Generating Interpretable Fuzzy Controllers using Particle Swarm Optimization and Genetic Programming

Daniel Hein*
Siemens AG, Corporate Technology,
Munich, Germany
hein.daniel@siemens.com

Steffen Udluft
Siemens AG, Corporate Technology,
Munich, Germany
steffen.udluft@siemens.com

Thomas A. Runkler
Siemens AG, Corporate Technology,
Munich, Germany
thomas.runkler@siemens.com

ABSTRACT

Autonomously training interpretable control strategies, called policies, using pre-existing plant trajectory data is of great interest in industrial applications. Fuzzy controllers have been used in industry for decades as interpretable and efficient system controllers. In this study, we introduce a fuzzy genetic programming (GP) approach called *fuzzy GP reinforcement learning* (FGPRL) that can select the relevant state features, determine the size of the required fuzzy rule set, and automatically adjust all the controller parameters simultaneously. Each GP individual's fitness is computed using model-based batch reinforcement learning (RL), which first trains a model using available system samples and subsequently performs Monte Carlo rollouts to predict each policy candidate's performance. We compare FGPRL to an extended version of a related method called *fuzzy particle swarm reinforcement learning* (FPSRL), which uses swarm intelligence to tune the fuzzy policy parameters. Experiments using an industrial benchmark show that FGPRL is able to autonomously learn interpretable fuzzy policies with high control performance.

CCS CONCEPTS

• **Theory of computation → Reinforcement learning**; • **Software and its engineering → Genetic programming**; • **Computing methodologies → Vagueness and fuzzy logic**;

KEYWORDS

Interpretable reinforcement learning, fuzzy control, swarm optimization, genetic programming, industrial benchmark

ACM Reference Format:
Daniel Hein, Steffen Udluft, and Thomas A. Runkler. 2018. Generating Interpretable Fuzzy Controllers using Particle Swarm Optimization and Genetic Programming. In *GECCO '18 Companion: Genetic and Evolutionary Computation Conference Companion, July 15–19, 2018, Kyoto, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3205651.3208277

*Also with Technical University of Munich, Departement of Informatics.

GECCO '18 Companion, July 15–19, 2018, Kyoto, Japan

ACM ISBN 978-1-4503-5764-7/18/07…$15.00
https://doi.org/10.1145/3205651.3208277

1 INTRODUCTION

In typical industrial applications, such as controlling wind or gas turbines, control strategies, known as policies, that can be interpreted and controlled by humans, are of great interest [21]. However, using domain experts to manually design such policies is complicated and sometimes infeasible, since it requires the plant's system dependencies to be modeled in great detail with dedicated mathematical representations. Since such representations cannot be found for many real-world applications, policies have to be learned via reward samples from the plant itself. Reinforcement learning (RL) [29] is capable of determining such policies using only the available system data.

Recently, *fuzzy particle swarm reinforcement learning* (FPSRL) has been proposed, and it has been shown that an evolutionary computation method, namely particle swarm optimization (PSO), can be successfully combined with fuzzy rule-based systems to generate interpretable RL policies [13]. This can be achieved by first training a model on a batch of pre-existing state-action trajectory samples and subsequently conducting model-based RL. This step uses PSO to optimize a predefined set of fuzzy rule parameters.

FPSRL has been applied to several well-known RL benchmarks, such as the mountain car and cart-pole problems [13]. While such simple benchmark problems are well-suited to introducing a new method and comparing its performance to that of standard approaches, their easy-to-model dynamics and low-dimensional state and action spaces share few similarities with real-world industrial applications. Real applications usually have high-dimensional continuous state and action spaces. Applying FPSRL to systems with many state features often yields non-interpretable fuzzy systems since every fuzzy rule contains all the state dimensions, including redundant or irrelevant state dimensions, in its membership function by default.

In this paper, we propose an approach to efficiently determine the most important state features with respect to the optimal policy. Selecting only the most important features prior to policy parameter training makes the production of interpretable fuzzy policies using FPSRL possible again.

However, performing a heuristic feature selection initially and subsequently creating policy structures manually is a feasible but limited approach; in high-dimensional state and action spaces, the effort involved grows exponentially.

Instead, we propose as main contribution of our work *fuzzy genetic programming reinforcement learning* (FGPRL), an approach, such as FPSRL, that is based on model-based batch RL. By creating fuzzy rules using genetic programming (GP) rather than tuning the fuzzy rule parameters via PSO, FGPRL eliminates the manual feature selection process. This GP technique is able to automatically

select the most important features as well as the most compact fuzzy rule representation with respect to a certain level of performance. Moreover, it returns not just one solution to the problem but a whole Pareto front containing the best-performing solutions for many different levels of complexity.

Although genetic fuzzy systems have demonstrated their ability to learn and adapt to solve different types of problems in various application domains, GP-generated fuzzy logic controllers have never been combined with a model-based batch RL approach so far.

Combining a fuzzy system's approximate reasoning with an evolutionary algorithm's ability to learn allows the proposed method to learn human-interpretable soft-computing solutions autonomously. Cordón et al. [7] provide an extensive overview of previous genetic fuzzy rule-based systems. While most of the existing research in this area has focused on genetic tuning of scaling and membership functions as well as genetic learning of rule and knowledge bases, less attention has been paid to using GP to design fuzzy rule-based systems. Since GP is concerned with automatically generating computer programs [19], it should theoretically be able to both learn rule and knowledge bases as well as tune scale and membership functions simultaneously [10].

Fuzzy rule-based systems have been combined with GP for modeling [16] and classification [3, 6, 24, 26] tasks. In the optimal system control field considered in this paper, early applications that combine GP and fuzzy rule-based systems for mobile robot path tracking have been demonstrated [32]. Type-constraint GP has also been used to define fuzzy logic controller rule-bases for the cart-centering problem [1, 2]. Memetic GP, which combines local and global optimization, has been used to train Takagi-Sugeno fuzzy controllers to solve the cart-pole balancing problem [31]. Recently, based on the GP fuzzy inference system (GPFIS), GPFIS-control has been proposed [18]. They used multi-gene GP to automatically train a fuzzy logic controller and tested its performance on the cart-centering and inverted pendulum problems.

In this study, we apply FPSRL and FGPRL to two different benchmarks, namely the cart-pole swing-up and industrial benchmarks, to compare their RL policy performance and the interpretability of their fuzzy system controllers.

2 POLICY GENERATION METHODS

This paper compares two approaches to generate fuzzy RL policies from a batch of previously generated state-action trajectories (Fig. 1). FPSRL, first proposed in [13], tunes the fuzzy membership parameters of a predefined fuzzy set. Here, we extend FPSRL by adding an initial feature selection step, thus enabling its application to RL problems with high-dimensional state spaces. In addition, we compare FPSRL to a new approach, called FGPRL, that uses GP to create fuzzy RL policies using the same underlying model-based RL fitness function as FPSRL.

2.1 Model-based Reinforcement Learning

Inspired by behaviorist psychology, RL is concerned with the actions software agents should take in an environment to maximize their received accumulated rewards. In RL, the agents are not explicitly told the actions they are supposed to take; instead, they must learn the best strategy by observing the rewards given by the environment in response to their actions. In general, their actions can affect both the next reward and all subsequent rewards [29].

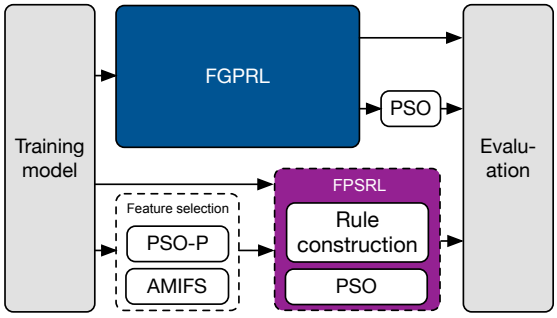


Figure 1: Comparing FPSRL to FGPRL

In the RL formalism, each agent observes the system state $\mathbf{s}_t \in \mathcal{S}$ at each discrete time step $t = 0, 1, 2, \ldots$ and takes an action $\mathbf{a}_t \in \mathcal{A}$, where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, respectively. In deterministic systems, state transitions can be expressed as function $g : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ with $g(\mathbf{s}_t, \mathbf{a}_t) = \mathbf{s}_{t+1}$. The corresponding rewards are given by reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$, with $r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) = r_{t+1}$. Thus, the RL problem's optimal solution is the policy that maximizes the expected accumulated rewards.

In the proposed approach, the goal is to find the best policy $\pi \in \Pi$, with Π being the set of all possible fuzzy RL policies. Policies associate every state $\mathbf{s}_t$ with an action $\pi(\mathbf{s}_t) = \mathbf{a}_t$, and their performance, for a given starting state $\mathbf{s}_t$, is measured by the return $\mathcal{R}(\mathbf{s}_t, \pi)$, i.e., the accumulated future rewards obtained by executing them. To account for the increasing uncertainties associated with future rewards, the reward r_{t+k} received k time steps in the future is weighted by γ^k, where $\gamma \in [0, 1]$. In addition, we adopt the common approach of only including a finite number $T > 1$ of future rewards in the return [29], as follows:

$$\mathcal{R}(\mathbf{s}_t, \pi) = \sum_{k=0}^{T-1} \gamma^k r(\mathbf{s}_{t+k}, \pi(\mathbf{s}_{t+k}), \mathbf{s}_{t+k+1}), \quad \text{with} \quad \mathbf{s}_{t+k+1} = g(\mathbf{s}_{t+k}, \mathbf{a}_{t+k}). \tag{1}$$

The overall state-independent policy performance $\mathcal{F}(\pi)$ is obtained by averaging $\mathcal{R}(\mathbf{s}_t, \pi)$ over all starting states $s_t \in S \subset \mathcal{S}$. Thus, the optimal solutions to the RL problem are the policies $\hat{\pi}$ where

$$\hat{\pi} \in \arg\max_{\pi \in \Pi} \mathcal{F}(\pi), \quad \text{with} \quad \mathcal{F}(\pi) = \frac{1}{|S|} \sum_{\mathbf{s}_t \in S} \mathcal{R}(\mathbf{s}_t, \pi). \tag{2}$$

In optimization terminology, the policy performance function $\mathcal{F}(\pi)$ is known as the fitness function.

For most real-world industrial control problems, the cost of executing a potentially bad policy is prohibitive. Therefore, in model-based RL [5], the state transition function g is approximated by a model $\tilde{g}$, that is either a first-principles model or has been created from previously recorded data. Substituting $\tilde{g}$ for the real system g in (1) allows us to obtain a model-based approximation $\tilde{\mathcal{F}}(\pi)$ of the true fitness function (2). Here, we consider models based on neural networks (NNs), but the proposed method could be extended to other models, such as Bayesian NNs [8] and Gaussian process models [25].

The model-based RL approaches considered in this paper are based on data sets $\mathcal{D}$ of state transition samples gathered from a real system. These samples are tuples $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, r_{t+1})$ that represent a start state $\mathbf{s}_t$ transitioning to a next state $\mathbf{s}_{t+1}$ owing to take action $\mathbf{a}_t$ and yielding a reward r_{t+1}. The set $\mathcal{D}$ can be generated using any policy (even a random one) prior to policy training and is subsequently used to generate world models $\tilde{g}$ that take inputs $(\mathbf{s}_t, \mathbf{a}_t)$ and predict $\mathbf{s}_{t+1}$ and r_{t+1}.

2.2 Fuzzy Controller

Fuzzy set theory was first proposed by Zadeh [34]. Based on this theory, Mamdani and Assilian [22] subsequently introduced so-called fuzzy controllers, specified by sets of linguistic if-then rules, whose membership functions can be activated independently to produce a combined output, computed by a suitable defuzzification function.

In a D-inputs-single-output system with C rules, the fuzzy rules $R^{(i)}$ can be expressed as follows:

$$R^{(i)}: \text{ IF } \mathbf{s} \text{ is } m^{(i)} \text{ THEN } o^{(i)}, \quad \text{with } i \in \{1, \ldots, C\}, \tag{3}$$

where $\mathbf{s} \in \mathbb{R}^D$ is the input vector (environment state, in our case), $m^{(i)}$ is the membership of a fuzzy set of the input vector in the premise part, and $o^{(i)}$ is a real number in the consequent part.

We use Gaussian membership functions [33]. These multivariate Gaussian functions are formed from products over all membership dimensions and yield smooth outputs, are local, and never produce zero activation. We define each rule's membership function as follows:

$$m^{(i)}(\mathbf{s}) = \mathrm{m}[\mathbf{c}^{(i)}, \sigma^{(i)}](\mathbf{s}) = \prod_{j=1}^{D} d(c_j^{(i)}, \sigma_j^{(i)}, s_j), \tag{4}$$

$$\text{with} \quad d(c, \sigma, s) = \exp\left\{-\frac{(c-s)^2}{2\sigma^2}\right\}, \tag{5}$$

where $m^{(i)}$ is the i-th parameterized Gaussian $\mathrm{m}(\mathbf{c}, \sigma)$, with center $\mathbf{c}^{(i)}$ and width $\sigma^{(i)}$.

The output is determined by the following equation:

$$\pi(\mathbf{s}) = \tanh\left(\alpha \cdot \frac{\sum_{i=1}^{C} m^{(i)}(\mathbf{s}) \cdot o^{(i)}}{\sum_{i=1}^{C} m^{(i)}(\mathbf{s})}\right), \tag{6}$$

where the hyperbolic tangent limits the output to be between -1 and 1, and the parameter α can be used to change the function's slope.

2.3 Fuzzy Particle Swarm Reinforcement Learning

FPSRL is a PSO-based approach to solve model-based batch RL problems [13]. PSO is a population-based, non-convex, stochastic optimization heuristic and can be applied to any search space that is a bounded sub-space of a finite-dimensional vector space [17].

The position of each particle in the swarm represents a potential solution to the given problem. The particles *fly* iteratively through the multidimensional search space, which is referred to as the fitness landscape. At each iteration, the particles move and receive fitness values for their new positions. These values are used to update each particle's velocity vector as well as those of all the other particles in a certain neighborhood.

For a given maximization problem, the best particle positions at iteration p are calculated as follows:

$$\mathbf{y}_i(p) = \begin{cases} \mathbf{x}_i(p), & \text{if } \mathcal{F}(\mathbf{x}_i(p)) > \mathcal{F}(\mathbf{y}_i(p-1)) \\ \mathbf{y}_i(p-1), & \text{else,} \end{cases} \tag{7}$$

where, in our framework, $\mathcal{F}$ is the fitness function given in (2) and the particle positions represent the policy parameters $\mathbf{x}$.

The parameter vector $\mathbf{x} \in \mathcal{X}$, where $\mathcal{X}$ is the set of valid Gaussian fuzzy parameterizations, is of size $(2D+1)\cdot C+1$ and can be presented as follows:

$$\begin{aligned} \mathbf{x} = (&c_1^{(1)}, c_2^{(1)}, \ldots, c_D^{(1)}, \sigma_1^{(1)}, \sigma_2^{(1)}, \ldots, \sigma_D^{(1)}, o^{(1)}, \\ &c_1^{(2)}, c_2^{(2)}, \ldots, c_D^{(2)}, \sigma_1^{(2)}, \sigma_2^{(2)}, \ldots, \sigma_D^{(2)}, o^{(2)}, \ldots, \\ &c_1^{(C)}, c_2^{(C)}, \ldots, c_D^{(C)}, \sigma_1^{(C)}, \sigma_2^{(C)}, \ldots, \sigma_D^{(C)}, o^{(C)}, \alpha). \end{aligned} \tag{8}$$

2.3.1 Rule Construction. To set up the FPSRL training process, we must take an assumption about the number of rules per policy during the rule construction step (Fig. 1). This requires either prior knowledge about the current problem, or testing different numbers of rules combinatorially [13].

2.3.2 Feature Selection. Industrial applications usually have dozens or even hundreds of possible state features, collected by different plant sensors. However, in our experience, very often only a small subset of them are required to create a policy that performs well. Unfortunately, determining the most important features is an expensive and ambiguous process, but it is nonetheless essential if we are to apply promising techniques such as FPSRL to industrial applications. Therefore, we propose a two-step approach that yields lists of features for each action, which are ordered by their relevance to an optimal policy.

First, an optimal trajectory is generated by applying the PSO-P receding horizon controller [12, 14] to the system model $\tilde{g}$. Here, PSO-P uses the model $\tilde{g}$ to determine action sequences $(\mathbf{a}_t, \mathbf{a}_{t+1}, \ldots, \mathbf{a}_T)$ starting from the state $\mathbf{s}_t$. The first action $\mathbf{a}_t$ in the optimal sequence is stored in the tuple $(\mathbf{s}_t, \mathbf{a}_t)$, and this process is repeated for all possible states in the data set $\mathcal{D}$. Note that no explicit policy representation is required to use the model to generate optimal actions.

Second, the AMIFS feature selection heuristic [30] is used to order the possible features $(s_1, s_2, \ldots) = \mathbf{s}$ of state $\mathbf{s}$ in terms of their relevance to the individual action dimensions $(a_1, a_2, \ldots) = \mathbf{a}$. AMIFS uses mutual information as a measure of feature relevance and redundancy to reveal non-linear feature-action relations.

Finally, the resulting ordered feature lists, e.g., $\{s_3, s_2, s_1\}_{a_1}$ and $\{s_2, s_1, s_3\}_{a_2}$ for the action dimensions a_1 and a_2, are used to construct compact and interpretable fuzzy rule representations, whose parameters will then be optimized by FPSRL.

2.4 Fuzzy Genetic Programming Reinforcement Learning

In this section, we introduce a GP-based approach that can generate interpretable fuzzy rule policies automatically using model-based batch RL. Analogously to FPSRL, FGPRL uses an approximate system model to predict the performance of policy candidates

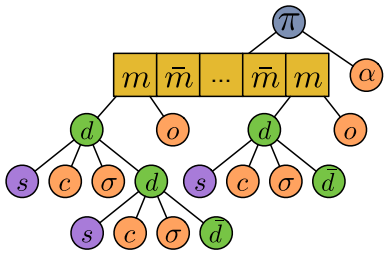


Figure 2: A fuzzy GP individual

Type		Complexity	
Variable	s	$\pi, \bar{m}, \bar{d}$	0
Floating-point number	c, σ, o, α	s	1
Dimension	$d, \bar{d}$	c, σ, o, α	1
Rule	$m, \bar{m}$	d	2
Policy	π	m	10

Table 1: Types and complexities of the GP building blocks. The terminal blocks $\bar{m}$ and $\bar{d}$ have values 0 and 1, respectively.

and subsequently uses this knowledge to iteratively generate high-performing policies. Unlike FPSRL, FGPRL not only optimizes the predefined policy parameters but also automatically selects the relevant state features, finds the necessary number of rules, and returns a Pareto front of policy candidates with different levels of complexity.

FGPRL is based on GP, which encodes computer programs as sets of genes and then modifies (evolves) them using a so-called genetic algorithm (GA) to drive the optimization of the population. The solution spaces comprise computer programs that perform well on the given tasks [19]. Since we are interested in using interpretable fuzzy controllers as RL policies, the genes in our case include membership and defuzzification functions, as well as constant floating-point numbers and state variables. These fuzzy policies can be represented as function trees (Fig. 2) and stored efficiently in arrays.

The GA drives the optimization process using selection and reproduction of population members, both of which are based on the members' fitness values $\mathcal{F}$. These represent how well each individual can perform the given task. Selection ensures that only the fittest individuals will survive into the next generation. Similar to the case of biological breeding process, pairs of individuals are selected for reproduction based on their fitness, and two offspring individuals are created for each pair by crossing their chromosomes. Technically, this is achieved by selecting compatible cutting points in the function trees and interchanging the subtrees below these cuts. Here, we apply tournament selection [4] to select the parent individuals. In addition, we use strongly-typed GP for FGPRL to avoid constructing ill-defined rules [2]. This means that each building block is assigned a type (Table 1). The different colors in Fig. 2 highlight the example GP individual's type structure. During the crossover process, only cutting points of equal type (color) are selected to ensure that only legal offspring are created.

We adopt the so-called Gaussian mutator as the mutation operator for floating-point terminals, which is common for evolutionary algorithms [27, 28]. In each generation, a given fraction of the best-performing individuals is selected for each level of rule complexity. Then, these individuals are copied, and their original floating-point values z are mutated by drawing replacement values z' from the normal distribution $\mathcal{N}(z, 0.1|z|)$. If the best copy's performance is better than that of the original individual, it is added to the new population. This allows us to conduct a local search in the policy space because it does not affect the individual's basic genotype structure.

To yield fuzzy rules with the structure described in Section 2.2, we must apply an additional tree correction. The GA can construct rules where two or more activation functions act on the same state variable, such as in Fig. 2 where the same s appears twice below the same rule m. Such rules are expected to be difficult to interpret since their shape does not conform the standard Takagi-Sugeno fuzzy inference model. For FGPRL, we decided to check every tree before evaluating its fitness by looking for recurring state variables and cutting out their corresponding activation functions. Note that the structures of the subsequent activation functions are not affected.

Since we are looking for interpretable solutions, we need to establish a suitable complexity measure. An individual's complexity can generally be measured in terms of its genotype (structural) or phenotype (functional) [20]. Here, we decided to use a simple node counting measurement strategy where different types of functions, variables, and terminals were weighted differently. Table 1 lists the weights (complexities) we decided to use in our experiments. Note that the weights yield a problem-specific balance between learning controllers consisting of more rules with less dimensions and vice versa.

Finally, we decided to create new generations for FGPRL using the following ratios: 45% crossover, 5% reproduction, 10% mutation, and 40% new random individuals.

2.4.1 Local Search. Since FGPRL's GP process searches the entire fuzzy policy structure space, it is prone to underestimate the importance of local fuzzy parameter tuning [23, 31]. We propose to counteract this by applying an additional parameter tuning step to all the terminals (c, σ, o, α) after optimization is complete (Fig. 1). Applying PSO to every individual in the final Pareto front yields an updated front comprising at most the same number of individuals with equal or higher fitness values.

3 EXPERIMENTS

In this section, we evaluate both approaches, FPSRL and FGPRL, using two benchmarks. The first is the so-called cart-pole swing-up problem (CP), a widely known RL benchmark, that was selected to demonstrate the methods' performance on a task where they could be easily compared with other RL methods. However, CP has a low-dimensional state space and its dynamics are deterministic and smooth. To investigate these methods' performance on real-world industry applications, we also selected a second benchmark, the industrial benchmark (IB). This combines a high-dimensional state space with stochastic dynamics, thus making its results increasingly

	FPSRL	FGPRL
Particles/Individuals	1,000	1,000
Iterations/Generations	× 10,000	× 10,000
Fitness value calculations (A)	1e+7	1e+7
Optimization result	Single policy	Pareto front of policies
	1	22 to 41
Additional local search	× 0	× 1e+6
Additional local fitness value calc. (B)	0	2.2e+7 to 4.1e+7
Runs for multiple complexities	4	1
(A)	× 1e+7	× 1e+7
Fitness value calc. for multiple compl. (C)	4e+7	1e+7
Total fitness value calc. (B)+(C)	4e+7	3.2e+7 to 5.1e+7

Table 2: Computational costs for FPSRL and FGPRL

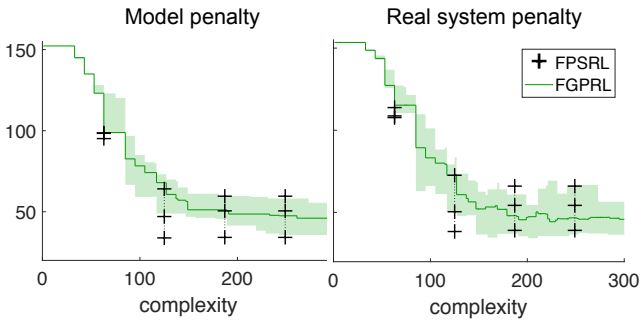


Figure 3: Comparison of FPSRL and FGPRL performance for the CP, for both the model and the real system. Here, the penalty is equal to the negative fitness. The green lines represent the median Pareto fronts over the FGPRL experiments. The semi-transparent green areas show the minimum and maximum penalties obtained from the experiments, while the black crosses indicate the FPSRL results. These show, from top to bottom, the maximum, median, and minimum penalties for each complexity level. The Pareto fronts resulting from training on the system model are on the left, while the right-hand plot shows the manner in which testing the same policies against the real system dynamics *altered* the fronts.

meaningful for industrial applications, such as controlling wind and gas turbines.

To compare the computational costs of FPSRL with FGPRL, we decided to use the parameter values shown in Table 2 for our experiments. Unlike FPSRL, FGPRL produces a whole Pareto front of solutions with different complexity and fitness values, while FPSRL only yields one solution, derived from its initial rule set. Note that although it is useful for FGPRL, FPSRL does not require any additional local optimization. Consequently, we chose the *additional local search* and *runs for multiple complexities* settings to produce similar *total fitness value calculations*, yielding the results presented below.

3.1 Cart-pole Swing-up

3.1.1 Dynamics. The objective of the CP benchmark is to apply forces to a cart moving along a one-dimensional track to keep a pole (hinged to the cart) in an upright position. The four Markov state variables are the pole's angle θ and angular velocity $\dot{\theta}$, and the cart's position ρ and velocity $\dot{\rho}$. These variables completely describe the Markov state; therefore, no additional information is required about the system's previous behavior. The RL agent's task is to find a sequence of force actions $a_t, a_{t+1}, a_{t+2}, \ldots$ that prevent the pole from falling over [9]. The CP experiments described in this paper were conducted using the CLS^2 software[1].

There are no restrictions on the cart's position or the pole's angle. Consequently, the pole can swing through, which is an important property of the CP. Since the pole's angle can initially be anywhere in the full interval $[-\pi, \pi]$, it is often necessary for the policy to swing the pole from one side to another to gain sufficient momentum to raise it and consequently receive the highest reward.

CP policies can apply actions of between −30 N and +30 N to the cart, and the reward function is given as follows:

$$r(\mathbf{s}) = \begin{cases} 0, & \text{if } |\theta'| < 0.5 \text{ and } |\rho'| < 0.5, \\ -1, & \text{otherwise.} \end{cases} \tag{9}$$

3.1.2 Benchmark Setup. Initially, we generated data set $\mathcal{D}$ by applying random actions using the real CP dynamics. Generating 100 state-action trajectories of length 100 gave us $|\mathcal{D}| = 10,000$. These trajectories' initial states $(\theta, \dot{\theta}, \rho, \dot{\rho})$ were sampled uniformly from $[-\pi, \pi] \times \{0\} \times \{0\} \times \{0\}$. Then, we trained five NN system models $\tilde{g}$, one for each state variable and one for predicting the probability of reaching the goal region [13].

These system models $\tilde{g}$ were subsequently used in model-based RL, with a time horizon T of 500 and a discount factor γ of 0.994. The training process involved 100 training states sampled from $[-\pi, \pi] \times \{0\} \times [-0.5, 0.5] \times \{0\}$. Solutions with $\mathcal{F} \approx -50$ or greater were considered to be successful because such policies could swing-up more than 99% of the given test states.

Finally, the best policies were tested against the real system dynamics using the same T and γ parameter values but a different set of 100 test states sampled from $[-\pi, \pi] \times \{0\} \times [-0.5, 0.5] \times \{0\}$.

3.1.3 Results. Here, we compare the results of CP experiments in which we ran the benchmark for each method 10 times. FPSRL produced 40 policies for 4 complexity levels, while FGPRL produced 278 policies for 96 complexity levels. Note that both methods involved a similar number of fitness value calculation (Table 2).

Since it is known that, for the CP, all four Markov state variables are required to produce policies that perform well, we skipped the feature selection step for FPSRL (Fig. 1) and invested the fitness value calculations budget in evaluating different numbers of rules, i.e., 2, 4, 6, and 8 rules yielding complexities of 63, 125, 187, and 249, respectively.

Fig. 3 shows that for problems such as the CP, which have low-dimensional state spaces and no irrelevant or redundant state variables, applying prior knowledge to the rule construction step yields a rule structure that can be easily tuned to produce high-performance, interpretable fuzzy policies. FPSRL can utilize all the available computational resources to tune a fixed set of parameters.

[1] http://ml.informatik.uni-freiburg.de/research/clsquare.

However, FGPRL has to employ the same resources to search a significantly larger space of possible solutions. Note that although FGPRL is theoretically able to produce exactly the same fuzzy policies for a complexity of 63 as FPSRL, it was unable to find a comparable solution for the system models in any of our experiments (Fig. 3). The best individual it produced with a complexity of 63 or less had a penalty value of 48.88, which was even above the median FPSRL penalty value of 47.05.

Comparing the model and real dynamics penalties yields another interesting observation. Even though the best FGPRL policies never surpassed FPSRL's performance for complexities of 300 and below on the system model, when they were evaluated using the real dynamics, some FGPRL policies actually performed better than the FPSRL policies. This could possibly be because the swarm optimization had already started to overfit the fuzzy parameters with respect to the system model; this means that FPSRL was exploiting model inaccuracies, thus reducing its performance on the real dynamics.

3.2 Industrial Benchmark

3.2.1 Dynamics. The IB[2] was designed to simulate several of the common challenges associated with many industrial applications [11]. It was not designed to approximate any specific real-world system but to be of a comparable hardness and complexity to many industrial applications.

The IB's state space is continuous, high-dimensional, and only partially observable. The actions are made up of three continuous components and affect three control inputs. In addition, the IB includes stochastic and delayed effects. The optimization task also involves multiple criteria; there are two reward components with opposite dependencies on the actions. Its dynamics are heteroscedastic, with state-dependent observation noise and probability distributions that are based on latent variables. Finally, it depends on an external driver that cannot be influenced by the actions.

At any given time step t, the RL agent can influence the state via actions $\mathbf{a}_t \in [-1, 1]^3$ that change the three observable state control variables, namely the velocity v, gain g, and shift h, i.e., $\mathbf{a}_t = (\Delta v_t, \Delta g_t, \Delta h_t)$.

The state $\mathbf{s}_t$ and successor state $\mathbf{s}_{t+1}$ are the Markov environment states. They can only be partially observed by the agent. In addition to the three control variables, v, g, and h, there is a setpoint p that simulates an external force, such as the load on a power plant or the speed of the wind driving a turbine that the agent cannot control but still has a significant influence on the system's dynamics. The system also suffers from a detrimental fatigue f_t, which depends on the setpoint p and the chosen control values $\mathbf{a}_t$, and consumes resources, such as power and fuel, represented by the consumption c_t. At each time step, it generates output values for c_{t+1} and f_{t+1}, which are part of the internal state $\mathbf{s}_{t+1}$. The reward is calculated as $r_{t+1} = -c_{t+1} - 3f_{t+1}$.

Note that the IB system's complete Markov state $\mathbf{s}$ is unobservable. Only the observation vector $\mathbf{o} = (v, g, h, p, c, f) \subset \mathbf{s}$ can be observed externally. The Markov state $\mathbf{s}_t$ can be approximated using a sufficient number of historic observations $(\mathbf{o}_{t-H}, \mathbf{o}_{t-H+1}, \ldots, \mathbf{o}_t)$ with a time horizon H. A system model $\tilde{g}(\mathbf{o}_{t-H}, \mathbf{o}_{t-H+1}, \ldots, \mathbf{o}_t, \mathbf{a}_t)$ that computes $(\mathbf{o}_{t+1}, r_{t+1})$ with $H = 30$, achieved adequate prediction performance during our IB experiments. Note that combining the six-dimensional observation vector with a time horizon H of 30 results in a 180-dimensional approximate Markovian state vector.

[2] http://github.com/siemens/industrialbenchmark

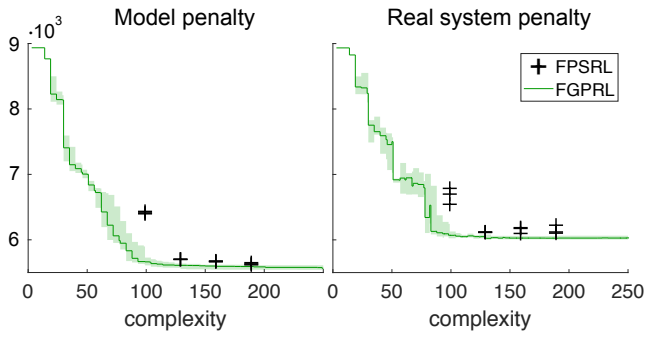


Figure 4: Comparison of FPSRL and FGPRL performance for the IB, for both the model and the real system

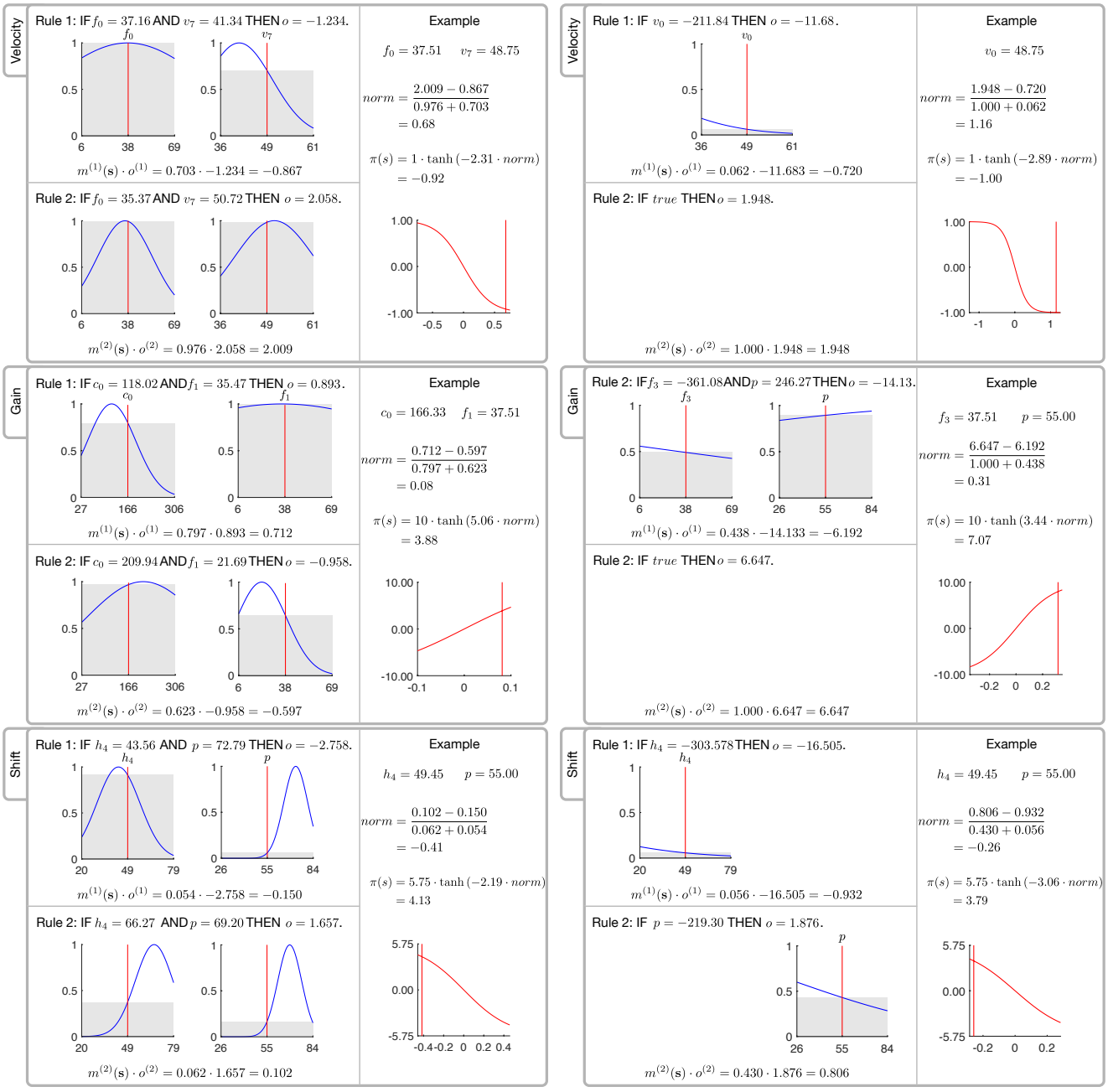
3.2.2 Benchmark Setup. The system was initialized for each setpoint p in $\{10, 20, \ldots, 100\}$ and then random trajectories of length 1,000 were produced. This process was repeated 10 times, resulting in a data set of size $|\mathcal{D}| = 100,000$. Following the approach reported in [15], we trained two recurrent NNs $\tilde{g}$ to predict the consumption c and fatigue f. These models $\tilde{g}$ were then used in model-based RL, with a time horizon T of 100 and a discount factor γ of 0.97. The training process involved 100 training states, drawn randomly from states in $\mathcal{D}$.

Finally, the best policies were tested against the real system dynamics, using the same T and γ parameter values, but a different set of 100 test states drawn randomly from states in $\mathcal{D}$.

3.2.3 Results. As with the CP experiments, we ran the IB benchmark 10 times for each method. FPSRL produced 40 policies for 4 complexity levels, while FGPRL produced 368 policies for 86 complexity levels.

To construct interpretable fuzzy rules using FPSRL, we have to select suitable features before swarm optimization of the fuzzy parameters (Section 2.3). The proposed method identified the following state variables as being the most important for each action dimension: Δv: $\{f_0, v_7, c_13, h_0\}$, Δg: $\{c_0, f_1, c_{15}, p\}$, and Δh: $\{h_4, p, h_{10}, h_0\}$. Here, the variables are listed in descending order of importance and the indices represent the time elapsed since the observation. Four different fuzzy rule structures were constructed based on these variables. The first policy with a complexity of 99, incorporated only the first variable in each list into two rules per action dimension, while the other policies, with complexities of 129, 159, and 189, incorporated the first two, the first three, or all four variables, respectively.

Using the proposed feature selection heuristic, FPSRL was able to generate policies with adequate performance ($\tilde{\mathcal{F}} = -5700$) for complexities of 129 or higher (Fig. 4). However, FGPRL was able to generate policies of a significant low complexity with a higher performance, achieving $\tilde{\mathcal{F}} = -5635$ at a complexity of 94 (Fig. 5). Moreover, the FGPRL search space covers all possible combinations



(a) FPSRL **(b) FGPRL**

Figure 5: Comparison of the fuzzy policies produced by FPSRL and FGPRL for the IB. Although both policies have similar fitness values (5a: -5700, 5b: -5635), their complexity is very different. The FGPRL policy only involves the minimum number of state variables required to yield adequate performance, meaning that the fuzzy controller is substantially more interpretable.

of state dimensions and numbers of rules for each individual action. For industrial problems where the state-to-action dependencies are not known a priori, we expect this ability to search autonomously to be highly valuable to control system designers and domain experts.

Comparing the policies' performance on the approximate IB model with their performance on the real IB dynamics shows that the results of training can be transferred to the real system as long as their regression and generalization quality is adequate (Fig. 4).

4 CONCLUSION

In this paper, we have evaluated two approaches to learn fuzzy control policies autonomously in terms of their performance and interpretability in industrial applications. We have considered applications with high-dimensional continuous state and action spaces and have proposed a feature selection heuristic that enables the previously presented FPSRL approach to be applied successfully in such industrial domains. Our second contribution is a GP-based fuzzy policy learning approach called FGPRL that utilizes the same model-based batch RL technique as FPSRL. However, instead of only tuning the parameters of fixed fuzzy policy structures, FGPRL searches the full space of all possible fuzzy controllers by determining the important state variables and the number of rules required and subsequently tunes all the rule parameters.

Experiments using the standard CP RL benchmark showed that FPSRL has a significant advantage when no feature selection is necessary and the number of rules required can be easily determined by simply testing a few different options. In this case, FGPRL's significantly wide search space is a drawback and it was far less likely to converge to a solution with similar performance to that produced by FPSRL when using a similar number of fitness value calculations. However, FGPRL was occasionally able to produce high-performance solutions for the CP benchmark.

Experiments using the IB, a benchmark that mimics real industrial systems like gas or wind turbines, yielded significant advantage for FGPRL over FPSRL. This benchmark has a high-dimensional state space, a multidimensional action space, stochastic and delayed effects, and a reward function with multiple criteria. Applying feature selection to FPSRL and manually testing different fuzzy policy structures did not yield satisfactory performance for low complexity solutions. In contrast, FGPRL was able to find high-quality interpretable solutions of low complexity with a similar number of fitness value calculations.

These results indicate that FGPRL is better than FPSRL at creating interpretable fuzzy policies autonomously from existing transition samples.

ACKNOWLEDGMENTS

The project this report is based on was supported with funds from the German Federal Ministry of Education and Research under project number 01IB15001. The sole responsibility for the report's contents lies with the authors.

REFERENCES

[1] E. Alba, C. Cotta, and J.M. Troya. 1996. Type-constrained genetic programming for rule-base definition in fuzzy logic controllers. In *Proceedings of the 1st annual conference on genetic programming*. MIT Press, 255–260.

[2] E. Alba, C. Cotta, and J.M. Troya. 1999. Evolutionary design of fuzzy logic controllers using strongly-typed GP. *Mathware and Soft Computing* 6, 1 (1999), 109–124.

[3] F.J. Berlanga, A.J. Rivera, M.J. del Jesús, and F. Herrera. 2010. GP-COACH: Genetic Programming-based learning of COmpact and ACcurate fuzzy rule-based classification systems for High-dimensional problems. *Information Sciences* 180, 8 (2010), 1183–1200.

[4] T. Blickle and L. Thiele. 1995. A Mathematical Analysis of Tournament Selection. In *ICGA*. 9–16.

[5] L. Busoniu, R. Babuska, B. De Schutter, and D. Ernst. 2010. *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press.

[6] B.-C. Chien, J.Y. Lin, and T.-P. Hong. 2002. Learning discriminant functions with fuzzy attributes for classification using genetic programming. *Expert Systems with Applications* 23, 1 (2002), 31–37.

[7] O. Cordón, F. Gomide, F. Herrera, F. Hoffmann, and L. Magdalena. 2004. Ten years of genetic fuzzy systems: current framework and new trends. *Fuzzy sets and systems* 141, 1 (2004), 5–31.

[8] S. Depeweg, J.M. Hernández-Lobato, F. Doshi-Velez, and S. Udluft. 2016. Learning and policy search in stochastic dynamical systems with Bayesian neural networks. *arXiv preprint arXiv:1605.07127* (2016).

[9] I. Fantoni and R. Lozano. 2002. *Non-linear control for underactuated mechanical systems*. Springer.

[10] A. Geyer-Schulz. 1995. Fuzzy Rule-Based Expert Systems and Genetic Machine Learning. *Physica-Verlag, Heidelberg* (1995).

[11] D. Hein, S. Depeweg, M. Tokic, S. Udluft, A. Hentschel, T.A. Runkler, and V. Sterzing. 2017. A benchmark environment motivated by industrial control problems. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*. 1–8. https://doi.org/10.1109/SSCI.2017.8280935

[12] D. Hein, A. Hentschel, T.A. Runkler, and S. Udluft. 2016. Reinforcement Learning with Particle Swarm Optimization Policy (PSO-P) in Continuous State and Action Spaces. *International Journal of Swarm Intelligence Research (IJSIR)* 7, 3 (2016), 23–42.

[13] D. Hein, A. Hentschel, T.A. Runkler, and S. Udluft. 2017. Particle swarm optimization for generating interpretable fuzzy reinforcement learning policies. *Engineering Applications of Artificial Intelligence* 65 (2017), 87–98.

[14] D. Hein, A. Hentschel, T.A. Runkler, and S. Udluft. 2018. Particle swarm optimization for model predictive control in reinforcement learning environments. In *Critical Developments and Applications of Swarm Intelligence*, Y. Shi (Ed.). IGI Global, Hershey, PA, USA, Chapter 16, 401–427.

[15] D. Hein, S. Udluft, M. Tokic, A. Hentschel, T. A. Runkler, and V. Sterzing. 2017. Batch reinforcement learning on the industrial benchmark: First experiences. In *2017 International Joint Conference on Neural Networks (IJCNN)*. 4214–4221.

[16] F. Hoffmann and O. Nelles. 2001. Genetic programming for model selection of TSK-fuzzy systems. *Information Sciences* 136, 1-4 (2001), 7–28.

[17] J. Kennedy and R.C. Eberhart. 1995. Particle swarm optimization. *Proceedings of the IEEE International Joint Conference on Neural Networks* (1995), 1942–1948.

[18] A.S. Koshiyama, T. Escovedo, M.M.B.R. Vellasco, and R. Tanscheit. 2014. GPFIS-Control: A fuzzy Genetic model for Control tasks. In *Fuzzy Systems (FUZZ-IEEE), 2014 IEEE International Conference on*. IEEE, 1953–1959.

[19] J.R. Koza. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

[20] N. Le, H.N. Xuan, A. Brabazon, and T.P. Thi. 2016. Complexity measures in Genetic Programming learning: A brief review. In *Evolutionary Computation (CEC), 2016 IEEE Congress on*. IEEE, 2409–2416.

[21] F. Maes, R. Fonteneau, L. Wehenkel, and D. Ernst. 2012. Policy search in a space of simple closed-form formulas: towards interpretability of reinforcement learning. *Discovery Science* (2012), 37–50.

[22] E.H. Mamdani and S. Assilian. 1975. An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Man-Machine Studies* 7, 1 (1975), 1–13.

[23] P. Moscato. 1989. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report* 826 (1989), 1989.

[24] L.S. Ramos and J.A.C. González. 2000. A niching scheme for steady state GA-P and its application to fuzzy rule based classifiers induction. *Mathware and Soft Computing* 7, 2-3 (2000), 337–350.

[25] E. Rasmussen and C.K.I. Williams. 2006. *Gaussian processes for machine learning (adaptive computation and machine learning)*. Mit Press Ltd.

[26] L. Sánchez, I. Couso, and J.A. Corrales. 2001. Combining GP operators with SA search to evolve fuzzy rule based classifiers. *Information Sciences* 136, 1-4 (2001), 175–191.

[27] H.-P. Schwefel. 1981. *Numerical optimization of computer models*. John Wiley & Sons, Inc.

[28] H.-P. Schwefel. 1995. Evolution and optimum seeking. Sixth-generation computer technology series. (1995).

[29] R.S. Sutton and A.G. Barto. 1998. *Reinforcement learning: an introduction*. A Bradford book.

[30] M. Tesmer and P.A. Estévez. 2004. AMIFS: Adaptive feature selection by using mutual information. In *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, Vol. 1. IEEE, 303–308.

[31] A. Tsakonas. 2013. Local and global optimization for Takagi–Sugeno fuzzy system by memetic genetic programming. *Expert Systems with Applications* 40, 8 (2013), 3282–3298.

[32] E. Tunstel and M. Jamshidi. 1996. On genetic programming of fuzzy rule-based systems for intelligent control. *Intelligent Automation & Soft Computing* 2, 3 (1996), 271–284.

[33] L.-X. Wang and J.M. Mendel. 1992. Fuzzy basis functions, universal approximation, and orthogonal least-squares learning. *IEEE Transactions on Neural Networks* 3, 5 (1992), 807–814.

[34] L.A. Zadeh. 1965. Fuzzy sets. *Information and Control* 8 (1965), 338–353.