Learning Explanatory Models for Robust Decision-Making Under Deep Uncertainty

by

Brodderick Connor Rodriguez

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
May 2, 2020

Keywords: Feature-driven variability management, machine learning, exploratory modeling, deep uncertainty

Approved by

Levent Yilmaz, Professor of Computer Science and Software Engineering
Bo Liu, Assistant Professor of Computer Science and Software Engineering
Anh Nguyen, Assistant Professor of Computer Science and Software Engineering

Abstract

Decision-makers rely on simulation models to predict and investigate the implications of their decisions. However, the use of monolithic simulation models based on fixed assumptions lack the requisite adaptivity needed when the real-world system contains significant uncertainty. Exploratory modeling is a methodology that involves iterative and incremental exploration of alternative hypotheses about the underlying assumptions of the real-world system under a broad range of contextual conditions. Through exploration, decision-makers gain an understanding of the breadth of the system and pinpoint robust policies. However, exploratory modeling tools lack mechanisms to generate, evaluate, and learn from the results of simulating an ensemble of alternative, possibly competing models. Additionally, exploratory modeling over a population of models generates significant amount of data that may obscure fundamental system mechanics and their interaction with the context. This thesis introduces a modeling architecture with (1) a feature-oriented generative modeling mechanism for rapid derivation of alternative causal model structures and (2) a rule-based machine learning strategy in terms of a Learning Classifier System to produce explanatory models in the form of a population of rules and its associated visual heat-maps that convey the robustness and resilience of alternative system designs. The use of both of these mechanisms accelerates the decision-support exercise and yields more intuitive interpretations of system insights when modeling for decision-making under deep uncertainty.

Acknowledgments

I want to thank Dr. Bo Liu and Dr. Anh Nguyen for their role in my advisory committee. They have been treasured resources in my academic endeavors as a student and researcher at Auburn University. I want to express sincere gratitude to Dr. Levent Yilmaz for his invaluable guidance and role as my major advisor.

I dedicate this work to the many people that advised me in my self-doubt, confidence, frustration, success, and personal and academic development. This work is the product of years of self-taught programming and schooled Computer Science. I am blessed that I love what I do, and it is a passion first and career second.

"Find a job you enjoy doing, and you will never have to work a day in your life." - Mark Twain.

"The more I learn, the less I know" - Albert Einstein.

Dyslexia didn't stop me!

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Decision-making & Uncertainty

Decision-makers use simulation models as predictive surrogates to investigate the implications of varying assumptions on future events. Real-world applications include fleet management and market analysis. Models often have explicit and absolute mechanics designed with real-world knowledge. However, accurate model creation may involve implementing complex ideas that are inherently difficult to realize computationally [5].

Uncertainties arise when: real-world behavioral data is unavailable, it cannot be depicted accurately due to necessary simplification, or stochasticity is present [30, 36, 64]. Uncertainty in decision-making can be defined as the absence of adequate knowledge to inform a decision for future events [58]. Uncertainties disrupt the science of modeling when one must make implementation decisions derived from inadequate information. This type of uncertainty often injects models with assumptions and contingencies, which yield a systematic bias [5].

Structural uncertainty, or deep uncertainty, is a severe condition in which decision-makers do not know or cannot agree upon the relationship between variables [30, 35, 58]. 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 decision-makers can define the dynamics of a system empirically but are unaware of the parameter values associated with such dynamics [58]. 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. Decision-makers aware of this prejudice are skeptical of a model's genuineness [31]. Regardless, effective decisions must still be made [30, 17, 35].

1.2 Exploratory Modeling

The solution to uncertainty management is to explore plausible scenarios [5]. Exploratory modeling [5] deploys an ensemble of models that describe the system under varying conditions. A decision-maker searches the ensemble to identify which strategies are resilient [13, 6, 5, 58, 30]. This exercise does not encompass analysis of a single, ground-truth model as in consolidative modeling [5]. Instead, it is an iterative process where a decision-maker performs a series of experiments to understand the consequences of alternative hypotheses [6, 5, 30, 36].

Exploration addresses the issue of alternative hypotheses that classical approaches [5] cannot incorporate [16, 64]. From exploration, decision-makers gain awareness for the breadth of a system [13, 58, 35], pinpoint policies that perform adequately under a variety of conditions [58, 36], and divide up scenarios by those that are advantageous and those that are suboptimal [64]. Consequently, exploratory modeling has shown to be an attractive approach to deal with uncertainty in decision-making [13, 5, 29, 64, 58, 4, 15].

1.3 Limitations of Exploratory Modeling Practices

Exploratory modeling is achievable with sufficient computational power to evaluate numerous scenarios [5]. However, more computation does not necessarily decrease uncertainty [58, 56]. The use of multiple models increases the complexity of the methodology and can obscure system mechanics from the decision-maker [16]. Analytical approaches like sensitivity analysis, scenario planning, and prim are not conclusive when decision-makers use a large number of models. Thus, decision-makers are ill-equipped to generalize and explain the dynamics of the ensemble as a whole. If we address two inadequacies in current practice: (1) the explanation of large ensembles and, (2) the maintenance of alternative model structures, exploratory modeling methodology can mature further into science.

1.4 Contribution

An analytical tool built for generalization and explanation yields a proper explanation of large ensembles. It should provide insight into areas of the ensemble not otherwise explicitly evaluated and give the decision-maker a better understanding of how it generally behaves with results transposed into sensible chunks [16]. An explainable analysis of a model can accelerate decision-support by supplying more intuitive interpretations of experiments.

Furthermore, the maintenance of alternative model structures is advantageous because it allows decision-makers to ask questions with varying levels of abstraction. Exploratory modeling platforms [29] currently in use do not supply a mechanism for model structure variability. Therefore, the responsibility to incorporate alternative structures according to hypotheses falls on modelers to implement them, possibly causing exploration to evaluate structural uncertainties inadequately. Exploratory modeling methodology needs to be flexible and should manage structural variability subject to composition constraints [64]. Structural variability management allows for the maintenance of an ensemble that incorporates casual configurations of model structures to describe all plausible hypotheses [64]. The adoption of structural variability in exploratory modeling mitigates the burden on modelers and yields results concerning scenarios of interest more quickly.

In this work, we address and propose how to overcome each of these limitations. This thesis demonstrates how wrapping exploratory experiments with variability management can efficiently process hypotheses of varying structural constraints. Additionally, we demonstrate the use of an explainable analytical tool, namely a rule-based machine learning algorithm, that can generalize and explain model dynamics over both explored and unexplored portions of the ensemble. And in conclusion, we demonstrate how adopting both of these solutions enhance exploratory modeling methodology and solidify the scientific approach to decision-making under uncertainty.

1.5 Thesis Layout

The layout of this work is as follows. Chapter 2 describes background information on modeling for decision-making, exploratory modeling, and explainable machine learning. In Chapter 3, we propose a methodology that addresses two limitations in contemporary exploratory modeling practices. Chapter 4 outlines our implementation of the methodology introduced in Chapter 3. In Chapter 5, we perform validatory and exploratory experiments to form a case study. In Chapter 6 we summarize findings, and limitations of our work, and propose future research.

Chapter 2

Background

The genesis of this work lies in the limitations of modeling for decision-making when uncertainty is present and incorporates existing ideology in decision-making, uncertainty, exploration, and explanation in the context of modeling and simulation. In this chapter, we provide background information on these topics.

2.1 Decision-Making & Uncertainty

Generally speaking, a policy is a protocol enacted by a decision-maker to guide a system of interest to a desirable outcome. A bad policy choice by the decision-maker can have social and economic impacts that lead to amplified, undesirable outcomes. For this work, we must distinguish the responsibilities of a decision-maker from the responsibilities of a modeler. Decision-makers concern themselves with identifying the system scope, the possible outcomes and their values, and the consequences of their execution. A modeler, on the other hand, is concerned with realizing the system computationally with sufficient robustness to adequately evaluate plausible hypotheses. In both roles, fathoming and allowing for uncertainty can improve the intrinsic value of a modeling exercise.

Walker et al. [57] characterize uncertainty as "any departure from the unachievable ideal of complete determinism." In other words, if some event is not 100% likely, then the scenario inherently contains uncertainty. In Chapter 1, we accounted for two types of uncertainty: parametric and structural uncertainty. Here we further expand upon the ideology of uncertainty and articulate its connotation.

2.1.1 Calculating Uncertainty

Walker et al. [57] introduce three axes to describe uncertainties: the location of uncertainty, its severity, and its nature. Measuring uncertainties along these axes allows decision-makers to assess their consequences and allows modelers to wield them properly.

According to Walker et al. [57], 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 their definition of the target system. They attribute to the choice of target system boundaries [30] 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. They also address questions such as: "what is the outcome if we initially have 5% more of X?" Conversely, uncontrollable input 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.

The second axes proposed by Walker et al. [57] addresses the severity of uncertainty. The analysis allows decision-makers to anticipate the potency of uncertainty, and modelers should handle them appropriately not to dilute the quality of decision-making. Criteria to assess the severity of uncertainty may set a range from complete certainty to total ignorance with several intermittent levels in between. The first level of uncertainty is determinism or complete certainty. Determinism, where system mechanics are entirely understood, is an extreme that is unattainable in practice [58, 57]. The next level of uncertainty is statistical uncertainty, and it arises when system mechanics are reasonably understood and modeled sufficiently accurately, but the parametric values affecting mechanics are not precisely known and is described with statistical distributions [58, 57]. An even deeper level of uncertainty is scenario uncertainty,

and this is a situation where system mechanics are not well understood or agreed upon by decision-makers. Scenario uncertainty corresponds to the many plausible descriptions of the relationships between variables and results in sets of plausible scenarios and candidate policies [58, 57, 30]. Yet another level of uncertainty is called recognized ignorance. Ignorance falls into this category when familiarity with plausible scenarios is absent because decision-makers do not know the mechanisms at play or their probabilities. Recognized ignorance appears in two embodiments: reducible and irreducible. It is reducible if further research can clarify the uncertainty; if not, it is irreducible [58, 57]. The deepest level of uncertainty is total ignorance. Here, decision-makers only accept that they do not know what they do not know [58, 57].

The nature of uncertainty is the third axes proposed by Walker et al. [57] and is an exercise to analyze how uncertainty manifests, i.e., understand its root cause. In contrast to analysis of the location of uncertainty, the analysis of the nature of uncertainty is the study of how it arises and how to cope with it [57]. The nature of uncertainty comes in two variants: epistemic uncertainty and variability uncertainty. Epistemic uncertainty stems from imperfect knowledge, which may reduce after additional study of the target system [57]. Variability uncertainty develops from intrinsic differences of mechanisms within the system. Variability uncertainty commonly appears when the target system involves natural or human systems and their innate randomness [57]. In this case, such systems may react in unpredictable ways, and further research may not tame uncertainty.

2.1.2 Parametric & Structural Uncertainty

To decorate our definitions of parametric and structural uncertainty from Chapter 1, we project them onto the three axes of uncertainty. Parametric uncertainties located in inputs are a statistical-level of severity and epistemic. Structural uncertainties located in the context or model construe a scenario-level of uncertainty and are variable.

2.2 Exploration

Exploration is a practice within modeling methodology that embraces the presence of uncertainty by incorporating a variety of hypotheses about the mechanics of the target system [14, 5].

For decision-makers, the benefit of exploration is that they: (1) better understand the implications of various assumptions, and (2) gain a sense of when a policy is most effective [14]. Each of these characteristics lead decision-makers to a more informed policy choice, and exploration becomes a viable solution to uncertainty. Exploration is most suitable when systems exhibit irreducible levels of uncertainty and encompass inherent variability [57, 30]. As Davis [13] said, exploration "can help identify strategies that are flexible, adaptive, and robust." Interestingly, exploration exists in real-world applications such as climate prediction [17] and supply chain management [14]. These applications also highlight how exploration implementation yields a competitive edge and higher profits for industry [38].

2.2.1 Exploratory Modeling & Analysis

The practice of exploration involves two phases: exploratory modeling and exploratory analysis. Exploratory modeling conceptualizes, generates, and executes a series of computational experiments [5] that explore the implications of the hypotheses. Each experiment incorporates a plausible hypothesis about the target system: a "what if" scenario. The hypothesis may also encompass multiple model instances with varying structures and parametric values. The set of model instances that wholly represent all plausible hypotheses is known as the ensemble [58, 14, 29]. Exploratory analysis is the second phase of exploration and is analogical to decision-support [57, 29]. It synthesizes and interprets the results of multiple experiments that may contain millions of individual results from model instances. Participants are tasked in the analysis phase to deliver informative insight into the target system. For exploration to be a useful exercise in decision-making, both phases must adequately explore and inform on plausible hypotheses of the target system.

Consolidation The counterpart to exploration in modeling methodology is consolidation. In short, consolidative modeling is the practice of aggregating all known facts into a single package and using it as a predictive prosthesis [5]. [1] While exploration incorporates numerous

[1] Banks [5] provides an excellent theory that explains the challenges decision-makers and modelers face when they implement consolidative models. The Ultimate Combat Model illustrates the issues of inadequate robustness and scenario variability.

models, consolidation incorporates just one, and the severity of uncertainties are presumed negligible. While consolidation appears to be a more unequivocal practice, it is often not applicable to decision-making for real-world systems [30, 16, 64, 5].

Performing Exploration In exploration practice, a decision-maker first selects a general policy that conceptualizes the system and then observes the behavior of a subset of the ensemble. This process should adhere to the chosen policy. The decision-maker performs quantitative analysis on the observed results to identify the conditions under which the policy performs suboptimally. Since the policy improves with the new insight, decision-makers repeat this process until they reach a satisfactory policy [30, 5]; however, models are not predictive surrogates. They only provide partial insight into the system, and ideal policies become more apparent with subsequent experiments.

Robust Decision-Making Robustness is a critical pillar in the evaluation of alternative policies [35, 50, 39]. A policy is robust if it performs adequately when compared to alternatives in a large number of hypothetical scenarios [64, 36, 50]. Robust policies should minimize expected cost or regret [36, 51], but they do not necessarily maximize optimality [58]. 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 [36, 34]. Robust Decision-making [58, 64, 36, 35] is a derivative of the exploratory process but with additional objectives for the decision-maker. In Robust Decision-making, participants identify robust candidate policies and their vulnerabilities and trade-offs [36, 35].

2.2.2 Model Variability

Software Product Line Engineering Software Product Line Engineering enables the rapid development of low-cost, high-quality products by organizing software into reusable artifacts

[24, 44]. 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 [44]. Variability in software is the degree to which components are configurable, extendable, or exchangeable [10]. Flexible and diverse software products can be created just by managing variability [10]. Thus, a product can consist of a subset of reusable components expressed in terms of the features it delivers [24]. A feature is a distinctive functionality or mechanism in a software product that may encompass additional subfeatures [2, 64].

Features & Feature Models Well-defined software features facilitate a process known as feature-driven engineering [64, 53]. Feature-driven engineering is an adaptation of the software product line where variations and commonalities between components are described with a feature model [53]. The feature model, in turn, is a conceptual realization of mechanisms in a domain, and its features are defined with various levels of delivered functionality. Flexibility and configurability in the software are introduced via the adaptive composition of expansive features from more fundamental features. 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. Features in a feature tree express interrelationships that decompose into mandatory, optional, OR (inclusive disjunction), and XOR (alternative) subfeature constraints [43]. Features at deeper levels of the hierarchy are mostly elementary in functionality so that higher-level features resemble aggregative functionality.

Feature-Driven Engineering in Decision-Making In the context of decision-making, the target system represents the product suite, and the features in the feature tree represent plausible mechanisms of the target system. The target system's mechanics are categorized based on their interdependencies and then mapped to features in a feature model with suitable constraints. Plausible mechanisms that intend to describe the same system phenomena neighbor each other in the feature tree and are described as alternative if they are competing hypotheses or desired as OR if they are complementary. Mechanisms understood to be within the context of the target

system are described as mandatory, and mechanisms that are not completely understood to be within the context of the target system are optional. Through adequate mapping, the feature tree represents the ensemble of all plausible hypotheses of the target system. Therefore, to realize a scenario of interest, one must select only the features which pertain to the scenario via synthesis of a subtree of the feature tree.

2.3 Explanation

An essential facet of exploratory modeling is the explanation of large ensembles [5, 4, 64]. 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 [14, 39, 18, 12].

As discussed in Section 2.2 Exploration, decision-makers employ exploration to better understand the assumptions about the target problem [14]. Exploration generates a substantial amount of data that may obscure essential system mechanics from decision-makers [16]. This obscurity is caused by the limitations of the human ability to interpret large amounts of data, especially with time constraints [23]. Decision-makers mitigate with analytical tools such as sensitivity analysis and scenario planning [14, 29, 35]. These tools analyze experiment results and transpose them into interpretable facts, which are used as decision-support artifacts [14]. 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 [14]. There is a need for a more robust explanatory and analytical tool to infer hypotheses across both explored and unexplored portions of the ensemble. Supplementary tools, such as machine learning, should help draw inferences from extensive data and extrapolate across the ensemble with precision.

2.3.1 Machine Learning

Machine learning is the subfield of computer science and artificial intelligence that focuses on pattern recognition in large data sets [40, 41, 55]. This class of algorithms generalizes patterns in observed data so they can be extrapolated and matched to unseen, future data [40, 45]. Through this process, a subset of these algorithms are a predictive apparati [40, 41]. In the context of exploratory modeling, these tools are used to produce symbolics that are both descriptive and prescriptive [7]. Machine learning is used to conclude from system phenomena, and its vital components or assumptions, to reach the desired outcome or avoid a suboptimal outcome. In this context, the desired algorithms are explanatory tools [18] because they explain observed mechanics in the target system and predict how unexplored portions of the ensemble might behave.

2.3.2 Explainable Machine Learning

The explainability of machine learning algorithms is a growing subject of study [18, 1, 52, 12]. Published research acknowledges that machine learning is becoming an integral part of society, and practitioners are exhorted to concern themselves with the trustworthiness of their systems [18, 1, 52, 12]. Research proves that there are deficiencies in how existing algorithms produce confidence in their output. Blind adoption of output occurs when the algorithmic complexity is unfamiliar [18, 1]. Doran et al. [18] characterizes three levels of machine learning explainability: (1) systems which are opaque, (2) systems which are interpretable, and (3) systems that are comprehensible. Opaque systems are black boxes with input-output mapping mechanisms invisible to end users. A closed-source deployed machine learning model is called opaque. Systems are interpretable when their mapping mechanism is visible but require a mathematical and technical understanding. Comprehensible systems output symbolics along with their predictions. The symbolics allow users to understand why a system made a decision.

2.3.3 Learning Classifier Systems

One such class of comprehensible algorithms is Learning Classifier Systems [55, 7, 33, 23]. Learning Classifiers build a set of rules that are validated through reinforcement and generalized through genetic operations. The rules specify input constraints for an expected amount of reward or outcome. The Learning Classifier System consists of three fundamental components: the environment, the reinforcement program, and the classification mechanism [55, 33, 23]. The environment generates new scenarios, and the classification mechanism applies an action associated with an existing rule to maximize an expected reward. The reinforcement program then improves on the rule with the observed reward. Learning Classifier Systems iteratively generate rules which are robust, general, and validated. The rules express conditions where the outcome is either high or low. Rules are described through plain text or explanatory models such as heat maps or coordinate plots. In exploratory modeling, the Learning Classifier is trained and validated on data from explored portions of the ensemble and its results extrapolated to predict how unexplored portions of the ensemble might behave. Thus, the utility in exploratory modeling is made clear. Learning Classifier Systems are a crucial component of the solution presented in this thesis. Additional description of Learning Classifier Systems is available in Chapter 5.

Chapter 3

Strategy Learning System Framework

The Strategy Learning System framework enables 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. In this chapter, we outline the components of the Strategy Learning System. Chapter 4 discusses an implementation-specific adoption of the Strategy Learning System, and in Chapter 5 we illustrate it as a case study.

3.1 Framework Purpose

In exploratory modeling, a decision-maker first identifies the context of the target system, then analyzes uncertainties to generate plausible hypotheses that resemble the target system. A decision-maker equips a modeler with specifications, and they implement a surrogate of the target system to address alternative hypotheses through an ensemble of models. A decision-maker uses the ensemble to perform a series of computational experiments to understand the consequences of alternative hypotheses [5, 30]. The proposed framework improves on this procedure to organize the ensemble and casually generate new structural scenarios without requiring a modeler to implement them explicitly. Additionally, the framework uses a rule-based machine learning algorithm to produce explainable models of the ensemble. Figure 3.1 is a diagrammatic representation of the Strategy Learning System framework and shows information flow.

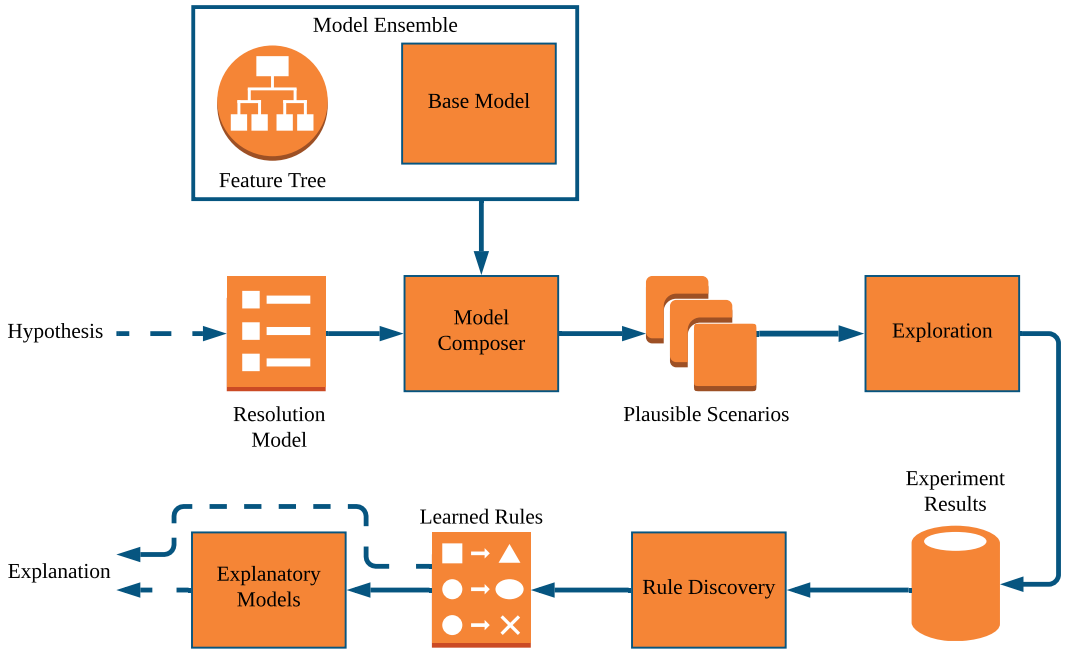

Figure 3.1: Strategy Learning System Framework

3.2 Feature Tree

A feature tree is the conceptual representation of the target system, and its features are symbolic aggregations of various levels of delivered functionality. A deep-level feature is the most basic in functionality, while a high-level feature is more complex and encapsulates lower-level features with constraints. Mechanisms within the context of the target system are mandatory, while mechanisms that are not completely understood to be within the context of the target system are optional. Additionally, plausible mechanisms that describe the same phenomena neighbor each other in the feature tree and are described as OR if they are complementary, and as XOR if they are competitive.

Figure 3.2 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 $f_{2,3}$.

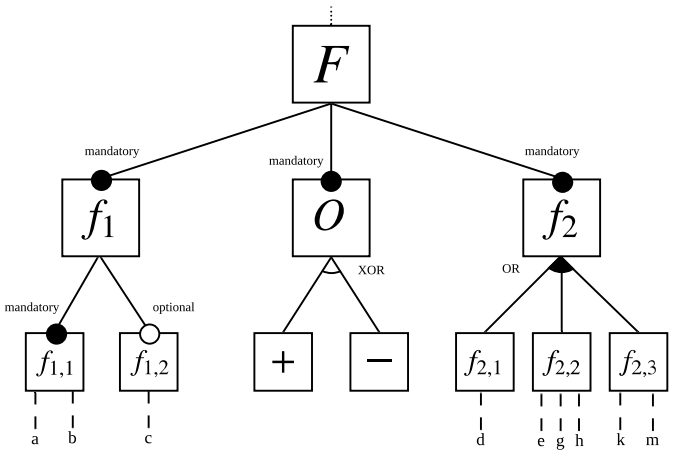

Figure 3.2: Feature Tree of Example Equation F

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.

3.3 Base Model

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. Built on top

are the increasingly more accumulative functionalities which may or may not be understood or agreed upon by decision-makers. Depending on the application, modelers can achieve greater functionality with (1) alternative flow controls, (2) inheritance and aggregation, and (3) decorator patterns. However, when the target system is complex, decorators are advantageous because they allow dynamic adoption of object behavior with no effect on inherited behavior. The implementation of the base model is hierarchical and emergent; it delivers a robust ensemble. Feature-oriented implementation allows casual automatic realizations of scenario models via a metaprogram to parse source code.

3.4 Resolution Model

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. Since the resolution model can be described with fluctuating levels of abstraction with both high- and deep-level features, they are included in generated scenarios— as long as their constraints are not violated. Subsequent experiments are performed by tweaking or exchanging the resolution model. This allows decision-makers to assess hypotheses on portions of the ensemble quickly with little overhead. A possible resolution model for equation F is:

Example Resolution Model for Equation F

include_feature(+)

include_feature($f_{2,1}$)

include_feature($f_{2,3}$)

3.5 Model Composer

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 walks the feature tree. Along a walk, the model composer performs feature composition and compiles the corresponding source code. For equation F, the model composer produces the following competing realizations with the above resolution model:

$$F = f_1(f_{1,1}) + f_2(f_{2,1}, f_{2,3})$$
$$F = f_1(f_{1,1}) + f_2(f_{2,1}, f_{2,2}, f_{2,3})$$
$$F = f_1(f_{1,1}, f_{1,2}) + f_2(f_{2,1}, f_{2,3})$$
$$F = f_1(f_{1,1}, f_{1,2}) + f_2(f_{2,1}, f_{2,2}, f_{2,3})$$

3.6 Exploration

Once the model composer has generated the plausible structural scenarios, the exploration component samples along parametric uncertainties. Sampling methods such as Monte Carlo Sampling or Hypercube Sampling are used to generate parametric values for the uncertainties adequately. This is essential to avoid an exhaustive search, reduce the computational cost, and maintain an adequate exploration of parametric uncertainties. This produces many model instances. 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.

3.7 Rule Discovery

The analysis of experimentation 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. This is described in greater detail in Chapter 5. 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 genetic operations. The rule creation process in Learning Classifier Systems is iterative, and it produces rules which are as-general-as-possible while maintaining resemblance to the patterns seen in experimentation results. The output of the Learning Classifier System is a set of rules which are easily 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 >
```

3.8 Explanatory Models

The rules generated by the rule discovery component are compiled into an explanatory model. In this framework, the explanatory model is a heat map, and the heat map illustrates which parametric and structural inputs in an experiment lead to an expected outcome. Outcome is visualized by a range of colors; warmer colors constitute a spectrum of desirable outcomes, while cooler colors constitute a spectrum of undesirable outcomes. The heat map is organized into two axes against which uncertainties are mapped. The horizontal axis plots uncertainties

under the control of a decision-maker. The vertical axis plots uncertainties not under the control of a decision-maker. In our example equation F, the horizontal axis is composed of the parametric values, and the vertical axis is composed of the structural compositions of F. The rules and heat maps allow a decision-maker to infer how the target system behaves under a hypothesis. The materials generated by the framework are used as decision-support artifacts and provide partial insight into a decision-making exercise. A decision-maker can construct subsequent experiments from this insight. The framework then provides an iterative process where the understanding of target system behavior is gradually improved through a series of experiments.

Chapter 4

Strategy Learning System Design & Implementation

In this chapter, we adapt the Strategy Learning System framework to a concrete instantiation. In the implementation we utilize: (1) a 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. In Chapter 5 we illustrate it as a case study.

4.1 Contaminant Plume Model: Overview, Design Concepts, & Details (ODD)

Agent-based models often illustrate behavior that is not easily described by mathematics. The absence of a mathematical formulation yields ambiguity and can be enigmatic to a model-user unfamiliar with its specifications. Large or complex models underscore this. Ambiguity in models leads to marginal reproducibility and expansion. Grimm et al. [22] proposed ODD to amend this issue.

ODD has three parts: Overview, Design Concepts, and Details. The Overview lists the model's purpose, state variables, and metaprocess scheduling. Design Concepts describe the model's characteristics. Together, Overview and Design Concepts construct a high-level explanation of the model. The last part is Details, and it describes the initialization process, model inputs, and low-level behavior of agents.

4.1.1 Overview

Purpose

In recent years, commercial and governmental use of Unmanned Aerial Vehicles (UAVs) has added a wide range of applications [9, 42]. The Contaminant Plume model (the model) [47] simulates a practical managerial application of UAVs. The model is based on previous work [37, 47], and is a proxy to demonstrate how a collection of UAVs can cooperate for a UAV Swarm. The intra-swarm cooperation breeds emergent behavior because a single UAV's decision affects the decisions made by other UAVs. 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.

State Variables and Scales

The model is written in both NetLogo [25] and Scala [60], where the Scala 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 popular adaptation in Modeling practice. For this documentation, we reference the model as a single implementation. The model contains a hierarchy of four entities: Contaminant Plume, UAV, a Swarm—composed of two or more UAVs, and the Environment. Figure 4.1 is a Conceptual High-level Class Diagram that represents the entity relationships.

In the following subsections, we describe the state variables of the four entities. It is important to note that these are not exhaustive lists of all the attributes these entities contain. In section 3.1 Inputs, we list additional attributes. The distinguishing factor between the attributes listed in this section and Inputs is that the state variables are detached from user interaction while the inputs are varied by the user to identify diverse emergent behavior.

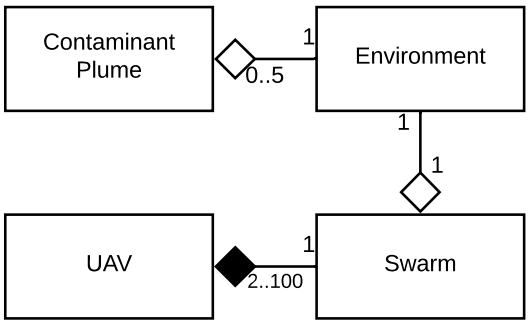

Figure 4.1: Contaminant Plume Model Conceptual Hierarchical High-level Class Diagram

Contaminant Plume A Contaminant Plume mimics an undesirable pollutant, toxin, or synthetic biological weapon that is mapped and decontaminated. In this scenario, the Contaminant Plumes are circular with a linearly decreasing density that radiates from its center. In this context, it is the role of the UAVs to map and sanitize the Contaminant Plumes. The Contaminant Plumes are composed of the following state variables:

- Plume spread patches is the radius of the Contaminant Plume in Environment patches (patches are defined in 1.2.4 Environment).

UAV UAVs are aircraft with no pilot on board. They are either autonomous or controlled from the ground. UAVs possess an array of sensors that allow them to perceive and react to their surroundings. In this example scenario, UAVs are autonomous and are composed of the following state variables:

- Location is the coordinate where the UAV is located in the Environment.
- Heading is the UAV's trajectory. It is measured in compass headings.
- Velocity is the rate at which a UAV moves in a single time step and is set to 0.5 patches per step.
- Detection time is the time step when the UAV first detects a Contaminant Plume.
- Sensor reading is the sensor reading which measures the density of a Contaminant Plume at the UAV's location.

- Flockmates is the group of UAVs within their respective vision (vision is described in 3.1 Inputs).

- Nearest neighbor is the flockmate that is physically closest.

- Best neighbor is the flockmate with the highest plume reading.

- Desired heading is the trajectory to which a UAV is maneuvering.

- Random search time is the number of time steps a UAV will continue on its current heading before turning when the global search policy is random search or symmetric search (search policies are described in Process Overview).

- Region is the portion of the Environment in which a UAV is responsible for mapping when the global search policy is symmetric search.

- Symmetric search max region reading is the maximum plume reading a UAV has detected while mapping its current region when the global search policy is symmetric search.

- Symmetric search region time is the amount of time a UAV will spend in its current region before it considers moving to another region when the global search policy is symmetric search.

Swarm A UAV Swarm is a collection of UAVs that are controlled by one or more ground-based systems. High-level commands sent to the Swarm are distributed and carried out by individual UAVs. Emergent behavior is exhibited due to intra-Swarm cooperation that would not appear from examining a single UAV operating companionless [37]. This scenario contains a single Swarm of homogenous UAVs that have three distinct C2 policies. The C2 policies are discussed in detail in Submodels. The Swarm is composed of the following state variable:

- Coverage all is a list that contains the real-time and historical coverage sensor readings of the Swarm. Given coverage data decay (defined Swarm Inputs) denoted as CDD, coverage all is defined as follows:

$$\{\text{sensor\_reading}(t, \text{UAV}_i) : t \in [t_n - CDD, ..., t_n], \text{UAV}_i \in Swarm\} \tag{4.1}$$

Environment The Environment functions as a commandant's designated area of interest and is where one or more Contaminant Plumes reside. The Environment is composed of the following state variables:

- Patches are units of area and are set to 5x5 pixels. The Environment is 195x100 patches.
- Ticks are a measurement of time native to NetLogo. A tick can be considered a single iteration– the model's entities are updated and performance is logged. Time intervals have thus far been described as time steps, but they can be considered in terms of ticks.
- The Contaminant Plumes reside in the Environment and there may be one to five Contaminant Plumes in an episode.
- The Swarm resides in the Environment and may contain two to 100 individual UAVs.

Figure 4.2 is a Conceptual Class Diagram that illustrates the state variables for the four entities described above. This Conceptual diagram will evolve as the additional model specification is introduced.

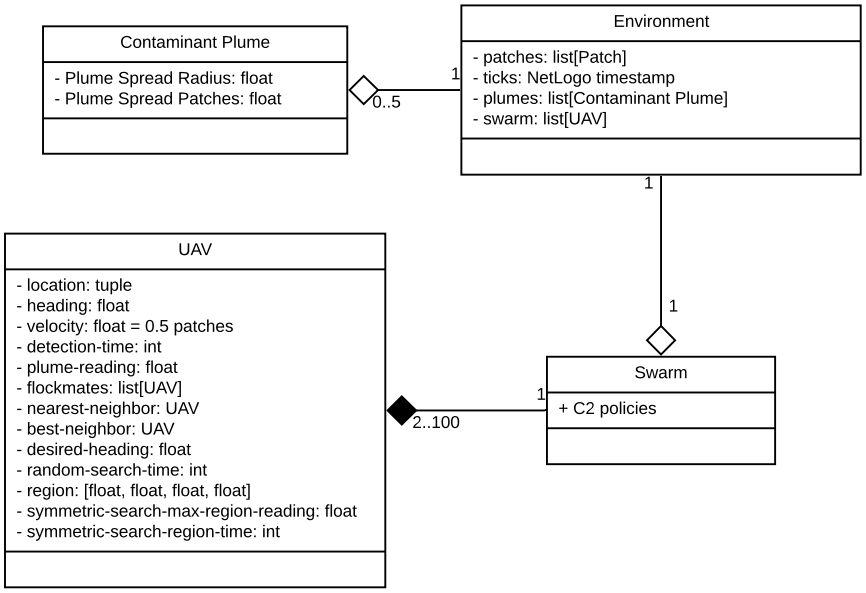

Figure 4.2: Contaminant Plume Model Conceptual Detailed Class Diagram for the model entities

Process Overview

C2 policy influence on Swarm collaboration varies. One of these policies is commissioned at the start of an episode.

Random Search Policy Random search [25] 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 [37, 46] is a policy influenced by behavior observed in Animalia, namely in flocks of birds, herds of land animals, and schools of fish [59]. In this scenario, flocking behavior emerges from UAVs using three BOIDS [59] rules:

- Separation - an Agent steers to avoid overcrowding its flockmates.
- Alignment - an Agent aligns its heading with the average heading of its flockmates.
- 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 [25] divides the Environment into symmetrical regions and initially assigns a UAV to each region where it searches for Contaminant Plumes. UAVs search their assigned region 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.

4.1.2 Design Concepts

Emergence In both flock and symmetric search policies, swarming phenomena emerges from individual UAV traits— where a UAV's location, plume reading, and flockmates dictate its short-term impermanent actions. These actions are observed as the Swarm converges around one or more Contaminant Plumes. Conversely, in random search, no system-level phenomena emerge because UAVs behave stochastically and are desensitized by their surroundings. In all cases, there are no explicit rules to control the Swarm.

Adaption Adaptation is observed in symmetric search when individual UAVs maneuver to regions with higher Contaminant Plume densities. Nevertheless, UAVs never change their behavioral rules.

Fitness The model is fitness-seeking as UAVs are explicitly tasked with exploring the Environment to maximize their plume reading. The model has three performance metrics:

- Coverage std is the standard deviation of coverage all.
- Coverage mean is the average of coverage all.
- Coverage percentage is the number of patches that contain the Contaminant Plume, and have been visited by a UAV, divided by the number of patches that contain the Contaminant Plume. Coverage percentage is defined as follows:

$$f_t \leftarrow \frac{\sum_{i=1}^{|\text{patches}|}(\text{plume\_density}(\text{patch}_i) > 0) \times (\text{visited}(\text{patch}_i) = 1)}{\sum_{i=1}^{|\text{patches}|}(\text{plume\_density}(\text{patch}_i) > 0)} \tag{4.2}$$

Sensing UAVs know their location, heading, velocity, vision, plume reading, Environmental wind heading, and wind speed (vision, wind heading, and wind speed are described in section 3.1 Inputs).

Interaction A UAV can interact with its flockmates, best neighbor, and nearest neighbor. Additionally, UAVs may decontaminate the Contaminant Plume. Interactions are detailed in Submodels.

Stochasticity Stochasticity is prevalent in the model. The Contaminant Plume's location is initially randomly assigned as this best model's uncertainty; the random search policy is wholly stochastic, and the number of ticks a UAV spends in its region for symmetric search is randomly assigned with two variables.

Collectives UAVs collectively form the Swarm, and UAVs accredit a subset of the Swarm as distinct flockmates.

Observation At each tick, all performance measurements are logged. Fitness analysis of the model is enabled through the fitness metrics.

4.1.3 Details

Inputs

The feature tree [63] in Figure A.1 demonstrates the hierarchy and dependence of all model features. Elements at the deepest level are the most rudimentary, while elements at the highest level are the most aggregate. The pivotal criteria to construct the feature model is dependency and concreteness.

Contaminant Plume Inputs Contaminant Plumes have the following input variables:

- Plume spread radius is the radius of the Contaminant Plume as a percentage of the Environment width.
- Plume decay rate is the amount that a Contaminant Plume naturally decreases in size in a single tick.
- Plume decontamination threshold is a percentage of the Contaminant Plume's original size. When the Contaminant Plume reaches this size, the episode terminates.

UAV Inputs UAVs have the following input variables:

- Decontamination strength is the amount a single UAV can decontaminate the Contaminant Plume size in a single tick.

- Vision defines the radius of the circle around the UAV where it can detect other UAVs.

Swarm Inputs The Swarm has the following input variables:

- Population is the total number of UAVs in an episode and the Swarm.
- World edge threshold is the minimum distance allowed between a UAV and the Environment edge before it must turn.
- Max world edge turn is the maximum angle a UAV can turn away from the Environment edge after it enters the world edge threshold.
- Coverage data decay is a temporal threshold for calculating the Swarm's coverage. For example, given a coverage data decay of two, coverage all contains the sensor reading at the current tick and the previous tick for each UAV.
- Global search policy is the search policy which the Swarm adheres to in the current episode.
- Random Search Inputs:
    - Random search max heading time is the maximum allowed time for a UAV to continue on its trajectory before turning when the global search policy is random search.
    - Random search max turn is the maximum allowed angle a UAV can turn when the global search policy is random search.
- Flock Search Inputs:
    - Minimum separation is the minimum distance allowed between any two UAVs within the Swarm.
    - Max align turn is modeled after BOIDs flocking [37] and is how UAVs align their heading with their flockmates.
    - Max cohere turn is modeled after BOIDs flocking [37] and is how UAVs steer towards their flockmates.

    - Max separate turn is modeled after BOIDs flocking [37] and, contrary to alignment, is how UAVs steer away from their nearest neighbor if they get too close.

- Symmetric Search Inputs:

    - Symmetric search max turn is the maximum allowed angle a UAV can turn.
    - Symmetric search region threshold is the distance allowed between a UAV and its region border before it must turn.
    - Symmetric search min region time is the minimum allowed amount of time a UAV must spend in its region before considering moving to another region.
    - Symmetric search max region time is the maximum allowed amount of time a UAV can spend in its region before considering moving to another region.

Environment Inputs The Environment encompasses the following input variables:

- Number plumes is the number of Contaminant Plumes in the current episode.
- Wind Speed is the velocity at which the wind is moving. Wind speed affects the location of Contaminant Plumes as well as UAV regions.
- Wind heading is the wind's trajectory.

A complete specification of model inputs can be found in Table A.1. Figure 4.3 is a complete Conceptual Class Diagram and contains both high-level methods, inputs, and state attributes of the model.

Initialization

On initialization, the Environment is first setup followed by the Contaminant Plumes then the UAV Swarm. Once NetLogo's tick counter and all the entity state variables are reset, the Environment is considered initialized. Next, the Environment creates the Contaminant Plumes, and they are then placed randomly in a central area that makes up one-quarter of the Environment. Lastly, the Swarm is initialized, and the UAVs are dispersed randomly around the Environment. Figure 4.4 is a High-level Sequence Diagram depicting the initialization process.

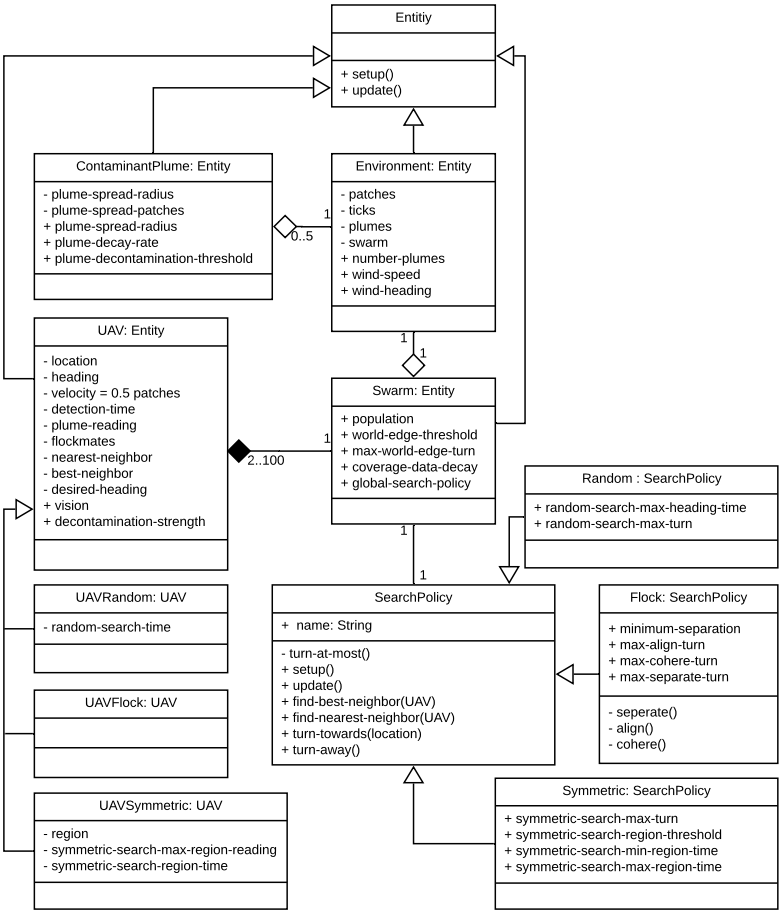

Figure 4.3: Contaminant Plume Model Conceptual Detailed Class Diagram for the Extended Plume Model

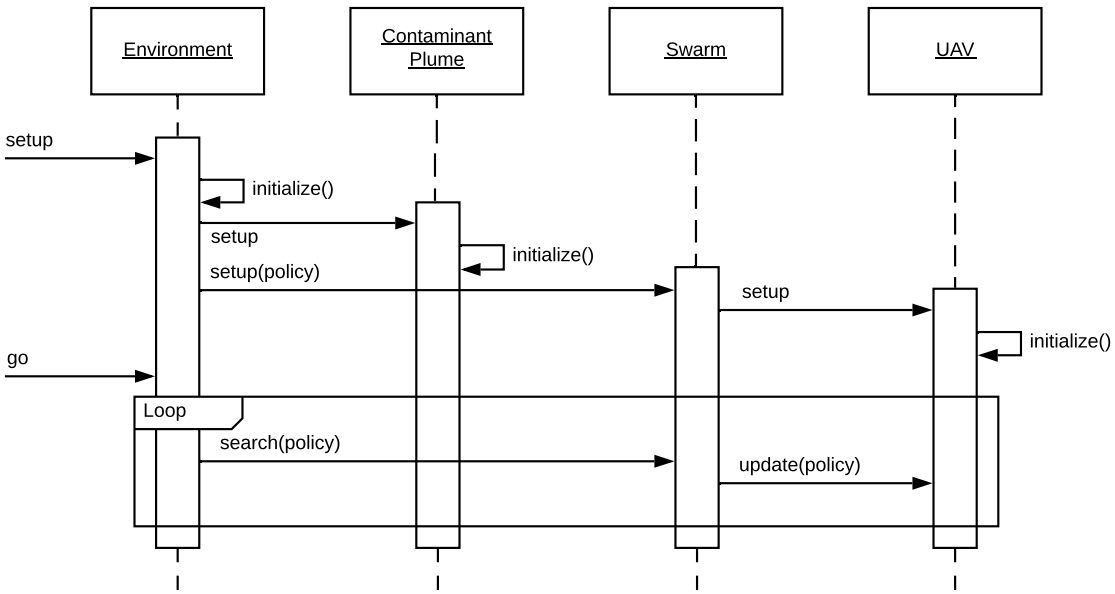

Figure 4.4: Contaminant Plume Model High-level Sequence Diagram of Initialization

Figure 4.5 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 ticks with the upper right being random search policy, the lower left being flock search policy, and the bottom right being symmetric search policy.

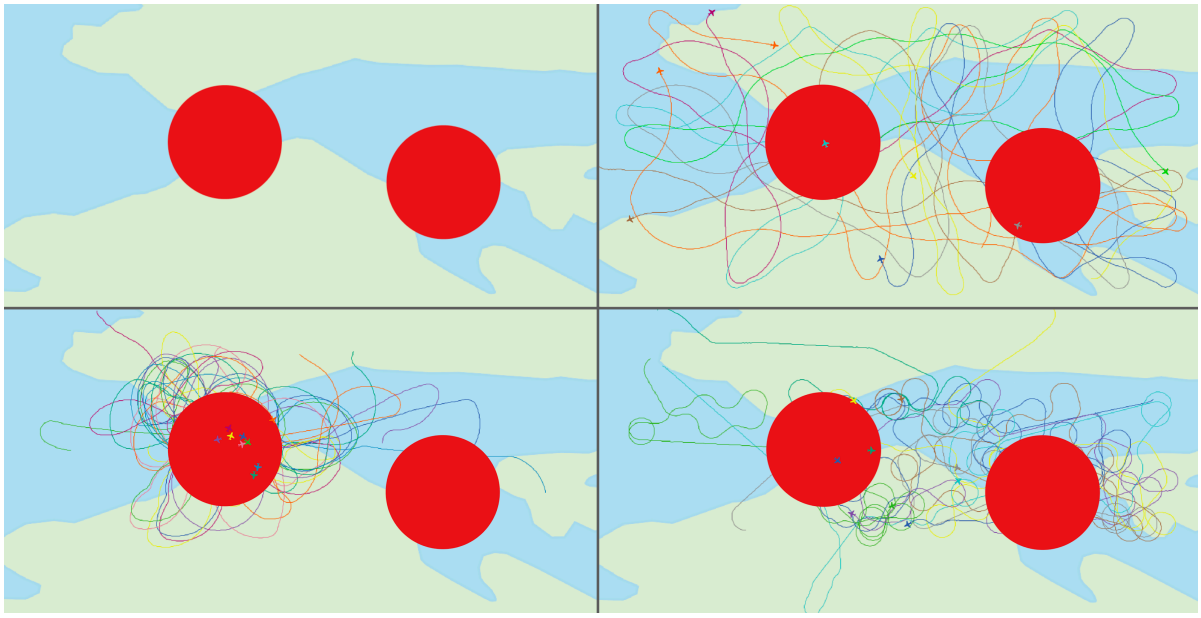

Figure 4.5: Contaminant Plume Model Screenshots

Submodels

The policies make use of helper functions which form universal conduct between policies. Universal conduct enforces restrictions on where UAVs are allowed to operate and how to handle transgressions on the restrictions. The helper functions are:

- `find-flockmates()` sets a UAV's flockmates to a list of other UAVs within this UAV's vision.

- `inside-of-world-edge-threshold()` returns true if a UAV is within the world edge threshold.

- `move-back-in-world-bounds()` sets a UAV's desired-heading to a random point within the Environment's center one-half.

- `turn-uav(turn-allowed)` adjusts a UAV's heading towards its desired-heading by the amount turn-allowed if a UAV's desired-heading is not its current heading.

Random Search Policy A UAV first sets its random search time to a random integer between zero and the input argument random search max heading time. After the number of ticks has elapsed, a UAV sets its desired heading to a random angle between zero and 360 degrees, then sets a new random search time. Figure 4.6 is a Detailed Sequence Diagram that describes the random search policy.

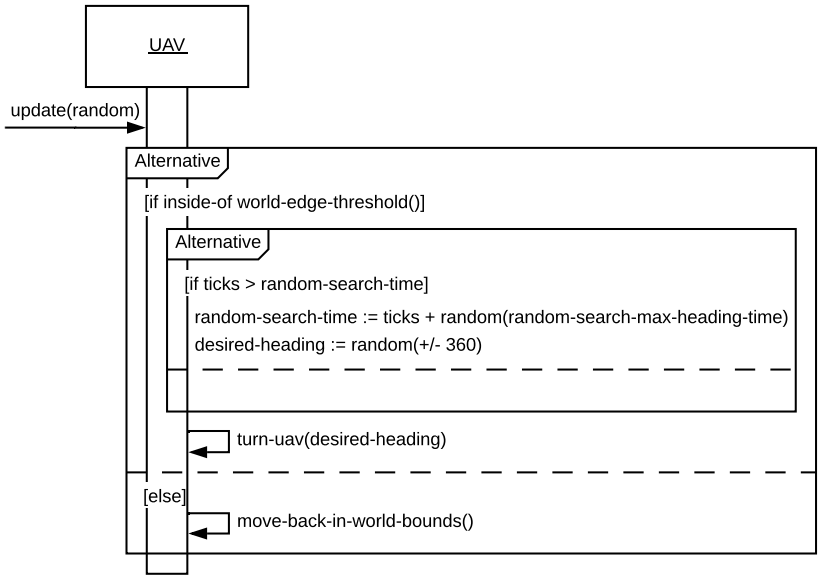

Figure 4.6: Contaminant Plume Model Random Search Policy Detailed Sequence Diagram

Flock Search Policy A UAV first determines its flockmates. If a UAV has no flockmates, then it advances on its current trajectory as flock search is entirely collaborative. Otherwise, a UAV continues to determine both its best neighbor and nearest neighbor. If the distance between a UAV and its nearest neighbor is less than minimum separation, then the UAV uses the BOIDS Separation rule to veer in an opposite trajectory. If the plume reading of a UAV's best neighbor is higher than its own, then it veers towards the average heading of its flockmates and uses the BOIDS alignment rule to veer towards the average heading of its flockmates. Figure 4.7 is a Detailed Sequence Diagram that graphically describes the Flock Search policy.

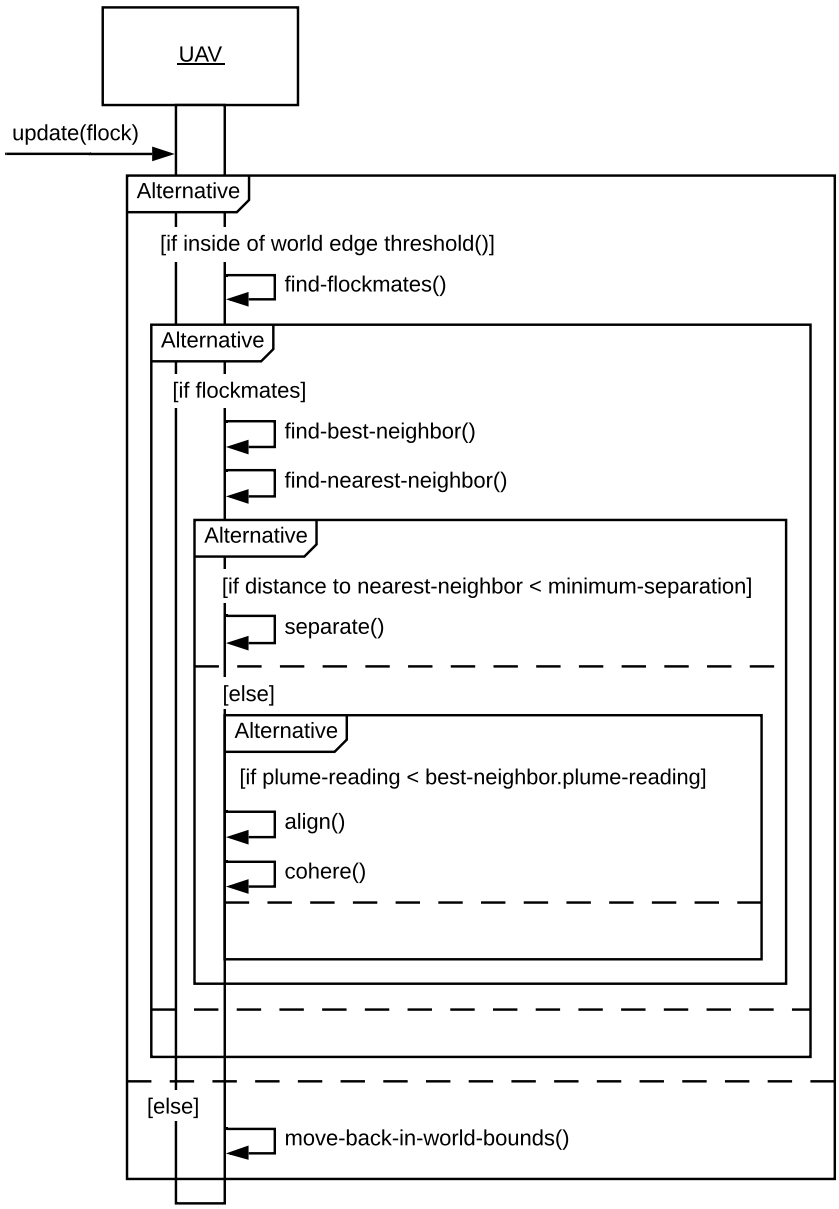

Figure 4.7: Contaminant Plume Model Flock Search Policy Detailed Sequence Diagram

Symmetric Search Policy A UAV first determines if it is within its assigned region and then checks if symmetric search region time has elapsed. If symmetric search region time has elapsed, it sets a new symmetric-search-region-time and moves to the region of its best neighbor, but only if the plume reading of its best neighbor is higher than its own. Alternatively, if symmetric search region time has elapsed, the UAV checks if its random search time has elapsed. If random search time has elapsed, the UAV sets its desired heading to a random angle between zero and 360 degrees, then sets a new random search time. All regions drift

in the direction of wind heading. Figure 4.8 is a Detailed Sequence Diagram describing the Symmetric Search policy.

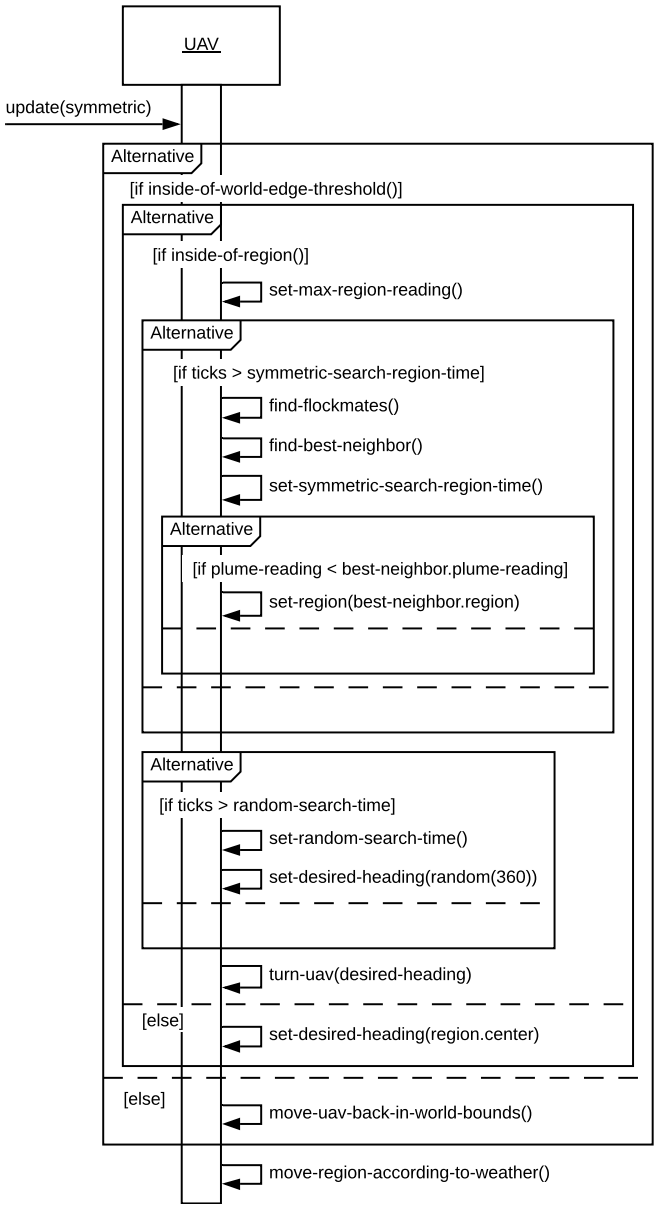

Figure 4.8: Contaminant Plume Model Symmetric Search Policy Detailed Sequence Diagram

4.2 EMA: Exploratory Modeling & Analysis Workbench

The Exploratory Modeling and Analysis Workbench [29, 27] is a Python package that supports robust and multi-objective optimization for exploratory modeling under uncertainty. The workbench accommodates generation of computational experiments for base models written in

Vensim, Netlogo, and Excel. Experiments are executed sequentially or in parallel with replications in either a single computer or cluster environment. The workbench treats the base model as a function of its parametric and structural uncertainties, and parameters are boolean, integer, real, or categorical. Parametric values stem from sampling methods such as Monte Carlo sampling, Latin Hypercube sampling, and Full Factorial sampling. Structural uncertainties are obtained by policy levelers external to the base model and the workbench. Experiment results are scalar, array, or time-series outcomes which are stored externally. Additionally, the workbench incorporates classification and regression trees, PRIM, sensitivity analysis, and feature scoring for the analysis of experiment results. A detailed specification for the Exploratory Modeling and Analysis Workbench can be found in [29, 28].

The Strategy Learning System uses a light-weight fork [32] 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 cluster [3] [1] 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, then rule discovery is performed.

4.3 XCS: Accuracy-based Learning Classifier System

This section outlines two Accuracy-based Learning Classifier Systems, XCS and XCSR. XCS is the traditional formulation, while XCSR is modified to allow for real-valued inputs.

XCS

The Accuracy-based Learning Classifier System (XCS) [55, 7, 33, 61, 8, 26] is a Michigan-style classifier which incorporates both reinforcement learning and genetic algorithms to form a set of rules that describe an environment or dataset. It is an online supervised learning algorithm

[1] We acknowledge the Auburn University Hopper Cluster for support of this work.

which can operate on both single-step and Markovian multistep problems with delayed rewards [55]. The XCS consists of three fundamental components: the environment, the reinforcement program, and the classification mechanism. This is depicted in Figure 4.9 which is modeled after one found in [8].

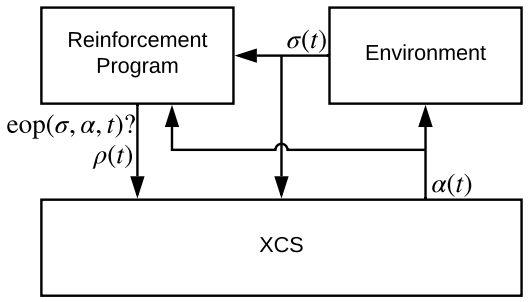

Figure 4.9: XCS Components

The environment is a surrogate of the scenario we wish to describe. It can resemble problems such as a simple boolean operation such as XOR, or more complex problem such as the n-bit Multiplexer, Woods2 [61, 26], or determining aircraft maneuvers [33]. It is a source of training where the classification mechanism learns actions to maximize its expected delayed reward ρ. The reinforcement program behaves as the supervised component of XCS which informs the classification mechanism on the adequacy of its action selection for the current state σ. The classification mechanism's fundamental attribute is a set of rules known as the population of classifiers [P]. The population is a set of condition-action-prediction tuples (if-then rules). Classifiers are evolved iteratively and represent the collective knowledge of XCS. A classifier's predicate (sometimes referred to as its condition) C specifies the σ values to which it applies. Predicates are discrete strings, $C \in \{0, 1, \#\}^L$, where each predicate attribute C_i can be assigned only a discrete value: zero, one or a wildcard $\#$ that matches both zero and one.

As in all Michigan-style classifiers, the whole population describes the environment (versus Pittsburgh-style where a single Classifier describes the environment). A classifier's action α specifies the proposed response to σ. A classifier's predicted payoff p is the learned reward for

applying α given σ: $(\sigma, \alpha) \rightarrow p$. In addition to C, α and p, classifiers contain other attributes that describe their proficiency. An exhaustive list of classifier attributes can be found in [8].

Learning Cycle For a given time step t, XCS perceives the environment through σ_t. XCS then forms the match set, $[\text{M}]_t$, using σ_t:

$$[\text{M}]_t = \{cl : cl \in [\text{P}]_t,\ \text{matches}(cl, \sigma_t)\}, \tag{4.3}$$

where:

$$\text{matches}(cl, \sigma_t) = cl.C_i == \sigma_{t,i} \text{ or } \#,\ \forall\ i \in [1, ..., L]. \tag{4.4}$$

If the number of classifiers in $[\text{M}]_t$ is less than a threshold or no classifiers in [P] match σ_t, then covering occurs. Covering is the process to generate a new classifier that does match and assign it a random action. Next, action selection is done by computing the expected payoff for each action present in $[\text{M}]_t$ and selecting one through a policy. The selected action is used to form the action set $[\text{A}]_t$, which is subset of $[\text{M}]_t$ where classifier actions are the chosen action:

$$[\text{A}]_t = \{cl : cl \in [\text{M}]_t,\ \ cl.\alpha = \text{chosen action}\}. \tag{4.5}$$

The chosen action is applied in the environment where σ_{t+1} is observed. The reinforcement program provides feedback on the effectiveness of the action through ρ_t and returns a flag that signals if the episode has terminated. The classification mechanism then updates the classifier's in $[\text{A}]_t$ using ρ_t. ρ_t updates the attributes of classifiers in $[\text{A}]_t$ to enforce favorable actions and punishes unfavorable ones. Rule discovery is performed by selecting two parents to generate an offspring through a genetic algorithm. The offspring is then added to [P]. Lastly, t is incremented and the process is repeated if the episode has not terminated.

XCS Deficiency XCS is limited in the problems it is applicable to due to the discrete space of C. The limitation is related to how classifiers evolved in the genetic algorithm and the immense state-space of continuous values in real-world problems [62]. This adversary causes XCS to apply only to a definitive number of environments because encoding many environments as binary strings is intractable. [21, 20, 19, 62, 54, 11] propose several adaptations of XCS for real-valued inputs (XCSR) where classifier predicates contain real values: $C_i \in \mathbb{R}$. For this

framework, we adopt ideas from varying XCSR concepts which promote building an optimal population for real-valued environments.

4.3.1 XCSR: XCS for Real-valued Inputs

Recent research into derivatives of XCS allow for real-valued scenarios [21, 20, 19, 62, 54, 11]. We adopt several ideas from proven methods to construct XCSR and contrast them with methods found in Wilson's XCS. For the XCSR implementation [49] in the Strategy Learning System framework, we combine the environment and the reinforcement program to increase cohesion. The XCSR learning cycle is shown in Figure 4.10, and a Detailed Class Diagram for XCSR can be found in Figure B.1.

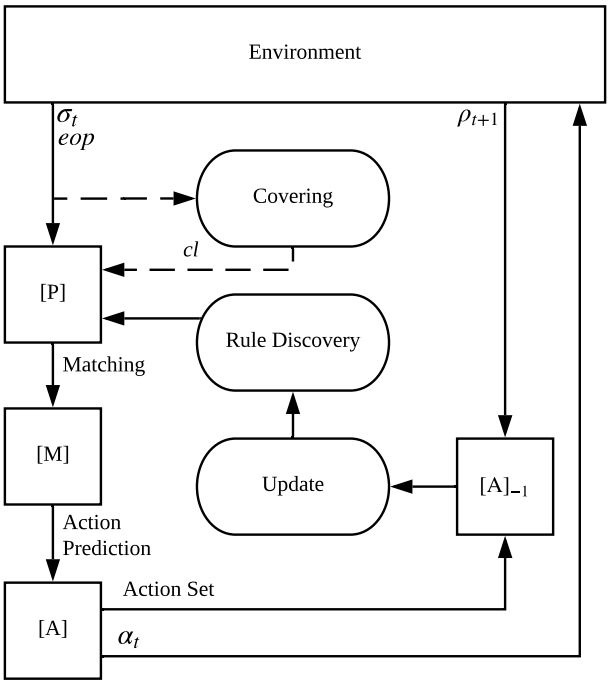

Figure 4.10: XCSR Learning Cycle

Classifier Predicates 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:

$$\text{matches}^{'}(cl, \sigma_t) = C_i[0] \leq \sigma_{t,i} \leq C_i[1], \ \ \forall \ \ i \in [1, ...L]. \tag{4.6}$$

Predicate attributes initially set to (0, 1) 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 genetic operation to modify the range of accepted values.

Deletion Procedure In XCS, deletion is performed when the size of [P] has exceeded a threshold and is roulette-wheel selection. In XCSR, we substitute this with an epsilon-greedy policy using a beta distribution. Here, classifiers are sorted from worst-to-best accuracy. Classifiers with low accuracy have a higher chance of deletion while classifiers with high accuracy have a very low chance of deletion.

Offspring Selection Procedure In XCS, offspring selection is achieved using roulette-wheel selection. In XCSR, we substitute an epsilon-greedy policy using a beta distribution as we did for the deletion procedure. Here, we sort classifiers from best-to-worst accuracy so those with high accuracy have a higher chance of being selected to breed an offspring.

4.4 Strategy Learning System Implementation

The Strategy Learning System is implemented [48] in Python and acts as a mediator between the components in the architecture. Figure 4.11 is an adaptation to Figure 3.1 that substitutes (1) the base model for the Contaminant Plume model, (2) the exploration component for the EMA Lite, and (3) the rule discovery component for XCSR. Figure 4.12 is a High-level Class Diagram of the Strategy Learning System implementation.

The Contaminant Plume model feature tree is encoded as Strategy Learning System parameters. This representation of the feature tree is shown in Figure 4.13. Here, features under control of the decision-maker are shown in green while features not in the control of the decision-maker are shown in red. The Contaminant Plume model performance metrics are shown in blue.

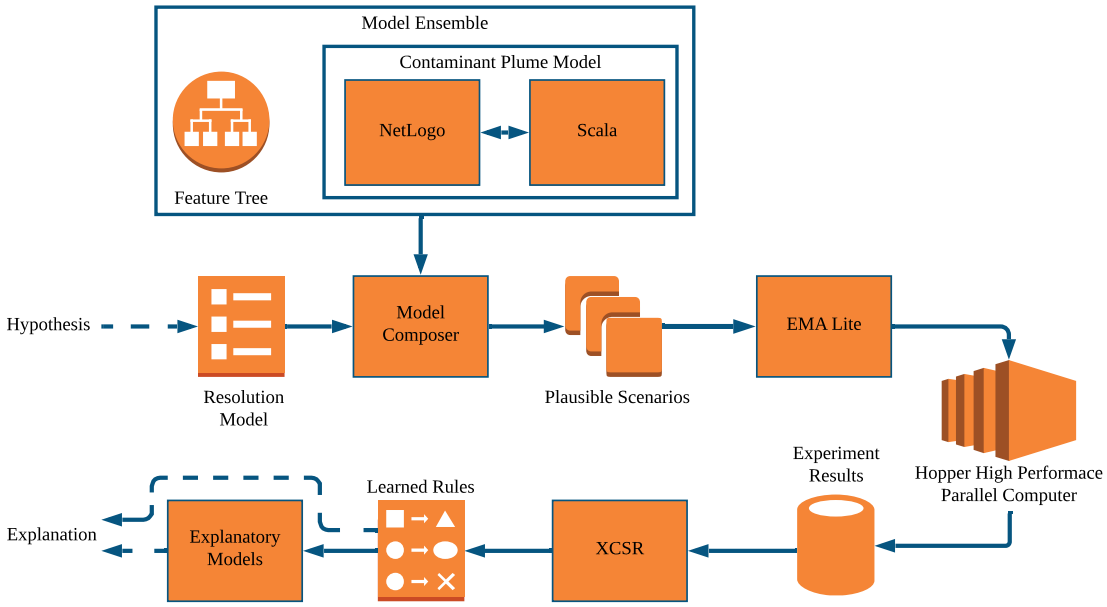

Figure 4.11: Strategy Learning System Implementation Information Flow

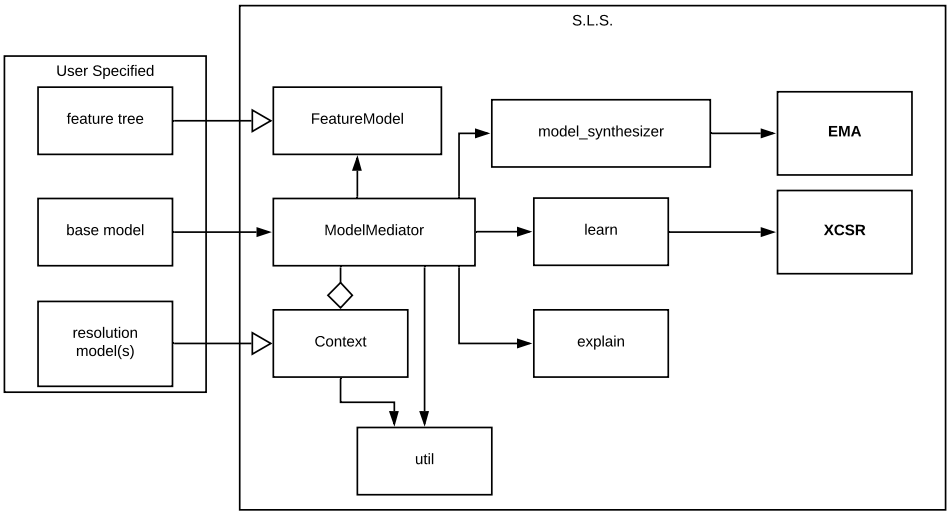

Figure 4.12: Strategy Learning System High-level Class Diagram

In the implementation realization 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 EMA Lite are processed by the Strategy Learning System. Controllable and uncontrollable feature argument

Figure 4.13: Contaminant Plume Model Feature Tree Implementation Realization

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:

$$R(\sigma_t, \alpha_t) \leftarrow AUC_f(a, b) = \sum_{t=a+1}^{b} (t_i - t_{i-1}) \times \frac{f(t_i) + f(t_{i-1})}{2}, \tag{4.7}$$

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 are then 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:

$$\rho_{t+1} \leftarrow R(\sigma_t, \alpha_t) - \left[\zeta R(\sigma_t, \alpha_t) \frac{||\alpha_t - \hat{\alpha}_t||_2}{\max_{\alpha \in A} ||\alpha||_2}\right], \tag{4.8}$$

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.

Chapter 5

Strategy Learning System Validation & Evaluation

In this chapter, we use the Strategy Learning System implementation from Chapter 4 to perform experiments in two category types. The first type of experiments are validatory and validate the theoretical tractability of the Strategy Learning System and the correctness of its implementation. The second type of experiments are exploratory and provide 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. Hyperparameters for EMA Lite and XCSR are determined by the complexity of the resolution model. 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:

- The number of model instances to execute from the ensemble.
- The number of replications for each model instance.
- The run length of each instance in ticks.

The experiment hyperparameters for XCSR are:

- Episode length determines how many instances we present to XCSR.
- The number of bins determines the granularity of feature arguments. For example, if bins is set to 3, then feature arguments are converted to low, medium, or high depending on their value.

- The maximum number of classifiers in the population [8].

- ζ is the discount factor for equation 4.8.

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 XCSR learned rules. For validatory experiments, we supply additional heat maps generated by a Multi-Layer Perceptron clustering algorithm to further corroborate the observation rule sets. The Multi-Layer Perceptron is 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.

5.1 Validation Experiments

5.1.1 Validation Experiment 1

Validation Experiment 1 was designed as simply as possible to easily understand both the exploratory results and the learned rules 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, the aim of this experiment was to merely test the capabilities of the Strategy Learning System.

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 5.1 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

Table 5.1: Validation Experiment 1 Parameters

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

experimentation with the base model, we determined that 1,000 ticks was 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 [26] and a lightweight grid-search.

Experiment Predictions Equations 4.2, 4.7, and 4.8 will reward high populations since they present the opportunity for the Swarm to cover a larger portion of the model environment. Due to this characteristic, we expect that outcome increases proportionally to population.

Observations & Discussion

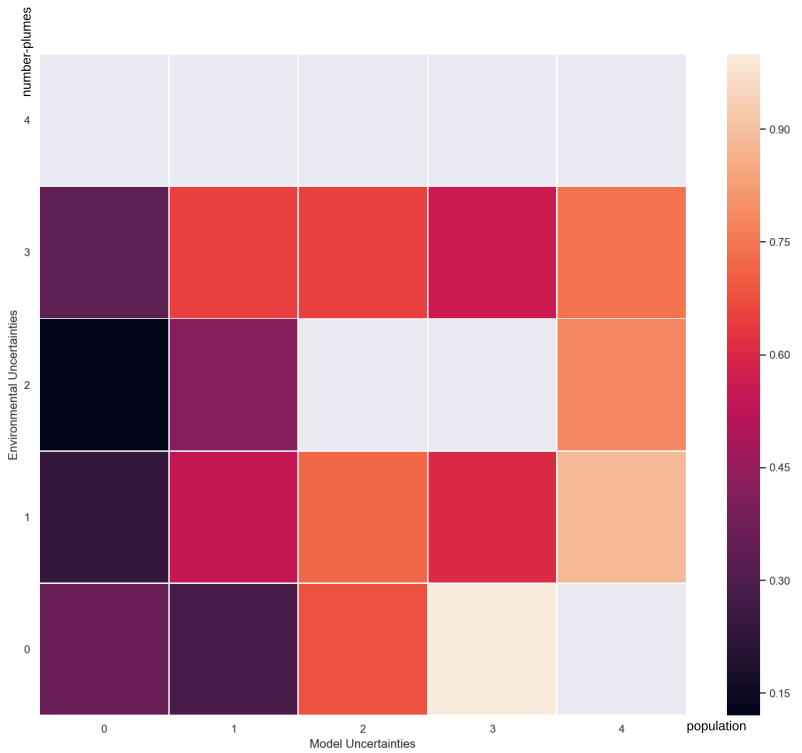

Figure 5.1: Validation Experiment 1 EMA Lite Exploration Heat Map

Figure 5.1 visualizes the EMA Lite exploratory results for Validation Experiment 1. The column data corresponds to values for 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 5.1, and all subsequent heat maps, darker colored cells correspond to less desirable outcomes, while lighter colored cells correspond to more desirable outcomes. In Figure 5.1, cells are gray if model instances that fall in that region were not explicitly explored. In Figure 5.1, 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 outcome is low because both the number of contaminant plumes and population is low.

Figure 5.2 is a visualization of the learned rules produced by training XCSR on the exploratory results for Validation Experiment 1. The rules are visually similar to the results in Figure 5.1. Figure 5.2 implies 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 5.2 where the desirability of outcome coincides with the row below it and parallels our prediction.

Notably, Figure 5.2 demonstrates generalization errors. For example, in the lower right region of Figure 5.1, where population is 0 and 1 and number of contaminant plumes is 1, we see that outcome is undesirable. In Figure 5.2, we see that XCSR incorrectly deemed this area significantly more desirable than the adjacent cells. This is due to the formulation of equation 4.8 which 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. Equation 4.8 is discussed in Chapter 6. There, we highlight its limitations and suggest possible improvements.

The level of outcome throughout Figure 5.2 is lower than outcome in Figure 5.1. This is because multiple rules overlap. The level of outcome assigned to a cell in Figure 5.2 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

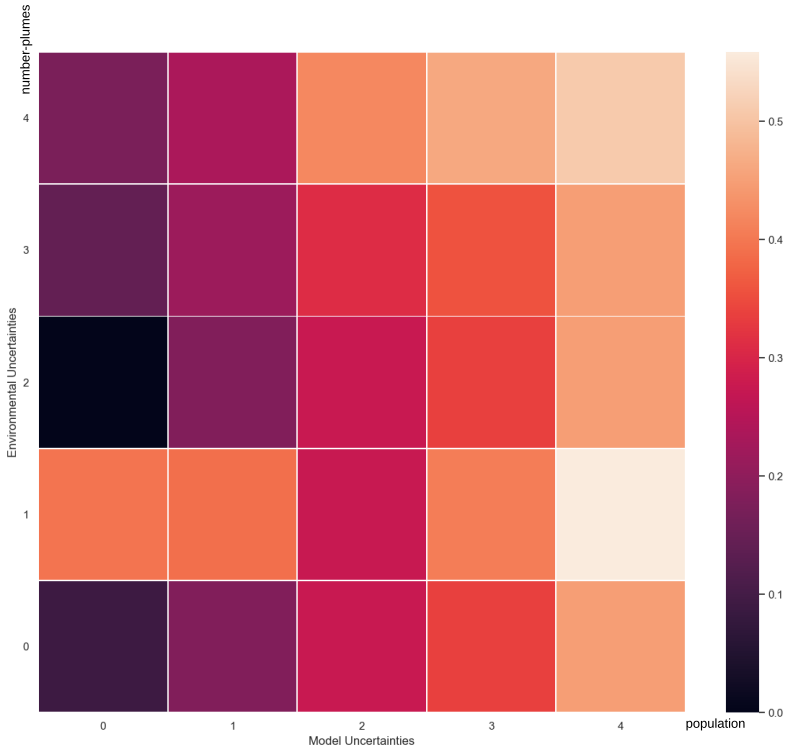

Figure 5.2: Validation Experiment 1 Learned Rules Heat Map

their outcomes are used to determine cell color. When rules contradict each other, it hampers the decision-support artifact. A solution to this is discussed in Chapter 6. Alternatively, a decision-maker can analyze the relative colors within the heat map to determine which regions are desirable.

Figure 5.3 visualizes the rules produced by the Multi-Layer Perceptron clustering algorithm. Figure 5.3 compliments both the exploratory results and the XCSR learned rules, and we see the appearance of the same generalized patterns. Figure 5.3 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 may be inapt for decision-support. Instead, we use Figure 5.3 visualizations as a validatory artifact to confirm the XCSR rules accurately describe the experiment space. We conclude from Figures 5.1, 5.2, and 5.3 that the Strategy Learning System correctly captured model behavior of select model instances and generalizes their desirability over the experiment space, as anticipated.

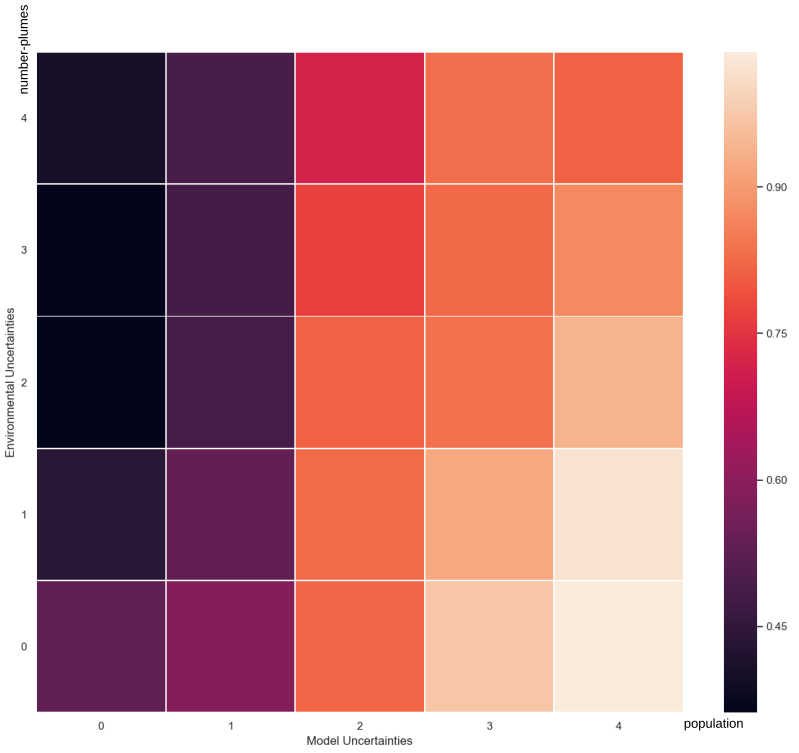

Figure 5.3: Validation Experiment 1 Learned MLP HAC Heat Map

5.1.2 Validation Experiment 2

Validation Experiment 2 was designed to be slightly more complex than Validation Experiment 1 but maintain simplicity in hypothesis so observations can be easily understood and compared to our predictions.

Experiment Hypothesis How is coverage percentage affected when we vary population, coverage data decay, the number of contaminant plumes, and wind speed?

From our hypothesis, we construct the following resolution model:

Validation Experiment 2 Resolution Model

```
include_feature(population)
include_feature(coverage-data-decay)
include_feature(number-plumes)
include_feature(wind-speed)
include_outcome(coverage-percentage)
```

Table 5.2: Validation Experiment 2 Parameters

EMA Lite			XCSR			
Model Instances	Replications	Ticks	Episodes	Bins	Classifiers	ζ
62	30	1,000	20,000	3	50	0.25

Table 5.2 is similar to Table X1, however, we double the number of model instances explored, decrease the number of bins, and decrease ζ for a slightly larger search space.

Experiment Predictions For Validation Experiment 2, we maintain and expand on the predictions from Validation Experiment 1: outcome increases proportionally to population. Additionally, we expect that outcomes are higher when wind speed is low and that coverage data decay has a marginal impact on coverage percentage.

Observations & Discussion

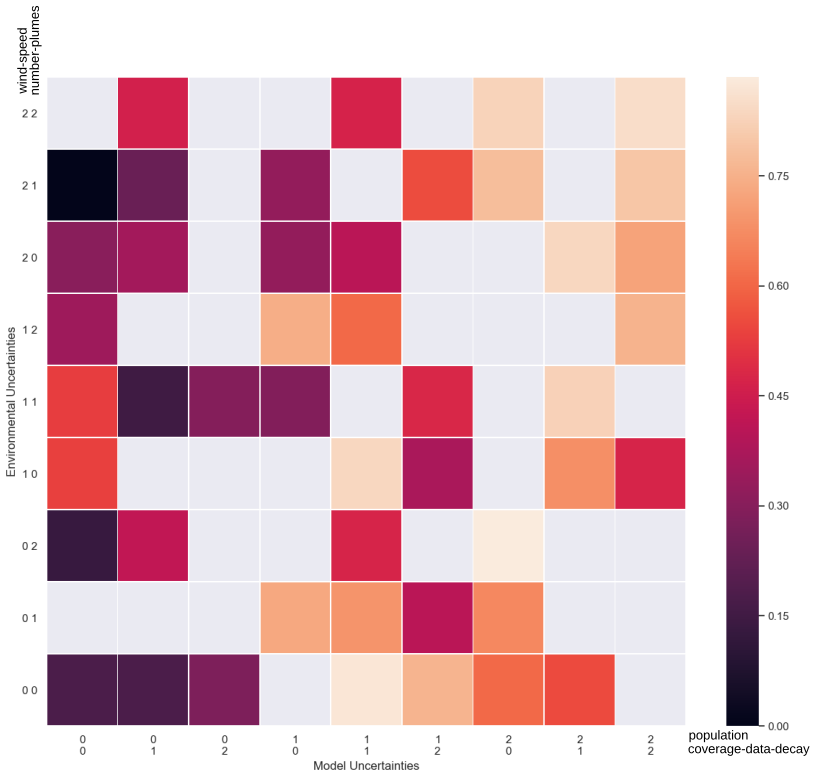

Figure 5.4: Validation Experiment 2 EMA Lite Exploration Heat Map

Figure 5.4 visualizes the EMA Lite exploratory results for Validation Experiment 2. In Figure 5.4, the columns correspond to values for population and coverage data decay. The

column values increase from left to right. Similarly, the rows correspond to the number of contaminant plumes and wind speed, and the values increase from the bottom to the top. Analysis of the columns in Figure 5.4 indicates that higher populations result in higher outcomes. Additionally, we see that the desirability of outcome is impacted more by population than by coverage data decay. Analysis of the rows indicates that when population is low, outcome is impacted by wind speed more than by the number of contaminant plumes, and when the population is high, the opposite is true.

In contrast to Figure X1 from Validation Experiment 1, Figure 5.4 shows how larger numbers of contaminant plumes lead to higher outcome. This is due to sampling variation, coupled with the stochasticity of the Contaminant Plume model, and the precision lost by binning. Higher outcomes align more with the formulation [equation 4.2] because a larger population yields the opportunity for more substantial mapping. Additionally, we see that the hot spot in Figure 5.4 is located in the upper right region where population, coverage data decay, wind speed, and number plumes are all notably high.

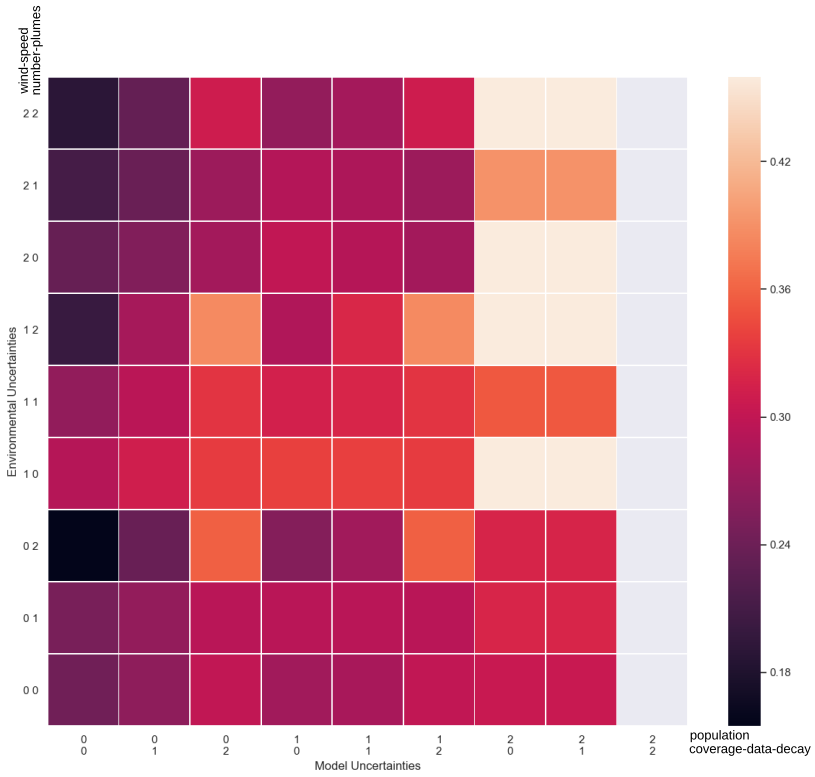

Figure 5.5: Validation Experiment 2 Learned Rules Heat Map

Figure 5.5 is a visualization of the learned rules produced by training XCSR on the exploratory results for Validation Experiment 1. The rules are visually similar to the results in Figure 5.4. The visualization implies that XCSR captured patterns in the exploratory results and generalized them into the portion of the ensemble not explicitly explored. In Figure 5.5, we again see the slight generalization error caused by equation 4.8. And, the amount of outcome in Figure 5.5 is lower than the amount of outcome in Figure 5.4 for the same reasons discussed in Validation Experiment 1.

The hot spot in Figure 5.5 is located in the upper right region where feature values are high. Analysis of the columns in Figure 5.5 reveals that the learned rules capture that change in population more than coverage data decay. The rows in Similar to Figure 5.4, Figure 5.5 shows that the learned rules capture that for low population, the wind speed highly impacts coverage percentage and for large populations the number of contaminant plumes highly impacts coverage percentage.

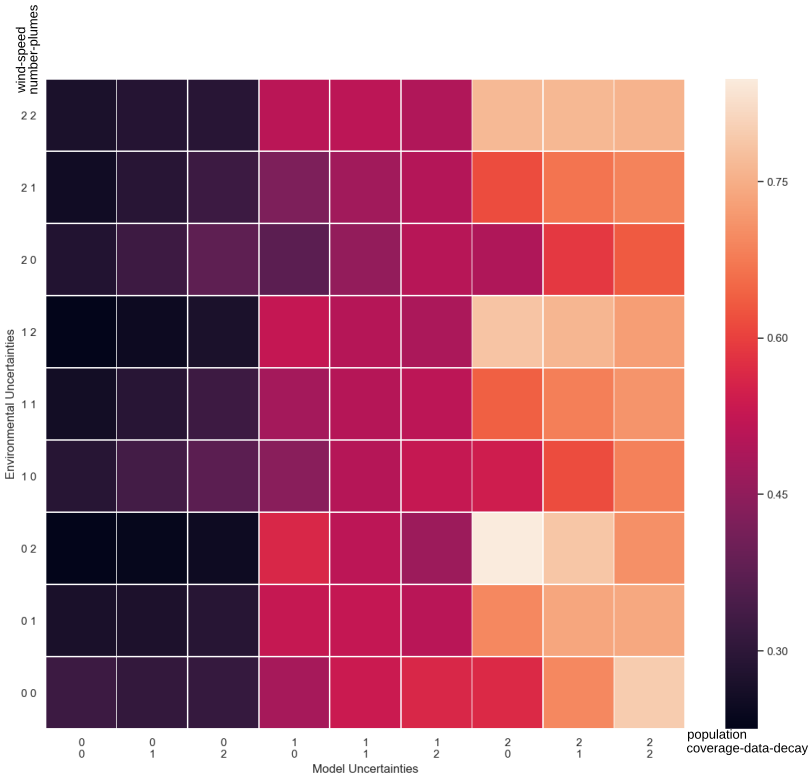

Figure 5.6: Validation Experiment 2 Learned MLP HAC Heat Map

Figure 5.6 visualizes the rules produced by the Multi-Layer Perceptron clustering algorithm for Validation Experiment 2. Figure 5.6 compliments both Figure 5.4 and Figure 5.5, and we see the appearance of the same generalized patterns. In Figure 5.6 columns, the rules produced by the Multi-Layer Perceptron clustering algorithm highlight that population is the highest impact feature. The rules underestimate the impact of wind speed and number of contaminant plumes. Nonetheless, as a validatory artifact Figure 5.6 aligns with the observed results in Figure 5.4 and Figure 5.5. This implies that the Strategy Learning System operates as anticipated.

5.2 Exploratory Experiments

5.2.1 Global Search Policy Experiment

The structural variation in the Contaminant Plume model is described via the three C2 policies. We constructed our exploratory experiments around the Swarm's performance and robustness given these three policies.

Experiment Hypothesis How is coverage percentage affected by both the number of plumes and wind speed when the UAVs follow each of the three C2 policies?

From our hypothesis, we construct the following resolution model. Here `include_children=False` informs the Strategy Learning System that we want to explicitly experiment with only the three C2 policies. We remove the subfeatures from the explainable models.

Global Search Policy Experiment Resolution Model

```
include_feature(global-search-policy, include_children=False)
include_feature(population)
include_feature(number-plumes)
include_feature(UAV-vision)
include_outcome(coverage-percentage)
```

Table 5.3: Global Search Policy Experiment Parameters

EMA Lite			XCSR			
Model Instances	Replications	Ticks	Episodes	Bins	Classifiers	ζ
200	30	1,000	20,000	3	50	0.5

Experiment Predictions For the Global Search Policy Experiment, we expect to see that for all C2 policies, percent coverage is high when population is high. Furthermore, due to the abundant stochasticity in the Random Search policy, we expect coverage percentage to be unrelated to the wind speed and number of contaminant plumes, and be inferior to Flock Search and Symmetric Search.

Observations & Discussion

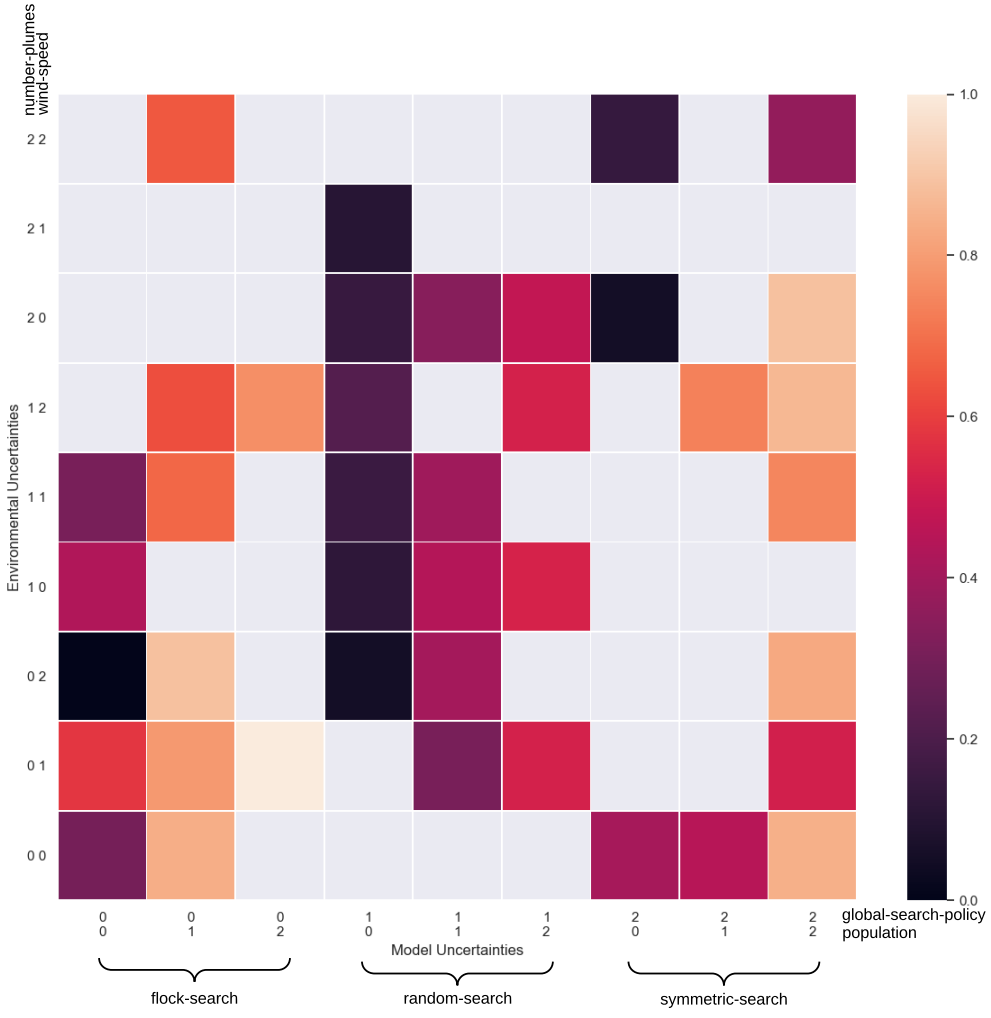

Figure 5.7: Global Search Policy Experiment EMA Lite Exploration Heat Map

Figure 5.7 visualizes the EMA Lite exploratory results for the Global Search Policy Experiment. In Figure 5.7, the columns correspond to population values and the selected global search policy. The rows correspond to values for the number of contaminant plumes and wind speed. As we analyze Figure 5.7 columns, we see that when population is low coverage percentage is low, regardless of C2 policy.

For the region of Figure 5.7 that describes flock search, we see that outcome is highest when population is high and the number of contaminant plumes is low. The desirability of the outcomes decreases as the number of contaminant plumes increases. For the region of Figure 5.7 that describes random search, we see that outcome is suboptimal regardless of the parametric values for uncontrollable features. Outcome increases with population, yet the only clear advantage to random search is that its performance is invariant. In this sense, random search is robust, but does not satisfactorily map the contaminant plumes. For the region of Figure 5.7 that describes symmetric search, we see that outcome is most desirable when both population and the number of contaminant plumes is high. Comparing all three C2 policies, we see that outcome is most consistent when the global search policy is symmetric search.

Figure 5.8 is a visualization of the learned rules produced by training XCSR on the exploratory results from the Global Search Policy Exploratory Experiment. In Figure 5.8, gray cells indicate that no rules address that region.

The region of Figure 5.8 that describes flock search reveals the policy results as a more desirable outcome when the number of contaminant plumes is low, regardless of the wind speed. A decision-maker can then conclude that flock search is robust with wind speed, but not with the number of contaminant plumes, and that it is most applicable when the number of contaminant plumes is understood to be low. The region of Figure 5.8 that describes random search shows that outcome is invariant to the parametric values for uncontrollable features. Decision-makers may deploy random search if they desire to achieve a minimum desirable outcome and care little for mapping the contaminant plumes optimally. The region of Figure 5.8 that describes symmetric search shows that outcome increases as the number of contaminant plumes increase. This makes sense because symmetric search disperses the Swarm throughout the environment, and more contaminant plumes present the opportunity to map a larger region. When the global

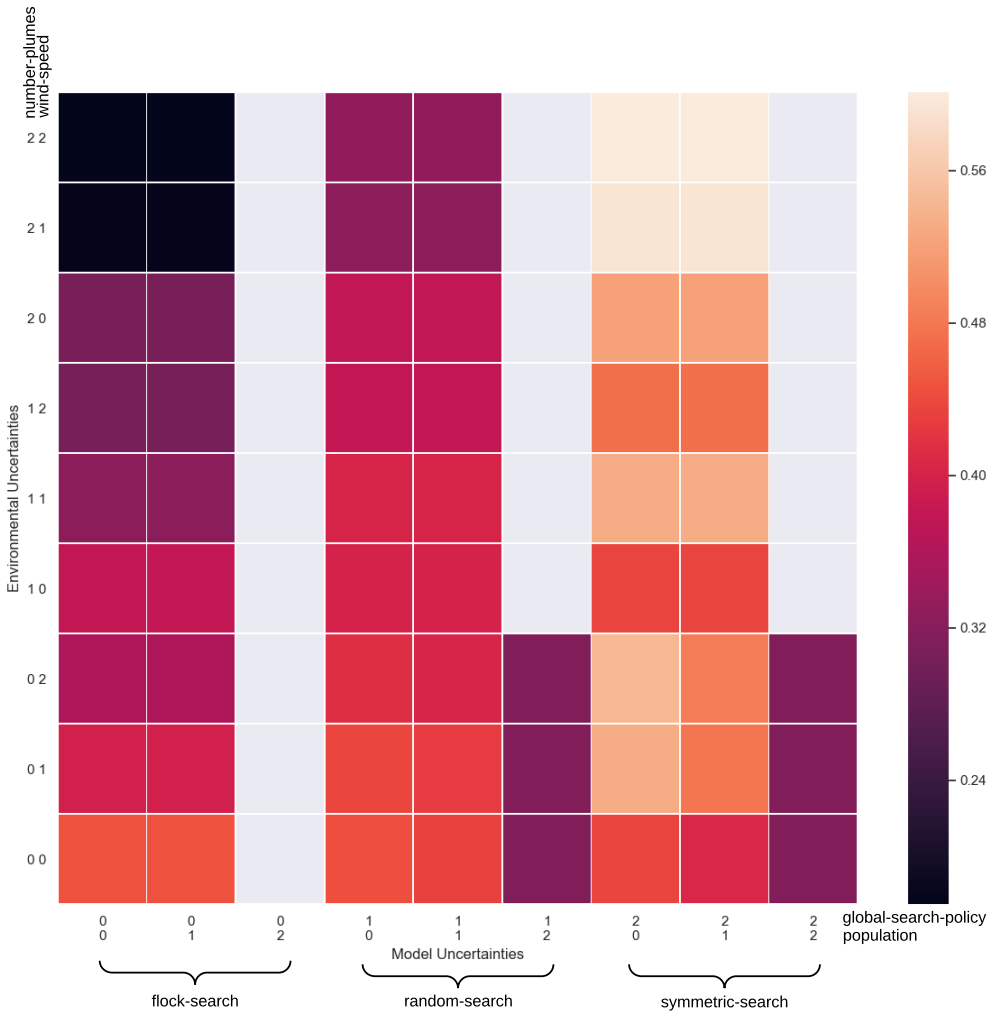

Figure 5.8: Global Search Policy Experiment Learned Rules Heat Map

search policy is symmetric search, the desirability of outcomes is correlated to the number of contaminant plumes and population size. Contrary to both flock search and random search, outcome desirability increases when the parametric values of uncontrollable features are more extreme. Therefore, symmetric search shows the highest amount of robustness when compared to the other C2 policies.

When we compare Figure 5.7 and Figure 5.8, a generalization error seems to appear. However, this is the XCSR pattern capture of the exploratory results. The patterns overpower outlier noise. A decision-maker can conclude that the Global Search Policy Exploratory Experiment provides insight into which C2 policy is the most robust against both wind speed and the number of contaminant plumes. For example, flock search and symmetric search demonstrate a

higher level of capability than random search. Flock search performs best when the number of contaminant plumes is low, while symmetric search performs best when the number of contaminant plumes is high. Now we can construct a subsequent experiment to analyze the robustness of symmetric search.

5.2.2 Symmetric Search Policy Experiment

From the insight of the Global Search Policy Experiment, we construct a subsequent experiment into the robustness of symmetric search.

Experiment Hypothesis Hypothesis: How is coverage percentage affected by the number of contaminant plumes and UAV vision when the Swarm adheres to the symmetric search policy?

From this hypothesis, we construct the following resolution model.

Symmetric Search Policy Experiment Resolution Model

```
include_feature(symmetric-search)
include_feature(number-plumes)
include_feature(UAV-vision)
include_outcome(coverage-percentage)
```

Table 5.4: Symmetric Search Policy Experiment Parameters

EMA Lite			XCSR			
Model Instances	Replications	Ticks	Episodes	Bins	Classifiers	ζ
150	30	1,000	20,000	3	50	0.5

Experiment Predictions For the Symmetric Search Policy Experiment, we expect that coverage percentage will be the most desirable when the number of contaminant plumes is high, as observed in the Global Search Policy Experiment. Additionally, we expect that higher parametric values for UAV vision will lead to more desirable outcomes because vision controls the distance that a UAV can perceive its environment.

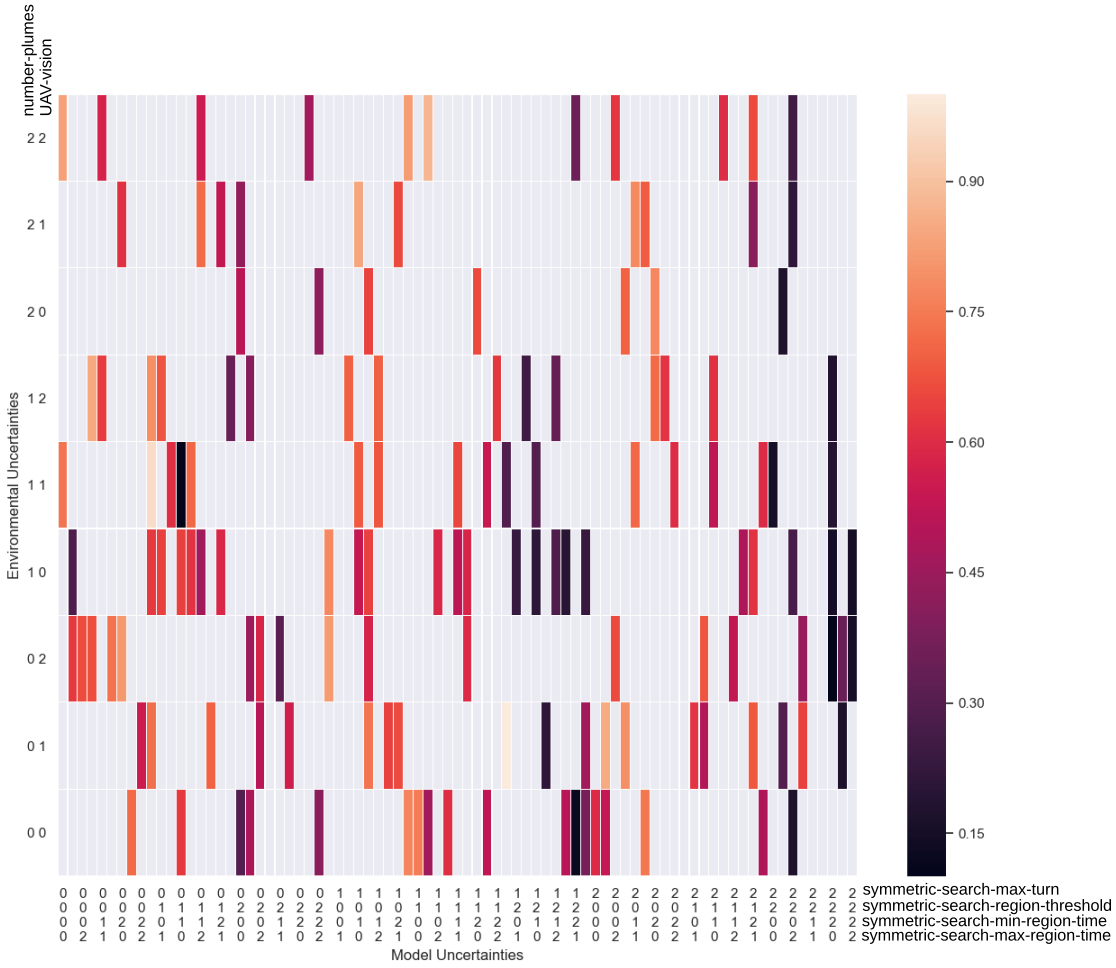

Figure 5.9: Symmetric Search Policy Experiment Exploration Heat Map

Observations & Discussion

Figure 5.9 visualizes the EMA Lite exploratory results for the Symmetric Search Policy Experiment. In Figure 5.9, the columns correspond to the parametric values for symmetric search which were discussed in Section 4.1. They are: symmetric search max turn, symmetric search region threshold, symmetric search max region time, and symmetric search max region time. Additionally, the rows in Figure 5.9 correspond to the number of contaminant plumes and UAV vision, and increase from the bottom up. An analysis of Figure 5.9 reveals two hot spots. The first is located in the upper left region, and the second is located in the center left region where the number of contaminant plumes and UAV vision is medium to high. More desirable outcomes are achieved when the parametric values for symmetric search are low. The analysis of

Figure 5.9 reveals that this is credited mostly to symmetric search min region time. Search min region time dictates how long a UAV must stay in its region before considering a move to a more desirable region. Symmetric search min region time reduces the desirability of outcomes when the UAV is required to map a region where there are no contaminant plumes. The mission is to avoid wasting Swarm resources.

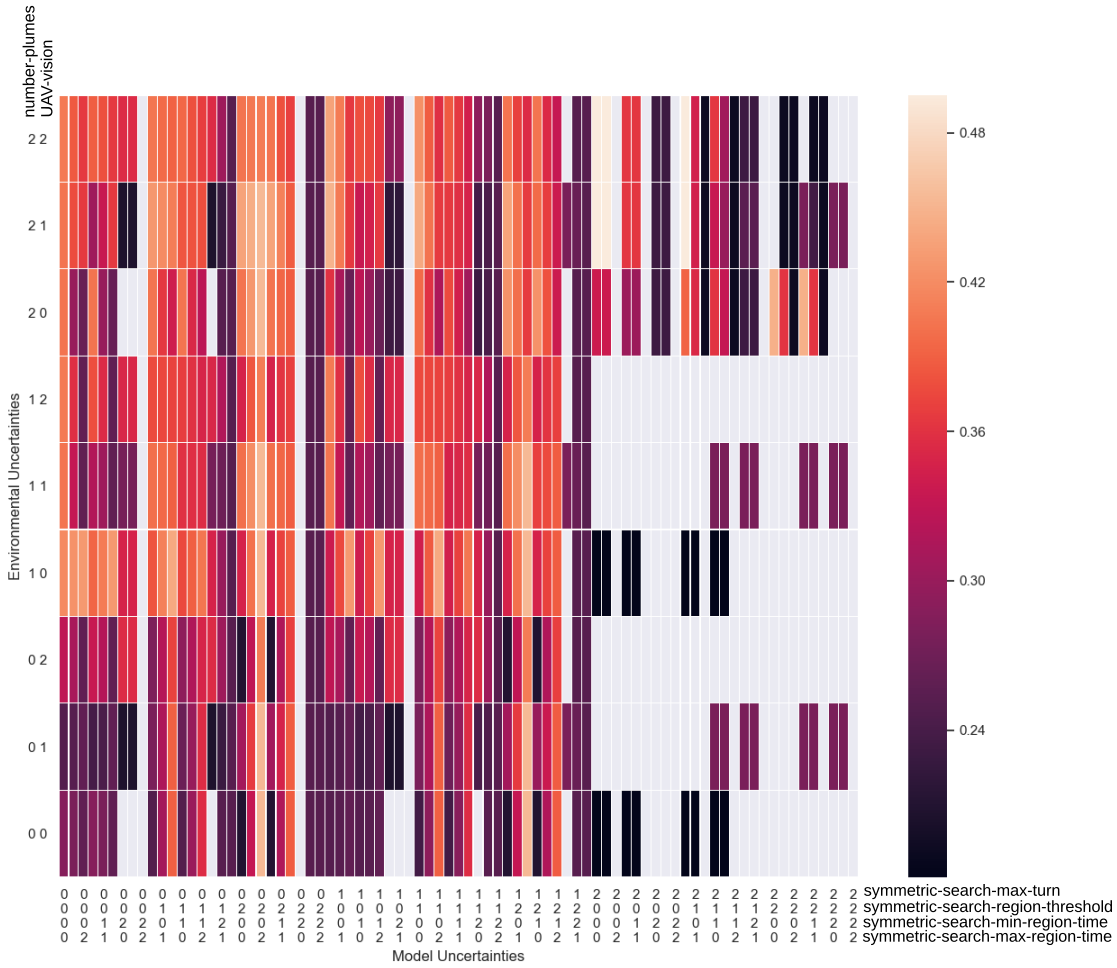

Figure 5.10: Symmetric Search Policy Experiment Learned Rules Heat Map

Figure 5.10 is a visualization of the learned rules produced by training XCSR on the exploratory results from the Symmetric Search Policy Experiment. As in Figure 5.8, the gray cells in Figure 5.10 indicate that no rules address that region. An analysis of Figure 5.10 reveals the same hot spots from Figure 5.9, and that higher parametric values for symmetric search decreases the desirability of outcomes. The hot spot in the upper left region shows

that the desirability of outcomes is relatively stable. On the other hand, the hot spot in the middle left region shows higher levels of outcome desirability than the hot spot in the upper left region. However, that region is slightly more volatile to symmetric search parametric values. From Figure 5.9 and Figure 5.10 observations, a decision-maker can conclude that a desirable outcome is achieved when the parametric values for symmetric search are low. The highest level of desirability is achieved when the number of contaminant plumes and UAV vision is medium while containing volatility. We conclude that symmetric search is most robust when the number of contaminant plumes and UAV vision is high.

Chapter 6

Conclusions

6.1 Summary

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 the solution 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 mechanical structures rapidly. Such tools excel with parametric uncertainties but require a modeler to implement alternative structures manually because they lack a mechanism to vary model structure. 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.

6.2 Accomplishments

The Strategy Learning System expands exploratory modeling to incorporate feature-driven variation and machine learning. Feature-driven variation incorporates a systematic approach to generate alternative model structures from a single base model, and the base model describes

the behavior of all features in the real-world target system. The features describe either competitive or complementary behaviors. Feature tree mapping delineates a hierarchy of behaviors. The tree allows a single model to represent alternative hypotheses about a target system. After mapping, the feature tree becomes a conceptual representation of the target system, and its features are symbolic aggressions of various levels of delivered functionality. The feature tree describes all plausible hypotheses of the target system. By parsing the base model and walking the feature tree, a model composer can generate scenarios according to a hypothesis. Exploration of each generated scenario is evaluated in terms of its outcome desirability to understand how the ensemble behaves. The scenario structures and their desirability feed into a rule-discovery machine learning algorithm that learns patterns and generates a set of rules that describe how the target system behaves under the hypothesis. The rule set can describe non-convex relationships and competitive or complementary strategies a decision-maker should follow to reach a desirable outcome. The rule set generates heat map visual aids that are easily interpretable. The Strategy Learning System improves the exploratory modeling decision-support because it (1) allows for the rapid development of new scenarios, and (2) accelerates the question-answer process by employing machine learning to extrapolate observed ensemble behaviors in the larger unexplored ensemble. Both of these mechanisms allow for a more robust and rapid decision-support exercise.

In this work, we demonstrated the advantages of the Strategy Learning System through the Contaminant Plume model case study. The Contaminant Plume model describes a hypothetical scenario where a Swarm of UAVs are tasked with mapping and decontaminating contaminant plumes. The model contains various parametric uncertainties, and its structural uncertainty lies in the C2 policy of the Swarm. The case study consists of two validatory experiments and two exploratory experiments. The validatory experiments show that the Strategy Learning System functions as designed, while the exploratory experiments exhibit enhanced insight into the mechanics of the model. In the Global Search Policy Experiment, we explored the three alternative Swarm C2 policies. The goal was to discover which policy was robust in the desirable rapid mapping of the contaminant plumes. The experiment shows that under some conditions, Flock

Search performed well but that Symmetric Search was the most robust and, therefore, the most reliable policy.

6.3 Limitations

The known limitations of the Strategy Learning System implementation are: (1) an inadequate reward function for the XCSR, and (2) the constraints presented by NetLogo. Equation 4.8 yields the reward function for the XCSR. For convenience, we repeat it here:

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

As described in section 4.4, the reward function converges to the observed reward when the distance is minimized between α_t and $\hat{\alpha}_t$. Here, α and $\hat{\alpha}$ are the expected and proposed actions, respectively, where actions are the controllable uncertainties. An issue arises when the rule set condition that $\hat{\alpha}_t$ is associated with differs substantially from σ_t. The effect is that the function rewards generalizations across the uncontrollable uncertainties while it punishes generalizations across the controllable uncertainties. This result is visible in Figure 5.8 and Figure 5.9. Note that the rules span across the vertical axis and not the horizontal axis. We believe a possible solution is to modify the information the XCSR presents to the generic environment. The remedial information should include the uncontrollable uncertainties with which $\hat{\alpha}_t$ is associated. Then, the improved reward function looks at the area in which the XCSR rule describes and calculates a reward based on observations in that region. The reward allows ρ_{t+1} to be computed based on the observed reward from both controllable and uncontrollable uncertainties.

Another limitation of the Strategy Learning System is the NetLogo implementation of the Contaminant Plume Model. NetLogo lacks inheritance, aggregation, and decorator patterns. When Scala functions are implemented as aggregations, limitations are slightly alleviated; however, we experience little flexibility with this. To meet the needs of the Strategy Learning System, the Contaminant Plume model implements with alternative logic flows. The

alternative logic flows meet the same objectives, and the implementation is observed to be less malleable and less tightly coupled.

6.4 Future Work

Adaptability The Strategy Learning System is compatible with NetLogo models. A decision-maker can use the Strategy Learning System as-is with their model, as long as their implementation describes features and specifies the feature tree. Moreover, by modifying the model synthesizer, a decision-maker can use the Strategy Learning System with models written in languages other than NetLogo.

Technology Development A data percolation study provides analysis into how much exploratory data is required for the rule discovery algorithm to produce accurate rule sets that describe the ensemble. The study discovers the minimum number of explorations required, and therefore can further accelerate the decision-support exercise. Furthermore, the use of additional machine learning algorithms improves the quality of insight. In this work, we demonstrated the use of a Learning Classifier System and its capacity to describe complementary and competitive patterns. With the employment of decision trees or a supervised clustering algorithm that is flexible with the number of clusters, a single instance with which it is associated an equivalent outcome is learned. Both of these algorithms easily explain predictions. With other machine learning algorithms consideration it is possible to explain their predictions to decision-makers who may be unfamiliar with their mathematical basis.

Bibliography

[1] Amina Adadi and Mohammed Berrada. Peeking inside the black-box: A survey on explainable artificial intelligence (xai). *IEEE Access*, 6:52138–52160, 2018.

[2] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. *Feature-oriented software product lines*. Springer, 2016.

[3] Office of Information Technology Auburn University. Auburn hopper cluster, 2016.

[4] SC Bankes, Warren E Walker, and Jan H Kwakkel. Exploratory modeling and analysis. *Encyclopedia of operations research and management science*, pages 532–537, 2013.

[5] Steve Bankes. Exploratory modeling for policy analysis. *Operations research*, 41(3):435–449, 1993.

[6] Alan L Brown and Paul K Davis. New challenges for defense planning: Rethinking how much ls enough. *Naval War College Review*, 49(2):13, 1996.

[7] Jason Brownlee. *Clever algorithms: nature-inspired programming recipes*. Jason Brownlee, 2011.

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

[9] Bill Canis. Unmanned aircraft systems (uas): Commercial outlook for a new industry, 2015.

[10] Lianping Chen, Muhammad Ali Babar, and Nour Ali. Variability management in software product lines: a systematic review. In *Proceedings of the 13th international conference on Software Product Lines*. Association for Computing Machinery, 2009.

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

[12] Hoa Khanh Dam, Truyen Tran, and Aditya Ghose. Explainable software analytics. In *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*, pages 53–56, 2018.

[13] Paul K Davis. Analytic architecture for capabilities-based planning, mission-system analysis, and transformation. Technical report, RAND NATIONAL DEFENSE RESEARCH INST SANTA MONICA CA, 2002.

[14] Paul K Davis. Exploratory analysis and implications for modeling. *RAND-PUBLICATIONS-MR-ALL SERIES-*, pages 255–284, 2003.

[15] Paul K Davis, Steven C Bankes, and Michael Egner. *Enhancing strategic planning with massive scenario generation: Theory and experiments*, volume 392. Rand Corporation, 2007.

[16] Paul K Davis, Angela O'Mahony, and Jonathan Pfautz. *Social-Behavioral Modeling for Complex Systems*. John Wiley & Sons, 2019.

[17] Suraje Dessai, Mike Hulme, Robert Lempert, and Roger Pielke Jr. Do we need better predictions to adapt to a changing climate? *Eos, Transactions American Geophysical Union*, 90(13):111–112, 2009.

[18] Derek Doran, Sarah Schulz, and Tarek R Besold. What does explainable ai really mean? a new conceptualization of perspectives. *arXiv preprint arXiv:1710.00794*, 2017.

[19] Nugroho Fredivianus and Kurt Geihs. Classifier systems with native fuzzy logic control operation. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1341–1348, 2017.

[20] Nugroho Fredivianus, Kais Kara, and Hartmut Schmeck. Stay real! xcs with rule combining for real values. In *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*, pages 1493–1494, 2012.

[21] Nugroho Fredivianus, Holger Prothmann, and Hartmut Schmeck. Xcs revisited: a novel discovery component for the extended classifier system. In *Asia-Pacific Conference on Simulated Evolution and Learning*, pages 289–298. Springer, 2010.

[22] Volker Grimm, Uta Berger, Finn Bastiansen, Sigrunn Eliassen, Vincent Ginot, Jarl Giske, John Goss-Custard, Tamara Grand, Simone K Heinz, Geir Huse, et al. A standard protocol for describing individual-based and agent-based models. *Ecological modelling*, 198(1-2):115–126, 2006.

[23] James Hendler. Computers play chess; humans play go. *IEEE Intelligent Systems*, 21(4):2–3, 2006.

[24] Kyo C Kang, Jaejoon Lee, and Patrick Donohoe. Feature-oriented product line engineering. *IEEE software*, 19(4):58–65, 2002.

[25] Michael A Kovacina, Daniel Palmer, Guang Yang, and Ravi Vaidyanathan. 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, pages 2782–2788. IEEE, 2002.

[26] Tim Kovacs. Xcs classifier system reliably evolves accurate, complete, and minimal representations for boolean functions. In *Soft computing in engineering design and manufacturing*, pages 59–68. Springer, 1998.

[27] Jan Kwakkel. Exploratory modeling workbench implementation. `https://github.com/quaquel/EMAworkbench`, 2010-2020.

[28] Jan Kwakkel. Exploratory modeling workbench read the docs. https://emaworkbench.readthedocs.io/, 2010-2020.

[29] Jan H Kwakkel. 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, 2017.

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

[31] Jan H Kwakkel, Warren E Walker, and Vincent AWJ Marchau. Classifying and communicating uncertainties in model-based policy analysis. *International journal of technology, policy and management*, 10(4):299–315, 2010.

[32] Rodriguez Kwakkel. Ema lite. https://github.com/brodderickrodriguez/ema_lite, 2019.

[33] Pier L Lanzi. *Learning classifier systems: from foundations to applications*. Springer Science & Business Media, 2000.

[34] Robert Lempert, Steven Popper, and Steven Bankes. Confronting surprise. *Social Science Computer Review*, 20(4):420–440, 2002.

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

[36] Robert J Lempert, David G Groves, Steven W Popper, and Steve C Bankes. A general, analytic method for generating robust strategies and narrative scenarios. *Management science*, 52(4):514–528, 2006.

[37] Alexander G Madey and Gregory R Madey. Design and evaluation of uav swarm command and control strategies. In *Proceedings of the Agent-Directed Simulation Symposium*, pages 1–8, 2013.

[38] Peter Mensah, Yuri Merkuryev, Jelena Pecerska, and Francesco Longo. Analysing uncertainties and their impacts on deliveries of a logging company: simulation model to foster supply chain resilience. *International Journal of Simulation and Process Modelling*, 14(3):251–260, 2019.

[39] Bert Metz, Ogunlade Davidson, Rob Swart, Jiahua Pan, et al. *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, 2001.

[40] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[41] P Russel Norvig and S Artificial Intelligence. *A modern approach*. Prentice Hall, 2002.

[42] Adrian N Phillips. A secure group communication architecture for a swarm of autonomous unmanned aerial vehicles. Technical report, AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH GRADUATE SCHOOL OF ..., 2008.

[43] Klaus Pohl, Günter Böckle, and Frank J van Der Linden. *Software product line engineering: foundations, principles and techniques*. Springer Science & Business Media, 2005.

[44] Klaus Pohl and Andreas Metzger. Variability management in software product line engineering. In *Proceedings of the 28th international conference on Software engineering*, pages 1049–1050, 2006.

[45] Anand Rajaraman and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2011.

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

[47] Brodderick Rodriguez. Contaminant plume model. `https://github.com/broddericksrodriguez/contaminant_plume_model`, 2018–2019.

[48] Brodderick Rodriguez. Strategy learning system. `https://github.com/broddericksrodriguez/strategy_learning_system`, 2019.

[49] Brodderick Rodriguez. xcsr. `https://github.com/broddericksrodriguez/xcsr`, 2019.

[50] Jonathan Rosenhead, Martin Elton, and Shiv K Gupta. Robustness and optimality as criteria for strategic decisions. *Journal of the Operational Research Society*, 23(4):413–431, 1972.

[51] Leonard J Savage. *The foundations of statistics*. Courier Corporation, 1972.

[52] Raymond Ka-Man Sheh. " why did you do that?" explainable intelligent robots. In *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[53] Thomas Stahl, Markus Voelter, and Krzysztof Czarnecki. *Model-driven software development: technology, engineering, management*. John Wiley & Sons, Inc., 2006.

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

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

[56] Marjolein BA Van Asselt and Jan Rotmans. Uncertainty in integrated assessment modelling. *Climatic change*, 54(1-2):75–105, 2002.

[57] Warren E Walker, Poul Harremoës, Jan Rotmans, Jeroen P Van Der Sluijs, Marjolein BA Van Asselt, Peter Janssen, and Martin P Krayer von Krauss. Defining uncertainty: a conceptual basis for uncertainty management in model-based decision support. *Integrated assessment*, 4(1):5–17, 2003.

[58] Warren E Walker, Robert J Lempert, and Jan H Kwakkel. Deep uncertainty. *Delft University of Technology*, 1:2, 2012.

[59] Uri Wilensky. Flocking. `http://ccl.northwestern.edu/netlogo/mod-els/Flocking`, 1998.

[60] Uri Wilensky et al. Center for connected learning and computer-based modeling. In *NetLogo*. Northwestern University, 1999.

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

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

[63] Levent Yilmaz. Featuresim: Feature-driven simulation for exploratory analysis with agent-based models. In *2017 IEEE/ACM 21st International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, pages 1–8. IEEE, 2017.

[64] Levent Yilmaz. Managing structural variability in agent-based models with feature coherence graphs. *accepted, to appear in International Journal of Simulation and Process Modeling*, (2020 -in press).

Appendices

Appendix A

Contaminant Plume Model Inputs

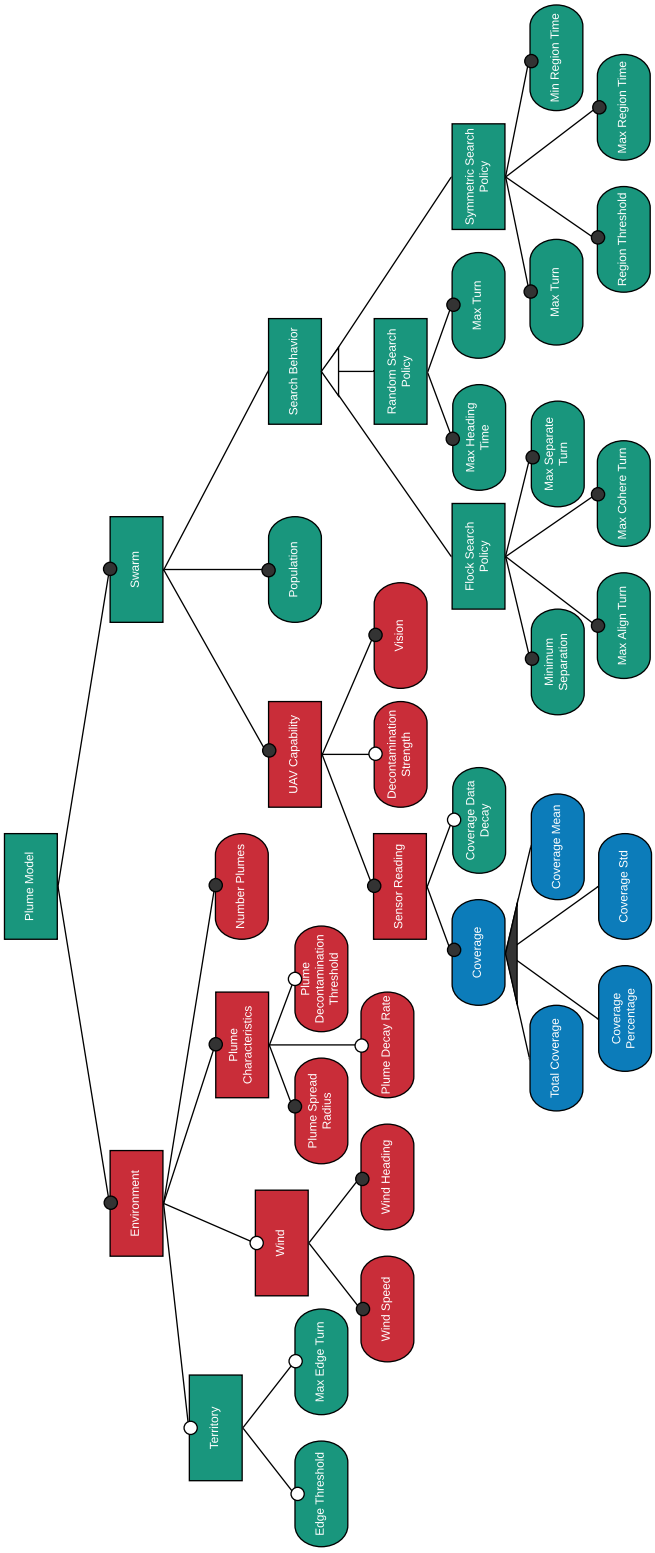

Figure A.1: Contaminant Plume Model Feature Model of the model's features and Input Parameters

Table A.1: Contaminant Plume Model Inputs

Category	Parameter Name	Min Value	Max Value	Default Value	Increment	Unit
Contaminant Plume	plume-spread-radius	0	1		0.01	Env. Width
Contaminant Plume	plume-decay-rate	0	1E-03		1E-11	Patches / Tick
Contaminant Plume	plume-decontamination-threshold	0.01	1		0.01	plume-spread-patches
UAV	UAV-vision	0	195		0.5	Patches
UAV	UAV-decontamination-strength	0	0.01		1E-05	Patches
Swarm	population	2	100		1	UAVs
Swarm	global-search-policy	N/A	N/A		N/A	N/A
Swarm - Flock Search	minimum-separation	0	5		0.25	Patches
Swarm - Flock Search	max-align-turn	0	20		0.25	Degrees
Swarm - Flock Search	max-cohere-turn	0	10		0.1	Degrees
Swarm - Flock Search	max-separate-turn	0	20		0.25	Degrees
Swarm - Random Search	random-search-max-heading-time	0	100		1	Ticks
Swarm - Random Search	random-search-max-turn	0	5		0.05	Degrees
Swarm - Symmetric Search	symmetric-search-max-turn	0	20		0.1	Degrees
Swarm - Symmetric Search	symmetric-search-region-threshold	-10	25		0.1	Patches
Swarm - Symmetric Search	symmetric-search-min-region-time	1	1E+03		1	Ticks
Swarm - Symmetric Search	symmetric-search-max-region-time	100	5E+05		1	Ticks
Environment	number-plumes	1	5		1	Plumes
Environment	wind-speed	0	0.1		1E-03	Patches
Environment	wind-heading	0	360		1	Degrees
Environment	world-edge-threshold	0	25		0.5	Patches
Environment	max-world-edge-turn	0	20		0.5	Degrees
Environment	coverage-data-decay	1	60		1	Ticks

Appendix B

XCSR Supplementary

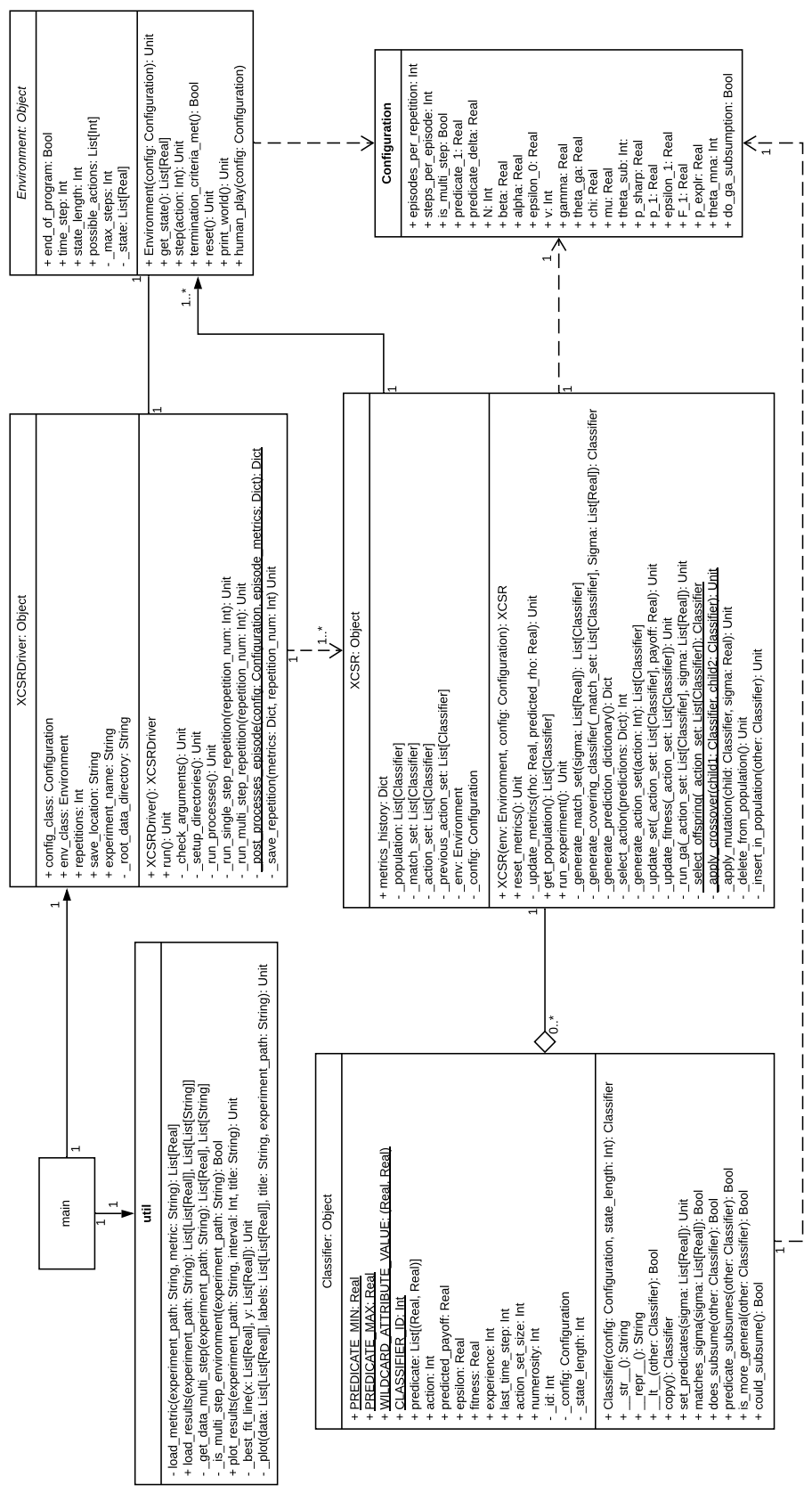

Figure B.1: XCSR Detailed Class Diagram