Proceedings of the 2020 Winter Simulation Conference
K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, eds.

LEARNING RULE-BASED EXPLANATORY MODELS FROM EXPLORATORY MULTI-SIMULATION FOR DECISION-SUPPORT UNDER UNCERTAINTY

Brodderick Rodriguez
Levent Yilmaz

Department of Computer Science and
Software Engineering
Auburn University
Auburn, AL 36849, USA

ABSTRACT

Exploratory modeling and simulation is an effective strategy when there are substantial contextual uncertainty and representational ambiguity in problem formulation. However, two significant challenges impede the use of an ensemble of models in exploratory simulation. The first challenge involves streamlining the maintenance and synthesis of multiple models from plausible features that are identified from and subject to the constraints of the research hypothesis. The second challenge is making sense of the data generated by multi-simulation over a model ensemble. To address both challenges, we introduce a computational framework that integrates feature-driven variability management with an anticipatory learning classifier system to generate explanatory rules from multi-simulation data.

1 INTRODUCTION

In simulation modeling, structural uncertainty, or deep uncertainty, manifests itself when model developers do not know or cannot agree upon the relationship between variables (Walker, Lempert, and Kwakkel 2012; Kwakkel, Walker, and Haasnoot 2016; Lempert 2003). In principle, this is ambiguity in the arrangement and composition of underlying structures as well as their perceived influence. Structural uncertainty should not be confused with parametric uncertainty. Parametric uncertainty arises when modelers can define the dynamics of a system but are unaware of the parameter values associated with such dynamics (Walker, Lempert, and Kwakkel 2012). The presence of either type of uncertainty leads to models that vary in utility due to the number of plausible alternative hypotheses and candidate policies. Due to the potential for bias and the presence of predispositions, decision-makers are often skeptical of the credibility of a single model (Kwakkel, Walker, and Marchau 2010).

Exploratory modeling is achievable with sufficient computational power to evaluate numerous scenarios (Bankes 1993). However, more computation does not necessarily decrease uncertainty (Walker, Lempert, and Kwakkel 2012; Van Asselt and Rotmans 2002). Also, the use of multiple models increases the complexity of the methodology and can obscure system mechanics (Davis, O'Mahony, and Pfautz 2019). Analytical approaches such as sensitivity analysis and scenario planning are not conclusive and scalable when large number of models are used. Thus, decision-makers are ill-equipped to generalize and explain the connection between the outcomes and the causal assumptions of models. We observe two primary inadequacies in current practice: (1) the explanation of outcomes from large model ensembles and (2) the synthesis and maintenance of alternative model structures that constitute an ensemble. Addressing these issues can facilitate exploratory modeling methodology to mature further into science.

An analytical tool built for generalization and explanation yields a proper interpretation of large ensembles. It can provide insight into areas of the ensemble not otherwise explicitly evaluated and give

decision-makers a better understanding of how systemic policies and strategies generally behave with results transposed into sensible chunks (Davis, O'Mahony, and Pfautz 2019). An explainable analysis of a model can accelerate the exploratory modeling process by supplying intuitive interpretations of experiments. Furthermore, streamlined maintenance of alternative model structures is advantageous because it allows decision-makers to ask a broad range of questions with varying levels of abstraction with minimal effort. Exploratory modeling platforms (Kwakkel 2017) lack mechanisms for causal structure and representation variability in a way that facilitates the evaluation and explanation of model variations. Therefore, the responsibility to incorporate alternative features and representations in accord with research hypotheses falls on modelers to implement them, possibly causing premature convergence to a single model and inadequate exploration of structural uncertainties. Exploratory modeling methodology needs to be flexible and should manage structural variability subject to composition constraints (Yilmaz 2020).

In this paper, we aim to make strides toward addressing these limitations and demonstrating how wrapping exploratory experiments with variability management can efficiently process hypotheses of varying structural constraints. Additionally, we develop an explainable analytical tool, namely a rule-based machine learning algorithm, which can generalize and explain model dynamics over both explored and unexplored portions of the ensemble. Our experiments illustrate how adopting both of these solutions enhance exploratory modeling methodology and solidify the scientific approach to simulation-based decision-making under uncertainty.

2 BACKGROUND

The genesis of the proposed Strategy Learning System lies in the limitations of modeling for decision-making when uncertainty is present. The Strategy Learning System integrates existing methods in decision-making, uncertainty, exploration, and explanation in the context of modeling and simulation.

2.1 Decision-making Under Uncertainty

According to (Walker, Harremoës, Rotmans, Van Der Sluijs, Van Asselt, Janssen, and Krayer von Krauss 2003), uncertainties can manifest in three locations: in the context of the target system, in its model representation, and its inputs. Contextual uncertainties arise when decision-makers question the completeness of the definition of the target system. They attribute to the choice of target system boundaries (Kwakkel, Walker, and Haasnoot 2016) and the factors that lie inside and outside of the target system. Model uncertainties question a model's ability to resemble the mechanisms at play and can emerge from either the analysis of the target system or the implementation of its surrogate. Input uncertainties question the external forces, which influence change in the system and categorize them as either controllable or uncontrollable. *Controllable inputs* are within the decision-maker's control and concern the impact of their actions. Conversely, *uncontrollable uncertainties* are those not under the control of the decision-maker. They can, for example, describe the impact force of actions from an adversary. Uncontrollable input uncertainties are inherently challenging to differentiate, and therefore, it is difficult to anticipate their influence.

Robustness is a critical pillar in the evaluation of alternative policies (Lempert 2003; Rosenhead, Elton, and Gupta 1972; Metz, Davidson, Swart, Pan, et al. 2001). A policy is robust if it performs adequately when compared to alternatives in a large number of hypothetical scenarios (Yilmaz 2020; Lempert, Groves, Popper, and Bankes 2006; Rosenhead, Elton, and Gupta 1972). Robust policies should minimize expected cost or regret (Lempert, Groves, Popper, and Bankes 2006; Savage 1972), but they do not necessarily optimize (Walker, Lempert, and Kwakkel 2012). When presented with multiple alternate policies within uncertainty, and to avoid vulnerabilities and undesirable outcomes, it is prudent for decision-makers to select among the policies which demonstrate sufficient robustness (Lempert, Groves, Popper, and Bankes 2006; Lempert, Popper, and Bankes 2002).

2.2 Feature-Driven Model Variability

Software Product Line Engineering enables the rapid development of low-cost, high-quality products by organizing software into reusable artifacts (Kang, Lee, and Donohoe 2002; Pohl and Metzger 2006). The mission of Software Product Line Engineering is to identify variations and commonalities in a product suite and then develop reusable components that meet those specifications (Pohl and Metzger 2006).

Well-defined software features facilitate a process known as feature-driven engineering (Kang, Lee, and Donohoe 2002; Yilmaz 2020; Stahl, Voelter, and Czarnecki 2006). Feature-driven engineering is an adaptation of the software product line where variations and commonalities between components are described with a feature model (Stahl, Voelter, and Czarnecki 2006). Through feature composition, a feature model represents a hierarchical structure that allows software developers to reuse functionality and reduce implementation complexity. A feature tree is a feature model where features are explicit and defined as aggregations. Elements in a feature tree express interrelationships that decompose into mandatory, optional, OR (inclusive disjunction), and XOR (alternative) subfeature constraints (Pohl, Böckle, and van Der Linden 2005). Early studies in Variable structural modeling for composing discrete-event models at different abstraction levels include the System-Entity-Structure formalism (Zeigler and Praehofer 1989).

2.3 Exploratory Modeling and Simulation

An essential facet of exploratory modeling is the explanation of large ensembles (Bankes 1993; Bankes, Walker, and Kwakkel 2013). Explanation, as defined by the Oxford English Dictionary, includes a reason or justification of an action or belief. In the context of decision-making, an explainable choice in policy is one that achieves the desired outcome and is logical, transparent, and justified in its social and economic ramifications (Davis 2003; Metz, Davidson, Swart, Pan, et al. 2001; Doran, Schulz, and Besold 2017; Dam, Tran, and Ghose 2018).

Exploration generates a substantial amount of data that may obscure essential system mechanics from decision-makers (Davis, O'Mahony, and Pfautz 2019). This obscurity is caused by the limitations of the human ability to interpret large amounts of data and evaluate an ensemble of models, especially with time constraints (Hendler 2006). Decision-makers mitigate with analytical tools such as sensitivity analysis and scenario planning (Davis 2003; Kwakkel 2017; Lempert 2003). These tools analyze experiment results and transpose them into interpretable facts, which are used as decision-support artifacts (Davis 2003).

These tools perform adequately when the experiment results sufficiently describe the ensemble; however, they decrease in utility when experiments run on only a fraction of the hypothesis space. The tools do not contribute insight into unexplored portions. Deficiencies heighten when exploration of the entire ensemble is impractical due to time or computational restraints.

3 THE CONCEPTUAL FRAMEWORK OF THE STRATEGY LEARNING SYSTEM

The Strategy Learning System (SLS) is designed to improve the explanation and management of large ensembles. It is an extension of exploratory modeling methodology and also solidifies a scientific approach to large ensembles in exploratory modeling for decision-making.

Figure 1 is a diagrammatic representation of the SLS framework and shows its information flow. A *feature tree* is the conceptual representation of the target system, and its features are symbolic aggregations of various levels of delivered functionality. The *base model* is the computational realization of the target system. It is not implemented as a single fixed artifact, but instead contains all individual features specified by decision-makers. The features are cohesive and loosely coupled, so they are easily exchanged and meet the criteria of the feature tree. The base model's fundamental functionalities are either in the context of the target system or are those that contain a negligible amount of uncertainty. The *resolution model* is a description of an experiment. It specifies which components from the base model decision-makers want to incorporate in a set of scenarios to assess the outcome of a hypothesis. The resolution model is a meta-description of a subset of features from the feature tree where features are identified, but not

described. Scenarios are generated only by including both mandatory features and features listed in the resolution model.

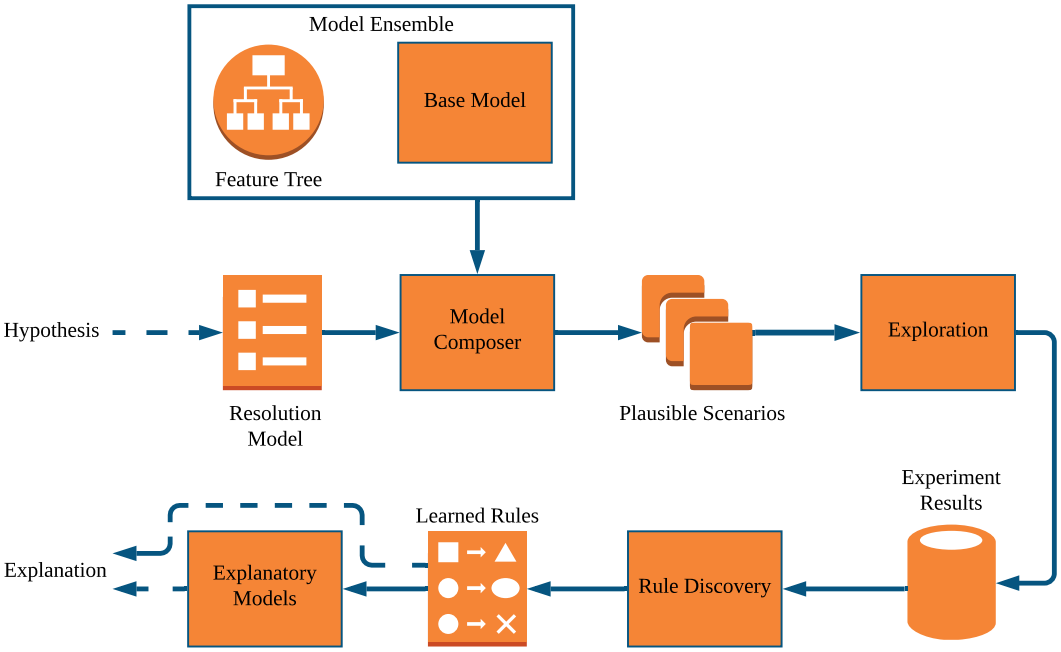


Figure 1: The data-flow architecture of the strategy learning system.

Figure 2 is a feature tree of a theoretical equation F. In this example, F is an aggregate feature composed of three subsequent mandatory subfeatures f_1, f_2, and o. f_1 has one mandatory subfeature and one optional subfeature. o is either $+$ or $-$ but not both. f_2 is any combination of $f_{2,1}$, $f_{2,2}$, and $f_{2,3}$.

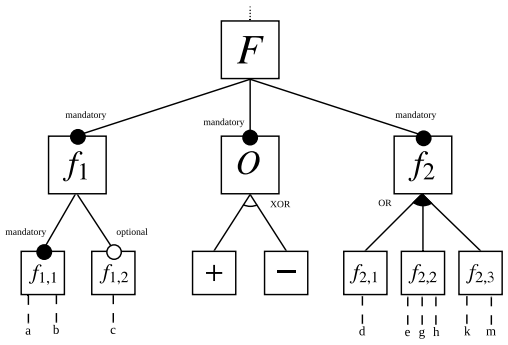


Figure 2: Feature tree of the example equation F.

This feature tree resemble the equation:

$$F = \begin{cases} f_1(\cdot) + f_2(\cdot) & \text{if } \; o \models + \\ f_1(\cdot) - f_2(\cdot) & \text{otherwise} \end{cases}$$

where $f_1 = \texttt{and}\big(f_{1,1}, \texttt{or}(f_{1,2})\big)$ and the cardinality of $f_2 = 2^3$. This example shows that from seven reusable components F can be configured into 2^5 unique equations.

The *model composer* generates plausible scenarios for an experiment. The base model, the feature tree, and the resolution model serve as inputs. It is a *metaprogram* that examines the resolution model and feature tree to identify which portions of the base model source code to include when the composer traverses the feature tree. Along a walk of the feature tree, the model composer performs feature composition and compiles the corresponding source code. Once the model composer has generated plausible structural scenarios, the exploration component samples over the parametric uncertainties. Monte Carlo or Hypercube Sampling is used to generate values for the parametric uncertainties adequately. This is essential to avoid an exhaustive search, reduce the computational cost, and maintain an adequate exploration of parametric uncertainties. A model instance is a single structural scenario with a set of corresponding parametric values. The exploration component executes the instances. Here a scenario runs multiple times with different parametric values and is represented by a series of model instances. If the base model contains stochasticity, then model instance execution can be repeated. The exploration component produces a set of inputs and outputs for each scenario called the experimentation results.

The analysis of experiment results yields rule discovery. A rule specifies constraints on the input where a particular outcome is expected. In this framework, rules are generated by a *Learning Classifier System*. The Learning Classifier System constructs tentative rules and validates them through reinforcement with the observed outcome in experimentation results. The rules are then generalized through the application of subsumption operators. The output of the Learning Classifier System is a set of rules which are transposed to plain-text. For the example equation F, a hypothetical rule could be:

```
IF:
        < f_{1,1}.a == LOW,
          f_{2,1}.d == HIGH,
          f_{2,3}.k == LOW,
          f_{2,3}.m == LOW >
WHEN:
        < F == f_1(f_{1,1}) + f_2(f_{2,1}, f_{2,3}) >
THEN:
        < outcome → HIGH >
```

The rules generated by the Learning Classifier System are compiled into an *explanatory model*. In this framework, the explanatory model is comprised of the population of rules and its visual *heat map* representation. The heat map illustrates which parametric and structural inputs in an experiment lead to an expected outcome. A color spectrum visualizes the outcome; lighter and warmer colors constitute a spectrum of desirable results, while cooler and darker colors constitute a spectrum of undesirable outcomes. The heat map is organized into two axes against which uncertainties are mapped.

4 DESIGN AND IMPLEMENTATION OF THE STRATEGY LEARNING SYSTEM

In this section, we map the Strategy Learning framework to a concrete instantiation. In the implementation, we utilize: (1) an agent-based Contaminant Plume model as the base model, (2) the Exploratory Modeling and Analysis Workbench as the exploration component, and (3) an Accuracy-based Learning Classifier System for the rule discovery mechanism.

4.1 Agent-Based Contaminant Plume Model

In recent years, commercial and governmental use of Unmanned Aerial Vehicles (UAVs) has added a wide range of applications (Canis 2015; Phillips 2008). The Contaminant Plume model (Rodriguez 2019a) simulates a practical application of UAVs. The model is an extension of a previous model (Madey and Madey 2013), and is a proxy to demonstrate how a collection of UAVs can cooperate as a UAV Swarm to perform a task.

The model incorporates an array of both mission controllable and uncontrollable parameters (deemed model and environmental parameters) as well as structurally distinctive command and control (C2) policies. The model allows decision-makers to fluctuate both parametric and structural parameters and identify circumstances that result in a rapid mapping and decontamination of a contaminant. The model is written in both NetLogo (Kovacina, Palmer, Yang, and Vaidyanathan 2002) and Scala, where the Scala language implementation is an extension that contains aggregate methods used in the NetLogo implementation. The benefit of this structure is that we can leverage both Scala's object-oriented nature and ease of readability along with NetLogo's GUI and widespread adoption in Agent-Based Modeling practice.

Figure 3 is a Conceptual Class Diagram that illustrates the state variables for the selected critical elements of the model. C2 policy's influence on Swarm collaboration varies. One of these policies is commissioned at the start of an episode.

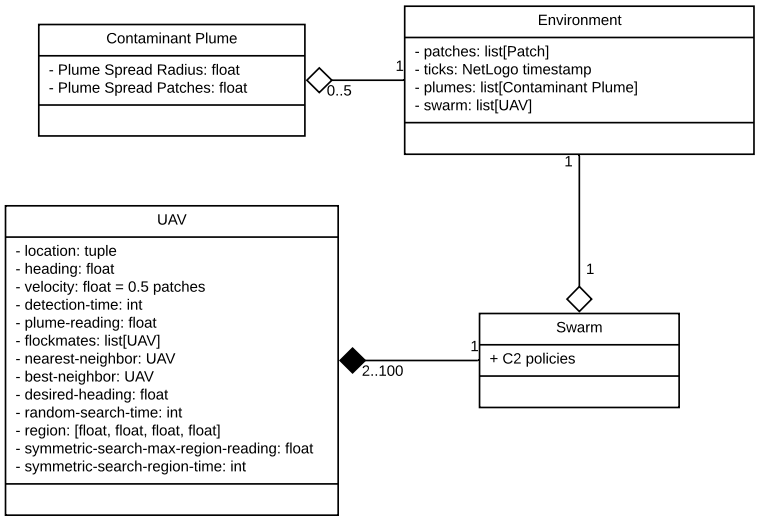


Figure 3: The conceptual class diagram for the model entities of the contaminant plume model.

Random Search Policy Random search (Kovacina, Palmer, Yang, and Vaidyanathan 2002) is the simplest of the three C2 policies. It pertains to a sense-free policy as the collective Swarm intelligence, and Environmental perception does not influence a UAV's behavior. Instead, UAVs make decisions independent of one another and use two random variables that determine how long a UAV will continue on its current trajectory and its next trajectory.

Flock Search Policy Flock search (Reynolds 1987) is a policy influenced by behavior observed in flocks of birds, herds of land animals, and schools of fish (Wilensky 1998). In this scenario, flocking behavior emerges from UAVs using three BOIDS (Wilensky 1998) rules: (1) Separation - an Agent steers to avoid overcrowding its flockmates; (2) Alignment - an Agent aligns its heading with the average heading of its flockmates; (3) Cohesion - an Agent changes its trajectory to move towards the average position of its flockmates. UAVs broadcast their plume reading to their flockmates. If a pair of UAVs are too close, they use separation to veer in opposite directions. Otherwise, once a UAV receives the plume reading of its flockmates, it uses both alignment and cohesion if the plume reading of its best neighbor is higher than its own.

Symmetric Search Policy Symmetric search (Kovacina, Palmer, Yang, and Vaidyanathan 2002) divides the Environment into symmetrical regions and initially assigns a UAV to each region where it searches for Contaminant Plumes. UAVs search their assigned territory for a random number of ticks, and after that, it may switch to the region of its best neighbor if the plume reading of its best neighbor is higher than its own.

On initialization, the Environment is set up, followed by the distribution of Contaminant Plumes and the deployment of the UAV Swarm. Once NetLogo's timestep counter and the entity state variables are reset, the Environment is considered initialized. Next, the Environment creates the Contaminant Plumes, and they are placed randomly in a central area that makes up one-quarter of the Environment. Finally, the Swarm is initialized, and the UAVs are dispersed randomly around the Environment. Figure 4 is a High-level Sequence Diagram depicting the initialization process.

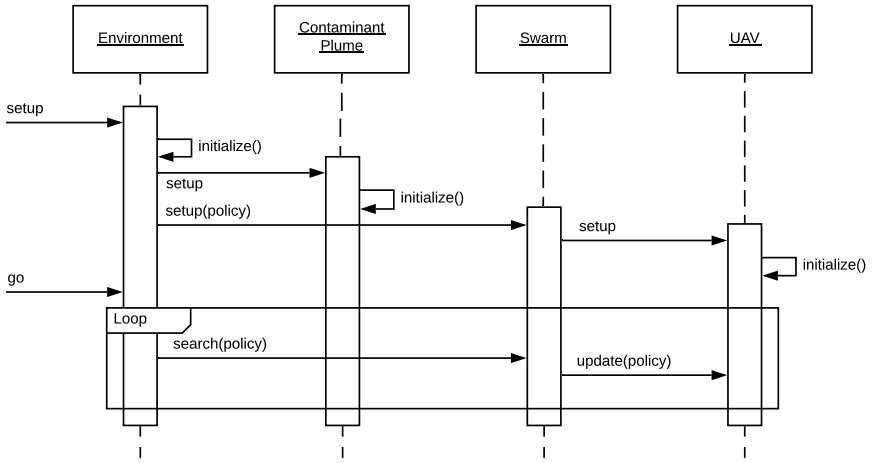


Figure 4: The high-level sequence diagram for initializing the contaminant plume model.

Figure 5 shows four distinct instances of the model. The upper left figure demonstrates the appearance of the Environment and the Contaminant Plumes (red area) before the Swarm is initialized. The remaining three figures illustrate the model after 2,000 timesteps with the upper right being random search policy, the lower left being flock search policy, and the bottom right being symmetric search policy.

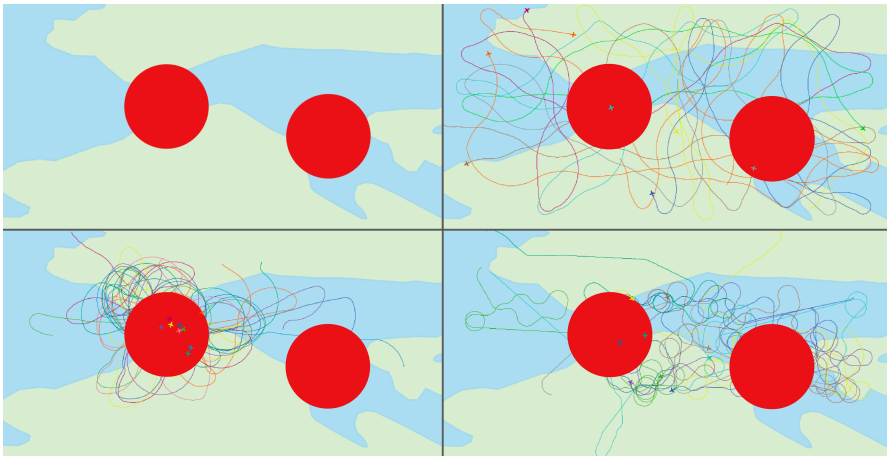


Figure 5: The screenshots of the simulation of alternative contaminant plume control strategies.

4.2 Exploratory Analysis Workbench

The Exploratory Modeling and Analysis Workbench (Kwakkel 2017) is a Python package that supports robust and multi-objective optimization for exploratory modeling under uncertainty. The workbench accommodates the generation of computational experiments for base models written in Netlogo and Excel. Experiments are executed sequentially or in parallel with replications in either a single computer or cluster environment. A detailed specification for the Exploratory Modeling and Analysis Workbench can be found in (Kwakkel 2020).

The Strategy Learning System uses a light-weight fork (Rodriguez 2019b) of the Exploratory Modeling and Analysis Workbench. While the Exploratory Modeling and Analysis Workbench is designed for a wide range of applications, *EMA-Lite* is intended to be only a mechanism to execute experiments in parallel on a high-performance computing cluster and obtaining raw experiment results. The Strategy Learning System interfaces with *EMA-Lite* and offers a seamless experimental procedure. The model composer generates structural variants of the base model that are passed to *EMA-Lite* and samples paramedic uncertainties and executes the model instances. The raw experimental results are returned to the Strategy Learning System, which applies the rule discovery and generalization process.

4.3 Accuracy-Based Learning Classifier

The Accuracy-based Learning Classifier System (XCS) (Urbanowicz and Browne 2017; Brownlee 2011; Lanzi 2000; Wilson 1995; Butz and Wilson 2000; Kovacs 1998) is a Michigan-style classifier which incorporates both reinforcement learning and evolutionary computing in the form of genetic algorithms to discern a set of rules that describe an environment or dataset.

It is an online supervised learning algorithm that can operate on both single-step and Markovian multistep problems with delayed rewards (Urbanowicz and Browne 2017). Recent research into derivatives of XCS allow for real-valued scenarios (Fredivianus and Geihs 2017). We adopt several ideas from proven methods to construct XCSR and contrast them with methods found in Wilson's XCS. For our XCSR implementation (Rodriguez 2019c) in the SLS framework, we combine the environment and the reinforcement program to increase cohesion. To this end, classifier predicates are expanded to accept values in the range $[0,1]$. A classifier predicate is a tuple where $C_i[0]$ and $C_i[1]$ are the lower and upper bounds, respectively. For example, a predicate of length 6 contains 12 learned parameters. We update the matching procedure as follows:

$$\text{matches}'(cl,\sigma_t) = C_i[0] \leq \sigma_{t,i} \leq C_i[1],\ \forall\ i \in [1,...L]. \tag{1}$$

Predicate attributes initially set to (0, 1) to encourage generalization as this matches any value of σ_t and by design replaces the wildcard # found in XCS. Classifier predicates are mutated with a generic operation to modify the range of accepted values.

In the implementation of the Strategy Learning System, the resolution model contains a list of features parsed by the model composer. EMA-Lite samples parametric uncertainties and executes the model insurances. The experiment results from the EMA-Lite are processed by the Strategy Learning System. Controllable and uncontrollable feature argument values are binned with fuzzy logic to reduce state space complexity. Additionally, in the Contaminant Plume model, all outcome metrics are time series data. A time series outcome f is transformed to a scalar with the Area Under the Curve Trapezoidal Rule:

$$R(\sigma_t,\alpha_t) \leftarrow AUC_f(a,b) = \sum_{t=a+1}^{b} (t_i - t_{i-1}) \times \frac{f(t_i)+f(t_{i-1})}{2}, \tag{2}$$

where a and b are the lower and upper bounds of the domain of f, respectively. Outcomes are normalized using min-max normalization. Experiment results consist of tuples of uncontrollable uncertainties, controllable uncertainties, and $R(\sigma_t,\alpha_t)$.

To interface XCSR with the Strategy Learning System framework, we implement a generic environment that feeds in experiment results and assesses action selection. The generic environment further processes experiment results and splits them into states and actions based on which features are under the control of the decision-maker and which are not. XCSR ρ_{t+1} is determined by:

$$\rho_{t+1} \leftarrow R(\sigma_t, \alpha_t) - \left[\zeta R(\sigma_t, \alpha_t) \frac{||\alpha_t - \hat{\alpha}_t||_2}{\max_{\alpha \in A} ||\alpha||_2}\right], \tag{3}$$

where ζ is a discount factor, and A is the set of actions in experimental results. This equation converges to 1 when the distance between α_t and $\hat{\alpha}_t$ is minimized and the observed reward is 1. It converges to 0 when the distance between the α_t and $\hat{\alpha}_t$ is large or the observed reward is low. Multiplying the distance by $R(\sigma_t, \alpha_t)$ proportionally reduces ρ_t depending upon the desirability of α_t.

5 EVALUATION OF THE STRATEGY LEARNING SYSTEM

We use the SLS implementation to perform experiments in two category types. The first type of experiment, which is illustrated in this paper, aims to validate the theoretical tractability of the SLS framework and the correctness of its implementation. The second type of experiment is exploratory and provides insight into the Contaminant Plume model's mechanics. Each experiment begins with a hypothesis that parallels a "what if" scenario. The hypothesis explains which base model features are included in the experiment's resolution model. The complexity of the resolution model determines hyperparameters for EMA-Lite and XCSR. More complex resolution models require a larger portion of the ensemble to be explored, and subsequently, the XCSR requires more iterations to identify accurate rules.

The experiment hyperparameters for EMA-Lite are: (1) The number of model instances to execute from the heterogeneous ensemble that includes distinct models that vary in terms of their structure and parameter values; (2) The number of replications for each model instance; (3) The run length of each instance in time steps. The experiment hyperparameters for XCSR are: (1) Episode length determines how many instances we present to XCSR; (2) The value of the variable *bins* determines the granularity of feature arguments. For example, if the variable *bins* is set to 3, then feature arguments are converted to low, medium, or high depending on their value; (3) The maximum number of classifiers in the population; (4) ζ is the discount factor in reinforcement learning component of the XCSR. Additionally, we predict how the model will behave and what the expected insight the experiment will yield.

Once the experiment is performed, we discuss and supply the produced explanatory heat maps from both EMA-Lite exploration results and the XCSR rules learned. For validation experiments, we provide additional heat maps generated by a Multi-Layer Perceptron clustering algorithm to corroborate the observation rule sets further. The Multi-Layer Perceptron trained on the exploratory results then extrapolates outcome into the entire experiment space. A Hierarchical Agglomerative clustering algorithm then builds rules from the extrapolated data. Validation Experiment was designed as simple as possible to understand both the exploratory results and the learned rules easily and to confirm that the Strategy Learning System behaves as anticipated. Due to the non-complexity of this experiment, it was feasible to do an exhaustive search of the experiment space; however, this experiment aimed to test the capabilities of the Strategy Learning System merely.

Experiment Hypothesis: How is coverage percentage affected when we vary the Swarm population and the number of contaminant plumes?
From our hypothesis, we construct the following resolution model.

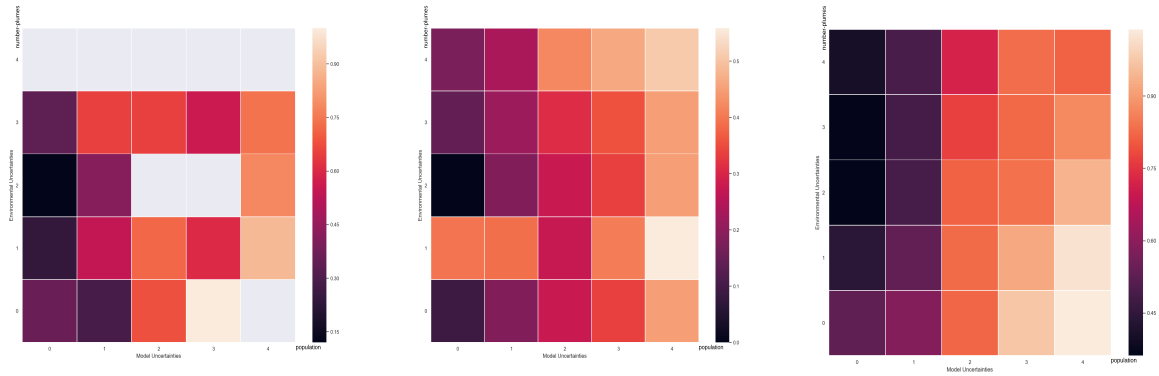
Validation Experiment 1 Resolution Model

```
include_feature(population)
include_feature(number-plumes)
include_outcome(coverage-percentage)
```

Table 1: The parameters of the illustrative validation experiment.

EMA Lite			XCSR			
Model Instances	Replications	Ticks	Episodes	Bins	Classifiers	ζ
30	30	1,000	20,000	5	50	0.5

Table 1 shows the hyperparameter values for the experiment. The number of model instances is low due to the small number of possible feature combinations. Through preliminary experimentation with the base model, we determined that 1,000 ticks were significant enough to assess Swarm performance. Setting bins to five allows us a more granular understanding of the experiment. The episode length, number of classifiers, and ζ were chosen from prior research results (Kovacs 1998) and a lightweight grid-search.



(a) EMA Map (b) XCSR Map (c) MLP with Clustering

Figure 6: The Heat Map representation of the explanatory model.

Figure 6a visualizes the EMA-Lite exploratory results for Validation Experiment 1. The column data corresponds to values for the population; they increase from left to right. Similarly, the row data corresponds to the number of contaminant plumes, and they increase from the bottom to the top. In Figure 6a, and all subsequent heat maps, darker colored cells correspond to less desirable outcomes, while lighter colored cells correspond to more desirable outcomes. In Figure 6a, cells are gray if model instances that fall in that region were not explicitly explored. In Figure 6a, we see that higher populations result in higher outcomes– as predicted. The hot spot is located on the right where populations are high, and the number of contaminant plumes is relatively low. The dark colors in the lower left region signify that the outcome is low because both the number of contaminant plumes and the population is low.

Figure 6b is a visualization of the learned rules produced by training XCSR on the exploratory results for the Validation Experiment. The rules are visually similar to the results in Figure 6a, indicating that XCSR captured patterns in the exploratory results and generalized them into the portion of the ensemble not explicitly explored. This generalization can be seen in the top row of Figure 6b, where the desirability of outcome coincides with the row below it and parallels our prediction.

Notably, Figure 6b demonstrates generalization errors. For example, in the lower right region of Figure 6a, where the population is 0 and 1, and the number of contaminant plumes is 1, we see that the outcome is undesirable. In Figure 6b, we see that XCSR incorrectly deemed this area significantly more desirable than the adjacent cells. This is due to the formulation of a reward function that solely evaluates the action proposed by XCSR. In other words, the closer the proposed action is to the expected action, the higher the reward. This encourages generalization over uncontrollable features while punishing generalization over controllable features. We believe this improperly evaluates the XCSR and yields contradictory results.

The level of outcome throughout Figure 6b is lower than the outcome in Figure 6a. This is because multiple rules overlap. The level of outcome assigned to a cell in Figure 6b is based on an accuracy-weighted average of the expected outcome of all the rules which apply to the region. Thus, if there are two rules which apply to the same experiment space region, then both their outcomes are used to determine cell color. When rules contradict each other, it hampers the decision-support artifact. Alternatively, a decision-maker can analyze the relative colors within the heat map to determine which regions are desirable.

Figure 6c visualizes the rules produced by the Multi-Layer Perceptron clustering algorithm. Figure 6c compliments both the exploratory results and the XCSR learned rules, and we see the appearance of the same generalized patterns. Figure 6c visualizes gradual changes in outcome desirability and reveals the presence of the more precise rules produced by the Multi-Layer Perceptron Clustering algorithm– especially when compared to the rules produced by XCSR. Precise rules lack generalization and maybe inapt for decision-support. Instead, we use Figure 6c visualizations as a validatory artifact to confirm the XCSR rules describe the experiment space.

6 CONCLUSIONS

The genesis of this work is the necessity to address structural uncertainties when modeling real-world systems for decision-making. Structural uncertainties arise when decision-makers do not entirely understand the relationship between variables and, in real-world systems, this results in many plausible descriptions of system mechanics. Exploratory modeling methodology is an effective strategy to address uncertainty. In an exploratory modeling exercise, a decision-maker iteratively explores hypotheses about the real-world system to understand better how the system behaves. Nevertheless, contemporary exploratory modeling tools cannot evaluate alternative causal models rapidly. Such tools excel with parametric uncertainties but require a modeler to implement alternative causal structures manually. Notably, exploratory modeling produces large data due to multiple model instance execution, and therefore may obscure fundamental system mechanics from the decision-maker. This work presents a candidate solution to address both of these exploratory modeling voids and improves the decision-support exercise.

REFERENCES

Bankes, S. 1993. “Exploratory modeling for policy analysis”. *Operations research* 41(3):435–449.

Bankes, S., W. E. Walker, and J. H. Kwakkel. 2013. “Exploratory modeling and analysis”. *Encyclopedia of operations research and management science*:532–537.

Brownlee, J. 2011. *Clever algorithms: nature-inspired programming recipes*. Jason Brownlee.

Butz, M. V., and S. W. Wilson. 2000. “An algorithmic description of XCS”. In *International Workshop on Learning Classifier Systems*, 253–272. Springer.

Canis, Bill 2015. “Unmanned aircraft systems (UAS): Commercial outlook for a new industry”.

Dam, H. K., T. Tran, and A. Ghose. 2018. “Explainable software analytics”. In *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*, 53–56.

Davis, P. K. 2003. “Exploratory analysis and implications for modeling”. *RAND-PUBLICATIONS-MR-ALL SERIES-*:255–284.

Davis, P. K., A. O’Mahony, and J. Pfautz. 2019. *Social-Behavioral Modeling for Complex Systems*. John Wiley & Sons.

Doran, D., S. Schulz, and T. R. Besold. 2017. “What does explainable AI really mean? A new conceptualization of perspectives”. *arXiv preprint arXiv:1710.00794*.

Fredivianus, N., and K. Geihs. 2017. “Classifier systems with native fuzzy logic control operation”. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 1341–1348.

Hendler, J. 2006. “Computers play chess; humans play go”. *IEEE Intelligent Systems* 21(4):2–3.

Kang, K. C., J. Lee, and P. Donohoe. 2002. “Feature-oriented product line engineering”. *IEEE software* 19(4):58–65.

Kovacina, M. A., D. Palmer, G. Yang, and R. Vaidyanathan. 2002. “Multi-agent control algorithms for chemical cloud detection and mapping using unmanned air vehicles”. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Volume 3, 2782–2788. IEEE.

Kovacs, T. 1998. “XCS classifier system reliably evolves accurate, complete, and minimal representations for Boolean functions”. In *Soft computing in engineering design and manufacturing*, 59–68. Springer.

Kwakkel, Jan 2010-2020. “Exploratory Modeling Workbench Read The Docs”. https://emaworkbench.readthedocs.io/.

Kwakkel, J. H. 2017. "The Exploratory Modeling Workbench: An open source toolkit for exploratory modeling, scenario discovery, and (multi-objective) robust decision making". *Environmental Modelling & Software* 96:239–250.

Kwakkel, Jan H and Walker, Warren E and Haasnoot, Marjolijn 2016. "Coping with the wickedness of public policy problems: approaches for decision making under deep uncertainty".

Kwakkel, J. H., W. E. Walker, and V. A. Marchau. 2010. "Classifying and communicating uncertainties in model-based policy analysis". *International journal of technology, policy and management* 10(4):299–315.

Lanzi, P. L. 2000. *Learning classifier systems: from foundations to applications*. Springer Science & Business Media.

Lempert, R., S. Popper, and S. Bankes. 2002. "Confronting surprise". *Social Science Computer Review* 20(4):420–440.

Lempert, R. J. 2003. *Shaping the next one hundred years: new methods for quantitative, long-term policy analysis*. Rand Corporation.

Lempert, R. J., D. G. Groves, S. W. Popper, and S. C. Bankes. 2006. "A general, analytic method for generating robust strategies and narrative scenarios". *Management science* 52(4):514–528.

Madey, A. G., and G. R. Madey. 2013. "Design and evaluation of UAV swarm command and control strategies". In *Proceedings of the Agent-Directed Simulation Symposium*, 1–8.

Metz, B., O. Davidson, R. Swart, J. Pan et al. 2001. *Climate change 2001: mitigation: contribution of Working Group III to the third assessment report of the Intergovernmental Panel on Climate Change*, Volume 3. Cambridge University Press.

Phillips, A. N. 2008. "A secure group communication architecture for a swarm of autonomous unmanned aerial vehicles". Technical report, Air Force Institute Of Technology.

Pohl, K., G. Böckle, and F. J. van Der Linden. 2005. *Software product line engineering: foundations, principles and techniques*. Springer Science & Business Media.

Pohl, K., and A. Metzger. 2006. "Variability management in software product line engineering". In *Proceedings of the 28th international conference on Software engineering*, 1049–1050.

Reynolds, C. W. 1987. "Flocks, herds and schools: A distributed behavioral model". In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, 25–34.

Rodriguez, Brodderick 2019a. "Contaminant Plume Model". https://github.com/broddericksrodriguez/contaminant_plume_model.

Rodriguez, Brodderick 2019b. "EMA Lite". https://github.com/broddericksrodriguez/ema_lite.

Rodriguez, Brodderick 2019c. "xcsr". https://github.com/broddericksrodriguez/xcsr.

Rosenhead, J., M. Elton, and S. K. Gupta. 1972. "Robustness and optimality as criteria for strategic decisions". *Journal of the Operational Research Society* 23(4):413–431.

Savage, L. J. 1972. *The foundations of statistics*. Courier Corporation.

Stahl, T., M. Voelter, and K. Czarnecki. 2006. *Model-driven software development: technology, engineering, management*. John Wiley & Sons, Inc.

Urbanowicz, R. J., and W. N. Browne. 2017. *Introduction to learning classifier systems*. Springer.

Van Asselt, M. B., and J. Rotmans. 2002. "Uncertainty in integrated assessment modelling". *Climatic change* 54(1-2):75–105.

Walker, W. E., P. Harremoës, J. Rotmans, J. P. Van Der Sluijs, M. B. Van Asselt, P. Janssen, and M. P. Krayer von Krauss. 2003. "Defining uncertainty: a conceptual basis for uncertainty management in model-based decision support". *Integrated Assessment* 4(1):5–17.

Walker, W. E., R. J. Lempert, and J. H. Kwakkel. 2012. "Deep uncertainty". *Delft University of Technology* 1:2.

Uri Wilensky 1998. "Flocking". http://ccl.northwestern.edu/netlogo/mod-els/Flocking.

Wilson, S. W. 1995. "Classifier fitness based on accuracy". *Evolutionary computation* 3(2):149–175.

Yilmaz, L. 2020. "Managing Structural Variability in Agent-Based Models with Feature Coherence Graphs". *International Journal of Simulation and Process Modelling - in press*.

Zeigler, B. P., and H. Praehofer. 1989. "Systems theory challenges in the simulation of variable structure and intelligent systems". In *International Conference on Computer Aided Systems Theory*, 41–51. Springer.

Acknowledgments

Levent Yilmaz's research is partially funded by the National Science Foundation (NSF) under grant NSF IIS1910794.

AUTHOR BIOGRAPHIES

BRODDERICK RODRIGUEZ is a Graduate Student in the Department of Computer Science and Software Engineering at Auburn University. His research interests are in Machine Learning and Modeling & Simulation. He received his B.S. and M.S. degrees in Computer Science and Software Engineering from Auburn University. His email address is bcr@brodderick.com.

LEVENT YILMAZ is Professor of Computer Science and Software Engineering with a courtesy joint appointment at the Department of Industrial and Systems Engineering at Auburn University. He holds M.S. and Ph.D. degrees in Computer Science from

Virginia Tech. His research interests are Theory and Methodology of Modeling & Simulation, Agent-Directed Simulation, and Complex Adaptive Systems. He is a Fellow of the Society for Modeling and Simulation International (SCS) and is the founding organizer and General Chair of the Annual Agent-Directed Simulation Symposium series. His email address is yilmaz@auburn.edu.