A Genetic Fuzzy System for Interpretable and Parsimonious Reinforcement Learning Policies

Jordan T. Bishop
The University of Queensland, Australia
j.bishop@uq.edu.au

Marcus Gallagher
The University of Queensland, Australia
marcusg@uq.edu.au

Will N. Browne
Queensland University of Technology, Australia
will.browne@qut.edu.au

ABSTRACT

Reinforcement learning (RL) is experiencing a resurgence in research interest, where Learning Classifier Systems (LCSs) have been applied for many years. However, traditional Michigan approaches tend to evolve large rule bases that are difficult to interpret or scale to domains beyond standard mazes. A Pittsburgh Genetic Fuzzy System (dubbed Fuzzy MoCoCo) is proposed that utilises both multiobjective and cooperative coevolutionary mechanisms to evolve fuzzy rule-based policies for RL environments. Multiobjectivity in the system is concerned with policy performance vs. complexity. The continuous state RL environment Mountain Car is used as a testing bed for the proposed system. Results show the system is able to effectively explore the trade-off between policy performance and complexity, and learn interpretable, high-performing policies that use as few rules as possible.

CCS CONCEPTS

• **Computing methodologies** → **Rule learning**; **Genetic algorithms**; **Reinforcement learning**; **Vagueness and fuzzy logic**;

KEYWORDS

Genetic Fuzzy Systems, Reinforcement Learning, Multiobjective Optimisation, Cooperative Coevolution, Direct Policy Search

ACM Reference Format:
Jordan T. Bishop, Marcus Gallagher, and Will N. Browne. 2021. A Genetic Fuzzy System for Interpretable and Parsimonious Reinforcement Learning Policies. In *2021 Genetic and Evolutionary Computation Conference Companion (GECCO '21 Companion), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3449726.3463198

1 INTRODUCTION

Genetics-Based Machine Learning (GBML) [13] has a long history of being applied to reinforcement learning (RL) problems where new methods are needed to take advantage of the renewed interest in such domains. Genetic Fuzzy Systems (GFSs) [5] are a type of GBML that aim to evolve fuzzy rule-based systems (FRBSs). Also under the umbrella of GBML, Learning Classifier Systems (LCSs) are a family of evolutionary rule-based systems that create solutions to machine learning problems.

Within both families, there are two broad types of systems that represent different ways to solve a problem: the Michigan and Pittsburgh approaches. Both approaches utilise population-based evolutionary mechanisms. In a Michigan system, each individual in the population is an element of the solution; all individuals act in ensemble to create the entire solution. In contrast, a Pittsburgh system treats each individual in the population as an entire solution to the problem [5, 23]. Within GFSs, a wide array of works have focused on the Pittsburgh approach [5, 8], while in the LCS literature the predominant paradigm is Michigan [23].

Both LCSs and GFSs can be applied to RL problems. LCSs were originally designed to perform RL, and much work has been done in this area already, particularly in maze-like environments, e.g. [14, 16]. In contrast, most GFS work has focused on supervised learning: classification or regression [8], with some work being done on "control" problems, e.g. [9]. However, such control problems are often not formulated under the RL framework; this framework prescribes problems that are multi-step, involve delayed rewards, and are characterised by two fundamental issues: the explore-exploit dilemma and temporal credit assignment. Michigan and Pittsburgh systems address these issues at different levels of abstraction. Michigan systems learn in an online fashion, and they address both issues at the level of individual state-action pairs within a stream of experience. Particularly in problems where exploration is difficult and/or reward signals are sparse, this can be difficult to achieve. On the other hand, Pittsburgh systems assign credit to entire solutions, and address the explore-exploit problem in the more abstract policy parameter space.

Generally, there is a lack of work applying LCSs to common environments from the RL literature that are not maze-like, e.g. Mountain Car or Cart Pole [21], an exception being [20]. Such environments often have continuous state spaces. Since LCSs prescribe a paradigm of learning rather than a specific algorithm, they enable the representation of rule conditions to be flexibly chosen to suit the problem domain. For continuous domains, there are a variety of choices available, some examples being hyperrectangles [16], hyperellipsoids [2], and fuzzy logic [24]. Fuzzy logic attempts to perform inference in a way that better emulates how a human expert may solve a problem by including degrees of truth rather than absolute values. It is an attractive representation to use if the purpose of the system is to produce a human-understandable explanation of how a problem is solved.

An issue for both LCSs and GFSs is how to deal with the complexity of rule bases that are evolved in order to prefer parsimonious (low complexity) models. In the Michigan approach, a post hoc

GECCO '21 Companion, July 10–14, 2021, Lille,France

ACM ISBN 978-1-4503-8351-6/21/07…$15.00
https://doi.org/10.1145/3449726.3463198

compaction mechanism is often employed to remove rules that do not contribute much to the solution [23]. In contrast, more options are available for Pittsburgh systems, some being: i) limiting the size of candidate solutions, ii) employing fitness penalisation based on complexity ([1] is an approach that uses the Minimum Description Length principle in this manner), iii) multiobjective (MO) formulation of solution performance vs. complexity [11, 15]. The third strategy is particularly attractive if the practitioner desires to understand the trade-off between performance and complexity; and to understand *how many rules are needed to achieve a given performance value.*

Therefore, the first objective of this work is to address an RL problem that incorporates both i) difficult exploration, and ii) a continuous state space (a candidate is Mountain Car, see justification Section 4). Objective two is to understand the trade-off between rule base complexity and performance, through employing a Pittsburgh GFS that performs MO optimisation of FRBSs. Finally, since an FRBS can be naturally decomposed into a rule base (RB) and a data base (DB) (see Section 2.2.1), the third objective is to show that cooperative coevolution (CoCo) can be employed to jointly optimise the RB and DB. Thus, the overall aim of this work is to develop a Pittsburgh GFS that utilises CoCo and MO mechanisms to produce parsimonious and interpretable policies for RL problems. As a proof of concept, we show that this system is able to produce compact and interpretable policies for the Mountain Car problem.

2 BACKGROUND

2.1 Reinforcement Learning

In RL, an agent interacts with an (episodic) environment $\mathcal{E}$ to maximise the expected amount of cumulative reward it receives. Let $\mathcal{E} = (S, A, P, R, \gamma, t_{\max})$, where S is the state space, A is the action space, $P(s'|s, a)$ is the transition function, $R(s, a, s')$ is the reward function, $0 \leq \gamma \leq 1$ is the discount factor, and $t_{\max}$ is the maximum number of episode time steps [21]. In this work, we assume S is continuous and A is discrete. The agent takes the form of a policy, which is a mapping $\pi : S \rightarrow A$. A common way to address RL problems is for the agent to construct an *action-value function* $Q : S \times A \rightarrow \mathbb{R}$ which represents the expected cumulative reward obtainable from each state-action pair. A policy can then be constructed by acting greedily with respect to Q. This is the approach followed by Michigan systems.

An alternative way to address the problem is to treat the environment as a black box and perform *direct policy search*; an approach taken by Pittsburgh systems. In this view, the agent receives feedback about its performance via a collective sum of discounted rewards, termed the *return*: G [21]. The task is to construct a policy that directly maximises the *expected return*, without decomposing the return into individual rewards. The expected return (performance) of a policy is measured over a set of initial states Z of cardinality η drawn from an initial state space $S_I \subseteq S$. Using this formulation, the performance of a policy, abbreviated perf, can be measured as:

$$\text{perf} = \frac{1}{\eta} \sum_{z \in Z} G(z) \tag{1}$$

where $G(z)$ is the return yielded by performing a rollout of the policy, starting at initial state z.

2.2 Fuzzy Rule-Based Systems

We use the following terminology when discussing aspects of fuzzy reasoning: *Linguistic variable* — analogous to an environmental feature; includes both the name of the feature and a fuzzy partition along its domain. A fuzzy partition is composed of multiple fuzzy sets. *Fuzzy set* — defined by a membership function along the domain of a linguistic variable; has an associated name or *linguistic value* to linguistically describe the set [5].

2.2.1 FRBS Structure. The specific FRBS type considered is a zero-order Takagi-Sugeno-Kang system [22]. The FRBS is composed of two components, that together form the knowledge base: the rule base (RB) and data base (DB) [5]. The rule base contains the fuzzy rules that act in the context of the fuzzy partitions contained in the DB. In the RB, we use fuzzy rules that are individually expressed in Conjunctive Normal Form (CNF) [5]. Assuming the dimensionality of S is d and there are k possible actions ($|A| = k$), each rule has the structure:

$$\begin{gathered}\text{IF } x_1 \text{ is } \widetilde{L_1} = \{L_{(1,1)} \text{ or } \ldots \text{ or } L_{(1,m_1)}\} \text{ and } \ldots \\ \text{and } x_d \text{ is } \widetilde{L_d} = \{L_{(d,1)} \text{ or } \ldots \text{ or } L_{(d,m_d)}\} \\ \text{THEN } y_1 = \alpha_1, \ldots, y_k = \alpha_k\end{gathered}$$

where, for $i \in \{1, \ldots, d\}$:

- $x_1, \ldots, x_d$ — components of input vector x; linguistic variables
- m_i — num. linguistic values belonging to i^{th} linguistic variable
- $L_{(i,j)}, j \in \{1, \ldots, m_i\}$ — j^{th} linguistic value of i^{th} linguistic variable
- $\widetilde{L_i}$ — non-empty set of linguistic values for i^{th} linguistic variable
- $y_1, \ldots, y_k$ — components of consequent vector y
- $\alpha_1, \ldots, \alpha_k$ — voting weights for each action in consequent

$\alpha_1, \ldots, \alpha_k$ are constrained to be either 0 (inactive) or 1 (active), with exactly one weight active in every rule, all others inactive, i.e. a one-hot encoding. Such a scheme represents each rule voting (fully) for a single action in its consequent. When writing rule consequents we simply specify the action whose weight is active (*a is k*). This type of rule allows for flexible levels of generalisation. Selecting all linguistic values of a linguistic variable is equivalent to a "don't care", denoted by #. Note that it is not possible to select *zero* linguistic values; as stated above the set of linguistic values must be non-empty. As an example, assuming that $d = 2$, $k = 2$, $m_1 = m_2 = 3$, the following CNF rule generalises partially over the first feature and fully over the second feature:

$$\text{IF } x_1 \text{ is } \{L_{(1,1)} \text{ or } L_{(1,2)}\} \text{ and } x_2 \text{ is \# THEN } a \text{ is } 2$$

and can be encoded using *GABIL encoding* [6] as: 110|111|2, where each clause of the antecedent is a binary mask, followed by the action to vote for, separated by vertical bars. In the inference engine of the FRBS, we use $f_{\text{and}} = \min$ for conjunction (ANDing) and $f_{\text{or}} = \max$ for disjunction (ORing) of membership values. Let n be the number of rules in the RB. Given an input vector $\vec{x}$, a voting strength g_a is calculated for each $a \in A$ via:

$$g_a(\vec{x}) = \frac{\sum_{i=1}^{n} y_{(i,a)} \cdot \tau_i(\vec{x})}{\sum_{i=1}^{n} \tau_i(\vec{x})}$$

where $y_{(i,a)}$ is the voting weight for action a in the consequent of the i^{th} rule, and $\tau_i(s)$ is the overall antecedent truth value (rule

firing strength) of the i^{th} rule in the context of $\vec{x}$; calculated through application of f_{or} and f_{and} to the membership values computed in the rule antecedent. The action to select is then determined via:

$$\text{action} = \arg\max_{a \in A} g_a(\vec{x})$$

2.2.2 Measuring FRBS Complexity. There are many possible ways to measure the complexity of an FRBS, including: number of rules in the RB, longest antecedent of any rule in the RB [11]. We choose an option that is based on the number of "decision points" represented in the system. For a d-dimensional feature space, let a *fuzzy subspace* be defined as the intersection of d fuzzy sets over the features: $\left(L_{(1,j)} \cap \ldots \cap L_{(d,j)}\right), i \in \{1, \ldots, d\}, j \in \{1, \ldots, m_i\}$. Such an intersection of fuzzy sets represents an *elementary fuzzy rule* that is only capable of representing conjunctions, i.e. a single decision point in feature space. A CNF rule represents possibly many elementary rules, because disjunctions in such a rule represent generalisations over fuzzy subspaces. For an RB containing n CNF rules, if the number of linguistic values specified in the j^{th} clause of the i^{th} rule's antecedent is given by $l_{(i,j)}$, then the total number of decision points embodied in the RB is:

$$\text{complexity} = \sum_{i=1}^{n} \prod_{j=1}^{d} l_{(i,j)} \tag{2}$$

This is the measure of complexity that we use for an RB. The complexity of the overall FRBS is equal to the complexity of its RB.

3 RELATED WORK

Many of the ideas required in this work have been considered previously in small combinations and in non-RL domains. The cooperative coevolution architecture originally described in [19] has been adopted by GFSs to jointly evolve FRBS components, where one population (species) is dedicated to RBs and the other to DBs. For example, Fuzzy CoCo [18] used this architecture to address the well-known classification problem of Wisconsin Breast Cancer Diagnosis. This particular system employed a fitness penalty for RB complexity, and so did not utilise multiobjectivity. However, it was able to evolve compact and interpretable FRBSs to address the problem, and it set a strong example for how CoCo could be used within a GFS.

A number of Pittsburgh GFSs have been designed to use MO mechanisms according to a survey conducted by Ishibuchi [11]. An apposite example is the work of Ishibuchi et al. [12], where an MO evolutionary algorithm evolves FRBSs to address various classification problems; finding trade-offs between three objective functions: i) maximise classification accuracy, ii) minimise the number of fuzzy rules, iii) minimise the total number of fuzzy rule antecedent conditions. These ideas need development to RL domains, especially adapting to credit assignment in multi-step problems.

In the broader evolutionary computation context, MO and CoCo have also been combined in a single system, such as in the work of Iorio and Li [10]. The validity of this system was demonstrated on a number of benchmark function optimisation problems, but not yet RL. Peripherally related work includes Michigan style LCSs that use fuzzy logic rule representations, such as the Fuzzy Classifier System in [24] and Fuzzy-XCS in [3]. The former was applied to multi-step control problems (true RL), while the latter was only applied to single-step problems (function approximation and robot control). What is missing in all of these works is the combination of CoCo, MO, and FRBSs to address multi-step RL problems, and the intention of our work is to make a first attempt at addressing this gap.

4 MOUNTAIN CAR ENVIRONMENT

In Mountain Car (MC), the agent must push a car out of a valley to the top of a mountain, as shown in the top plot of Figure 4. State features are the position of the car on the horizontal axis: $x \in [-1.2, 0.5]$, and the horizontal component of the car's velocity: $\dot{x} \in [-0.07, 0.07]$. $A = \{1, 2\}$, representing push car 1: Left or 2: Right. R yields -1 at every time step, with $t_{\max} = 200$. Discounting is not used (effectively $\gamma = 1$). The goal is reached when $x \geq 0.5$. Let $S_I = \{[-0.6, -0.4], 0\}$ with Z constructed by sampling uniformly at random from S_I, such that the agent starts around the bottom of the valley with zero velocity. $\eta = 30$, with samples being drawn from S_I using a fixed RNG seed, such that all performance evaluations use the same initial states. These initial conditions make exploration difficult; if learning online (Michigan approach), the agent must somehow explore to the goal in order to learn how to escape the valley, then reinforce this path over time. In contrast, Pittsburgh approaches *may find the task easier* if they are able to construct coherent policies that escape the valley, then improve them over successive generations.

The minimum possible performance of a policy in MC is -200, indicating all η rollouts were unable to reach the goal within $t_{\max}$ steps. To calculate an upper bound on performance, we obtained a policy that was approximately optimal, and calculated the expected return achieved by it. To find this policy, we performed value iteration on a finely discretised version of MC (1000 bins per feature), yielding an approximately optimal action-value function $\widetilde{Q}$, then constructed an approximately optimal policy $\widetilde{\pi}$ by querying $\widetilde{Q}$ for each discretised state. The performance of $\widetilde{\pi}$ was -96 (rounded up to nearest integer).

5 COOPERATION AND SUBSPECIATION

As previously mentioned, natural decomposition of an FRBS leads to the concept of a DB cooperating with an RB. We evolve FRBSs where both the DB and RB are subject to adaptation via a CoCo algorithm where one population represents DBs and the other RBs. These populations are termed *species* [19]. We choose to label the DB species as the *first* population and the RB species as the *second* population, and for the remainder of this section refer to them as O_1 and O_2 respectively.

In implementing this CoCo paradigm we must primarily ask: *what kinds of structures are the two species searching over, and why?* A related secondary question is: *how are individuals of each species genetically encoded?* An answer to the primary question has to take into account the goal of the evolutionary process. In our case, we are performing an MO search over performance and complexity of FRBSs. Since the performance of an FRBS is governed by its interaction with the environment (and is outside of our control), diversity in this objective is a natural consequence of searching over many possible FRBSs of varying complexity. Therefore, there must be mechanisms built into the evolutionary process to support

diversity in FRBS complexity. To achieve this, and to provide an answer to the primary question, we include a niching mechanism in our algorithm in the form of *subspecies*. Since each population represents a species, a subspecies is a subpopulation.

To explain exactly what a subspecies is, and how it works in our CoCo paradigm, we have to begin to answer the secondary question posited above: *how are individuals of each species genetically encoded?* Let $\text{idv}_1 \in O_1$ be a DB and $\text{idv}_2 \in O_2$ be an RB. In order to form a solution, idv_1 must cooperate with idv_2; however, it must actually be possible for these individuals to form a valid FRBS. For example, assuming $d = 2$, let there be an idv_1 for which $m_1 = m_2 = 2$, i.e. two fuzzy sets defined on each feature. Next, assume there is an idv_2 containing the following rule:

$$\text{IF } x_1 \text{ is } L_{(1,1)} \text{ and } x_2 \text{ is } L_{(2,3)} \text{ THEN } \ldots$$

idv_2 does not make sense in the context of idv_1 because there is no fuzzy set $L_{(2,3)}$ defined in idv_1; therefore cooperation cannot occur. There must be a mechanism in the algorithm to prevent situations like this from occurring.

To accomplish this, each individual in both populations is assigned a (non-alterable) *subspecies tag* σ from a set of possible subspecies tags Σ, that indicates what subspecies it belongs to. A subspecies tag is a tuple of d integers each ≥ 2 representing the number of fuzzy sets defined on each of the d feature domains (at least two on each). For example, the subspecies tag of idv_1 from the previous example is $(2, 2)$. The subspecies tag determines the granularity of the fuzzy partitions over the feature space. Within each possible level of granularity, many levels of RB complexity are possible, as we explain in Section 5.2. For a given problem, this setup allows the evolutionary process to produce FRBSs with *appropriate granularity and complexity* to address the problem. Diversity in granularity drives diversity in complexity, which enables diversity in performance; thus subspeciation is a critical mechanism in our MO search. Subspecies tags are primarily used to coordinate cooperation between individuals of both species. Let $\text{idv}.\sigma$ denote the subspecies tag of idv. Cooperation is *restricted to be performed intra-subspecies*, such that idv_1 and idv_2 can only cooperate if $\text{idv}_1.\sigma = \text{idv}_2.\sigma$.

To genetically represent individuals of a given subspecies, we use a *fixed-length positional vector encoding*, where the length of an individual's genotype is dependent on its subspecies tag. For individuals in O_1, the genotype encodes "reference coordinates" along each feature domain that are used to construct fuzzy sets, as described in Section 5.1. For individuals in O_2, the genotype encodes the advocation of actions in fuzzy subspaces, as detailed in Section 5.2.

5.1 DB Genetic Representation

For an individual $\text{idv} \in O_1$, each element σ_i of $\text{idv}.\sigma$ represents the number of fuzzy sets used to partition feature i. To encode $m = \sigma_i$ fuzzy sets, m values are required, as explained below. Therefore, the length of such an individual's genotype is given by:

$$\lambda_1(\sigma) = \sum_{i=1}^{d} \sigma_i \tag{3}$$

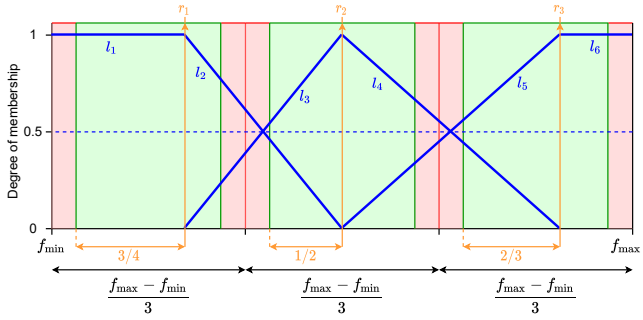

Figure 1: Example decoding of alleles from a DB individual over a single feature domain to produce fuzzy sets.

Since the genotype is a vector, its genes can be logically split into sections that are responsible for a specific feature. Alleles in the genotype are real numbers in the range $[0, 1]$. Figure 1 depicts an example of genotype decoding for an individual in O_1, on a single feature domain $[f_{\min}, f_{\max}]$. This process is repeated with the appropriate alleles for each feature domain to create fuzzy sets for all linguistic variables. Applicable alleles in this example are $(\frac{3}{4}, \frac{1}{2}, \frac{2}{3})$, shown in orange at the bottom of the figure. Since there are three alleles, three fuzzy sets are constructed along the domain; $m = 3$. The outer two fuzzy sets on the extremes of the domain are trapezoidal in shape and the inner fuzzy set is triangular in shape. In general, for $m = 2$ there is no inner triangular fuzzy set and for $m \geq 4$, there are multiple inner triangular fuzzy sets.

First, the domain is split into m equal width subdomains. Next, a fraction ω of the center of each subdomain is marked as a *valid* region (shaded green, red areas are *invalid*). We set $\omega = 0.75$, i.e. 75% of the middle of each subdomain is valid, 12.5% on each side (remaining 25%) is invalid. Each allele specifies a relative fraction *of the width of the valid region — measuring from the left hand side of the region* — at which a "reference coordinate" is placed along the domain. m reference coordinates $r_1, \ldots, r_m$ are placed, shown in orange at the top of the figure.

Using these reference coordinates, lines are drawn to construct the fuzzy sets. To construct the outer two trapezoidal fuzzy sets, lines are drawn between the following pairs of points:

$$\big((f_{\min}, 1), (r_1, 1)\big),\ \big((r_1, 1), (r_2, 0)\big),\ \big((r_{m-1}, 0), (r_m, 1)\big),\ \big((r_m, 1), (f_{\max}, 1)\big)$$

which correspond to lines l_1, l_2, l_5, l_6 in our example. Next, lines for the inner triangular membership functions are created: for $r_i, i \in \{2, \ldots, m-1\}$, lines are drawn from the point $(r_i, 1)$ to points $(r_{i-1}, 0)$ (positive gradient) and $(r_{i+1}, 0)$ (negative gradient), corresponding to lines l_3, l_4 in our example. Because of the concept of valid/invalid regions, this construction process has the desirable property of ensuring that there is a minimum amount of separation between neighbouring fuzzy sets, which reduces overlap and aids linguistic distinguishability [5].

5.2 RB Genetic Representation

An individual $\text{idv}_2 \in O_2$ must operate in the context of fuzzy sets specified by an individual $\text{idv}_1 \in O_1$. Because of this, there are only a certain number of fuzzy subspaces in which idv_2 can advocate actions. The number of fuzzy subspaces is the number of possible

fuzzy set intersections, i.e. the product of the number of fuzzy sets defined on each feature dimension. The length of idv_2's genotype is given by this number:

$$\lambda_2(\sigma) = \prod_{i=1}^{d} \sigma_i \tag{4}$$

Expressed as a vector, the genes in the genotype are ordered in the same order as nested for-loops over the fuzzy intersections, e.g. for $d = 2, m_1 = 3, m_2 = 2$, the genotype is of length six with genes specifying the following fuzzy subspaces:

$$\begin{gathered}\left(L_{(1,1)} \cap L_{(2,1)}\right),\ \left(L_{(1,1)} \cap L_{(2,2)}\right),\ \left(L_{(1,2)} \cap L_{(2,1)}\right),\\ \left(L_{(1,2)} \cap L_{(2,2)}\right),\ \left(L_{(1,3)} \cap L_{(2,1)}\right),\ \left(L_{(1,3)} \cap L_{(2,2)}\right)\end{gathered}$$

Each gene has alleles from the set: $A \cup \{0\}$, which select the action to advocate in the fuzzy subspace. The alleles from A are self-explanatory, but the 0 allele signifies that *no* action is specified, the fuzzy subspace is ignored. Thus, it is possible for an RB genotype to be *under-specified* in that actions do not have to be advocated in all fuzzy subspaces. Subspaces with a 0 allele are said to be *unspecified*, else they are *specified*. The possible underspecification of an RB is how different levels of complexity are achieved. This has important implications for measuring the performance of an FRBS, as we expand on in Section 6.

This genetic encoding represents a set of elementary fuzzy rules that only allow conjunction of fuzzy sets (c.f. Section 2.2.2). However, since we actually use CNF fuzzy rules in the RB phenotype, there is a mechanism to *merge fuzzy rules together* during genotype decoding in order to create CNF rules where commonalities are "factored out". A merge creates a disjunction that generalises over multiple fuzzy subspaces, and occurs when the binary encodings of two rules share the same bits in *all but one* clause — the differing clause creates the disjunction. This process is very similar to how a Karnaugh map is constructed for simplifying Boolean expressions. For example, using the same format of genotype as above and given the following specification of linguistic variables and their corresponding linguistic values:

$$x_1 : \{L_{(1,1)} : \text{L},\ L_{(1,2)} : \text{M},\ L_{(1,3)} : \text{H}\},\ x_2 : \{L_{(2,1)} : \text{L},\ L_{(2,2)} : \text{H}\}$$

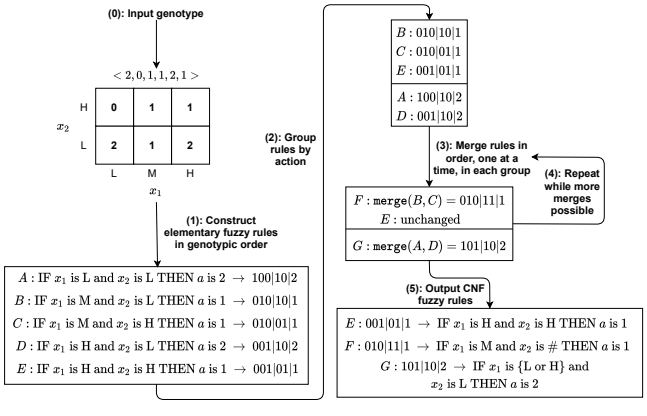

Figure 2: Example decoding of an RB genotype to produce a phenotype of CNF fuzzy rules.

Figure 2 depicts how the genotype $< 2, 0, 1, 1, 2, 1 >$ is decoded into a phenotype of CNF fuzzy rules. Note that in this example, only one application of step (3) is necessary, but in general step (3) may need to be repeated multiple times. Also note that the order of merging in step (3) is fixed; given our example rule B would always be merged with rule C, and it is not possible for rule C to be merged with rule E (even though this would result in an equally valid CNF rule).

Why use this encoding; why not just encode CNF rules directly? There are two good reasons: it is impossible to have rules that are *over-specified* in the sense of being either i) redundant, or ii) contradictory, since *at most* one action can be specified in every fuzzy subspace. Redundancy occurs when two rules A and B share the same consequent but rule A has an antecedent that logically subsumes B's. Contradictions occur when A and B specify common fuzzy subspace(s) in their antecedents but their consequents differ [5]. Both of these situations are problematic, contradictions more so than redundancies. There has been much attention in the literature on how to deal with these situations; some approaches allow them to genetically manifest and phenotypically deal with them via a conflict resolution procedure when evaluating the rule set, others employ corrections in genotype space to remove them if they occur [4]. We take the approach of making them unable to occur.

5.3 Complexity Bounds

In Section 2.2.2 we defined how the complexity of an RB of CNF rules is (phenotypically) calculated. However, given the RB genotype to phenotype decoding example shown in Figure 2, it is apparent that RB complexity can be measured genotypically as the *number of specified alleles* in the RB genotype. In the example, the number of specified alleles in the genotype is 5, which is exactly the complexity of the RB of CNF rules: applying Equation 2 gives $1 \times 1 = 1$ for rule E, $1 \times 1 = 1$ for rule F, $2 \times 1 = 2$ for rule G, $1 + 1 + 2 = 5$ total. Therefore we actually measure RB complexity genotypically.

We use the following bounds for complexity: minimum complexity for any RB genotype is equal to k (number of actions in A), meaning we give each RB the opportunity to advocate at least one rule for each possible action. Maximum complexity is equal to the *maximum possible number of specified alleles for any RB genotype*, equivalent to the length of the longest genotype of any RB subspecies.

6 FUZZY MOCOCO

We now present our algorithm for performing multiobjective cooperative coevolution of FRBSs. Our algorithm most closely resembles the system used in [10], in that it uses the same overarching framework for integrating the cooperative coevolutionary mechanisms from CCGA [19] with the Pareto multiobjective and elitist features of NSGA-II [7]. Algorithm 1 presents a top-level overview of our algorithm: Fuzzy MoCoCo (**M**ulti**o**bjective **Co**operative **Co**evolution). Algorithms 2–7 detail the main functions used in Algorithm 1. In these algorithms and for the remainder of this paper, the following notation is used:

- δ — a probability mass function (PMF) over subspecies tags
- P — a parent population

- Q — a child population
- R — a combined parent *and* child population
- O — a population: used in a general sense when *any* of (P, Q, R) could be expected
- $O_i, i \in \{1, 2\}$ — the i^{th} population
- $O^\sigma, \sigma \in \Sigma$ — a subpopulation; individuals in O with subspecies tag σ
- O_i^σ — a subpopulation of the i^{th} population
- S — a set of solutions (FRBSs)
- $\prec_{cc}$ – NSGA-II crowded comparison operator; partial ordering based on (Pareto front rank, crowding distance) pairs

Algorithm 1: `Fuzzy MoCoCo`

Input: $\mathcal{E}, \Sigma$

1 δ_1, δ_2 = `makeSubspeciesDists`(Σ)
2 P_1 = `initDBPop`(δ_1)
3 P_2 = `initRBPop`(δ_2)
4 $Q_1 = \emptyset$
5 $Q_2 = \emptyset$
6 $gen = 0$
7 **while** $gen <$ *numGens* **do**
8 $\quad \chi$ = `buildCollabrMap`(P_1, P_2, Σ, gen)
9 $\quad$ **if** $gen == 0$ **then**
10 $\quad\quad S$ = `buildSolnSet`(P_1, P_2, Σ, χ)
11 $\quad\quad$ `evalSolnSet`$(S, \mathcal{E})$
12 $\quad\quad$ `assignIndivsCredit`(P_1, S)
13 $\quad\quad$ `assignIndivsCredit`(P_2, S)
14 $\quad$ **else**
15 $\quad\quad S$ = `buildSolnSet`(Q_1, Q_2, Σ, χ)
16 $\quad\quad$ `evalSolnSet`$(S, \mathcal{E})$
17 $\quad\quad$ `assignIndivsCredit`(Q_1, S)
18 $\quad\quad$ `assignIndivsCredit`(Q_2, S)
19 $\quad R_1 = P_1 \cup Q_1$
20 $\quad R_2 = P_2 \cup Q_2$
21 $\quad P_1$ = `archiveParentPop`$(R_1, \delta_1, |P_1|)$
22 $\quad P_2$ = `archiveParentPop`$(R_2, \delta_2, |P_2|)$
23 $\quad Q_1$ = `breedChildren`(P_1, δ_1)
24 $\quad Q_2$ = `breedChildren`(P_2, δ_2)
25 $\quad gen = gen + 1$
26 **return** (R_1, R_2, S)

Algorithm 2: `buildCollabrMap`

Input: P_1, P_2, Σ, gen

1 χ = empty mapping of subpop specification pairs to indivs
2 **for** $(i, \sigma) \in (\{1, 2\} \times \Sigma)$ **do**
3 $\quad \chi[(i, \sigma)]$ = `selectCollabrs`(P_i^σ, gen)
4 **return** χ

As input to Algorithm 1, $\mathcal{E}$ and Σ must be specified. The main generational loop of Algorithm 1 can be split into two phases: the top half (lines 8–18, evaluation phase) builds solutions via cooperation, evaluates these solutions in the MO space, then assigns credit (objective values) to individuals based on their participation. The second half (lines 19–24, reproductive phase) archives the best solutions found so far, then breeds new child populations from these archives.

Algorithm 3: `buildSolnSet`

Input: O_1, O_2, Σ, χ

1 $S = \emptyset$
2 $popNums = \{1, 2\}$
3 **for** $(i, \sigma) \in (popNums \times \Sigma)$ **do**
4 $\quad subpop = O_i^\sigma$
5 $\quad j$ = opposite of i in $popNums$
6 $\quad collbars = \chi[(j, \sigma)]$
7 $\quad$ **for** $(idv, collabr) \in (subpop \times collabrs)$ **do**
8 $\quad\quad soln$ = `makeFRBS`$(idv, collabr)$
9 $\quad\quad S = S \cup \{soln\}$
10 **return** S

Algorithm 4: `evalSolnSet`

Input: $S, \mathcal{E}$

1 **for** $soln \in S$ **do**
2 $\quad soln.perf$ = `calcPerformance`$(soln, \mathcal{E})$
3 $\quad soln.comp$ = `calcComplexity`$(soln)$
4 `assignParetoFrontRanks`(S)
5 `assignCrowdingDists`(S)

Algorithm 5: `assignIndivsCredit`

Input: O, S

1 **for** $idv \in O$ **do**
2 $\quad C$ = set of solns in S that contain idv as a component
3 $\quad C'$ = `crowdedComparisonSort`(C)
4 $\quad best$ = first soln in C'
5 $\quad idv.perf = best.perf$
6 $\quad idv.comp = best.comp$

Algorithm 6: `archiveParentPop`

Input: $R, \delta, numParents$

1 R' = copy of R
2 $P = \emptyset$
3 **while** $|P| < numParents$ **do**
4 $\quad \sigma$ = draw sample from δ
5 $\quad subpop = R'^\sigma$
6 $\quad$ **if** *subpop not empty* **then**
7 $\quad\quad C$ = `crowdedComparisonSort`$(subpop)$
8 $\quad\quad best$ = first soln in C
9 $\quad\quad R' = R' - best$
10 $\quad\quad P = P \cup \{best\}$
11 **return** P

The reproductive phase is the same for every iteration of the loop, but the evaluation phase has different behaviour for the first vs. subsequent generations. In the first generation ($gen = 0$), Q_1 and Q_2 are empty and so P_1 must cooperate with P_2. This is a bootstrapping generation, where the initial parents are evaluated. In every subsequent generation, P_1 cooperates with Q_2 and P_2 cooperates with Q_1, with the purpose of evaluating individuals in Q_1 and Q_2. This means each individual is evaluated exactly once: in the

Algorithm 7: `breedChildren`

Input: P, δ
1 $Q = \emptyset$
2 **while** $|Q| < |P|$ **do**
3 σ = draw sample from δ
4 *subpop* = P^{σ}
5 *parentA* = `selection`(*subpop*)
6 *parentB* = `selection`(*subpop*)
7 *childA, childB* = `crossoverMutate`(*parentA, parentB*)
8 $Q = Q \cup \{childA, childB\}$
9 **return** Q

first generation for the initial parents, and in the generation after its conception for every child. Elitism appears in the form of archiving individuals from the combined populations R_1 and R_2 as parents. The function `makeSubspeciesDists` (Algorithm 1, line 1) creates a subspecies PMF for both populations: δ_1 and δ_2. These PMFs specify what fraction of the search should be (probabilistically) dedicated to each subspecies: subspecies with larger search spaces receive a larger fraction of the available resources. For $i \in \{1, 2\}, \sigma \in \Sigma$, δ_i is initialised as:

$$\delta_i[\sigma] = \frac{\beta^{\lambda_i(\sigma)}}{\sum_{\sigma' \in \Sigma} \beta^{\lambda_i(\sigma')}}$$

where $\lambda_i(\sigma)$ calculates the length of the genotype used by subspecies σ in population i (Equation 3, Equation 4). $\beta \geq 1$ is a hyperparameter that controls disparity in probability mass allocation, larger values of β allocate more mass to subspecies with longer genotypes. P_1 and P_2 are initialised in the functions `initDBPop` and `initRBPop`. These functions create *dbPopSize* and *rbPopSize* individuals respectively, determining which subspecies to create by drawing samples from δ_1 and δ_2, respectively. `initDBPop` initialises alleles randomly from $\mathcal{U}(0, 1)$. For `initRBPop`, a hyperparameter *rbPUnspec* controls the probability of initialising an allele as unspecified. The remaining alleles ($a \in A$) each have an initialisation probability of $\frac{1-rbPUnspec}{k}$.

Algorithms 2 and 3 implement cooperation between individuals as previously described. To select collaborators for cooperation, the function `selectCollbars` (Algorithm 2, line 3) has two different behaviours, depending on the generation counter. During the first generation there is no information about the objective values of any individual, so Pareto front ranks and crowding distances cannot be computed, and $\prec_{cc}$ cannot be applied. Therefore *two* collaborators are randomly selected in each subpopulation. During subsequent generations, again *two* collaborators are selected from each subpopulation, but are taken as: i) the best individual according to $\prec_{\text{cc}}$, ii) a random individual from the remainder of the subpopulation.

In Algorithm 4, the performance and complexity of solutions in S is evaluated. Because we allow RBs to be underspecified (see Section 5.2), it is possible that an FRBS can *fail* its performance evaluation, i.e. an input state is reached that is not covered by any rule. This is an inherent disadvantage of a Pittsburgh system as opposed to a Michigan system, the latter makes this impossible via a covering mechanism. If such a scenario is encountered, the FRBS is assigned *a performance equal to the environmental lower bound* (see Section 4). Evaluation of complexity is done as per Section 5.3.

Following performance and complexity evaluation, the function `assignParetoFrontRanks` determines the Pareto front ranks of solutions in S, implemented as the same fast non-dominated sort from NSGA-II. A Pareto front of rank i is denoted by $\mathcal{F}_i$, $\mathcal{F}_1$ representing the set of non-dominated solutions on the frontier of the search. Finally, `assignCrowdingDists` is used to determine the crowding distance of solutions, again in the same fashion as NSGA-II. To do this, lower and upper bounds for both performance and complexity must be known. Performance bounds are a property of $\mathcal{E}$ (see Section 4). Complexity bounds are as discussed in Section 5.3.

Algorithm 5 assigns credit to individuals in a population according to the *best* solution they participated in. The notion of best is determined via application of $\prec_{cc}$ (`crowdedComparisonSort` function). Algorithm 6 performs an NSGA-II style archiving procedure, selecting a new P. The distribution of subspecies tags in P is reflective of δ, in that the main loop (lines 4–10) firstly draws a subspecies tag from δ. Then, the best individual from the corresponding subpopulation is archived in P, and removed from the set of candidate parents R'. This is repeated until P is full.

Algorithm 7 generates a new child population Q, via application of a GA on P. The selection operator is tournament selection with a tournament size of 2, using $\prec_{\text{cc}}$ to rank individuals. Like Algorithm 6, δ is sampled to preserve the distribution of subspecies tags in Q. Reproduction is done *intra-subspecies*: the subspecies tag drawn from δ determines the subpopulation to select parents from. We use a real-coded GA on P_1 (since its genotypes are vectors of real numbers), with crossover implemented as line recombination [17], performed with probability *dbPCross*. Mutation is Gaussian noise, zero mean, standard deviation controlled by *dbMutSigma*.

For P_2 we use uniform crossover, performed with probability *rbPCross* per allele. Mutation allows each allele (being one of $k + 1$ values from $A \cup \{0\}$) to switch to one of the other k alleles, each with probability $\frac{1}{k}$. The probability of performing such a mutation is *rbPMut* per allele. Crossover and mutation probabilities are deliberately constant across subspecies, inciting more exploration (more frequent crossover and mutation) in subspecies with longer genotypes. Because we specify a minimum RB complexity, we include a *repair* operator, applied after mutation in Q_2. This operator rectifies situations where the number of specified alleles in an RB genotype is less than the minimum complexity — altering the genotype to be *of minimum complexity* by randomly selecting an appropriate number of unspecified alleles and assigning them random values from A.

7 RESULTS

We conducted thirty independent runs of Fuzzy MoCoCo[1] on the OpenAI Gym implementation of MC[2], in order to determine if parsimonious, high-performing policies could be found. Hyperparameters for each run were: *rbPUnspec*=0.1, *numGens*=50, *dbPCross*=0.75, *dbMutSigma*=0.02, *rbPCross*=0.25, *rbPMut*=0.05, η=30, Σ={(2, 2), (3, 3), (4, 4), (5, 5)}, β=1.125, *dbPopSize*=300, *rbPopSize*=600. β, *dbPopSize*, and *rbPopSize* were set in tandem to allocate each subspecies approximately ten times as many individuals as the dimensionality of its search space. Figure 3 (middle)

[1] Source code for algorithm available at: https://github.com/jtbish/fuzzy-mococo
[2] https://gym.openai.com/envs/MountainCar-v0/

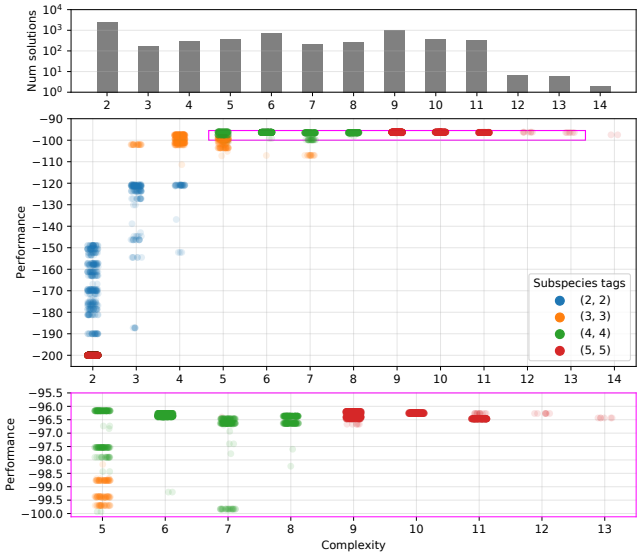

Figure 3: Middle: scatter plot of merged $\mathcal{F}_1$s over thirty Fuzzy MoCoCo runs on MC. Bottom: zoomed in view of highlighted region in middle plot. Top: histogram of complexity values in middle plot.

shows a scatter plot of $\mathcal{F}_1$ yielded by each run (thirty $\mathcal{F}_1$s, plotted on one set of axes). Each solution is plotted as an individual point with high transparency, giving some indication of solution density in a particular area. A small amount of jitter is used on the complexity axis to make individual points in low density areas more visible. Points are coloured by their subspecies tag. Note that in areas of very high density (e.g. around $(2, -200)$), multiple points are plotted on top p of one another and so cannot be distinguished; this is a limitation of the plotting technique. The top plot of the figure shows the frequency of each level of complexity. The bottom plot of the figure shows a zoomed in view of the magenta area in the middle plot. From this figure, we observe the following:

(1) A large number of solutions are of minimum complexity, minimum performance: $(2, -200)$.
(2) Increasing complexity up to 5 provides increased performance, but from then on provides no improvement.
(3) Solutions offering the best tradeoff between performance and complexity are of complexity 5, subspecies $(4, 4)$.

The first observation manifests because the minimum performance attainable in MC is -200, indicating that no performance rollouts were able to reach the goal. This can easily occur, via an FRBS that either i) has rules that do not cooperate enough to reach the goal, or ii) is too underspecified and so fails its performance evaluation.

We chose one of the solutions from the complexity 5 subspecies $(4, 4)$ group as the *best* solution found by any run. The performance of this best solution was -96.17, while the performance of our approximately optimal policy was -96. This is notable: *our algorithm has produced a policy that has almost attained the upper bound of performance.* Due to limited available space, a discussion of the performance and computational complexity of Fuzzy MoCoCo vs. other learning approaches is omitted. Such a discussion is a task for future work.

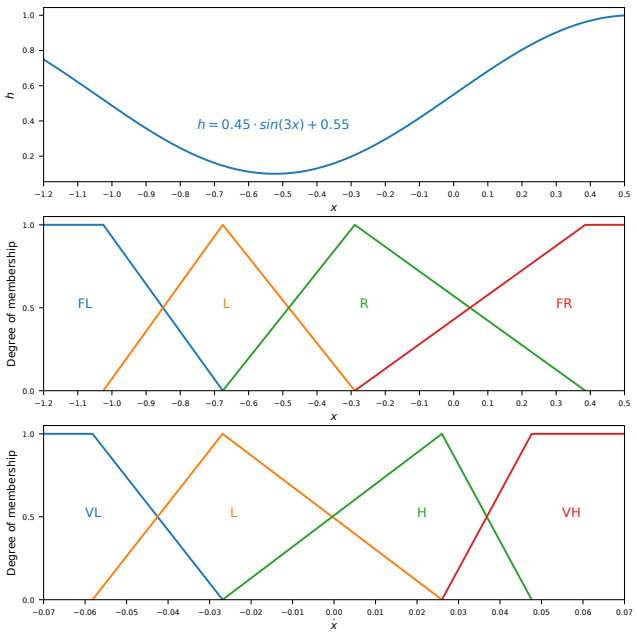

Figure 4: Top: curvature of the MC valley. Middle, bottom: fuzzy sets used by the best evolved FRBS in MC.

Figure 4 shows the fuzzy sets used by the best solution (middle, bottom), accompanied by the curvature of the valley (top) to make fuzzy sets over x easier to interpret. Linguistic values are as follows: x : {FL: Far Left, L: Left, R: Right, FR: Far Right}, $\dot{x}$: {VL: Very Low, L: Low, H: High, VH: Very High}. The RB used by the best solution is:

(A) IF x is L and $\dot{x}$ is L THEN a is 1 (Left)
(B) IF x is {FL or L or FR} and $\dot{x}$ is H THEN a is 2 (Right)
(C) IF x is R and $\dot{x}$ is VH THEN a is 2 (Right)

Rule (A) is responsible for pushing the car up the LHS mountain. Rule (B) pushes the car right when it has a moderate amount of positive velocity and is either i) on the LHS mountain, ii) in the valley, or iii) towards the top of the RHS mountain (almost at the goal). This rule deliberately omits the case where the car is on the steeper (lower) part of the RHS mountain, because there is not enough momentum to reach the goal by pushing right. Rule (C) covers this scenario, pushing the car to the right when it is on the steep part of the RHS mountain if there is a large amount of positive velocity.

8 CONCLUSION

We proposed a novel Pittsburgh GFS that utilises both MO and CoCo mechanisms to learn FRBSs that act as policies in RL environments. The system was tested on the Mountain Car environment, as it was a prime candidate due to its combination of a continuous state space and difficult exploration. Results show that the system was able to effectively balance resources and explore the tradeoff between FRBS performance and complexity. Analysis of a selected "best" (near optimal performance) FRBS showed that its rules were both interpretable and parsimonious.

REFERENCES

[1] Jaume Bacardit and Josep Maria Garrell. 2007. Bloat Control and Generalization Pressure Using the Minimum Description Length Principle for a Pittsburgh Approach Learning Classifier System. In *Learning Classifier Systems (Lecture Notes in Computer Science)*, Tim Kovacs, Xavier Llorà, Keiki Takadama, Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson (Eds.). Springer, Berlin, Heidelberg, 59–79.

[2] Martin V. Butz. 2005. Kernel-based, ellipsoidal conditions in the real-valued XCS classifier system. In *Proceedings of the 2005 conference on Genetic and evolutionary computation - GECCO '05*. ACM Press, Washington DC, USA, 1835–1842.

[3] Jorge Casillas, Brian Carse, and Larry Bull. 2007. Fuzzy-XCS: A Michigan Genetic Fuzzy System. *IEEE Transactions on Fuzzy Systems* 15, 4 (Aug. 2007), 536–550.

[4] Jorge Casillas and Pedro Martinez. 2007. Consistent, Complete and Compact Generation of DNF-type Fuzzy Rules by a Pittsburgh-style Genetic Algorithm. In *2007 IEEE International Fuzzy Systems Conference*. 1–6.

[5] Oscar Cordón (Ed.). 2001. *Genetic fuzzy systems: evolutionary tuning and learning of fuzzy knowledge bases*. Number 19 in Advances in fuzzy systems. World Scientific, Singapore.

[6] Kenneth A. de Jong, William M. Spears, and Diana F. Gordon. 1993. Using Genetic Algorithms for Concept Learning. *Machine Learning* 13, 2 (Nov. 1993), 161–188.

[7] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (April 2002), 182–197.

[8] Francisco Herrera. 2008. Genetic fuzzy systems: taxonomy, current research trends and prospects. *Evolutionary Intelligence* 1, 1 (March 2008), 27–46.

[9] A. Homaifar and E. McCormick. 1995. Simultaneous design of membership functions and rule sets for fuzzy controllers using genetic algorithms. *IEEE Transactions on Fuzzy Systems* 3, 2 (May 1995), 129–139.

[10] Antony W. Iorio and Xiaodong Li. 2004. A Cooperative Coevolutionary Multiobjective Algorithm Using Non-dominated Sorting. In *Genetic and Evolutionary Computation – GECCO 2004 (Lecture Notes in Computer Science)*, Kalyanmoy Deb (Ed.). Springer, Berlin, Heidelberg, 537–548.

[11] Hisao Ishibuchi. 2007. Multiobjective Genetic Fuzzy Systems: Review and Future Research Directions. In *2007 IEEE International Fuzzy Systems Conference*. 1–6.

[12] Hisao Ishibuchi, Tomoharu Nakashima, and Tadahiko Murata. 2001. Three-objective genetics-based machine learning for linguistic rule extraction. *Information Sciences* 136, 1 (Aug. 2001), 109–133.

[13] Tim Kovacs. 2012. Genetics-Based Machine Learning. In *Handbook of Natural Computing*, Grzegorz Rozenberg, Thomas Bäck, and Joost N. Kok (Eds.). Springer, Berlin, Heidelberg, 937–986.

[14] Pier Luca Lanzi, Daniele Loiacono, Stewart W. Wilson, and David E. Goldberg. 2005. XCS with computed prediction in multistep environments. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation (GECCO '05)*. Association for Computing Machinery, Washington DC, USA, 1859–1866.

[15] Xavier Llorà and David E. Goldberg. 2003. Bounding the Effect of Noise in Multiobjective Learning Classifier Systems. *Evolutionary Computation* 11, 3 (Sept. 2003), 279–298.

[16] Daniele Loiacono and Pier Luca Lanzi. 2008. Recursive least squares and quadratic prediction in continuous multistep problems. In *Proceedings of the 10th annual conference companion on Genetic and evolutionary computation (GECCO '08)*. Association for Computing Machinery, Atlanta, GA, USA, 1985–1992.

[17] Sean Luke. 2013. *Essentials of metaheuristics: a set of undergraduate lecture notes* (online version, second ed.). lulu.com, Morrisville, N.C.

[18] C.A. Pena-Reyes and M. Sipper. 2001. Fuzzy CoCo: a cooperative-coevolutionary approach to fuzzy modeling. *IEEE Transactions on Fuzzy Systems* 9, 5 (Oct. 2001), 727–737.

[19] Mitchell A. Potter and Kenneth A. De Jong. 2000. Cooperative Coevolution: An Architecture for Evolving Coadapted Subcomponents. *Evolutionary Computation* 8, 1 (March 2000), 1–29.

[20] Anthony Stein, Roland Maier, Lukas Rosenbauer, and Jörg Hähner. 2020. XCS classifier system with experience replay. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference (GECCO '20)*. Association for Computing Machinery, New York, NY, USA, 404–413.

[21] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement learning: an introduction* (second edition ed.). The MIT Press, Cambridge, MA.

[22] Tomohiro Takagi and Michio Sugeno. 1985. Fuzzy identification of systems and its applications to modeling and control. *IEEE Transactions on Systems, Man, and Cybernetics* SMC-15, 1 (Jan. 1985), 116–132.

[23] Ryan J. Urbanowicz and W. N. Browne. 2017. *Introduction to learning classifier systems*. Springer Berlin Heidelberg, New York, NY.

[24] Manuel Valenzuela-Rendón. 1998. Reinforcement learning in the fuzzy classifier system. *Expert Systems with Applications* 14, 1 (Jan. 1998), 237–247.