Kurian Tharakunnel and David E. Goldberg. XCS with average reward criterion in multi-step environment, 2002.

[121] Edward C. Tolman. Cognitive maps in rats and men. *Psychological review*, 55(4):189, 1948.

[122] Filip Tóth, Kristína Rebrová, Gregor Zat'ko, Pavol Krasňanskỳ, and Boris Roha'l-Ilkiv. Mobile robot control using XCS. In *2013 International Conference on Process Control (PC)*, pages 504–509. IEEE, 2013.

[123] Wen-Chih Tsai and An-Pin Chen. Using the XCS classifier system for portfolio allocation of MSCI index component stocks. *Expert Systems with Applications*, 38(1):151–154, 2011.

[124] Michalis T Tsapanos, Kyriakos C Chatzidimitriou, and Pericles A Mitkas. A zeroth-level classifier system for real time strategy games. In *2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*, volume 2, pages 244–247. IEEE, 2011.

[125] Olgierd Unold. Context-free grammar induction with grammar-based classifier system. *Archives of Control Science*, 15(4):681, 2005.

[126] Olgierd Unold and Marcin Mianowski. Real-valued ACS classifier system: A preliminary study. In *Proceedings of the 9th International Conference on Computer Recognition Systems CORES 2015*, pages 203–211. Springer, 2016.

[127] Olgierd Unold, Edyta Rogula, and Norbert Kozłowski. Introducing action planning to the anticipatory classifier system ACS2. In *International Conference on Computer Recognition Systems*, pages 264–275. Springer, 2019.

[128] Ryan J. Urbanowicz and Will N. Browne. *Introduction to learning classifier systems*. Springer, 2017.

[129] Ryan J. Urbanowicz and Jason H. Moore. Learning classifier systems: a complete introduction, review, and roadmap. *Journal of Artificial Evolution and Applications*, 2009, 2009.

[130] Ronald L. Wasserstein and Nicole A. Lazar. The ASA statement on p-values: context, process, and purpose, 2016.

[131] Christopher Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Oxford, 1989.

[132] Stewart W. Wilson. Knowledge growth in an artificial animal. In *Adaptive and Learning Systems*, pages 255–264. Springer, 1986.

[133] Stewart W. Wilson. ZCS: A zeroth level classifier system. *Evolutionary computation*, 2(1):1–18, 1994.

[134] Stewart W. Wilson. Classifier fitness based on accuracy. *Evolutionary computation*, 3(2):149–175, 1995.

[135] Stewart W. Wilson. Generalization in the XCS classifier system. In *Genetic Programming. Proceedings of the Third Annual Conference*, 1998.

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

[137] Stewart W. Wilson. State of XCS classifier system research. In *International workshop on learning classifier systems*, pages 63–81. Springer, 1999.

[138] Stewart W. Wilson. Mining oblique data with XCS. In *International Workshop on Learning Classifier Systems*, pages 158–174. Springer, 2000.

[139] Stewart W. Wilson. Compact rulesets from XCSI. In *International Workshop on Learning Classifier Systems*, pages 197–208. Springer, 2001.

[140] Stewart W. Wilson. Function approximation with a classifier system. In *Proc. 3rd Genetic and Evolutionary Computation Conf.(GECCO'01)*, pages 974–981, 2001.

[141] Stewart W Wilson. Classifiers that approximate functions. *Natural Computing*, 1(2):211–234, 2002.

[142] Stewart W. Wilson and David E. Goldberg. A critical review of classifier systems. In *Proceedings of the third international conference on Genetic algorithms*, pages 244–255, 1989.

[143] Ian H. Witten and Eibe Frank. Data mining: practical machine learning tools and techniques with java implementations. *Acm Sigmod Record*, 31(1):76–77, 2002.

[144] David Wyatt and Larry Bull. Building compact rulesets for describing continuous-valued problem spaces using a learning classifier system. In *Adaptive computing in design and manufacture VI*, pages 235–246. Springer, 2004.

[145] Ying Yang, Geoffrey I Webb, and Xindong Wu. Discretization methods. In *Data mining and knowledge discovery handbook*, pages 101–116. Springer, 2009.

[146] Lotfi A. Zadeh. Fuzzy sets. In *Fuzzy sets, fuzzy logic, and fuzzy systems: selected papers*, pages 394–432. World Scientific, 1996.

[147] Zhaoxiang Zang, Dehua Li, Junying Wang, and Dan Xia. Learning classifier system with average reward reinforcement learning. *Knowledge-Based Systems*, 40:58–71, 2013.