



POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Self-Adaptation under Uncertainty using Bayesian Model Averaging and Many Objective Search

TESI DI LAUREA MAGISTRALE IN
COMPUTER SCIENCE AND ENGINEERING - INGEGNERIA
INFORMATICA

Author: **Umberto Messuti**

Student ID: 971412
Advisor: Prof. Raffaella Mirandola
Co-advisors: Matteo Camilli
Academic Year: 2022-23

Abstract

This thesis proposes a novel approach to tackle the problem of *Self-adaptation* in the presence of model selection *Uncertainty* in self-adaptive systems. These systems are designed to adapt to changing environmental conditions without human intervention. However, due to their complexity, building accurate models that can capture the wide range of environmental conditions that the system may encounter is challenging. This uncertainty can lead to suboptimal performance and even system failures.

To address this issue, the proposed approach uses two techniques: Bayesian Model Averaging and Many Objective Search. The former is a statistical technique that allows for combining multiple models through a weighted average to obtain a more accurate and robust prediction. This technique can effectively handle the aforementioned Uncertainty. The latter is used to search for a new configuration of the self-adaptive systems that allow them to achieve an equilibrium condition, an optimal adaptation in a wide search space. To complete the adaptation, the proposed approach uses the Non-dominated Sorting Genetic Algorithm III (NSGA-III), a state-of-the-art many objective optimization algorithm that can handle conflicting objectives.

A simulator is introduced to evaluate the effectiveness of the proposed approach using a case study from the robotics domain. The results show that the proposed idea is effective in detecting and enforcing equilibrium constraints during execution and improving the adaptation performance of self-adaptive systems.

Keywords: cyber-physical systems, self-adaptive systems, model uncertainty, Bayesian model averaging, many objective search

Abstract in lingua italiana

Questa tesi propone un nuovo approccio per affrontare il problema dell'*Auto-adattamento (Self-adaptation)* in presenza di *Incertezza (Uncertainty)* nella selezione del modello nei sistemi auto-adattivi. Questi sistemi sono progettati per adattarsi alle mutevoli condizioni ambientali senza l'intervento umano. Tuttavia, a causa della loro complessità, è difficile costruire modelli accurati in grado di catturare la vasta gamma di condizioni ambientali che il sistema può incontrare. Questa incertezza può portare a prestazioni non ottimali e persino a guasti del sistema.

Per affrontare questo problema, l'approccio proposto utilizza due tecniche: Bayesian Model Averaging e Many Objective Search. La prima è una tecnica statistica che consente di combinare, attraverso una media ponderata, più modelli per ottenere una previsione più accurata e robusta: questa tecnica può gestire efficacemente l'Incertezza *de qua*. La seconda viene utilizzata per cercare una nuova configurazione dei sistemi auto-adattivi, che ci consenta di raggiungere una condizione di equilibrio quale adattamento ottimale in un ampio spazio di ricerca. Per completare l'adattamento, l'approccio proposto utilizza il Non-dominated Sorting Genetic Algorithm III (NSGA-III), algoritmo di ottimizzazione multi-obiettivo all'avanguardia in grado di gestire più obiettivi in conflitto.

Viene introdotto un simulatore per valutare l'efficacia dell'approccio proposto utilizzando un esempio del dominio robotico. I risultati mostrano che l'idea proposta è efficace nel rilevare e far rispettare i vincoli di equilibrio durante l'esecuzione e nel migliorare le prestazioni di adattamento dei sistemi auto-adattivi.

Parole chiave: sistemi cyber-physical, sistemi auto-adattivi, model uncertainty, Bayesian model averaging, many objective search

Contents

Abstract	i
Abstract in lingua italiana	iii
Contents	v
Introduction	1
1 Preliminaries	5
1.1 Computing Systems	5
1.1.1 Cyber-Physical Systems	6
1.1.2 Self-Adaptive Systems	6
1.2 Parametric Markov Decision Process	10
1.3 Bayesian Inference	14
1.4 Bayesian Model Averaging	15
1.5 Meta-Heuristic Optimizing Search	17
2 Starting Point	21
2.1 Run-Time Equilibrium Verification	21
2.1.1 Specifying with Partial Knowledge	22
2.1.2 Offline Computation of Equilibrium Constraints	23
2.1.3 Run-Time Equilibrium Verification	25
2.1.4 Run-Time Equilibrium Enforcement	26
2.2 Taming Model Uncertainty in Self-Adaptive Systems	26
3 Our Proposal	29
3.1 Problem Formulation	30
3.2 Simulator	31
3.2.1 Input Definition	31
3.2.2 Simulator Architecture	38

3.2.3	Execution Result	41
3.3	BMA and Meta-Heuristic Search for Adaptation	42
3.3.1	Dataset	43
3.3.2	BMA	45
3.3.3	Meta-Heuristic Search	47
4	Evaluation	51
4.1	Research Questions	51
4.2	Simulator	52
4.2.1	Evaluation Design	52
4.2.2	RQ1: Violation Detection Ability	53
4.2.3	RQ2: Violation Detection Effectiveness	57
4.2.4	RQ3: The Effectiveness of Policies in Preventing Disequilibria	60
4.3	BMA for Adaptation	62
4.3.1	Design of the Evaluation	62
4.3.2	RQ4: Assessing the Accuracy of BMA Estimates	68
4.3.3	RQ5: Evaluating the Cost of BMA Model Calculation	71
4.3.4	RQ6: Optimizing Search for Restoring Equilibrium Constraints	73
5	Conclusion and Future Developments	75
	Bibliography	77
	List of Figures	81
	List of Tables	83
	List of Symbols	85
	Ringraziamenti	87
	Acknowledgements	89

Introduction

Cyber-Physical Systems are systems that combine the physical and digital worlds, integrating sensors, actuators, data processing systems, and communication infrastructures. They are capable of monitoring and controlling physical processes in real-time, using control and data analysis algorithms to make autonomous decisions. These systems offer numerous advantages, such as increased efficiency, and are used in various sectors, including health-care, automotive with intelligent vehicles and aircraft avionics systems.

The intrinsic goal of a Cyber-Physical System is to maintain continuous and reliable functioning. This means that a Cyber-Physical System must be able to constantly monitor the physical environment, collect data, process information, and make real-time decisions to ensure the correct functioning of the system itself, i.e. find a new internal configuration, in order to guarantee continuous operation while avoiding potential safety risks. For example, a rescue robot used in natural disasters must be able to adapt to changing environmental conditions such as fluctuating temperatures, debris or rubble obstructing its path, and the presence of survivors or dangerous obstacles to avoid potential safety or reliability issues while still being able to locate and rescue victims efficiently.

This goal can be defined as the system's ability to maintain continuous and reliable functioning even in the presence of unforeseen or adverse events, such as malfunctions, errors, or changes in the physical environment. This means that a resilient Cyber-Physical System must be able to recognize and mitigate any problems, adapt to unexpected situations, and restore its processes quickly and efficiently, in order to maintain its intrinsic goal of continuous and reliable functioning of the system. In other words, the resilience of a Cyber-Physical System is closely linked to its ability to maintain the equilibrium.

The classical approach to resilience in a Cyber-Physical System is based on identifying a limited number of conditions in which the system should operate and ensuring that the system operates reliably under such conditions. In other words, the classical approach seeks to define a set of possible scenarios to which the system could be subjected and, under such assumptions, ensure its proper functioning, that is making sure that all the requirements are met.

However, this approach has some limitations. In particular, in a dynamic and complex environment such as that of Cyber-Physical Systems, it is difficult to predict *all* possible situations to which the system may be subjected. For this reason, in recent years a more dynamic and flexible approach to achieve resiliency in Cyber-Physical Systems has been developed, which involves the use of statistical models to allow the system to learn from past events and adapt to new situations autonomously. That is, in case of disequilibrium, find a new configuration of the CPS that is most likely to satisfy all the requirements.

Nevertheless, the use of these models introduces a new difficulty known as “Model Selection Uncertainty”: the uncertainty that arises from choosing the best statistical model to analyze a particular set of data. In other words, when working with statistical models, it can be difficult to determine which is the best one for the data being analyzed: there may be many options to choose from, and the wrong choice can lead to inaccurate or even erroneous results.

In this context, Bayesian Model Averaging emerges as a useful technique for managing the uncertainty in model selection: it allows multiple models to be combined to obtain a more accurate and robust prediction. This technique uses a Bayesian approach to calculate the probability that each model is the best for analyzing the current dataset, taking into account available empirical evidence and using Bayesian inference techniques. Thus, instead of choosing a single model, Bayesian Model Averaging assigns a weight to each selected model, taking into account their probability of being the best one. Specifically, we use Bayesian Model Averaging to mitigate the model selection uncertainty in selecting an appropriate statistical model for each requirement.

Our approach consists of two main blocks, the first focusing on the development and implementation of a Cyber-Physical system simulator, and the second focusing on the application of Bayesian Model Averaging and many-objective search for enforcing the resiliency of these very Cyber-Physical systems. More in detail, the former is used to simulate Cyber-Physical systems in order to check for the requirement fulfillment; the latter represents the application of an optimization search problem that, in case of requirement unfulfillment, is used to find out a set of optimal configurations for a Cyber-Physical system capable of satisfying as many requirements as possible.

The evaluation phase consisted of tests to assess the effectiveness of our proposed approach in recognizing violations and its ability to mitigate them efficiently. The results demonstrated that the approach significantly improved the simulator’s ability to recognize violations, and policies efficiently reduced the rate of violations pursued outside of equilibrium. Moreover, Bayesian Model Averaging outperformed logistic models in precision,

recall, and F1 score, and the cost of obtaining a Bayesian Model Averaging model through Markov Chain Monte Carlo was significantly lower than using a deterministic technique. In addition, the use of both Bayesian Model Averaging and meta-heuristic had a high rate of requirements satisfied while minimizing costs.

The thesis has been organized as follows:

- Chapter 1 encompasses fundamental notions that are crucial to understand the context and technologies employed throughout the work, providing an introduction to computer systems and different types of systems, including cyber-physical and self-adaptive ones. Furthermore, it introduces other concepts such as Parametric Markov Decision Processes, Bayesian Inference, Bayesian Model Averaging, and meta-heuristic optimizing search.
- Chapter 2 addresses the starting point of the research, which focuses on Run-time Equilibrium Verification and Taming Model Uncertainty in Self-adaptive Systems.
- Chapter 3 presents our proposed solution, including the problem's formulation, the simulator's architecture, the execution results, and the use of Bayesian Model Averaging and many objective search for adaptation purposes.
- Chapter 4 deals with the evaluation of the proposed solution, including the research objectives and evaluation design. Additionally, it reports the evaluation results, such as the effectiveness of the simulator, Bayesian Model Averaging, and many objective search for adaptation.
- Chapter 5 draws conclusions from the research and suggests future developments that may be undertaken in the field.

1 | Preliminaries

In this section, we will introduce the preliminary concepts necessary to fully understand our proposal, which was briefly mentioned in the introduction and will be thoroughly explored in chapter 3.

We will start with the general definition of a computing system and then move on to two of its subsets, cyber-physical systems and self-adaptive systems. Next, we will introduce Markov Decision Processes containing parameters θ , which are known as parametric Markov Decision Processes (pMDP). These are a formal modeling framework widely accepted for specifying and verifying the dependability of software systems. We will then introduce Bayesian inference and Bayesian Model Averaging, which are used for data analysis and for identifying models that best describe the data. Finally, we will conclude with the presentation of Meta-Heuristic optimizing search, which is necessary for finding the best solutions in complex problems.

1.1. Computing Systems

The term “system” generally denotes a conglomeration of interconnected elements that work together to achieve a particular outcome or form a cohesive entity. A computing system is a complex assemblage of hardware, software, and data, designed to process, store, and transmit information in an efficient and reliable manner, and collaborate to perform specific functions while tackling challenges such as security, scalability, and efficiency to ensure their proper operation.

The computing systems that we will mainly focus on are:

- Cyber-physical systems, and
- Self-adaptive systems

1.1.1. Cyber-Physical Systems

The vast majority of systems around us can be defined as Cyber-Physical Systems (CPSs) [4, 23]. The peculiarity of these systems lies in the combination of physical components and computational components in order to achieve a specific goal. Typically, CPSs contain interconnected sensor networks, actuators, processors, and communication devices that are integrated with physical systems such as vehicles, buildings, or machinery.

Of course, all this innovation encounters many challenges to overcome. Among the main complexities, we can include:

- **Integration:** as already mentioned, CPSs integrate components of different nature (physical and computational) and, precisely for this reason, implementing coordination and synchronization mechanisms involves studying and implementing additional systems.
- **Security, Protection and Reliability:** working in physical environments, CPSs can have significant (or even catastrophic) impacts on the environment and on those who live there. In fact, they require careful design, testing, and validation to ensure that, from the point of view of both individual components and the system as a whole, they are able to function continuously while maintaining high reliability in the presence of disturbances, attacks, or variable circumstances.
- **Data management:** working within an environment that is often subject to rapid changes and with the need to react in real-time to each of them means generating a huge amount of data, potentially with heterogeneous structures, which must be efficiently processed, analyzed, and saved.
- **Interoperability:** the individual components underlying CPSs can be from different manufacturers and, consequently, can implement different communication protocols or data formats that require careful integration and coordination.

1.1.2. Self-Adaptive Systems

With the continuous increase in power and complexity of various computer systems, it has become necessary to find new approaches to software development. As examples of these difficulties, resulting from both external and internal factors, there may be the size of the system, dynamics in the availability of resources and services, system errors that can be difficult to predict and changes in user objectives during their operation.

As a result Self-Adaptive Systems (SAS) [27] were born: these are computer systems

that have the ability to adapt themselves without requiring explicit human intervention in response to changes in the environment, user requirements, or system state. These systems have a wide range of applications such as adaptive robotics, self-driving cars and intelligent home automation systems. However, the effectiveness and reliability of SASs depend on addressing several challenges such as designing effective monitoring and control mechanisms, ensuring system reliability and safety and managing the complexity of these systems. Even though testing and validating the system's behavior under different environmental conditions and user requirements can present challenges, SASs offer numerous benefits. In particular, the ability to pursue increased system efficiency, improved user experience, better fault tolerance, higher levels of automation and autonomy and increased system scalability and flexibility. Moreover, they also improve system adaptability and responsiveness to changing environmental conditions, ensuring better system performance and reliability.

Conceptual Model of a Self-Adaptive System

To better understand what is a Self-Adaptive System and what are its main components, let us review the conceptual model introduced by Weyns et al. [26]. The model consists of four basic elements: the environment, the managed system, adaptation goals and the managing system.

According to Jackson et al. [20], the environment refers to the external world in which a self-adaptive system interacts and the effects of the system are observed and evaluated. The environment can be physical or virtual and it is differentiated from the self-adaptive system based on the control: the system has control over itself, but not over the environment. The environment is something that the system has to work with and respond to, but it can't change it directly since it is not under the control of the software engineer of the system. The Environment can be sensed and affected through sensors and effectors, but there may be uncertainty [16] in terms of what is sensed, of what might happen or what the outcomes of effecting the effectors will be. The managed system is the part of the self-adaptive system that performs the specific functionality and needs to be equipped with sensors and actuators to enable monitoring and adaptations. The managing system, on the other hand, manages the managed system and comprises the adaptation logic that deals with one or more adaptation goals. The adaptation goals are the concerns of the managing system driven by the so called self-* properties, including, for instance, self-configuration, self-optimization, self-healing and self-protection.

Imagine a rescue robot system that has an objective to optimize its movement and battery consumption while maintaining its operational effectiveness and safety. The system's

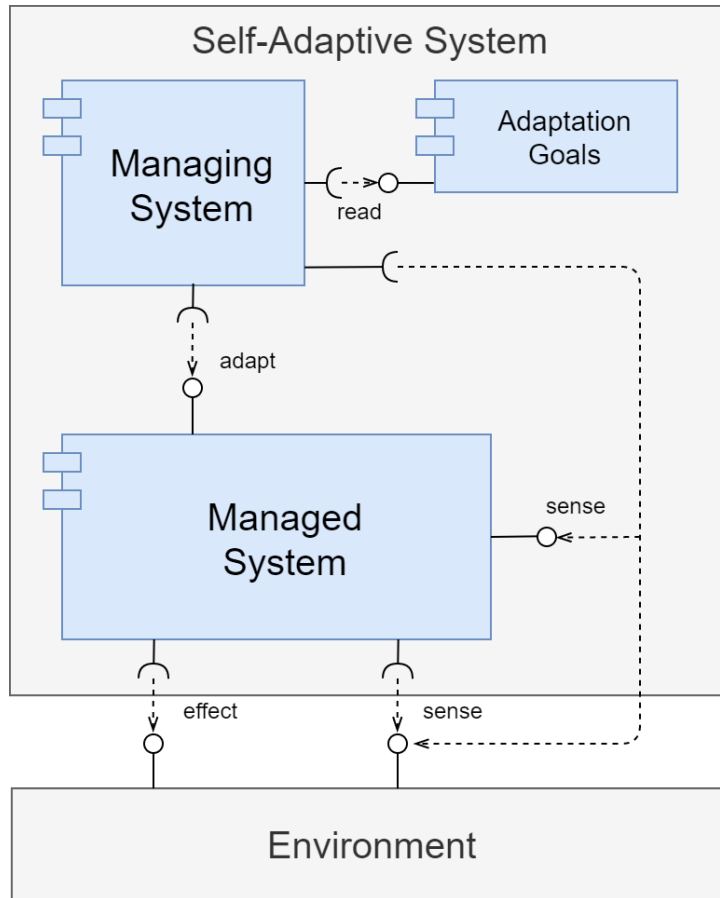


Figure 1.1: Conceptual model of a self-adaptive system

managing system is responsible for achieving this adaptation goal by making runtime decisions about the configuration of various components in the managed system, such as the movement system, the sensor system or the power system, which affect the robot's movement and battery consumption.

The managed system of the rescue robot, which comprises all software and hardware components that directly affect the robot's movement and behavior, is equipped with sensors and actuators that monitor and control the robot's surroundings and its internal states. For instance, the managed system uses cameras, sensors, and radars to sense the environment and detect obstacles, hazardous materials and survivors.

However, the environment is uncertain, and there might be unexpected events and unknown scenarios such as debris or unstable structures. These uncertainties can introduce obstacles and disturbances that may affect the robot's performance, effectiveness and battery life. Therefore, the managing system must be capable of handling the uncertainty in the environment and making adaptive decisions to ensure that the system can still meet its adaptation goals. For example, in case of a collapsed building or a narrow passage,

the managing system may decide to switch to a different movement mode or to adjust the robot's sensors, in order to optimize its movement and battery consumption while still ensuring its effectiveness and safety. Alternatively, if the robot's battery level is low, the managing system may choose to activate power-saving modes or temporarily reduce the robot's movement, while still ensuring its operational effectiveness and safety.

MAPE-K

Now that we have an overview of a SAS as a whole, we can focus on its primary building block for which later, in section 2.2, we will introduce enhancements aimed at improving its adaptability: the MAPE-K loop [11, 22].

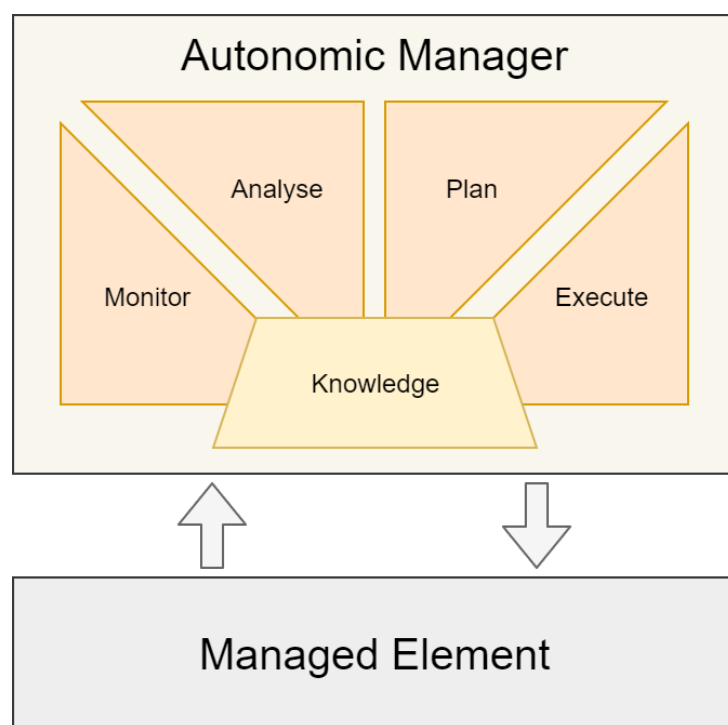


Figure 1.2: Structure of autonomic manager [26].

In the following we dive deeper into the managing system illustrated in the conceptual model of SASs in Figure 1.1. The basic functions of any self-adaptive system are performed by four elements sharing common Knowledge.

These four elements are:

1. Monitor: this component observes the system and, through its sensors, collects data about its current state and the environment in which it operates.
2. Analyze: this component analyzes the data collected by the monitor component to

assess the current state of the system and to determine whether any adaptation is necessary.

3. Plan: based on the analysis, this component creates a plan for adaptation, identifying the specific changes that need to be made to the system in order to improve its performance or to address any problems or threats.
4. Execute: this component implements the plan created by the planner component, making the necessary changes to the system through its actuators.

The shared knowledge (K) component, is responsible for storing the knowledge learned by the system during the MAPE-K loop, including data collected by the monitor component, analysis performed by the analyze component, plans generated by the plan component and the outcomes of previous execution cycles. This knowledge can be used to improve the accuracy and efficiency of the self-adaptive system over time.

1.2. Parametric Markov Decision Process

The modeling formalism that we will use to schematize the systems introduced in section 1.1 and that is now widely accepted for specification and verification of software system dependability [1] is the Markov Decision Process (MDP) [25].

An MDP is a mathematical framework used to model decision-making problems in which the outcome depends on probabilistic transitions between different states. It is named after the Russian mathematician Andrey Markov, who developed the concept of a Markov chain in the early 20th century. MDPs were originally developed as a tool for studying stochastic processes, but their applications have since expanded to many fields, including engineering, finance, and artificial intelligence. In essence, an MDP models a decision-making problem as a sequence of states, actions and rewards, in which the goal is to find a policy (a mapping from states to actions) that maximizes the expected cumulative reward over time.

Often MDPs are depicted as a state transition graph where the nodes correspond to states and (directed) edges denote transitions (a complete example can be seen in Figure 1.3).

Formally, an MDP is defined as a tuple $\langle S, A, T, R \rangle$ in which S is a finite set of states, A a finite set of actions, T a transition function and R a reward function.

By giving a series of parameters θ and a series of polynomial functions $\mathbb{Q}[\theta]$ to an MDP, we define a pMDP. Graphically, they are defined in the same way, except that they have parametric values.

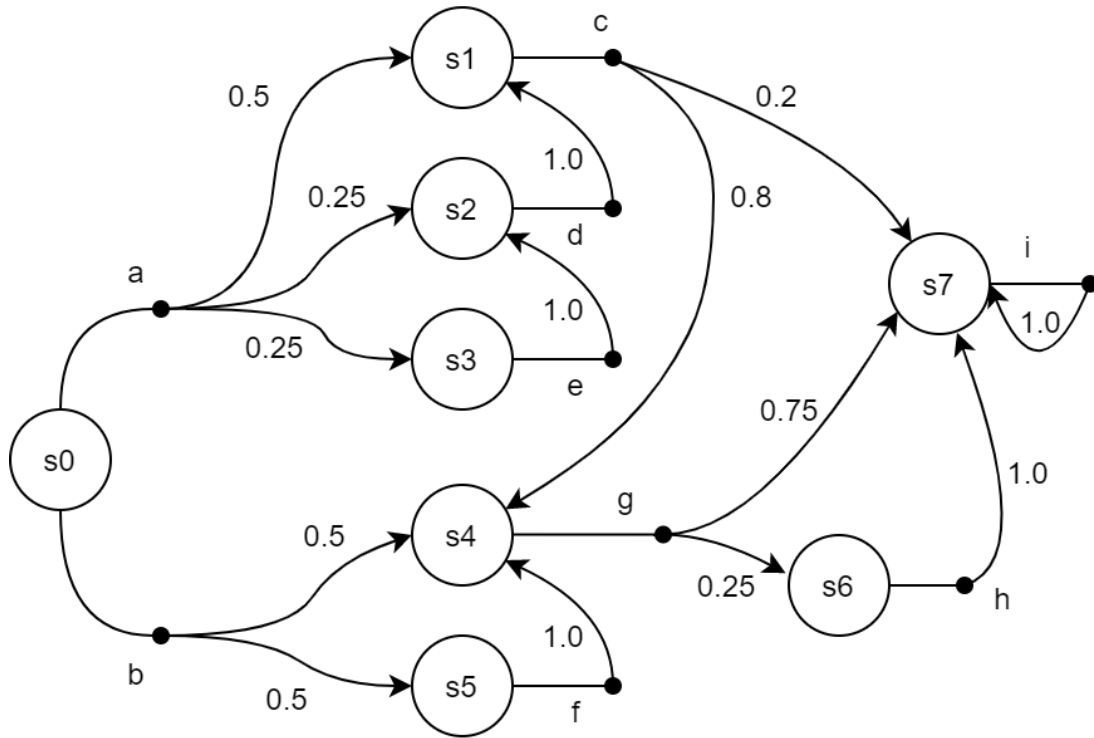


Figure 1.3: An example of an MDP with eight states and nine actions.

Formally, a pMDP is defined as a tuple $\langle S, s_0, \theta, A, \delta, R \rangle$ in which S is a finite set of states ($s_0 \in S$ is the initial state), θ is a finite set of parameters, A a finite set of actions, $\delta : S \times A \times S \rightarrow \mathbb{Q}[\theta] \cup [0, 1]$ is a partial transition function (T) and R a reward function.

State

In an MDP, a state represents a particular configuration of the system being modeled. It can be thought of as a snapshot of the system at a particular point in time which captures all the relevant information about the system that is needed to make decisions about what action to take next. For example, this representation can include the location of a robot in a navigation task or the health status of a patient in a medical diagnosis problem. In order for a state to be considered Markovian, it must satisfy the Markov property, which states that the future evolution of the system depends only on the current state and action, and not on the history of past states and actions. This property is crucial because it allows us to model complex systems in a computationally efficient way by reducing the state space to a manageable size.

In Figure 1.3, there are eight states ($s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7$) and they are represented as nodes of the graph.

Action

In an MDP, an action, which represents a decision made by an agent at a particular state of the system being modeled, can be used to control the system state. They could include, for example, selecting a particular location for a robot to move to in a navigation task, choosing which product to order in a supply chain or selecting a particular treatment for a patient in a medical diagnosis problem.

In Figure 1.3, there are nine actions $(a, b, c, d, e, f, g, h, i)$ and they are represented as edges of the graph. If, for example, we consider the state s_0 , there are two possible actions: a and b .

The Transition Function

In an MDP, the transition function describes the probability of moving from one state to another when a particular action is taken. It is a fundamental component of the MDP model, as it captures the dynamics of the system being modeled. It maps each state-action pair to a probability distribution over the next states: given the current state and the action taken by the agent, the transition function specifies the probability of transitioning to each possible next state.

Mathematically, the transition function is denoted as follows:

$$T(s, a, s') = P(s_{t+1} = s' | s_t = s, a_t = a)$$

where s is the current state, a is the action taken by the agent, s' is the next state, P denotes the probability of an event and t is a discrete global clock: s_t means the state at the moment t and s_{t+1} means the state at the moment $t + 1$. $T(s, a, s')$ represents the probability of transitioning from state s to state s' when action a is taken.

Moreover, the transition function has also to satisfy the Markov Property: the probability to get from a state s through an action a to a state s' has to be the same regardless of the previous actions and visited states. Mathematically:

$$P(s_{t+1} | s_t, a_t, s_{t-1}, a_{t-1}, \dots) = P(s_{t+1} | s_t, a_t) = T(s_t, a_t, s_{t+1})$$

In the Figure 1.3, the transition function determines the probability values represented on each edge. If, for example, we move from the state s_1 to the state s_4 through the action c , the transition function asserts that the probability of this event is 0.8.

The Reward Function

The reward function, whose result is a scalar value, specifies the immediate incentive or punishment received by the agent for taking a particular action in a particular state. In general, it's used to evaluate the quality of the actions performed by the agent, whose goal is to maximize the expected cumulative reward over time.

There are three different definitions:

$$R : S \longrightarrow \mathbb{R} \quad (1.1)$$

$$R : S \times A \longrightarrow \mathbb{R} \quad (1.2)$$

$$R : S \times A \times S \longrightarrow \mathbb{R} \quad (1.3)$$

The first definition (1.1) specifies the reward obtained in a specific state. The second definition (1.2) specifies the reward obtained by selecting a specific action in a specific state. The third definition (1.3) specifies the reward obtained by landing in a specific state coming from a specific state-action pair.

The reward function can be either positive, negative or zero, depending on the nature of the problem being modeled. For example, in a game, winning a point might be rewarded with a positive score, while losing a point might be penalized with a negative score.

Policy

A policy is a function that specifies what action the agent should take in each state in order to maximize its expected cumulative reward. Essentially, it's a mapping from states to actions, that the agent uses to make decisions and navigate through the state space.

There are two main types of policies: deterministic and stochastic. A deterministic policy specifies a single action to be taken in each state, while a stochastic policy specifies a probability distribution over actions to be taken in each state.

Formally, a policy is denoted by the symbol π and it is defined as follows (deterministic (1.4) and stochastic (1.5)):

$$\pi : S \longrightarrow A \quad (1.4)$$

$$\pi : S \times A \longrightarrow [0, 1] \quad (1.5)$$

In order to get the highest possible expected cumulative reward, the agent's objective is

to find an optimal policy. There are different methods for accomplishing such a goal in an MDP, including value iteration and policy iteration algorithms. Value iteration involves iteratively computing the optimal value function, which represents the expected cumulative reward from each state, and using it to derive the optimal policy. Policy iteration, on the other hand, involves iteratively improving an initial policy until it converges to the optimal policy.

1.3. Bayesian Inference

Now that we have defined the pMDPs, we need to introduce a method that allows us to estimate their parameters without accessing the real values. The only information we can rely on is prior knowledge and evidence collected during the execution of the pMDP. Therefore, starting from prior knowledge of the behavior of a stochastic phenomenon of interest (the pMDP), the goal is to be able to identify the parameters that generated it (the set θ). The approach chosen is called Bayesian Inference.

The Bayesian approach to statistical inference [6] is a method for updating our beliefs or knowledge about a parameter of interest based on available data. At the heart of Bayesian inference is Bayes' theorem, which states that the posterior probability distribution of a parameter is proportional to the product of the prior probability distribution and the likelihood function. Mathematically, it can be expressed with the following equation:

$$f(\theta|y) \propto f(\theta) \cdot f(y|\theta) \quad (1.6)$$

The prior distribution $f(\theta)$ represents our beliefs or knowledge about the parameter before observing any data, available in the form of expert information based on past experience or previous studies, while the likelihood $f(y|\theta)$ function represents the probability of observing the data given a particular value of the parameter, in other words represents the compatibility of the data with the hypothesis.

Starting from the posterior distribution of our multivariate case, we can compute either a point or an interval estimation. The point estimation can be expressed as the mean of the distribution without considering the notion of confidence encoded into it, formally:

$$\mathbb{E}[f(\theta|y)] = \int \theta \cdot f(\theta|y) d\theta$$

The interval estimation, calculated using the Highest Density Region (HDR), which represents the region containing $100(1 - \alpha)\%$ of the posterior distribution that in our case

encodes a set of credible intervals, provides further information by accounting for the notion of confidence. Given $\text{HDR}[f(\theta|y)] = C$, formally:

$$P(\theta \in C|y) = \int_C f(\theta|y) d\theta = 1 - \alpha$$

typically using $\alpha = 0.05$. These resulting intervals represent the region within which each variable value falls with probability $1 - \alpha$. As described in [24], we can measure the confidence in the inference process using the magnitude of the HDR: the smaller the region, the higher the confidence.

1.4. Bayesian Model Averaging

A model is a mathematical representation of a system, process or phenomenon and, in statistical modeling, it is used to make predictions or inferences based on data. It can be composed of multiple parameters which are responsible for its outcome and finding their right value can be difficult and time consuming.

An example useful to illustrate the concepts of model representation, parameters, and the difficulty in finding their right values can be a linear regression model. A linear regression model is a mathematical representation of the relationship between two variables: let us call them x and y . It assumes that there is a linear relationship between x and y , which can be described by a straight line equation. The equation takes the form of $y = mx + b$, where m is the slope of the line (i.e., the change in y for a unit change in x) and b is the y -intercept (i.e., the value of y when x is zero). The model assumes that there is some random error or noise in the relationship that is not captured by the equation. Let us suppose that we want to predict a person's weight (y) based on their height (x). We can collect data on the heights and weights of a sample of individuals and use it to estimate the parameters of the linear regression model (i.e., the slope and intercept). However, the model might not always fit the data well, leading to model uncertainty. For example, if there is a non-linear relationship between height and weight, then a linear regression model might not be appropriate and another model may need to be used instead.

Bayesian Model Averaging (BMA) [18, 19] is a statistical method used to estimate the model parameters and to make predictions in the presence of model uncertainty. Unlike traditional statistical methods, that focus on selecting a single "best" model, BMA allows for the simultaneous consideration of multiple models and the idea behind it is to compute weighted averages of the models considered, where the weights are proportional to the posterior probabilities of the models. The posterior probability of a model is the probabil-

ity of the model given the data and the prior information. The weights reflect the relative plausibility of the models, and the weighted averages provide a way to make predictions that account for the uncertainty in the model selection process. This approach has been adopted in a range of disciplines to incorporate model-selection uncertainty into statistical inference and prediction [14, 17, 19, 21, 28]. An example of BMA can be illustrated in the context of predicting the sale price of a house based on its features.

Suppose we have a dataset of environments with various features such as obstacles, hazardous materials, victims, and structural damage, along with their corresponding rescue plans. We want to build a model to predict the best rescue plan for a new environment based on its features. To use BMA, we first need to specify a set of candidate models, each with a different set of predictor variables. For example, we might consider a model with only the obstacles as a predictor, another model with only hazardous materials as a predictor, and a third model with all the features as predictors.

$$\begin{aligned}
 \text{Rescue Plan} &= \beta_1 \cdot \text{obstacles} \\
 \text{Rescue Plan} &= \beta_1 \cdot \text{hazardous materials} \\
 \text{Rescue Plan} &= (\beta_1 \cdot \text{obstacles}) + (\beta_2 \cdot \text{hazardous materials}) + (\beta_3 \cdot \text{victims}) + \\
 &\quad (\beta_4 \cdot \text{structural damage})
 \end{aligned} \tag{1.7}$$

Next, we need to assign prior probabilities to each of the models based on our prior knowledge or assumptions. For example, we might assign higher prior probabilities to the models that include more features, assuming that more features are likely to be relevant for predicting the best rescue plan for a given environment.

Once we have assigned prior probabilities, we can use the Bayesian approach to estimate the posterior probabilities of the models based on the data. The posterior probability of a model is the probability of the model given the data and the prior information. This reflects the relative plausibility of each model given the available data and our prior beliefs.

After estimating the posterior probabilities of the models, we can use BMA to compute weighted averages of the model parameters. These weighted averages account for the uncertainty in the model selection process and provide a way to make predictions that incorporate the information from all the candidate models.

For example, suppose we have estimated the posterior probabilities of three models: Model 1 (obstacles only) with a posterior probability of 0.3, Model 2 (hazardous materials only)

with a posterior probability of 0.2, and Model 3 (all features) with a posterior probability of 0.5. We can compute weighted averages of the model parameters, where the weights are proportional to the posterior probabilities of the models. The weighted averages provide a way to make predictions that account for the uncertainty in the model selection process.

To compute the weighted average of the best rescue plan for a new environment, we would use a formula such as:

$$\text{Best Rescue Plan} = 0.3 \cdot \text{model}_1 + 0.2 \cdot \text{model}_2 + 0.5 \cdot \text{model}_3$$

The prior and posterior probabilities play a crucial role in BMA. The prior probability reflects our prior knowledge or assumptions about the relative plausibility of each model before we have seen the data. The posterior probability reflects the updated probability of each model given the data and our prior information.

1.5. Meta-Heuristic Optimizing Search

An optimization problem involves a mathematical calculation that aims to determine the best possible outcome for a given objective function while following a set of constraints. The objective function represents the variable that is to be either maximized or minimized while the constraints determine the parameters or requirements of the problem. The ultimate goal is to identify the optimal values for the decision variables while satisfying the constraints and achieving the desired objective function outcome.

Without any loss of generality an optimization problem can be defined by:

$$\begin{aligned} \min \quad & f_m(x) & m = 1, \dots, M \\ \text{s.t.} \quad & g_j(x) \leq 0 & j = 1, \dots, J \\ & h_k(x) = 0 & k = 1, \dots, K \\ & x_i^L \leq x_i \leq x_i^U & i = 1, \dots, N \\ & x \in \Omega \end{aligned} \tag{1.8}$$

where x_i represents the i -th variable to be optimized, x_i^L and x_i^U its lower and upper bounds, f_m the m -th objective function, g_j the j -th inequality constraint and h_k the k -th equality constraint. The objective function(s) f_m are supposed to be minimized by satisfying all equality and inequality constraints. If a specific objective function is maximized (if instead of \min we had \max), one can redefine the problem to minimize its negative value ($\min - f_i$).

To better explain an optimization problem, let us use a simple example related to a rescue robot: in such a case, we would be interested in maximizing the efficiency of the robot while minimizing the time taken to rescue the victims. As constraints, we could have a limited battery life (which limits the maximum distance it can travel) and the need to reach each victim within a certain time window. The objective of the problem is to maximize the number of victims rescued by the robot over the course of the mission, while minimizing the total time spent traveling between the rescue sites. To solve this optimization problem, the robot would need to make a series of decisions, which are the homonym variables, about where to go, how to reach each victim, and how to navigate between the rescue sites.

Based on the number of objective functions involved, an optimization problem can be classified as a single objective, multi-objective or many-objective optimization problem. In this context, we will focus on multi and many-objective optimization problems.

Multi-objective optimization problems are a type of optimization problem that involves optimizing multiple objective functions simultaneously subject to a set of constraints. In such problems, objective functions may be conflicting and finding a single optimal solution that satisfies all objectives may not be possible. Instead, a set of Pareto-optimal solutions is often proposed, where no solution is dominated by any other solution in terms of all objective functions. Multi-objective optimization is useful when there are competing objectives that need to be balanced and when no single optimal solution satisfies all objectives.

Many-objective optimization problems, on the other hand, are a type of multi-objective optimization problem that involves optimizing a large number of objective functions simultaneously (more than 3 objectives). These problems are typically encountered in complex engineering and design problems where multiple performance criteria need to be considered. However, the high dimensionality of the problem makes it challenging to find a set of Pareto-optimal solutions that are diverse and informative.

Meta-heuristic optimizing search [7, 13] pertains to a category of computational optimization strategies that are used to identify appropriate solutions for complex optimization problems that cannot be reasonably and efficiently addressed by using precise mathematical methods. The technique employs meta-heuristic algorithms that mimic natural evolutionary processes to explore the search space in an intelligent manner, using heuristic or rule-of-thumb approaches to steer the search toward potential areas of the search space. These algorithms are capable of efficiently exploring large and complex search spaces and can find near-optimal solutions within a reasonable time frame.

Examples of meta-heuristic algorithms include: Genetic Algorithm, Particle Swarm Optimization, Water Waves Optimization, Clonal Selection Algorithm, Chemical Reaction Optimization, Harmony Search, Sine Cosine Algorithm, Simulated Annealing, Teaching–Learning–Based Optimization, League Championship Algorithm, Tabu Search, Variable Neighborhood Search and others [2].

We will focus our attention on the GA family of algorithms, which are a popular optimization method that utilizes natural selection and genetics principles to solve complex problems. One of the commonly used genetic algorithms for multi-objective optimization is NSGA-3 (Non-dominated Sorting Genetic Algorithm-3), which is an extension of the NSGA-2 algorithm designed to handle many-objective optimization problems. The main advantage of NSGA-3 is its ability to generate a well-distributed set of Pareto-optimal solutions that are diverse and informative across the entire objective space.

The NSGA-3 algorithm begins by creating an initial population of individuals at random. These individuals are then evaluated based on their fitness, which measures their performance with respect to the optimization problem’s objectives. To determine the fitness of an individual in NSGA-3, two measures are used:

- **Dominance rank:** It evaluates how well an individual dominates other individuals in the population based on their performance on different objectives. An individual is considered non-dominated if no other individual in the population outperforms it on all objectives.
- **Crowding distance:** It assesses the diversity of the population and ensures that the algorithm thoroughly explores the search space. It measures the crowdedness of an individual’s region in the search space, giving preference to individuals with higher crowding distances in the selection process to maintain diversity in the population.

After determining the fitness of each individual, NSGA-3 selects non-dominated individuals for reproduction using genetic operators such as crossover and mutation. The resulting offspring replace less fit individuals in the population, and the process is repeated for multiple generations until a set of Pareto-optimal solutions is obtained. These solutions represent trade-offs between the objectives of the optimization problem and offer decision-makers a range of options to choose from.

2 | Starting Point

This chapter serves as an introduction to two pivotal components of our proposal. We firstly delve into the concept of Run-Time Equilibrium Verification (RUNE) and examine how it facilitates the schematization of a system and guarantees its requirements are met by leveraging the notion of equilibrium and equilibrium constraints. Secondly, we explore the process of Taming model UNcErtainty (TUNE) in SASs, which involves the integration of Bayesian Model Averaging and Meta-heuristic optimizing search to improve the feedback control loop architecture for adaptability.

2.1. Run-Time Equilibrium Verification

Within the context of CPSs, it is expected that they remain safe and functional under any possible scenario, given that their main characteristic is the coexistence of heterogeneous components (physical and computational).

To this end, a new approach called RUNtime Equilibrium verification (RUNE) has been proposed [9], which leverages the notion of equilibrium, i.e., the ability of the system to maintain a behavior within its multidimensional viability zone [5], defined as the set of possible states in which the system operation is not compromised [3].

RUNE enables the verification at run-time of whether a system satisfies or not the equilibrium conditions and it is structured in four phases (a schematic overview is provided in Figure 2.1):

1. define a system with partial knowledge using a pMDP
2. calculate the equilibrium constraints offline
3. verify at run-time whether the equilibrium constraints are satisfied
4. enforce the equilibrium constraints at run-time

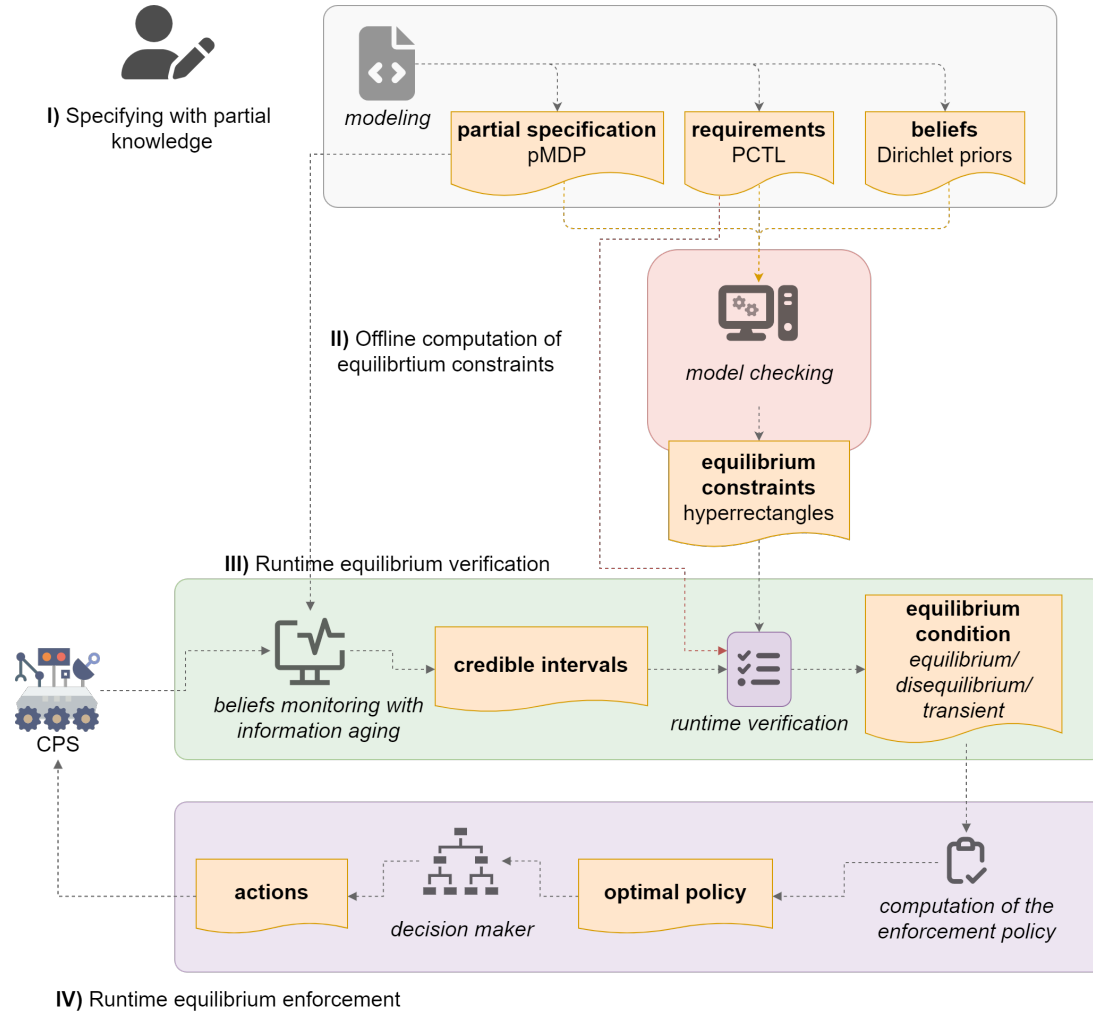


Figure 2.1: RUNE approach overview [9].

2.1.1. Specifying with Partial Knowledge

The first phase focuses on modeling non-deterministic and stochastic behaviors, respectively stimuli/events and their corresponding outcomes, using a formalism for the specification and verification of software system dependability that is widely accepted [1].

Essentially, it involves creating a schematization of the behavior of the system under study using five components:

- **pMDP:** a parametric state transition graph where the nodes correspond to states and (directed) edges denote transitions (see chapter 1.2).
- **Uncertain region:** the set of parameters attached to the edges of the pMDP.
- **Semantic space:** the set of configuration and environment dimensions describing some internal and external quantities, like the battery percentage or the weather

condition, which may have an impact on the uncertain regions.

- Prior knowledge: the initial beliefs or assumptions that we want to give to the system about the probability distribution of the uncertain regions before any data is observed.
- Probabilistic Computation Tree Logic (PCTL) dependability requirements: a formal language used to formalize the dependability constraints of interest.

The example provided in the paper defines a rescue robot system, which, during navigation, may encounter still human bodies or other obstacles. The success or failure of the robot’s visual perception depends on both the environment conditions and the system’s internal configuration and, due to uncertainties that exist both inside and outside the system, specifying the probabilities of these outcomes may be challenging. To this end, we utilize variables such as x_3 to represent the probability of misclassification and x_1 to represent the likelihood of a contact resulting from detection failures, which can potentially lead to safety concerns.

2.1.2. Offline Computation of Equilibrium Constraints

Once all the necessary components to model the system under study are defined, it is necessary to pre-compute the equilibrium constraints. These are intervals that ensure the satisfiability of all PCTL dependability requirements, regardless of any possible assignment of the semantic space. It should be noted that the parameters within the pMDP are variables and it is therefore necessary to define an interval that satisfies the requirements for each of them.

An example of equilibrium constraints, i.e. boundaries, for a set of variables inside the uncertain region θ_1 of the pMDP depicted in Figure 2.2a are visually shown in Figure 2.2.

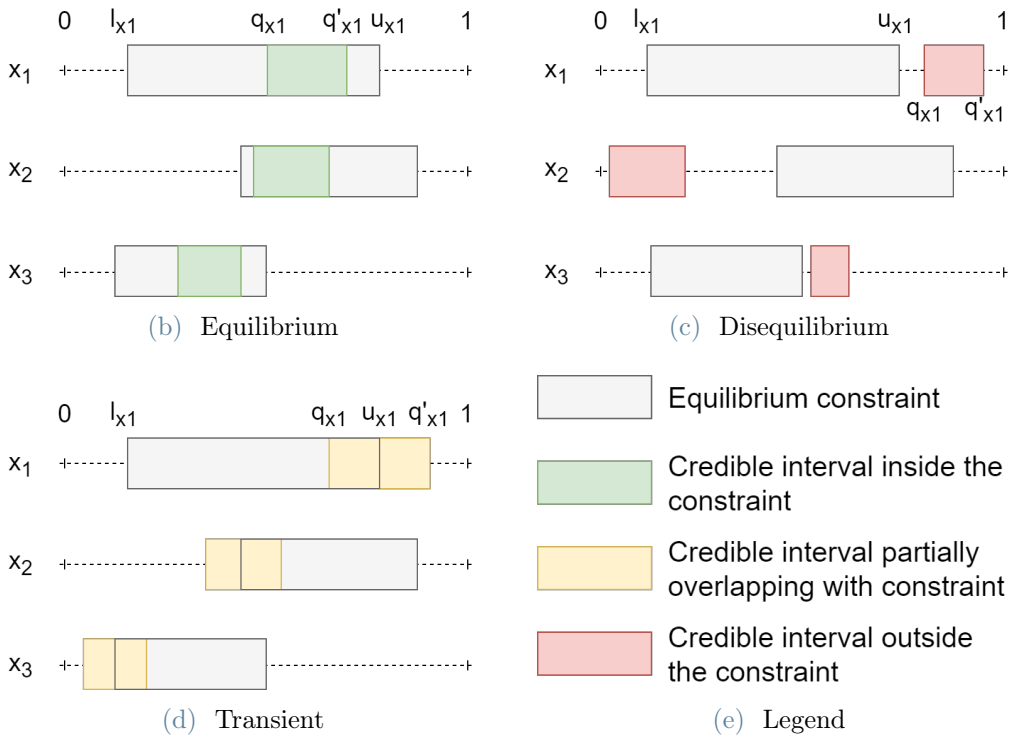
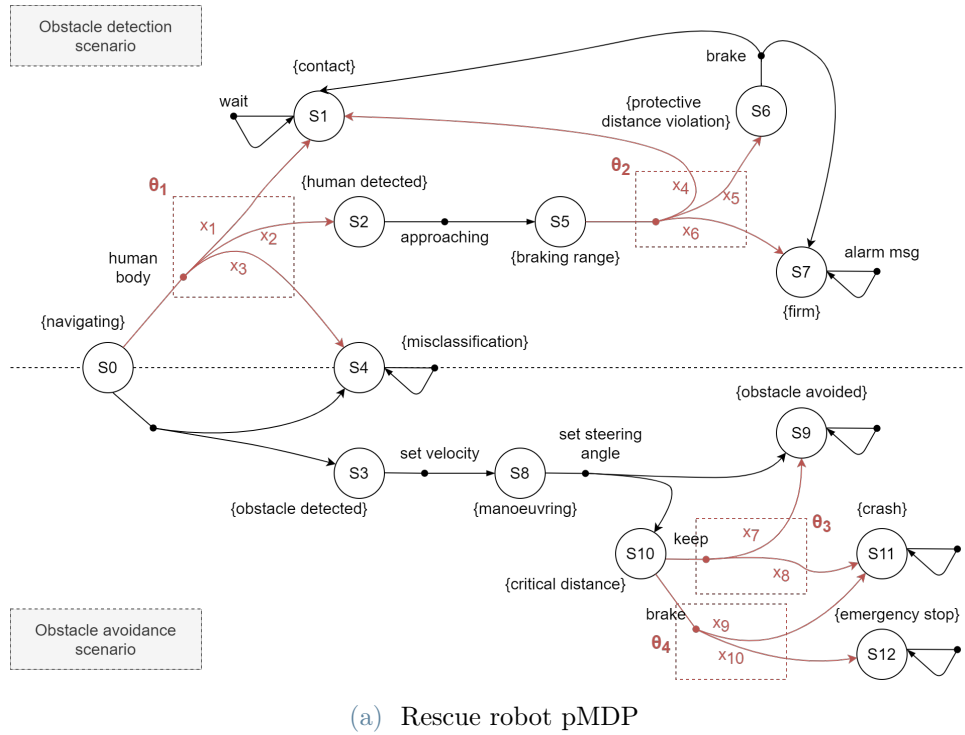


Figure 2.2: Visualization of the equilibrium conditions for the uncertain region θ_1 of the pMDP in Figure 2.2a in case of Equilibrium (2.2b), Disequilibrium (2.2c) and Transient (2.2d). [9]

2.1.3. Run-Time Equilibrium Verification

In this phase, the main objective is to control the CPS behavior at run-time: to comply with the required dependability properties, it must always remain in equilibrium.

For example, returning to the rescue robot, we want the system to comply with the dependability requirements even in moments when sudden changes occur in the semantic space, such as thick smoke, low light or any other event that may temporarily disturb the robot's sensors.

To observe and quantitatively monitor the effects of these changes, we collect run-time evidence. These pieces of evidence are later used to feed two timelines, one longer and one shorter. These two timelines allow us to have a more accurate estimate over the long term and a faster reaction to sudden changes, respectively. Both of these estimates are computed through Bayesian Inference (§ 1.3). The combination of these two timelines composes the *adaptive observation aging mechanism*, which is helpful to filter out old information about changes occurred in order to alleviate the negative effects of historical data: after a certain amount of inconsistencies between the estimates carried out from these timelines, the longer of the two is flushed.

Once these estimations, named q_x and q'_x , have been obtained, which we remember are relative to the pMDP parameters, we use the concepts of equilibrium, disequilibrium and transient to define whether the required dependability properties are met:

- Equilibrium: given the estimation $\hat{\theta}_i$ for each parameter θ_i and the equilibrium constraint \mathcal{H}^* , we say that the equilibrium condition holds if and only if:

$$\exists \prod_{x \in \theta} [l_x, u_x] \in \mathcal{H}^* : \forall x \in \theta, [q_x, q'_x] \subseteq [l_x, u_x]$$

- Disequilibrium: given the estimation $\hat{\theta}_i$ for each parameter θ_i and the equilibrium constraint \mathcal{H}^* , we say that the disequilibrium condition holds if and only if:

$$\forall \prod_{x \in \theta} [l_x, u_x] \in \mathcal{H}^*, \exists x \in \theta : [q_x, q'_x] \cap [l_x, u_x] = \emptyset$$

- Transient: we say that the system has transient behavior when none of the above conditions hold. In such a case, there is a significant degree of uncertainty and further evidence is necessary to reduce the size of the credible intervals.

Figure 2.2 displays three examples that focus on a single variable in an uncertain region

to better visualize the meaning of equilibrium (Figure 2.2b), disequilibrium (Figure 2.2c), and transient (Figure 2.2d).

2.1.4. Run-Time Equilibrium Enforcement

If, in the previous step, the disequilibrium condition holds, the enforcement mechanism is triggered. In such a case, a new optimal policy, aiming at maximizing the probability of enforcing the equilibrium, is computed based on the outcome of the verification phase.

First, rewards with low values are assigned to transitions leading to variables in disequilibrium and high values to those leading to variables in equilibrium/transient states. Subsequently, controllable states and their corresponding actions are identified to maximize the probability of not passing through transitions with low rewards.

2.2. Taming Model Uncertainty in Self-Adaptive Systems

Usually, mainstream approaches use model-based inference techniques inside the analysis and plan activities to enhance the feedback control loop architecture for adaptation, that is the mechanism used to adapt to changes in the system itself or in the environment. These kind of approaches select the "best" model representing the system without taking into account the existence of other plausible models. The issue caused by such an assumption is that poor predictions lead to failures in recognizing the need of an adaptation and might yield unnecessary adaptations affecting dependability attributes [12].

The goal of TUNE (Taming model UNcErtainty) [8] is to enhance the classical approach of a SAS with the ability to mitigate the uncertainty related to the model selection process using BMA.

TUNE's first innovative concept is the introduction of the BMA estimates into the Analyze component.

These estimates permit the prediction of possible violations of the requirements. They are obtained from the model space $M = \{M_1, \dots, M_n\}$, where $n = 2^{|X|}$ and M_i is the i^{th} model obtained by including in the predictor only a subset of the explanatory variables in X . Due to the fact that the total number of models can be huge, M cannot be exhaustively explored. To this end, techniques such as Markov Chain Monte Carlo (MCMC) [15] exist and can be helpful in exploring this space and identifying models that are most likely to yield accurate predictions by producing efficient random draws from a high dimensional

space.

The following modification introduced by TUNE and strictly bound to the first one, is adding meta-heuristic optimizing search inside the Plan component. If, through the previous step, a violation is predicted with high probability, the Plan component is triggered. The plan adopts meta-heuristic optimizing search, as described in [10]. This approach explores the semantic space looking for alternative assignments of the configuration variables (i.e., controllable phenomena) without modifying the assignment of the environment variables (i.e., observable phenomena) with the goal of minimizing the adaptation cost and, while also, maximizing the likelihood of satisfying the requirements.

For example, let us consider again the search and rescue robot presented in 2.1: in this case, each plausible model can be viewed as tailored to a specific scenario. At run-time, it is not necessary to know the exact scenario that the robot is in, instead, we can consider all the possible scenarios simultaneously and weigh them according to the likelihood of observing them given the evidence collected up to that point. Once a resulting model has been computed, we can use it to check whether or not a requirement is violated. If it is, the plan component tries to find a new set of configuration variables for the robot, such as a new cruise speed, without modifying the environmental variables, like the illuminance or the smoke intensity, focusing on minimizing the cost of such a change and maximizing the likelihood of finding a new assignment that satisfies all the requirements.

3 | Our Proposal

Let us consider a remotely controlled rescue robot that can operate in a wide range of rescue conditions, such as in disasters, in extreme climates, and in different terrains. This robot will have many parameters, both internal, such as its speed, mobility, or sensor angle, and external, such as the amount of debris, the condition of the terrain, or visibility, that can affect its behavior.

Now, let us imagine we want to install an emergency response module. During rescue operations, the robot must never come into contact with any obstacle, whether it be debris, walls, or people. Therefore, if the minimum safety distance is violated, the robot must autonomously stop or change its direction.

But what happens if the terrain is steep? And if visibility is poor? Or if the robot's mobility is reduced due to obstacles? What if multiple factors are combined? The system must adapt in real-time to always achieve its objective.

Let us now imagine we want to install a fully autonomous rescue module. Consequently, to ensure safe and efficient operation, the number of requirements will no longer be just one, but many others will be added, such as detecting and avoiding hazardous materials, maintaining a safe distance from rescue personnel, and avoiding collisions.

Compared to before, the problem has become much more complex, as finding a new parameter configuration capable of satisfying a single requirement is not enough. Rather, we need to find one that can satisfy as many requirements as possible to ensure the safety of both the robot and the rescue personnel.

In the subsequent chapters, our focus shall be directed towards the formulation of the problem that we aim to resolve. Specifically, we shall scrutinize the challenges and intricacies that our project must confront, as well as the objectives that we have aimed to achieve (§ 3.1). We shall delineate the input, output, and structure of our pMDP simulator, which constitutes the main building block of our proposal, and it enables the simulation of self-adaptive systems while validating their equilibrium functionality (§ 3.2). Eventually, our concentration shall pivot towards the application and utilization of BMA

and meta-heuristic optimizing search mechanism to tackle uncertainty in the selection of logistic models, aiming to discover new configurations of the semantic space that can re-establish balance to a self-adaptive system, thus guaranteeing its correct functionality (§ 3.3).

3.1. Problem Formulation

Given a generic CPS, a set of parameters and a series of requirements, the objective of the CPS is to satisfy as many requirements as possible by modifying its parameters.

The set of parameters is called configuration and is composed of a modifiable section (the internal system parameters) and a non-modifiable one (the parameters of the external environment).

The requirements are in the form of *TRUE* or *FALSE*, meaning that the requirement can only be satisfied or not satisfied, with no middle ground.

The CPS is modeled as a pMDP which, after performing its simulation with a given input configuration, allows us to determine whether the requirements are satisfied. Unfortunately, we cannot use this tool directly within our solution because its execution would take too long and consequently we would not be able to find a solution in real-time. Instead, we will use statistical models which, referring to each individual requirement, will provide us with values between 0 and 1 where 0 means “requirement not satisfied” and 1 means “requirement satisfied”. It is important to note that the type and choice of statistical model used can greatly impact the results obtained. While it may be tempting to choose the best model upfront, this approach may not always yield the best results: indeed, settling on a single model (even the best one) could involve the risk of neglecting possible scenarios.

Now that all the inputs have been defined, we can formulate the following optimization problem, whose constraints are the intervals of the modifiable parameters (those internal to the system) and the fixed assignments of the non-modifiable parameters (those external to it). As objective functions, we have the cost function, which returns the cost to apply a new configuration given the starting configuration, and the aforementioned models, each of which, referring to a specific requirement, tells us whether that requirement is satisfied or not given the new possible configuration.

Formally:

$$\begin{aligned}
 \min \quad & 1 - f_m(x_1, \dots, x_N, y_1, \dots, y_K) \quad m = 1, \dots, M \\
 & c(\bar{x}_1, \dots, \bar{x}_N, y_1, \dots, y_N) \\
 \text{s.t.} \quad & x_i^L \leq x_i \leq x_i^U \quad i = 1, \dots, N \\
 & \bar{y}_j = y_j \quad j = 1, \dots, K
 \end{aligned} \tag{3.1}$$

Where x_i with $i \in [1, N]$ are internal parameters, y_j with $j \in [1, K]$ are external parameters, \bar{x} identifies the starting values, and x_i^L and x_i^U denote the lower and upper bounds of the variable x_i . The cost function is $c(\dots)$ and the functions $f_m(\dots)$ are related to the M models and allow us to understand if the various requirements are satisfied.

Our study is divided into two main phases: in the first one, we focus on the development of a pMDP simulator, while in the second one, using the results obtained from the simulator, we concretely define the optimization problem just introduced.

3.2. Simulator

During the development of the first essential component for our proposal, we took inspiration from the RUNE approach, concretely implementing the definition of a system with partial knowledge using pMDP (§ 2.1.1), the runtime verification of the satisfaction of equilibrium constraints (§ 2.1.3) and their enforcement (§ 2.1.1).

3.2.1. Input Definition

As in the first phase of RUNE, we need to define the tools that allow us to model our CPS: specifically, we need to define the pMDP, its uncertain areas, the semantic space and the prior knowledge. Furthermore, in our implementation, we extend this initial configuration by including the equilibrium constraints: while in RUNE they had to be obtained by evaluating the PCTL dependability requirement, in our case, we assume they have already been computed.

To better visualize the architecture of our pMDP model, we begin with a class diagram (Figure 3.1) that includes the main classes involved in the system. The diagram illustrates the relationships and interactions between the classes, highlighting their attributes. The classes depicted in the diagram include the pMDP itself, the states, the actions, the semantic space and semantic space variables. For each of these elements, we are going to dive deeper into the details of their definition.

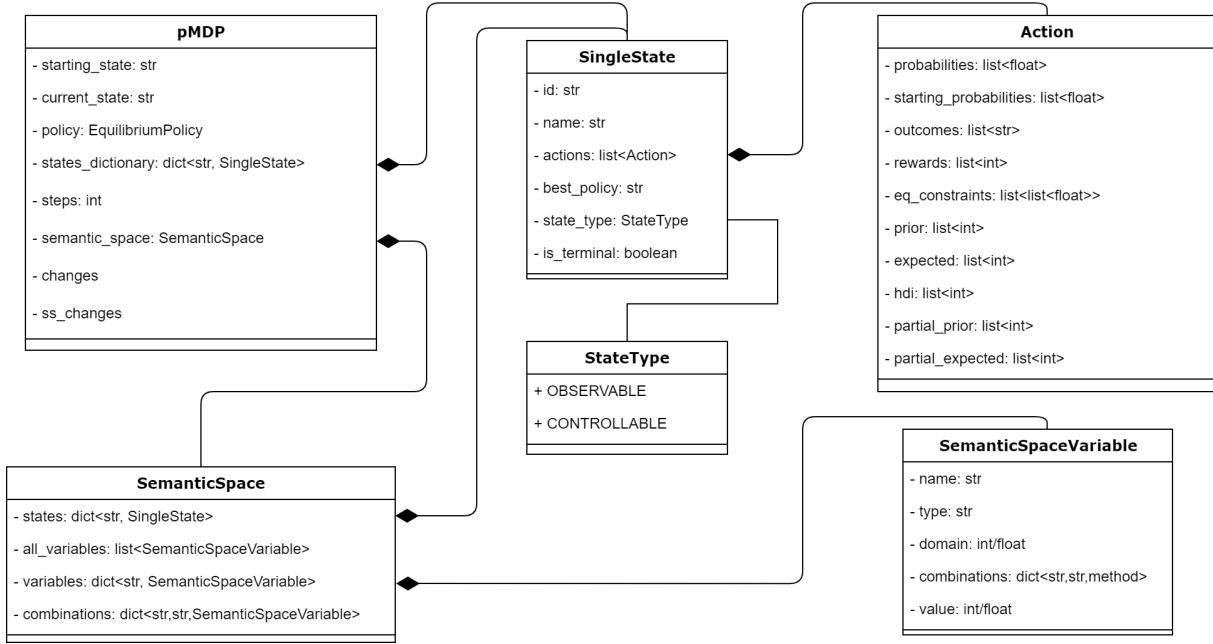


Figure 3.1: The main three classes composing the pMDP object

States, Actions, and Transition Functions (MDP)

As we said earlier, our proposal consists of integrating the MDP with additional parameters (θ). Therefore, it is essential and preliminary to define the MDP. This element (MDP) is of fundamental importance because it will serve as the pivot for all the other specifications that we will define later. The goal here is to define a language that allows us to schematize an MDP: therefore, the states, actions and transition functions must be describable.

Let us start by providing a template for a row that defines a state:

$$state\ id; state\ label; state\ type; [action;]^+ \quad (3.2)$$

- *state id*: a string containing a unique code that allows to identify the state.
- *state label*: a string containing the label to assign to the state. While for the *state id* the goal is to provide a unique code that allows the simulator to quickly and unambiguously identify each state, this attribute is designed to provide more immediate and easily understandable feedback to the simulator user.
- *state type*: an ENUM that specifies whether the state is observable or controllable. In the former case we will have `OBSERVABLE`, while in the latter `CONTROLLABLE`.

- *action*: each state must have one or more defined actions within it. In case it is a final state, i.e. a state from which no outgoing “arrow” is foreseen, it will still be necessary to define a self-loop: i.e. an action whose only outcome is the starting state. Actions are defined as tuples containing three values “*action id, probability, outcome*,”:
 - *action id*: just like for the state definition, it is necessary to uniquely define actions. However, unlike states, this must be unique only within the same state: two actions with the same identifier can coexist without negatively influencing the result if they are related to two different states.
 - *probability*: a float whose value must be between 0.0 and 1.0. This will define the result of the transition function with the input of the starting state id, the selected action id, and the outcome state id. In other words, it will be the probability that, once *action id* is selected, we will arrive at the *outcome* state (see below).
 - *outcome*: a string containing the identifier of the state that can be reached with *probability* by selecting *action id*.

To better understand a state definition in this language, we provide two examples:

$$s0; start; CONTROLLABLE; a, 0.5, s1; a, 0.3, s2; a, 0.2, s3; b, 0.4, s4; b, 0.6, s5; \quad (3.3)$$

$$s10; turn\ off\ engine; OBSERVABLE; i, 1.0, s10; \quad (3.4)$$

In (3.3), a state with id *s0* and label *start* is defined. This is a **CONTROLLABLE** state (meaning the agent can select their desired action) which has two available actions, *a* and *b*.

- The first action, *a*, has three possible outcomes, i.e., it can lead to three different states: *s1*, *s2*, and *s3*. The probabilities of reaching these states after this action are respectively 0.5, 0.3, and 0.2.
- The second action, *b*, has only two outcomes, i.e., it can lead to states *s4* and *s5* with probabilities of 0.4 and 0.6, respectively.

In (3.4), a terminal state, which has a self loop, is defined as an **OBSERVABLE** state with id *s10* and label *turn off engine*. Since it is observable, if multiple actions were available, one of them would be chosen randomly with a uniform distribution.

For completeness, we provide an example of a complete MDP followed by its respective graphical representation in Figure 3.2.

```

s0; start; CONTROLLABLE; a,0.5,s1;a,0.25,s2;a,0.25,s3;b,0.5,s4;b,0.5,s5;
s1; state 1; CONTROLLABLE; c, 0.5, s6; c, 0.25, s7; c, 0.25, s8;
s2; state 2; CONTROLLABLE; d, 1.0, s1;
s3; state 3; CONTROLLABLE; e, 1.0, s2;
s4; state 4; CONTROLLABLE; g, 0.5, s9; g, 0.25, s10; g, 0.25, s11;
s5; state 5; CONTROLLABLE; f, 1.0, s4;
s6; state 6; CONTROLLABLE; m, 1.0, s0;
s7; state 7; CONTROLLABLE; l, 1.0, s6;
s8; state 8; CONTROLLABLE; k, 0.7, s7; k, 0.3, s12;
s9; state 9; CONTROLLABLE; h, 0.3, s8; h, 0.7, s12;
s10; state 10; OBSERVABLE; i, 1.0, s7;
s11; state 11; OBSERVABLE; j, 1.0, s10;
s12; END; OBSERVABLE; n, 1.0, s12;

```

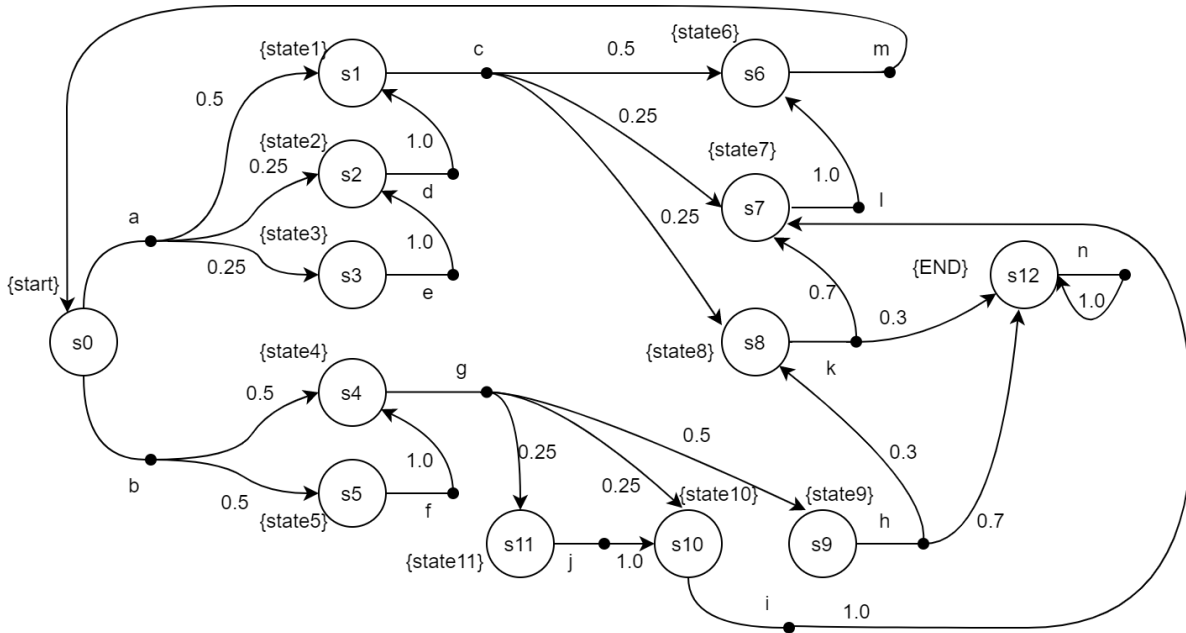


Figure 3.2: An example of MDP.

Uncertain Model Regions, Priors and Changes

After defining the states and actions of our MDP, we must move on to defining the uncertain regions, i.e., those regions where there are parameters θ that the simulator will estimate at run-time, and their possible evolutions. Therefore, just like in the case of states and actions, the goal is to define a language that allows us to outline the uncertain

areas, prior knowledge, equilibrium constraints and changes in transition functions.

We start by providing a template for a configuration line to specify the uncertain regions, prior knowledge and equilibrium constraints:

$$\textit{state id}; \textit{action id}; \textit{prior knowledge}; \textit{equilibrium constraints}; \quad (3.5)$$

- *state id*: a string that specifies the state to which this uncertain region refers.
- *action id*: a string that identifies the action relevant to the uncertain area being defined.
- *prior knowledge*: a series of integers with a length equal to the number of outcomes (each uncertain area must have two or more prior definitions within it; these definitions are to be understood as parameters of a Dirichlet prior density function) relative to the state-action combination, in the form:

$$\textit{value} [, \textit{value}]^+$$

- *equilibrium constraints*: a series of pairs of values that are also of length equal to the prior knowledge, defining the constraint intervals for each individual outcome. The values that compose them are floats ranging from 0.0 to 1.0. They are presented in the following form (the lower bound and upper bound, respectively):

$$\textit{value} - \textit{value} [, \textit{value} - \textit{value}]^+$$

As for both prior knowledge and equilibrium constraints, the values provided here will be assigned to the various outcomes in the same order as they were defined in the relative action. Let us see the following example to better understand the concept:

$$\begin{aligned} & s5; \textit{contact}; \textit{OBSERVABLE}; a, 0.3, s5; a, 0.15, s3; a, 0.55, s9; \\ & s5; a; 10, 20, 30; 0.1-0.9, 0.2-0.6, 0.3-0.8; \end{aligned}$$

- 10 and 0.1 – 0.9 are assigned to *s5*
- 20 and 0.2 – 0.6 are assigned to *s3*
- 30 and 0.3 – 0.8 are assigned to *s9*

Now we move on to the variations of the transition functions, providing two templates,

each related to a specific task:

$$\text{CHANGES } value: \quad (3.6)$$

$$state\ id; action\ id; probabilities; \quad (3.7)$$

The template (3.6) is used to define the step in which the change should be applied and consists of two values:

- *CHANGES*: a keyword used to indicate that a new change is being defined.
- *value*: an integer that defines the step number in which the change should occur.

The second template, (3.7), is used to define the actual variation to be applied and consists of three parameters:

- *state id*: a string containing the identifier of the state for which the change is being defined.
- *action id*: a string containing the identifier of the action whose outcome probabilities are to be modified.
- *probabilities*: a series of float values between 0.0 and 1.0, whose sum must be equal to 1.0, and whose length must correspond to the number of outcomes associated with the state-action combination (each change must have at least two probabilities, as they are intended to refer to uncertain areas, which have at least two outcomes). They are presented in the form:

$$value [, value]^+$$

Next, an example, defining areas of uncertainty and variations, is presented based on the same states, actions and transition functions described earlier:

PRIORS

s0; a; 50, 25, 25; 0.1-0.9, 0.1-0.6, 0.1-0.9;

s0; b; 50, 50; 0.2-0.7, 0.4-0.9;

s1; c; 50, 25, 25; 0.1-0.9, 0.1-0.6, 0.1-0.9;

s8; k; 25, 66; 0.1-0.3, 0.1-0.6;

CHANGES 500:

s0; a; 0.1, 0.1, 0.8;

s0; b; 0.77, 0.23;

CHANGES 1000:

s0; b; 0.95, 0.05;

s1; c; 0.01, 0.08, 0.91;

s8; k; 0.15, 0.85;

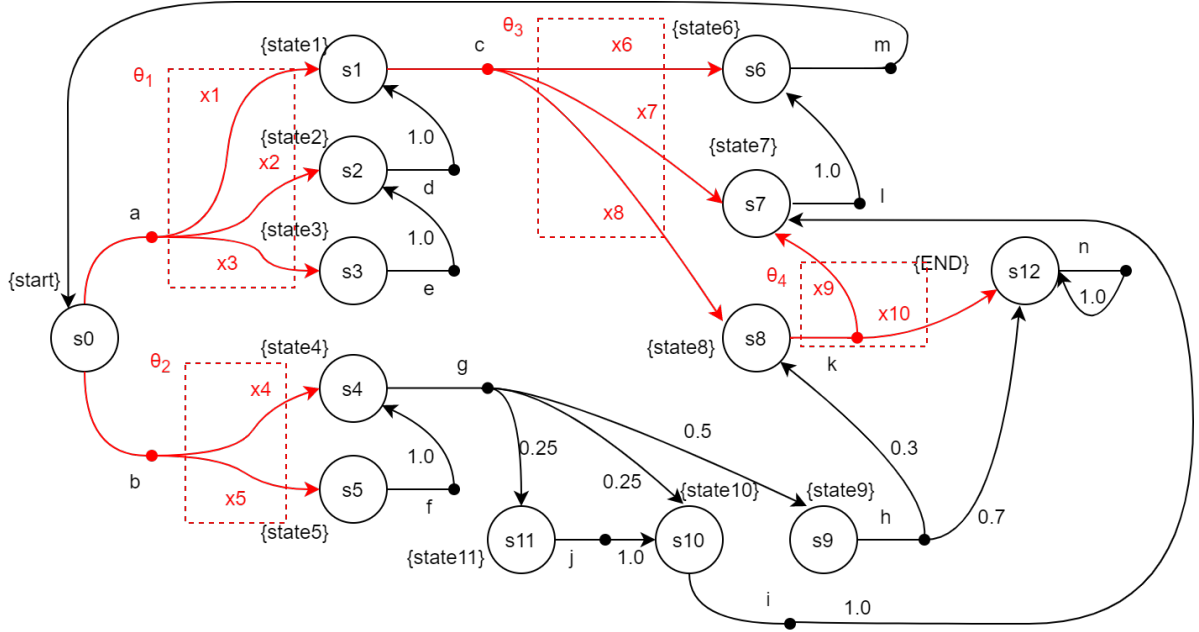


Figure 3.3: An example of MDP with the uncertain regions

Given these configurations, we will obtain four uncertain areas, namely θ_1 , θ_2 , θ_3 and θ_4 , and ten parameters to estimate at run-time, namely x_1, \dots, x_{10} . Additionally, changes will be applied to the previously defined transition functions (3.2) at steps 500 and 1000:

- first variation: x_1, x_2, x_3, x_4 and x_5
- second variation: $x_4, x_5, x_6, x_7, x_8, x_9$, and x_{10}

Semantic Space

Another useful tool to extend the functionalities of the pMDP is the Semantic space, which we defined in § 2.1.1 as the set of configuration and environment dimensions describing some internal and external quantities, like the battery percentage or the weather condition, which may have an impact on the uncertain regions.

Determining the semantic space (SS) means specifying the behavior of each of its single variables (SSV) on the various uncertain areas (not necessarily each SSV needs to operate

on every uncertain area), defining their initial values (if not defined, they will assume their default values) and their possible variations during the simulator execution.

Ignoring the implementation of variations and initial assignments of SSVs, which work in a very similar way to states and actions, we will focus on the more relevant part: modeling their behaviors and the application of their logic on the transition functions of the uncertain areas.

Each SSV is defined through a domain (int or float), an interval, a default value and a series of combinations. The latter specify, through state ids and action ids, the uncertain areas on which they will act along with the methods to apply. Each method can be seen as a function that takes as input a scalar (the current value of the SSV) and a vector (containing the values of the transition function of the action on which it is called) and returns the new transition function. Different SSV methods that act on the same uncertain area are applied consecutively and in the order of definition, with the first one starting from its own value and from the initial value of the transition functions of the uncertain area, and the subsequent ones starting from their own value and the result of the previously applied method.

3.2.2. Simulator Architecture

In this section we present the reasons that have motivated us to develop it both in the short and long term.

This simulator is not only a means to an end in completing our current research study but it is also intended to create a flexible and adaptable framework for future use and development. To achieve this goal, the “event-driven” design pattern has been implemented, which loosely couples the internal components of the simulator. In doing so, we aimed to create a system that is easy for future developers and users to approach and understand, as they may have different requirements and use cases.

Event-Driven

The Event-Driven design pattern is a software design approach that is based on the management of events, i.e. actions or notifications that can occur within the system. These events are exchanged between the various components of the system, and each component reacts specifically (actively or inactively) to each received event.

In addition to being useful in areas where there are many interactions between the various components of the system, this design pattern is particularly effective in situations where

the actions of various users or external systems cannot be predicted in advance. In these cases, the event management allows the system to be more flexible and adaptable to unforeseen situations, such as different future requirements and use cases.

In the Event-Driven design pattern, the components of the system are generally divided into two categories: “event producers,” i.e. components that generate events, and “event consumers,” i.e. components that react to received events. This division allows for the separation of responsibilities among the various components and ensures greater modularity of the system. In our case, we have one producer and multiple consumers.

To demonstrate the benefits of this approach, we provide a brief example with and without an event-driven architecture.

Suppose we have a security system for a building that has a single event producer (e.g. an alarm system) and three types of events that can be generated:

- Theft alarm
- Fire alarm
- Flood alarm

These events can be delivered to multiple consumers, each of which may react differently based on their own functionality. The first consumer could be an event logging system, which records all events generated by the producer without activating any specific response. The second consumer could be an evacuation system, which activates an audible alarm and sends notifications to personnel in case of a fire alarm. The third consumer could be a flood protection system, which automatically activates drainage pumps in case of a flood alarm.

In this way, the security system can handle different types of events generated by the producer and activate different responses based on the specific needs of the various consumers. Additionally, individual consumers do not necessarily have to intercept all notifications or respond to those of interest to them.

If the event-driven architecture is not used, a single central block should be implemented that is capable of recognizing and handling all three types of alarms (burglar, fire and flood) and activating appropriate responses based on the type of event detected.

Some of the main challenges of this approach include:

- Difficulty in future scalability: if the security system needs to be extended or customized in the future, the central block would need to be modified or replaced. This

could require a significant amount of time and resources.

- Complexity of event management: with a large number of events and actions, the code management quickly becomes complex and unclear, increasing the risk of programming errors.
- Inability to customize responses for each consumer: in a centralized system, all responses are controlled by the same central block, which means that it is not possible to customize responses for each consumer independently.

Event Producer

Returning to the simulator architecture, the event producer is the component responsible for executing the MDP steps: selecting actions to be taken, states to move to, computing policies, and updating transition functions and the semantic space. Essentially, it is designed to implement the basic logic of the MDP, expanding it to recalculate policies and control the equilibrium constraints (phases 3 and 4 of RENE are implemented in this module). Everything else is seen as an extension of it and therefore as external components that fulfill a specific task.

To achieve this result, i.e., the extension of the main component's functionalities, the following events are generated:

- `START_NEW_STEP` and `END_STEP`: triggered on the start or on the end of a step.
- `TERMINAL_STATE`: triggered when the MDP reaches a terminal state.
- `SELECT_NEXT_STATE` and `SELECTED_NEXT_STATE`: triggered before and after the state selection phase.
- `SELECT_NEXT_ACTION` and `SELECTED_NEXT_ACTION`: triggered respectively before and after the action selection phase.
- `PRIOR_ACTION_SELECTED`: triggered if the MDP chooses an action inside an uncertain region.
- `UPDATED_PRIOR_ACTION_SELECTED`: triggered if the inferences of the uncertain region selected are updated.
- `EQ_CONSTRAINT_UNSATISFIED`: triggered once the MDP lands inside an uncertain region whose equilibrium constraints are unsatisfied.
- `POLICY_UPDATED`: triggered after a policy re-computation.

- `END_CONDITION_SATISFIED`: triggered once the condition for the simulation end is satisfied.

Event Consumers

As just mentioned, all the extensions of the main component are intended to be event consumers. Before the start of the main component’s execution, they subscribe to one or more topics of interest (i.e., they prepare to intercept all events of a specific type). Once the execution of the main component begins, these consumers will synchronously react to all events (to which they have subscribed) that are produced.

With this mechanism, for the present work, two extensions have been implemented:

- **Monitor**: this component updates the estimates of uncertain areas when a specific threshold of passages is exceeded. That is, to avoid calculating all estimates (point and interval over long and short periods) related to an uncertain area for each new collected evidence, it waits to collect a batch before applying them all at once. The events on which it is based are two: *prior_action_selected* and *end_condition_satisfied*.
- **Log&Plot**: this component maintains a history of all the new estimates calculated for each uncertain area and, at the end of the execution, performs the plot of this data. The events on which it is based on are two: *updated_prior_action_selected* and *end_condition_satisfied*.

3.2.3. Execution Result

The simulator and its results are not the final goal of our proposal, but rather they can be seen as reusable building blocks, as further described in the next sections. Indeed, they are functional for the subsequent BMA and meta-heuristic search phase as they will allow us to simulate systems of interest, that is, build a dataset that links the configurations of the semantic space to the satisfaction of the requirements (in the next chapter, we will go into more detail).

More specifically, the products of the simulator are two:

- **The pMDP in its final state**: This final state (represented by the same class diagram in Figure 3.1) contains all the properties of the pMDP, such as states, actions, policy and the number of steps taken. Moreover, within all actions related to uncertain regions, we also have point and interval estimates, equilibrium constraints, rewards, and prior beliefs. This element allows us to conduct further post-mortem

investigations while maintaining maximum maneuverability and freedom in terms of consultable attributes. In our case, this is where we can trace the satisfaction of all individual requirements back to.

- The plots deriving from its execution: In each plot, all the computed estimates for each state-action-outcome combination for all uncertain areas are represented. An example plot is provided in Figure 3.4 where, given an action a , a starting state s_0 and an outcome s_2 , the evolutions of point and interval estimates, computed on both a longer and shorter timeline, are depicted, as well as the actual probability and the equilibrium constraint.

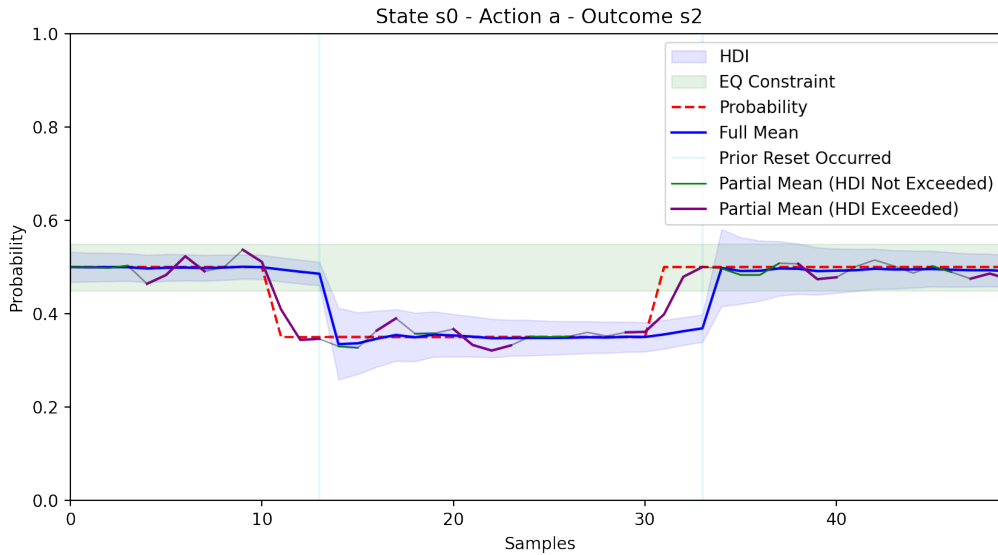


Figure 3.4: The plot for the state s_0 , action a and outcome s_2

3.3. BMA and Meta-Heuristic Search for Adaptation

After having seen in the previous chapter the implementation and functioning of the pMDP simulator, in this section we describe the approach adopted to support adaptation with the ultimate goal of keeping satisfied as many requirements as possible at run-time. To achieve this result, two fundamental building blocks are required in addition to the output of the simulator: BMA (introduced in chapter 1.4) and Meta-heuristic optimization search (introduced in chapter 1.5).

In this phase, we will recall the TUNE approach, already discussed in chapter 2.2, extending it with the simultaneous use of multiple models.

Starting from the output of the simulator, we will face in order the creation of a dataset, the training of multiple BMA models and solving optimization problems.

The dataset, whose features will be the configuration values of the semantic space, while the labels will indicate the satisfaction of each requirement, will be analyzed by the various models, each independently and separately, therefore generating as many results as requirements to be satisfied. With this forest of results, the meta-heuristic research problem can then be set.

The concatenation of these steps, each functional to the next and logical to the preceding one, will generate a set of Pareto-optimal solutions.

To illustrate this process more concretely, let us imagine a rescue robot. The robot needs to navigate through a disaster area and find survivors while avoiding obstacles and hazards. To accomplish this, we can gather data from previous disasters and create a dataset of various obstacles and hazards the robot may face, as well as the locations of survivors. Based on this data, we can create models that estimate the likelihood of encountering a particular obstacle or hazard, or the likelihood of finding a survivor in a given location. These models allow the robot to make more accurate decisions in real-time, maximizing the chances of success while minimizing risks to itself and any survivors it encounters. The estimates generated by these models can be expressed as:

- 0.0: indicating that the obstacle or hazard is present and should be avoided.
- 1.0: indicating that a survivor is present and should be located immediately.
- intermediate values indicating the level of risk or likelihood of success with some uncertainty.

Using these estimates, we can set an optimization problem whose goal is to find the safest and most efficient path for the robot to navigate through the disaster area, while maximizing the chances of finding survivors and avoiding obstacles and hazards.

3.3.1. Dataset

As mentioned earlier, to build the dataset the pMDP simulator has to be run multiple times, each time using different configurations of the semantic space. The elements of the resulting dataset correspond to each individual simulation run: the features consist of the configurations of the semantic space used, and the labels indicate the degree of satisfaction of the requirements associated with that configuration. The construction of the dataset is crucial for the subsequent training phase of the BMA models, as it allows us

to use a representative set of data from the semantic space we are interested in exploring.

As an example, in the following table we present the semantic space variables relative to the pMDP illustrated in Figure 3.5:

	space	type	domain
cruise speed	configuration	continuous	[0.0, 5.0]
power	configuration	discrete	[0, 100]
bandwidth	configuration	continuous	[10.0, 50.0]
quality	configuration	discrete	[0, 2]
illuminance	environment	continuous	[40.0, 120000.0]
smoke intensity	environment	discrete	[0, 2]
obstacle size	environment	continuous	[0.0, 120.0]
obstacle distance	environment	continuous	[0.0, 10.0]
firm obstacle	environment	discrete	[0, 1]

Table 3.1: Semantic Space Variables used in the simulation.

In this example, out of these nine variables, four are controllable and five are not. As for their sampling, a random distribution can be chosen, meaning that the assignments do not follow any coherent logic, with the single exception of avoiding duplicates.

Operationally, the dataset builder consists of two phases: in the first phase, all combinations of the semantic space to be executed are generated in a single thread; in the second phase, these combinations are divided into several processes for the actual execution of the simulator. The goal of this approach is to ensure a duplicate-free creation of configurations and their fast execution, making the most out of parallel computation.

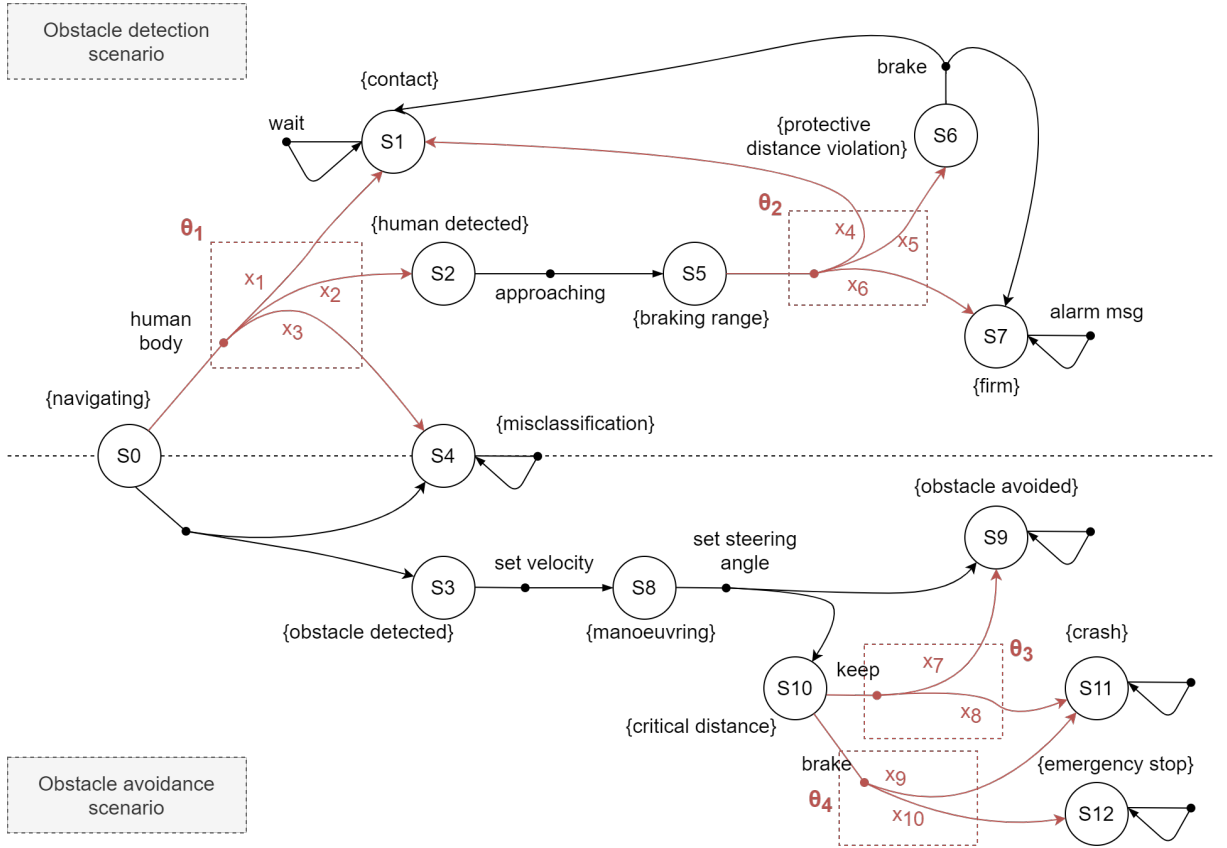


Figure 3.5: Rescue Robot pMDP [9]

3.3.2. BMA

Once the dataset has been fully generated, the next step is to obtain the models, each of which focuses on a unique requirement. This involves producing models capable of predicting whether a requirement will be satisfied given a specific configuration of the semantic space.

In TUNE, BMA is used as a method for model selection and uncertainty quantification. It is used to combine the predictions of different models and generate an improved prediction with more accurate uncertainty estimates. BMA assigns weights to each model based on their ability to fit the data and produce accurate predictions. The combination of these models allows for a more robust prediction while also accounting for the inherent uncertainty in the data.

However, compared to TUNE, we introduce a new approach to model selection and uncertainty quantification through the use of multiple BMA models. This is because we aim to find a new configuration that can satisfy as many requirements as possible, rather than just one.

Model Creation and Fit

When defining a new model, the only parameters needed are the response variable y and the predictor variables X : in our case the response variable is the satisfaction of a requirement, hence a *True* or *False* value, and the predictor variables are the parameters of the Semantic Space. It is worth noting that the X parameter has been normalized using standard scaler, which ensures that all predictor variables are on the same scale and thus have equal importance in the model.

The fit method performs the actual model selection using BMA starting by computing a normalization denominator in Bayes Theorem called the likelihood sum. It then iterates through all possible models of varying sizes, starting with models that have only one predictor variable and increasing the number of variables for each iteration.

For each model, it computes the regression using the selected predictor variables and computes the likelihood of the model by multiplying the prior probability of each variable with the likelihood of the model. The prior probabilities are provided as input using `**kwargs` and are assumed to be uniform if not provided.

Subsequently, as a criterion for selecting the best models in a way that balances between model complexity and model fit, Occam's window is applied: only models whose likelihood is greater than $1/20th$ of the maximum likelihood among all models are considered in the next iteration.

For each selected model, the likelihood sum is updated and the likelihood of each variable in the model is computed by adding the likelihood of the model to the running tally for each variable. Moreover, also the coefficients of the regression equation are computed for each variable in the model and the running tally is updated for each coefficient.

Eventually, the probabilities and coefficients are normalized by dividing each by the likelihood sum to ensure that the probabilities sum to one.

The result of fitting the model is an object capable of making predictions on new data: each prediction yields a value between 0.0 and 1.0, which respectively mean requirement not satisfied and requirement satisfied. It's important to note that the output values represent the probability of a requirement being satisfied, based on the predictor variables provided as input. Therefore, a prediction of 0.8, for example, means that the model estimates an 80% chance of the requirement being satisfied given the input values. Additionally, the output can be interpreted using a threshold value: for example, if the threshold is set to 0.5, any prediction value greater than or equal to 0.5 would be classified as requirement satisfied and any value less than 0.5 would be classified as requirement

not satisfied.

3.3.3. Meta-Heuristic Search

Once BMA models have been obtained, they can be used to tackle optimization problems through a technique known as Meta-heuristic search. This search method is based on a set of heuristics that enable efficient exploration of the solution space in order to find the one that maximizes or minimizes a certain objective function. In practice, heuristic meta-search offers an interesting alternative to traditional optimization methods, especially in complex contexts or when available information is incomplete or uncertain.

The second phase of TUNE is precisely aimed at implementing meta-heuristic search. This optimization procedure is carried out through a series of iterations, each of which involves selecting a point in the variable domain (a new configuration of the semantic space) and evaluating the values of the objective functions at that position (the BMA model prediction and the cost to set that new configuration). This procedure continues until a certain stopping criterion is met.

Our approach builds on this implementation by expanding the number of models to be considered during the evaluation phase: we will no longer rely on a single prediction provided by a single BMA model, but rather, as many BMA models as there are requirements to be satisfied. Consequently, our optimization problem will be of type many-objective.

The result of this meta-heuristic search will be a set of Pareto-Optimal Solutions, i.e., a set of solutions, each representing a possible configuration of the SS, in which none can improve one objective variable without worsening at least another one. Therefore, there is no solution that can be globally considered the best, but only a set of solutions that are all equally valid and reasonable.

There are several options available in Python to tackle optimization problems, especially those with multiple objectives (many-objective optimization problems). Some common libraries include Pyomo, PuLP, PyMOO, SciPy, and DEAP. Moreover, there is also a wide selection of specialized libraries for multi-objective problems, such as MOEA Framework and PlatEMO, which offer a wide range of meta-heuristic search techniques and evolutionary algorithms.

We chose to use PyMOO (Multi-objective Optimization in Python): a relatively new but very powerful and efficient library for solving multi-objective, as well as many-objective, optimization problems. The library offers diverse algorithms and search techniques, including evolutionary optimization, local search and swarm-based optimization.

Problem Definition and Solution

To define a many-objective problem, we start by creating a class that inherits from the *Problem* class and then implements the following methods:

1. `__init__(self)`: initializes the class and defines the problem parameters, such as the number of decision variables, the lower and upper bounds of the decision variables and the objective functions.
2. `_evaluate(self, x, out, *args, **kwargs)`: computes the value of the objective functions given the decision variables x . This method should set the values of the objective functions in the *out* parameter.

In our case, if we had six requirements to satisfy and nine variables in the semantic space, with only four variables being modifiable, we will have the following definitions inside the initialization (*init*):

- Number of variables (together with their bounds): 4 (only the modifiable variables)
- Number of objective functions: 7 (six models, one for each requirement, plus the cost function)

In the evaluation (*evaluate*) method, we receive an input array x of 4 elements (our decision variables) and we have to return all the results of our 7 objective functions in the *out* vector.

After defining the class that describes the problem, we need to define the algorithm we intend to use to solve it. In our proposal, we use NSGA-III, which requires:

- *ref_dirs*: represents the set of reference directions for the NSGA-III algorithm. To calculate this value, we use “Das-dennis,” a method for generating points uniformly on the hypersphere.
- *pop_size*: indicates the number of individuals in the population.
- *sampling*: indicates the sampling strategy of the points to be tested.
- *mating* and *eliminate_duplicates*: define the mating strategy for generating offspring and the elimination of duplicates.

The resulting Pareto-optimal solutions are obtained by specifying whether the problem is a maximization or minimization problem. In our case, it is a minimization problem, and the parameters are:

- *problem*: the problem that we want to solve.

- *algorithm*: the algorithm that we want to use to explore the space of possible solutions.
- *n_gen*: indicates the maximum number of generations that will be executed by the algorithm before stopping. A generation is a cycle of execution of the optimization algorithm that produces a new population (set of solutions) from the current population, using selection, crossover and mutation operators. Generally, the larger the number of generations, the better the optimization results, but it requires more time.
- *seed*: seed for the random number generator. Setting it to a specific value guarantees the repeatability of the results.

4 | Evaluation

The evaluation examination, a crucial moment to verify the effectiveness and efficiency of our study, as we shall see, has ensured - through an objective and quantitative assessment of the system's performance - that our proposal is capable of achieving the predetermined objectives.

In our case, we decided to divide the evaluation into two sections, as many as the structure of the work: the first section focused on the main aspects of the pMDP simulator (§ 4.2 in relation to §3.2), while the second section focused on the analysis of BMA and Meta-Heuristic Optimizing Search (§ 4.3 in relation to §3.3).

To ensure the robustness and consistency of the results, we performed the tests multiple times (50) for each individual situation. The test environment used had the following characteristics:

- Processor: Intel® Core™ i9-9900, 16M Cache, up to 5.00 GHz, 8 cores and 16 threads.
- RAM: 32GB at 3000MHz DDR4, Dual Channel (2x16GB)
- Interpreter: Python 3.11

4.1. Research Questions

Before delving into the individual results, it is essential to introduce the research questions that define the objectives we aim to achieve. In fact, focusing on the Simulator, we will inquire whether our approach can recognize violations in the equilibrium constraints and to what extent it is scalable and effective in this regard. We will investigate whether the application of a Policy can affect its ability to avoid disequilibrium. Moving on to the BMA, we will ask how much more accurate a BMA model is in making estimations as compared to simple logistic models. Eventually, we will evaluate how much more efficient a meta-heuristic approach is in mitigating violations of the equilibrium constraints as compared to a random approach.

Simulator

RQ1 How capable is our approach in detecting violations in the equilibrium constraints?

RQ2 How effective and scalable is the detection process?

RQ3 How effective is the usage of a Policy in avoiding disequilibria?

BMA and Meta-Heuristic Adaptation

RQ4 What is the accuracy of the BMA estimates?

RQ5 What is the cost of calculating the BMA models to mitigate the model uncertainty?

RQ6 How effective is the meta-heuristic optimizing search in computing new configurations of the semantic space in order to restore the equilibrium constraints?

4.2. Simulator

In this section, we will present the results of the tests carried out on the pMDP simulator we developed. These tests focus on evaluating the simulator’s ability to detect changes in the external environment using the adaptive observation mechanism and to apply a self-adaptation policy capable of neutralizing disturbances that may occur in the environment.

4.2.1. Evaluation Design

To answer RQ1-RQ3, we decided to conduct a series of controlled experiments using two carefully selected pMDPs. In particular, the first one (Figure 4.1) has only one action (a), one uncertain area (θ_1) associated with it, and two possible outcomes ($s1$, $s2$), while the second one (Figure 4.2) has 10 actions, each with only two possible outcomes, and 10 uncertain areas associated with them. Indeed, in both cases, the purpose of the test is not the complexity of the pMDP in terms of number of states, actions, and uncertain areas, but rather the behavior of individual uncertain areas in the analyzed cases.

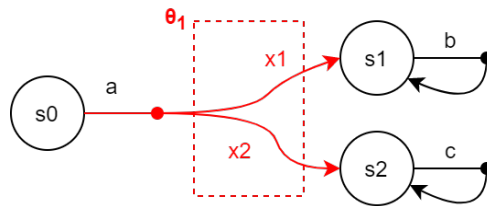


Figure 4.1: Graph representing the first pMDP

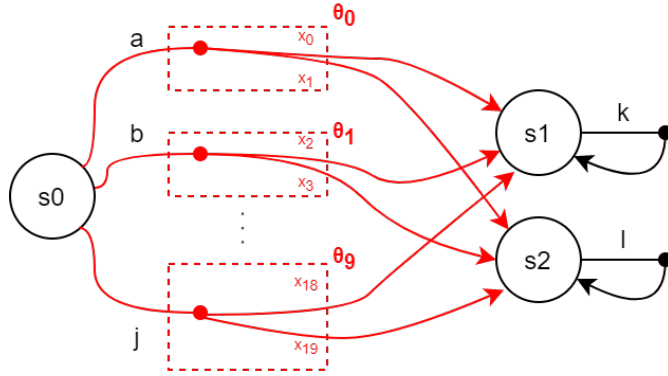


Figure 4.2: Graph representing the second pMDP

The different factors we manipulated during the execution of our tests are the x_i parameters of the individual uncertain areas, the use or not of the adaptive observation aging mechanism, and the alternation of two policies (*Random* and *Policy Iteration*) in order to verify the simulator’s ability to estimate the new values of the x_i parameters with respect to the changes made. When manipulating the parameters of the uncertain areas, we acted on magnitude (i.e., the external positive increment outside the equilibrium constraint boundaries) and duration (i.e., length of the perturbation in terms of the number of observations).

4.2.2. RQ1: Violation Detection Ability

In response to RQ1, we focused on the violation detection aspect of the simulator, studying its behavior with and without the adaptive observation aging mechanism in a pMDP consisting of a single action and 3 states, as shown in Figure 4.1. The objective was to understand whether, in the presence of perturbations of the uncertain area parameters, the system could recognize the variations and update its beliefs to reflect the new external conditions.

To achieve this, we conducted tests modifying the values of the uncertain area parameters θ_1 , injecting perturbations of varying magnitude and duration. The scenarios we considered are four, generated by alternating between short and long duration, small and large magnitude:

- substantial magnitude and unbounded duration
- small magnitude and unbounded duration
- substantial magnitude and short duration

- small magnitude and short duration

The test results showed a clear uncertainty in recognizing and adapting to changes in the case of the simulator **without the adaptive observation aging mechanism**: based on what is shown in Figures 4.3 and 4.4, it is evident that the presence of very old historical data negatively affects the ability to recognize the violation. In fact, in these two figures, the interval and pointwise inference approach the real value of the parameter x_1 very slowly (while the goal is to recognize the presence of a violation as soon as possible). In practice, although violations are “sensed” (in fact, in the graphs, the point estimate *tends* in the direction of the violation), reaching the real position is delayed to such an extent that the test becomes useless to conduct in the most difficult scenarios. In the following graphs, we have represented the point estimate processed and defined by the simulator taking into account *all* previous observations (blue line), and the real situation, to which the estimate must tend, represented by the dashed red line. In the first case (Figure 4.3), given a contained violation, not only does the simulator fail to reach the value of the violation (although it perceives the change), but it also fails to return to the real situation, even though it was encountered in the past. In the second case (Figure 4.4) we simulated a determined and infinite violation: in this case, the simulator only keeps on approach the real situation (like Achilles with the turtle).

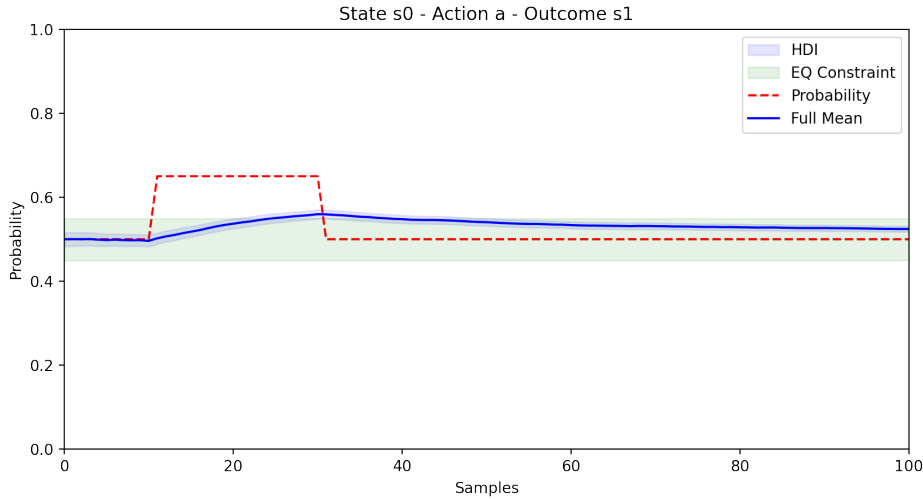


Figure 4.3: Simulation with $m=0.1$ and $d=20$ without using the adaptive observation aging mechanism

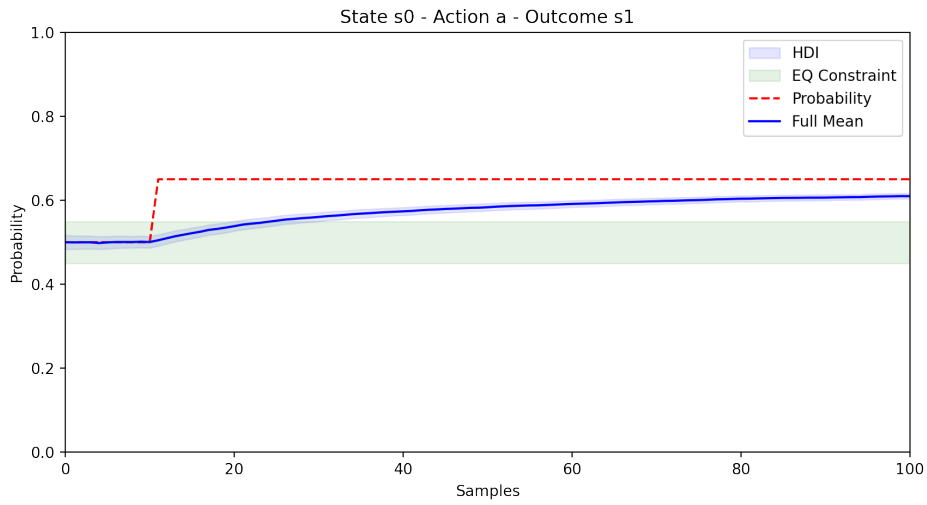


Figure 4.4: Simulation with $m=0.1$, $d=\infty$ and without using the adaptive observation aging mechanism.

The opposite effect is observed in the case of the simulator that utilizes the adaptive observation aging mechanism. As depicted in Figure 4.5, which represents the most critical scenario amongst the four considered, the simulator is able to estimate the new parameter value within a few observations.

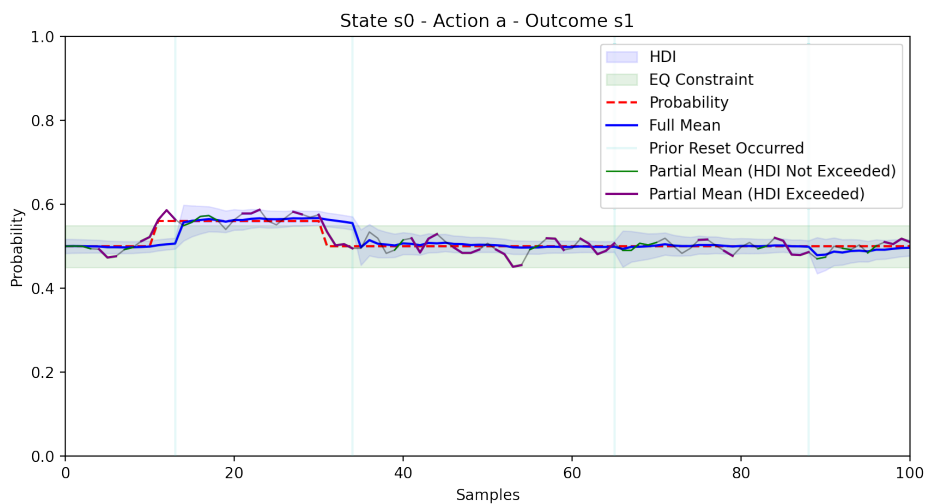


Figure 4.5: Simulation with $m=0.01$, $d=20$ using the adaptive observation aging mechanism

For a more in-depth analysis of the simulator's behavior, we represented the punctual estimate elaborated and defined by the simulator considering a determined historical

number of previous observations (blue line), i.e., the long-term historical timeline, and the punctual estimate elaborated and defined by the simulator considering a very limited historical number of previous observations (multi-color line), i.e., the short-term historical timeline. The actual situation, that the estimate must aim for, is always depicted by the dashed red line.

The second line (multicolor), determined by the short-term historical data, is very sensitive to noise and often deviates from the actual value, albeit remaining very close to it. Hence, when the simulator, according to the setup we have defined after several tests, detects 5 estimations that are too inconsistent between the two timelines, the long-term historical data gets flushed and, as a consequence, the blue line shifts to the last estimate of the shorter timeline, bringing it closer to the red line. Observing Figure 4.5, we note that the punctual estimate (blue) is almost superimposable on the actual value, except during the moment of its variation with a tolerable adjustment interval (reaction time).

The validity of our conclusions is confirmed by the tests performed on all the scenarios considered. In the following figures, it will be noted how the blue line reasonably and reliably follows the red line during infinite or short, large or small perturbations.

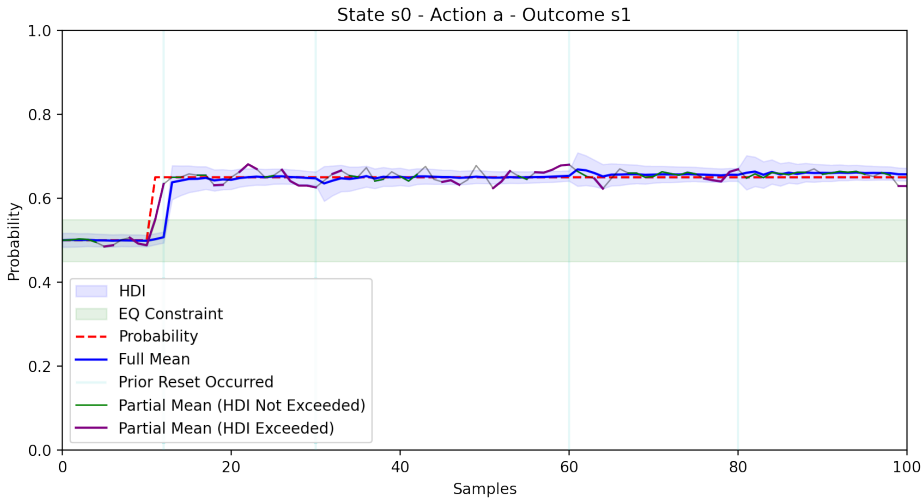


Figure 4.6: $m=0.1$, $d=\infty$ using the adaptive observation aging mechanism

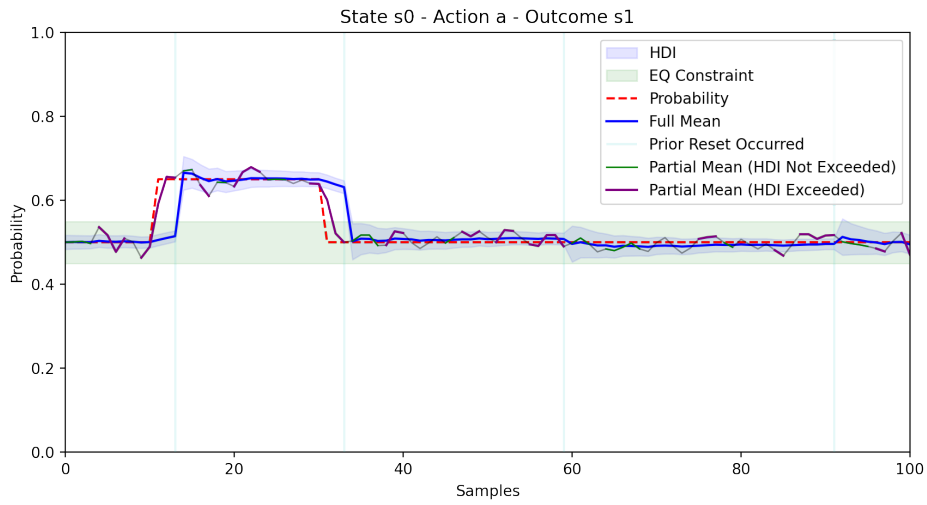


Figure 4.7: $m=0.1$, $d=20$ using the adaptive observation aging mechanism

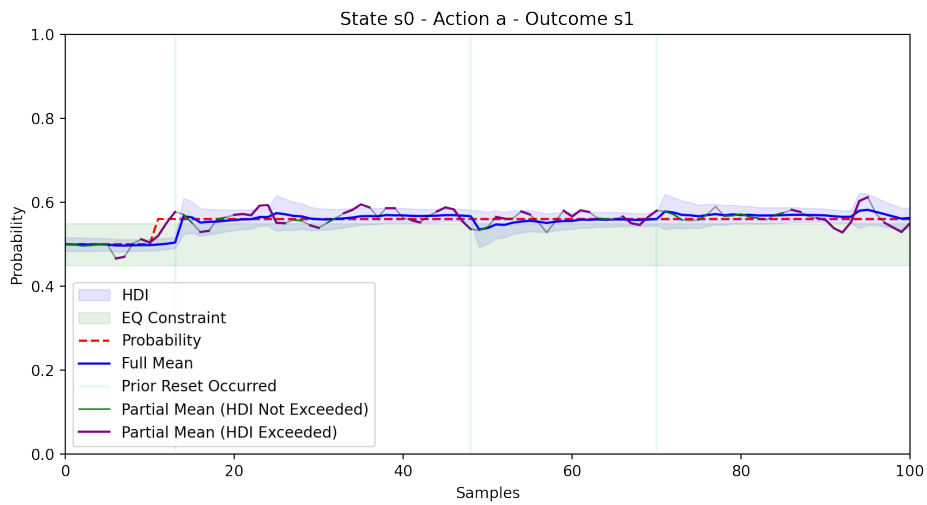


Figure 4.8: $m=0.01$, $d=\infty$ using the adaptive observation aging mechanism

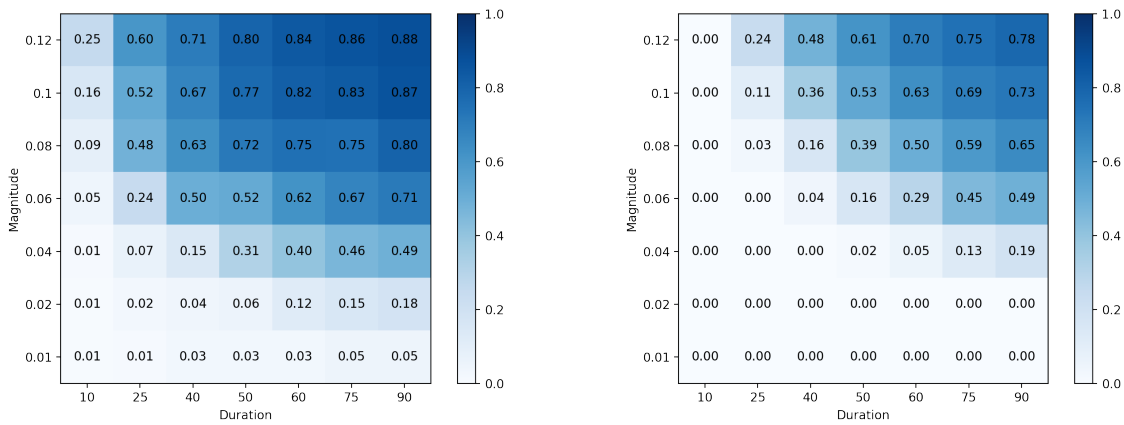
4.2.3. RQ2: Violation Detection Effectiveness

In the previous section, while answering RQ1, we observed that our approach recognizes changes in uncertain areas' parameters and provides consistent estimates, i.e., estimates that are very close to the real values. Now, we aim to quantify the simulator's ability to detect a violation of the equilibrium constraints by applying a range of values. The range of magnitude varies from 0.01 to 0.12, and the duration varies from 10 to 90.

Therefore, the question we ask is "how effective and scalable is the detection process?"

The metric we used measures the percentage of recognized violations out of total violations, i.e., the number of observations where a violation is detected over the number of all observations made during the perturbation.

The following heatmaps display the results of multiple violations as described in the previous paragraph. The two values under consideration (magnitude and duration) were varied. As it can be seen, each heatmap cell contains a detection rate value that is higher the more the violation is recognized. (For convenience and immediacy, the cells were colored with gradual shades of blue.)

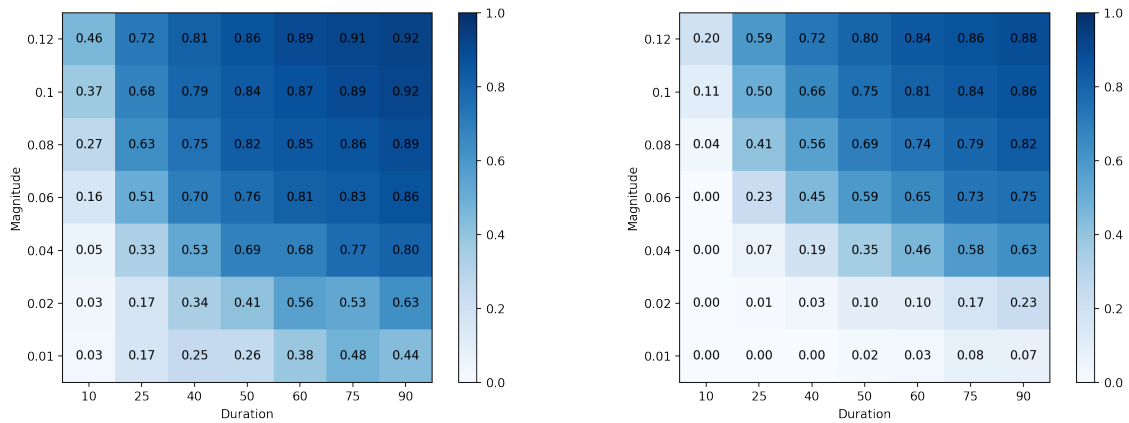


(a) With the adaptive observation aging mechanism.

(b) Without the adaptive observation aging mechanism.

Figure 4.9: Effectiveness of the Interval Estimates

The heatmaps shown in Figure 4.9 pertain to the interval estimation - which encodes a credible set of intervals, providing additional information by taking into account the notion of confidence (refer to § 1.3) - and confirm what was previously stated in the paragraph above: the heatmap in Figure 4.9a, which implements the the adaptive observation aging mechanism, exhibits a more accurate recognition of disturbances compared to the heatmap in Figure 4.9b, which does *not* implement the adaptive observation aging mechanism. The result is easily noticeable visually where the darker shaded cells assign greater recognition accuracy. It should be noted that the lighter cells refer to minimal variations in magnitude and time, whose deviation, even if imperceptible, is tolerable. By observing the heatmap, it can be concluded that magnitude is a more relevant and decisive element compared to time: a minimal magnitude, even if sustained over a long period, can be overlooked as it is less influential than a significant magnitude, even if limited in time, and as such, recognized by the simulator.



(a) With the adaptive observation aging mechanism.

(b) Without the adaptive observation aging mechanism.

Figure 4.10: Effectiveness of the Point Estimates

The heatmaps in Figure 4.10, which do not incorporate confidence (as discussed in § 1.3), refer to **point estimates**, and they recall the previous considerations regarding the greater reliability of the usage of the adaptive observation aging mechanism. The simple visual impact is proof of this, where, as we recall, a greater intensity of blue shading denotes a greater recognition capability.

Comparing now the previous heatmaps grouped not by estimate type but by the use of the adaptive observation aging mechanism, the visual impact (shading) could lead us to believe that point estimates are also more sensitive to small and short-term perturbations than interval estimates. However, in reality, relying on point estimates can give apparent, unreliable or even misleading results: in this case we speak of “false positives”. Consider the following representations:

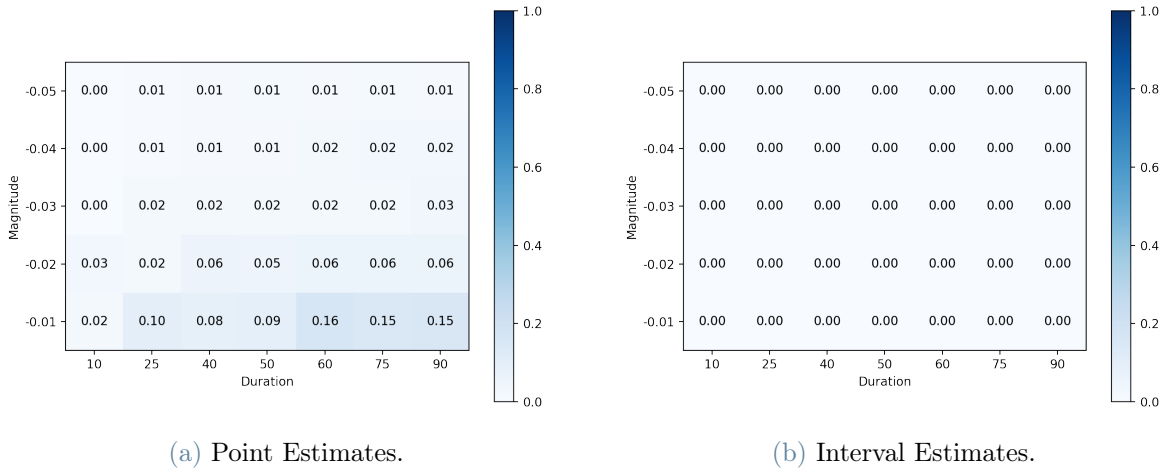


Figure 4.11: False Positives detection.

In Figure 4.11, we have assumed variations in magnitude *within* the equilibrium constraints, which, as we have discussed so far, do not represent violations of the same. Therefore, the only acceptable and correct result can only be 0.00.

This is found in all cells of the interval estimates heatmap (Figure 4.11b). This means that the simulator correctly does not recognize any violations; in other words, it considers the perturbation as a non-violation and therefore irrelevant. Obviously, this behavior assures the correct functioning of the simulator.

Conversely, with point estimates (Figure 4.11a), the simulator behaves by recognizing a perturbation within the equilibrium constraints as a violation of the equilibrium constraints. This is a contradiction.

Therefore, since the result of the point estimate test falsely indicates the presence of a positive result, i.e. the presence of a violation, where it is not actually present, we speak in this case of “false positives”. The consequences are obviously intuitive, as this error leads to an incorrect conclusion. Conversely, it is important to minimize false positives as they can lead to wrong decisions. In our case, this means choosing interval estimates.

4.2.4. RQ3: The Effectiveness of Policies in Preventing Disequilibria

The analysis conducted so far aimed to identify actions that violate equilibrium constraints. Having defined them, we now aim to investigate how the use of a policy can help us avoid these disequilibria. As a reminder, a policy is a function that specifies which

action the agent should take in each state to maximize its expected cumulative reward (see §1.2 sub *Policy*). Therefore, the reward is to avoid disequilibria.

The pMDP used in the following tests is the one represented in Figure 4.2, on which we introduced perturbations by manipulating the parameters of the uncertain regions θ_1 - θ_9 , or rather, the parameters x_2 - x_{19} . The purpose of the test is to verify the simulator's ability to recognize the violations introduced in the uncertain regions θ_1 - θ_9 and to favor the selection of the action related to the uncertain area θ_0 .

So, the question is: *How effective is the usage of a policy in avoiding disequilibria?*

The metric we used here identifies the percentage of actions pursued outside of equilibrium compared to the range of choices made. A lower value denotes a greater ability to avoid actions in disequilibrium.

The sum of the results of the tests carried out leads us to the conclusion that the use of a policy (in our case, PolicyIteration) limits this percentage value.

Observing the heatmap With Policy (Figure 4.12a), lower values indicate lower chances of pursuing actions outside of equilibrium. Of the two quantities considered, if it is logical to expect that the greater the magnitude, the greater the sensitivity of recognition, it is noted how the duration is the determining factor: its increase entails evident reductions in the rate of violations. This means that the duration of the perturbation is an important factor to consider in the evaluation of the probability of performing actions in disequilibrium, even more so than magnitude. In other words, the longer the perturbation duration, the lower the chances of performing actions outside of equilibrium, regardless of the variation itself.

In general, the figure suggests that the longer the perturbation duration, the lower the chances of selecting actions outside of equilibrium, even at relatively low magnitude. In fact, if the action has a too short duration, even a high magnitude might not be enough to recognize and therefore carry out actions in equilibrium. Note how the heatmap in Figure 4.12a behaves like the heatmap in Figure 4.12b in the case of a short-duration perturbation: 0.90 results from the probability of randomly selecting one of the 9 actions out of 10 in disequilibrium.

Thus, we can unequivocally assert that the utilization of a policy is pivotal in the event of perturbations.

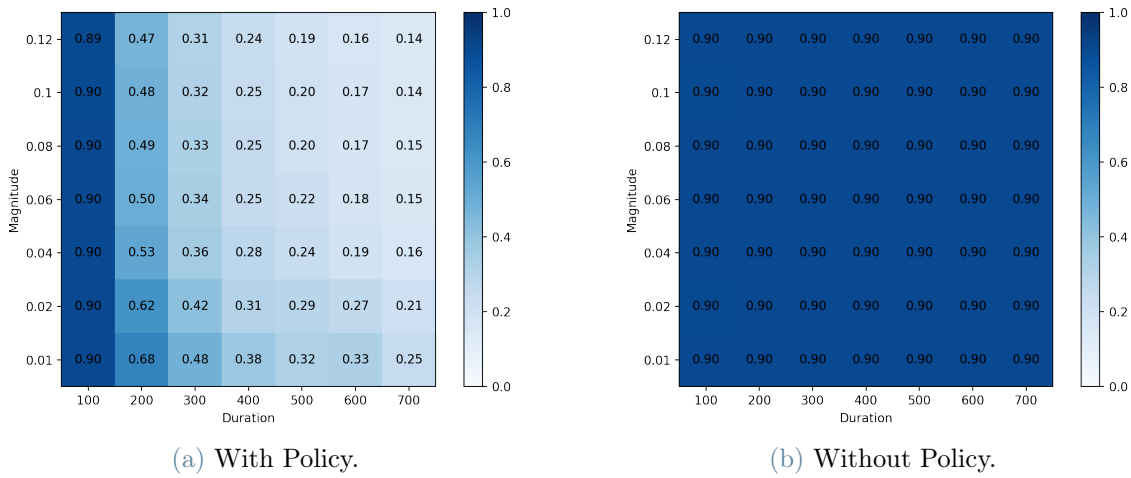


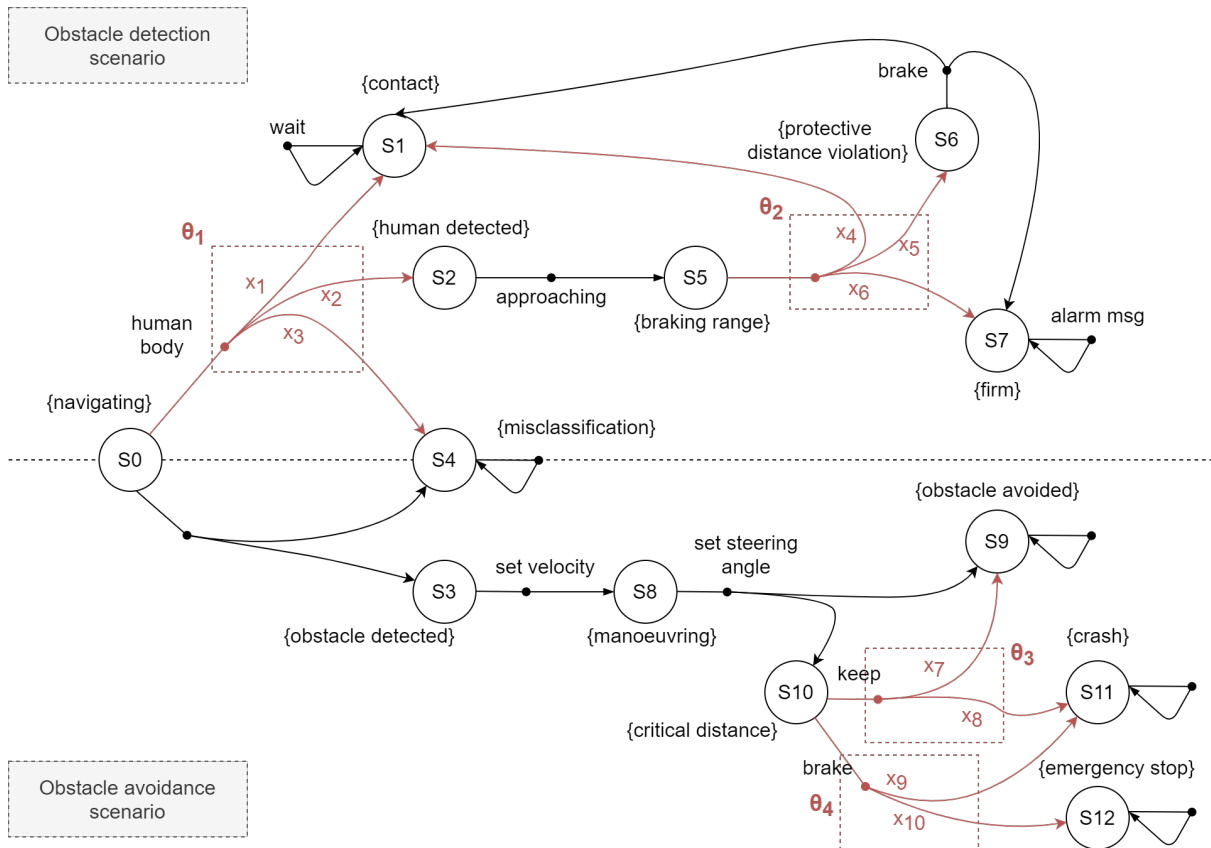
Figure 4.12: Violation rate.

4.3. BMA for Adaptation

In this section, we will present the outcomes of the tests performed on the output yielded by the pMDP simulator we developed. These tests are centered on evaluating the capability of BMA to estimate the satisfaction of requirements, starting from a Semantic Space configuration, and the capability of the meta-heuristic search approach to discover a new Semantic Space configuration, satisfying as many requirements as possible, when commencing with a configuration that does not satisfy at least one requirement.

4.3.1. Design of the Evaluation

To address the inquiries of RQ4, RQ5 and RQ6, we decided to conduct a series of controlled experiments utilizing the Rescue Robot pMDP (which we illustrated in Figure 3.5, included below).



While, in the previous chapter, we focused on analyzing cases with low complexity (few states, actions, and uncertain areas) because of our interest in the internal mechanics of a single uncertain area, the aim here is to observe what occurs when various states, actions, and uncertain areas exist simultaneously.

The factors that we modified during the execution of our tests were the parameters of the Semantic Space which were presented in Table 3.1, included below.

	space	type	domain
cruise speed	configuration	continuous	[0.0, 5.0]
power	configuration	discrete	[0, 100]
bandwidth	configuration	continuous	[10.0, 50.0]
quality	configuration	discrete	[0, 2]
illuminance	environment	continuous	[40.0, 120000.0]
smoke intensity	environment	discrete	[0, 2]
obstacle size	environment	continuous	[0.0, 120.0]
obstacle distance	environment	continuous	[0.0, 10.0]
firm obstacle	environment	discrete	[0, 1]

Furthermore, 10 synthetic requirements were employed for all tests, as defined below:

uncertain region	equilibrium constraints
θ_1	$x_1 \in [0.0, 0.075]$
	$x_2 \in [0.825, 1.0]$
	$x_3 \in [0.0, 0.125]$

Table 4.1: Requirement 1

uncertain region	equilibrium constraints
θ_2	$x_4 \in [0.0, 0.075]$
	$x_5 \in [0.0, 0.125]$
	$x_6 \in [0.825, 1.0]$

Table 4.2: Requirement 2

uncertain region	equilibrium constraints
θ_1	$x_1 \in [0.0, 0.075]$
	$x_2 \in [0.825, 1.0]$
	$x_3 \in [0.0, 0.125]$
θ_2	$x_4 \in [0.0, 0.075]$
	$x_5 \in [0.0, 0.125]$
	$x_6 \in [0.825, 1.0]$

Table 4.3: Requirement 3

uncertain region	equilibrium constraints
θ_2	$x_4 \in [0.0, 0.075]$
	$x_5 \in [0.0, 0.125]$
	$x_6 \in [0.825, 1.0]$
θ_4	$x_9 \in [0.0, 0.125]$
	$x_{10} \in [0.875, 1.0]$

Table 4.4: Requirement 4

uncertain region	equilibrium constraints
θ_1	$x_1 \in [0.0, 0.075]$
	$x_2 \in [0.825, 1.0]$
	$x_3 \in [0.0, 0.125]$
θ_2	$x_4 \in [0.0, 0.075]$
	$x_5 \in [0.0, 0.125]$
	$x_6 \in [0.825, 1.0]$
θ_4	$x_9 \in [0.0, 0.125]$
	$x_{10} \in [0.875, 1.0]$

Table 4.5: Requirement 5

uncertain region	equilibrium constraints
θ_2	$x_4 \in [0.0, 0.075]$ $x_5 \in [0.0, 0.125]$ $x_6 \in [0.825, 1.0]$
θ_3	$x_7 \in [0.875, 1.0]$ $x_8 \in [0.0, 0.125]$
θ_4	$x_9 \in [0.0, 0.125]$ $x_{10} \in [0.875, 1.0]$

Table 4.6: Requirement 6

uncertain region	equilibrium constraints
θ_1	$x_1 \in [0.0, 0.075]$ $x_2 \in [0.825, 1.0]$ $x_3 \in [0.0, 0.125]$
θ_2	$x_4 \in [0.0, 0.075]$ $x_5 \in [0.0, 0.125]$ $x_6 \in [0.825, 1.0]$
θ_3	$x_7 \in [0.875, 1.0]$ $x_8 \in [0.0, 0.125]$
θ_4	$x_9 \in [0.0, 0.125]$ $x_{10} \in [0.875, 1.0]$

Table 4.7: Requirement 7

uncertain region	equilibrium constraints
θ_1	$x_1 \in [0.0, 0.08]$ $x_2 \in [0.82, 1.0]$ $x_3 \in [0.0, 0.13]$
θ_2	$x_4 \in [0.0, 0.08]$ $x_5 \in [0.0, 0.13]$ $x_6 \in [0.82, 1.0]$
θ_3	$x_7 \in [0.87, 1.0]$ $x_8 \in [0.0, 0.13]$
θ_4	$x_9 \in [0.0, 0.13]$ $x_{10} \in [0.87, 1.0]$

Table 4.8: Requirement 8

uncertain region	equilibrium constraints
θ_1	$x_1 \in [0.0, 0.09]$ $x_2 \in [0.81, 1.0]$ $x_3 \in [0.0, 0.14]$
θ_2	$x_4 \in [0.0, 0.09]$ $x_5 \in [0.0, 0.14]$ $x_6 \in [0.8, 1.0]$
θ_3	$x_7 \in [0.86, 1.0]$ $x_8 \in [0.0, 0.14]$
θ_4	$x_9 \in [0.0, 0.14]$ $x_{10} \in [0.86, 1.0]$

Table 4.9: Requirement 9

uncertain region	equilibrium constraints
θ_1	$x_1 \in [0.0, 0.095]$
	$x_2 \in [0.805, 1.0]$
	$x_3 \in [0.0, 0.145]$
θ_2	$x_4 \in [0.0, 0.095]$
	$x_5 \in [0.0, 0.145]$
	$x_6 \in [0.805, 1.0]$
θ_3	$x_7 \in [0.855, 1.0]$
	$x_8 \in [0.0, 0.145]$
θ_4	$x_9 \in [0.0, 0.145]$
	$x_{10} \in [0.855, 1.0]$

Table 4.10: Requirement 10

4.3.2. RQ4: Assessing the Accuracy of BMA Estimates

Starting from a dataset generated by the simulator output (as described in our proposal in §3.3.1), we aim to comprehend and quantify the potential of BMA in improving inference on new data in comparison to individual Logit models. The latter, the total of which is 511, represent all logistic models that can be considered and grouped by the number of variables they consider.

In summary: *What is the accuracy of BMA estimates in comparison to the Logit models?*

For our purpose, we have created a dataset of 20,000 samples whose features are the semantic space configuration parameters, while the labels represent the satisfaction of a single requirement. Subsequently, we divided the dataset into two sets, namely a training set (80%) and a test set (20%), using stratified sampling on the labels, to respectively execute the two corresponding phases. The results below are derived from the testing phase.

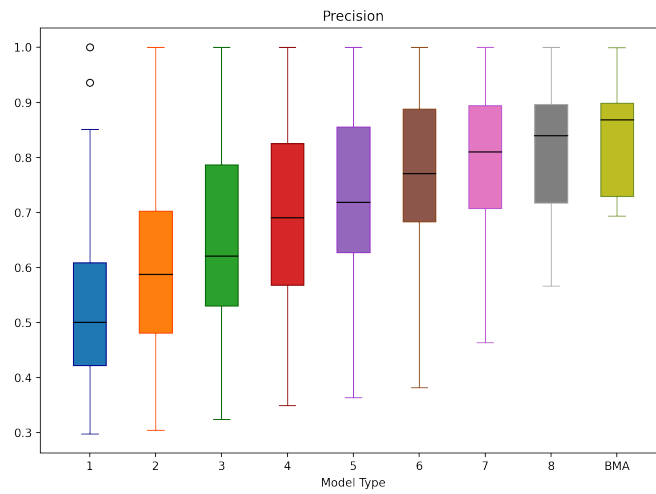
The attained results confirm our expectations. Firstly, the metrics we have used are precision, recall, and f1:

- Precision is defined as the ratio of the number of true positives to the sum of true positives and false positives. In other words, precision indicates the percentage of instances that the model has correctly classified as positive.
- Recall, also known as sensitivity or true positive rate, is defined as the ratio of

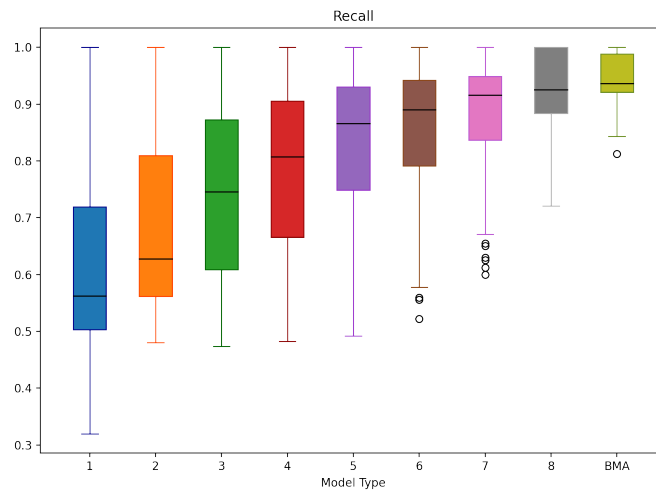
the number of true positives to the sum of true positives and false negatives and is employed to indicate the percentage of positive instances that the model has correctly classified.

- The f1 score is a harmonic mean between precision and recall. It is used when balancing precision and recall is desirable, in order to obtain a metric that evaluates performance comprehensively. The f1 score is defined as the ratio of the product of precision and recall to their sum.

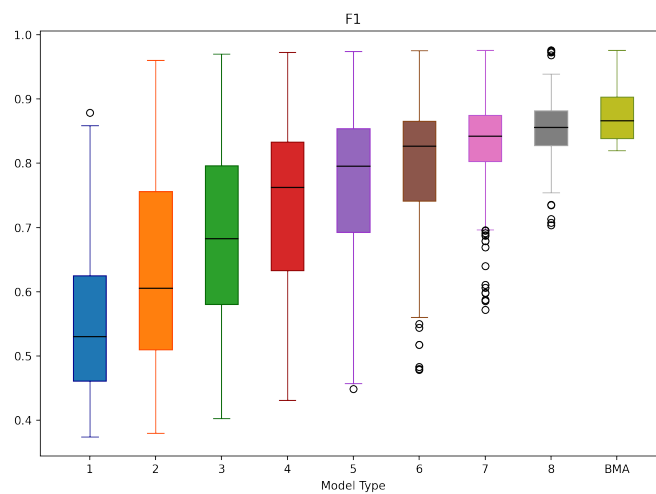
The Figure 4.13 illustrates how the BMA results (i.e., the rightmost rectangle of each box plot), for all metrics, are higher and more densely concentrated towards the top: on average, BMA models provide more accurate predictions.



(a) Precision



(b) Recall



(c) F1

Figure 4.13: Accuracy comparison between Logit and BMA

The depicted graphs showcase the performance distribution of Logit models and the BMA model in terms of precision, recall, and f1. The shaded rectangles delimit the interquartile range of the samples, i.e., the values between the 25% and 75% percentiles, while the median separates the data in half at the 50% percentile.

The observation that the BMA rectangle is consistently more tightly concentrated towards the top, compared to all the other Logit models, suggests that the BMA model outperforms the single Logit models in all three evaluation metrics. This advantageous behavior is due to BMA's ability to combine information from multiple Logit models more efficiently, resulting in superior predictions. In fact, BMA enables the simultaneous consideration of several models and computes their weighted average, where the weights are proportional to the models' posterior probabilities (see §1.4). Notably, the median value is more relevant than the interquartile range.

Regarding the Logit models, it is noticeable that increasing the number of considered variables leads to a higher accuracy in the results (though still never reaching or surpassing BMA's results).

4.3.3. RQ5: Evaluating the Cost of BMA Model Calculation

Now that we know that BMA is a more accurate approach compared to the individual Logit models, one of the requirements is certainly to obtain results as quickly as possible, without compromising accuracy. Therefore, we inquire: *What is the computational cost of calculating BMA models in order to mitigate model uncertainty?* In other words, does increasing the variables result in a proportionately constant duration?

It is known that among the various methods to obtain BMA, the deterministic and the MCMC methods deserve major consideration.

The former can be defined as the simplest in its conception: in fact, it calculates *all* possible models (based on the variables provided) and, in our case, since the considered variables were always 9, the time required was minimal (about one second).

Therefore, given the current question, we put the BMA approach to the test. We created a synthetic dataset consisting of 400 samples and 64 variables, gradually applying it to the fitting of models. The metric used to quantify the result is the duration in seconds of the training phase: the shorter the time, the better the result.

So, *what is the computational cost of using the deterministic method to calculate BMA models in order to mitigate model uncertainty?* Excessive.

Indeed, the time required doubled with just the increase of one variable, and then exponentially increased with the addition of even a single unit of variables (as seen in the orange line in the following figure).

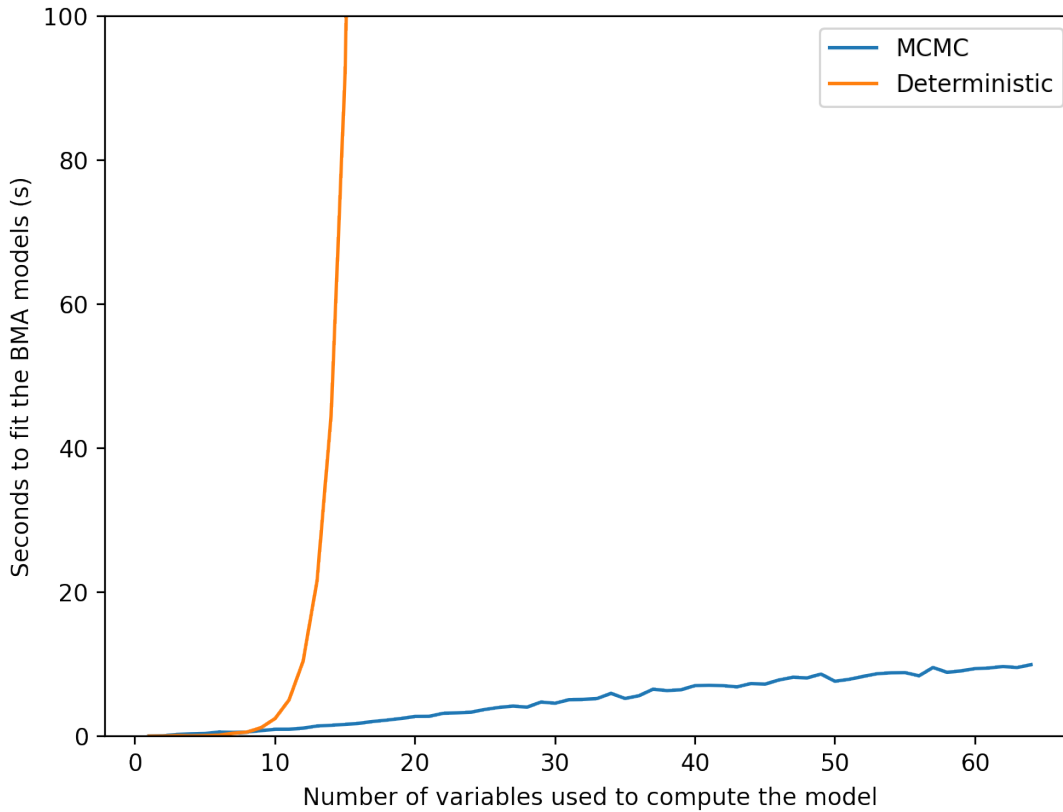


Figure 4.14: The time required in seconds to fit a BMA model using MCMC and Deterministic methods.

Using the MCMC method instead, up to the variables considered in our study (9), the *duration* result was substantially overlapping with the deterministic method. The best results in terms of duration were obtained with the increase of variables beyond the value of 9, which saw the time factor increase constantly but proportionally (as shown by the blue line in the previous figure).

However, everything has its Achilles' heel. The MCMC method evaluates only a fraction of the possible models based on the imposed variables, not all of them. It is intuitive that while the deterministic method (which, as previously stated, analyzes all possible models, regardless of the number of variables) requires a lot of time as the number of variables increases (stopping at a number slightly higher than that of our working hypothesis, and this is because each variable corresponds to innumerable models, calculable according to the following formula: $2^n - 1$, where n is the number of variables), MCMC, on the

other hand, which does not calculate all possible models related to the variables, makes the resulting time factor preferable over exhaustive enumeration, appearing impractical. Therefore, while in this case the choice is mandatory, in the hypothesis considered by us, both approaches appear as alternatives. However, it is also intuitive that exhaustive enumeration is preferable when the number of models is relatively small, and computational costs are not an obstacle.

4.3.4. RQ6: Optimizing Search for Restoring Equilibrium Constraints

In the previous section, in response to RQ4, we observed that BMA produces more consistent results compared to individual Logit models, i.e. results that are very close to the actual labels. Now, our aim is to verify whether the joint use of BMA and NSGA-3 is capable of addressing and neutralizing the perturbations that may occur during the execution of a SAS, thus ensuring the satisfaction of as many requirements as possible.

Therefore, we are wondering, *“how effective is our approach in computing new configurations of the semantic space in order to restore the equilibrium constraints?”*

We started from the same dataset considered in section 4.3.2. However, we split it differently and for different purposes: while previously we were interested in having a training set and a test set to be used exclusively for BMA, now we want to extract a training set to be used for BMA and a test set to evaluate the performance of NSGA-3. Specifically, the test set consists of a set of 100 configurations that do not satisfy any requirements, and the goal is to understand the percentage of requirements that can be satisfied by computing new configurations starting from them. To make the results more comprehensible, we will compare the results obtained from our approach with those generated randomly.

Recall that each configuration is composed of two sets of parameters: the first consists of all modifiable parameters, while the second is constituted of fixed parameters. Therefore, NSGA-3’s objective is to find a new combination of variable parameters that, together with the fixed ones, can satisfy as many requirements as possible.

The metrics used are two:

- Requirement Fulfillment Rate: quantifies the percentage of requirements satisfied after applying the new configuration.
- Costs: denotes the value that defines the distance between the new configuration found and the starting one.

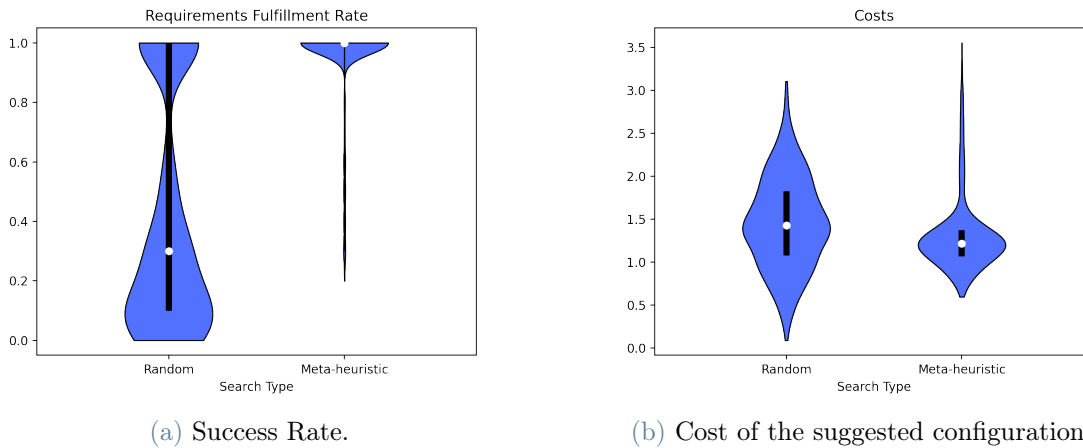


Figure 4.15: NSGA-3 vs RANDOM

Upon analyzing Figure 4.15a, it is readily apparent that the random method exhibits a high level of variability in results. Notably, as the objective of the method is to maximize success rate (represented by the optimum at 1.0), the failure of the test concerning RANDOM is evident, with results primarily concentrated around values close to zero. As a result, the median falls below expectations, rendering the method unsatisfactory.

On the other hand, the use of NSGA-3 has the advantage of aggregating almost all results close to the optimum, including the median, but at the cost of increased computational time. While random is virtually instantaneous, NSGA-3, in our testing environment, requires approximately 25 seconds for each set of Pareto-optimal solutions.

Although less evident, the NSGA-3 method proves preferable in identifying new configurations with a lower cost (Figure 4.15b). Indeed, the median of results obtained through this method is lower compared to that of the random approach and exhibits greater clustering of results around it. In contrast, the violin plot of the random method is highly elongated, highlighting a significant dispersion of results.

In general, it can be stated that the NSGA-3 approach tends to select new configurations with the aim of maximizing success rate while simultaneously minimizing, where possible, the cost of these new configurations.

5 | Conclusion and Future Developments

Considering the results obtained throughout all the experiments described in Chapter 4, in this last part we draw the conclusions of our work by assessing the effectiveness and limitations of our proposed method.

At the outset of this work, our aim was to implement RENE and TUNE and then extend them. Specifically, starting from a generic CPS with its own requirements and parameters in the semantic space, the objective was to satisfy as many requirements as possible.

In order to achieve this goal, we developed a simulator of CPSs using the pMDP formalism. This step was crucial to bring the uncertainty of the real world into MDPs through their integration with a range of parameters (pMDP). The positive results we achieved enabled us to use the simulator for our purposes.

We conducted numerous tests which allowed us to verify the system's ability to recognize short and long, large and small perturbations. We do not conceal that we also had negative results, which we nonetheless consider to be positive: with this oxymoron, we mean to say that negative results were not immediately discarded, but were evaluated, studied and modified until they became useful in order to consider the simulator reliable.

The next step was to verify the ability of our approach to identify these perturbations and inhibit their destabilizing effects on the equilibrium. Thus, having defined a pMDP and its semantic space, we acted by modifying the latter's parameters and, through the execution of the simulator, we verified the satisfaction of the requirements. These valuable results were collected in a dataset and used to build statistical models through the BMA approach. These models were essential to set up the many objective search which satisfied the proposed verification by finding new configurations of the semantic space.

More in details, our tests showed that the simulator, when applying our proposed adaptive observations ageing mechanism, was significantly more effective in detecting violations than without it. Additionally, the usage of a Policy was effective in avoiding disequilibria,

with the duration of the perturbations playing a more significant role than their magnitude. Furthermore, the BMA method outperformed all individual Logistic models in terms of precision, recall, and F1 score, while the Monte Carlo method was more efficient than the deterministic method in computing BMA models to mitigate model uncertainty. Finally, the joint use of BMA and NSGA-III proved to be effective in identifying new configurations of the semantic space while restoring equilibrium constraints. Overall, our approach showed promising results and can serve as an effective tool for decision-making in complex systems.

The key to understanding our proposal was the relationship between equilibrium and disequilibrium. As we have seen, a perturbation is an intervention that leaves a trace, and therefore has effects. To recognize the relationship between the latter and the parameters of the semantic space, we defined the Equilibrium Constraints, that is, the restrictions that must be respected to guarantee the equilibrium of a cyber-physical system.

At the end of this work, we can affirm that our approach has demonstrated the capacity to maintain and, when necessary, restore equilibria by neutralizing external environmental perturbations, reaching a state that can be defined as “meta-stable”, since it is still a partial, temporary equilibrium that is susceptible to attacks and interference, but is always able to recover.

There are still many opportunities to extend this work to other case studies in different domains of CPS application. In fact, *ex multis*, it could help better understand the potential of CPSs and develop new practical applications. In general, the hope is that this thesis will inspire future researchers to explore the numerous opportunities offered by CPSs in different contexts.

Bibliography

- [1] page 1–16. John Wiley and Sons, Ltd, 1994. ISBN 978-0-470-31688-7. doi: 10.1002/9780470316887.ch1. URL <https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470316887.ch1>.
- [2] M. Abdel-Basset, L. Abdel-Fatah, and A. K. Sangaiah. Chapter 10 - metaheuristic algorithms: A comprehensive review. In A. K. Sangaiah, M. Sheng, and Z. Zhang, editors, *Computational Intelligence for Multimedia Big Data on the Cloud with Engineering Applications*, Intelligent Data-Centric Systems, pages 185–231. Academic Press, 2018. ISBN 978-0-12-813314-9. doi: <https://doi.org/10.1016/B978-0-12-813314-9.00010-4>. URL <https://www.sciencedirect.com/science/article/pii/B9780128133149000104>.
- [3] J.-P. Aubin, A. Bayen, and P. Saint-Pierre. *Viability Theory: New Directions*. 01 2011. ISBN 978-3-642-16683-9. doi: 10.1007/978-3-642-16684-6.
- [4] A. Bennaceur, C. Ghezzi, K. Tei, T. Kehrer, D. Weyns, R. Calinescu, S. Dustdar, Z. Hu, S. Honiden, F. Ishikawa, Z. Jin, J. Kramer, M. Litoiu, M. Loreti, G. Moreno, H. Müller, L. Nenzi, B. Nuseibeh, L. Pasquale, W. Reisig, H. Schmidt, C. Tsigkanos, and H. Zhao. Modelling and analysing resilient cyber-physical systems. In *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 70–76, 2019. doi: 10.1109/SEAMS.2019.00018.
- [5] A. Bennaceur, C. Ghezzi, K. Tei, T. Kehrer, D. Weyns, R. Calinescu, S. Dustdar, Z. Hu, S. Honiden, F. Ishikawa, Z. Jin, J. Kramer, M. Litoiu, M. Loreti, G. Moreno, H. Müller, L. Nenzi, B. Nuseibeh, L. Pasquale, W. Reisig, H. Schmidt, C. Tsigkanos, and H. Zhao. Modelling and analysing resilient cyber-physical systems. In *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 70–76, 2019. doi: 10.1109/SEAMS.2019.00018.
- [6] J. Berger. *Statistical Decision Theory and Bayesian Analysis*. Springer Series in

- Statistics. Springer, 1985. ISBN 9780387960982. URL https://books.google.it/books?id=oY_x7dE15_AC.
- [7] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, sep 2003. ISSN 0360-0300. doi: 10.1145/937503.937505. URL <https://doi.org/10.1145/937503.937505>.
- [8] M. Camilli, R. Mirandola, and P. Scandurra. Taming model uncertainty in self-adaptive systems using bayesian model averaging. In *Proceedings of the 17th Symposium on Software Engineering for Adaptive and Self-Managing Systems*, page 25–35, Pittsburgh Pennsylvania, May 2022. ACM. ISBN 978-1-4503-9305-8. doi: 10.1145/3524844.3528056. URL <https://dl.acm.org/doi/10.1145/3524844.3528056>.
- [9] M. Camilli, R. Mirandola, and P. Scandurra. Enforcing resilience in cyber-physical systems via equilibrium verification at runtime. *ACM Transactions on Autonomous and Adaptive Systems*, page 3584364, Feb 2023. ISSN 1556-4665, 1556-4703. doi: 10.1145/3584364.
- [10] T. Chen, K. Li, R. Bahsoon, and X. Yao. FEMOSAA. *ACM Transactions on Software Engineering and Methodology*, 27(2):1–50, apr 2018. doi: 10.1145/3204459. URL <https://doi.org/10.1145%2F3204459>.
- [11] B. Cheng, R. De Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. Di Marzo Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. Müller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, and J. Whittle. Software engineering for self-adaptive systems: A research roadmap. In *Software Engineering for Self-Adaptive Systems*, volume 5525 LNCS of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pages 1–26, 2009. ISBN 3642021603. doi: 10.1007/978-3-642-02161-9_1.
- [12] J. Donckt, D. Weyns, M. Iftikhar, and R. Singh. Cost-benefit analysis at runtime for self-adaptive systems applied to an internet of things application. pages 478–490, 01 2018. doi: 10.5220/0006815404780490.
- [13] K. Du and M. Swamy. *Search and Optimization by Metaheuristics: Techniques and Algorithms Inspired by Nature*. Springer International Publishing, 2016. ISBN 9783319411927. URL <https://books.google.it/books?id=Jhy1DAAAQBAJ>.
- [14] T. M. Fragoso, W. Bertoli, and F. Louzada. Bayesian model averaging: A system-

- atic review and conceptual classification. *International Statistical Review*, 86(1):1–28, 2018. doi: <https://doi.org/10.1111/insr.12243>. URL <https://onlinelibrary.wiley.com/doi/abs/10.1111/insr.12243>.
- [15] D. Gamerman and H. Lopes. *Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference, Second Edition*. Chapman & Hall/CRC Texts in Statistical Science. Taylor & Francis, 2006. ISBN 9781584885870. URL https://books.google.it/books?id=yPvECi_L3bwC.
- [16] S. M. Hezavehi, D. Weyns, P. Avgeriou, R. Calinescu, R. Mirandola, and D. Perez-Palacin. Uncertainty in self-adaptive systems: A research community perspective, 2021.
- [17] M. Hinne, Q. F. Gronau, D. van den Bergh, and E.-J. Wagenmakers. A conceptual introduction to bayesian model averaging. *Advances in Methods and Practices in Psychological Science*, 3(2):200–215, 2020. doi: 10.1177/2515245919898657. URL <https://doi.org/10.1177/2515245919898657>.
- [18] J. A. Hoeting, D. Madigan, A. E. Raftery, and C. T. Volinsky. Bayesian model averaging: a tutorial (with comments by M. Clyde, David Draper and E. I. George, and a rejoinder by the authors). *Statistical Science*, 14(4):382 – 417, 1999. doi: 10.1214/ss/1009212519. URL <https://doi.org/10.1214/ss/1009212519>.
- [19] J. A. Hoeting, D. Madigan, A. E. Raftery, and C. T. Volinsky. Bayesian model averaging: A tutorial. *Statistical Science*, 14(4):382–401, 1999. ISSN 08834237. URL <http://www.jstor.org/stable/2676803>.
- [20] M. Jackson. The meaning of requirements. *Annals of Software Engineering*, 3(1): 5–21, Jan 1997. ISSN 1573-7489. doi: 10.1023/A:1018990005598. URL <https://doi.org/10.1023/A:1018990005598>.
- [21] D. Kaplan. On the quantification of model uncertainty: A bayesian perspective. *Psychometrika*, 86(1):215–238, Mar 2021. ISSN 1860-0980. doi: 10.1007/s11336-021-09754-5.
- [22] J. Kephart and D. Chess. The vision of autonomic computing. *Computer*, 36(1): 41–50, 2003. doi: 10.1109/MC.2003.1160055.
- [23] L. Pagliari, R. Mirandola, and C. Trubiani. Engineering cyber-physical systems through performance-based modelling and analysis: A case study experience report. *Journal of Software: Evolution and Process*, 32, 07 2019. doi: 10.1002/smr.2179.
- [24] C. Robert. *The Bayesian Choice: From Decision-Theoretic Foundations to Com-*

- putational Implementation*. Springer Texts in Statistics. Springer New York, 2007. ISBN 9780387715988. URL <https://books.google.it/books?id=6oQ4s8Pq9pYC>.
- [25] M. van Otterlo. Markov decision processes: Concepts and algorithms.
- [26] D. Weyns. *Software Engineering of Self-adaptive Systems*, pages 399–443. Springer International Publishing, Cham, 2019. ISBN 978-3-030-00262-6. doi: 10.1007/978-3-030-00262-6_11. URL https://doi.org/10.1007/978-3-030-00262-6_11.
- [27] D. Weyns, N. Bencomo, R. Calinescu, J. Cámara, C. Ghezzi, V. Grassi, L. Grunske, P. Inverardi, J.-M. Jézéquel, S. Malek, R. Mirandola, M. Mori, and G. Tamburrelli. Perpetual assurances for self-adaptive systems, 2019.
- [28] B. A. Wintle, M. A. McCarthy, C. Volinsky, and R. P. Kavanagh. The use of bayesian model averaging to better represent uncertainty in ecological models. *Conservation Biology*, 17, 2003.

List of Figures

1.1	Conceptual model of a self-adaptive system	8
1.2	Structure of autonomic manager [26].	9
1.3	An example of an MDP with eight states and nine actions.	11
2.1	RUNE approach overview [9].	22
2.2	Visualization of the equilibrium conditions for the uncertain region θ_1 of the pMDP in 2.2a	24
3.1	The main three classes composing the pMDP object	32
3.2	An example of MDP.	34
3.3	An example of MDP with the uncertain regions	37
3.4	The plot for the state s_0 , action a and outcome s_2	42
3.5	Rescue Robot pMDP [9]	45
4.1	Graph representing the first pMDP	52
4.2	Graph representing the second pMDP	53
4.3	Simulation with $m=0.1$ and $d=20$ without using the adaptive observation aging mechanism	54
4.4	Simulation with $m=0.1$, $d=\infty$ and without using the adaptive observation aging mechanism.	55
4.5	Simulation with $m=0.01$, $d=20$ using the adaptive observation aging mech- anism	55
4.6	$m=0.1$, $d=\infty$ using the adaptive observation aging mechanism	56
4.7	$m=0.1$, $d=20$ using the adaptive observation aging mechanism	57
4.8	$m=0.01$, $d=\infty$ using the adaptive observation aging mechanism	57
4.9	Effectiveness of the Interval Estimates	58
4.10	Effectiveness of the Point Estimates	59
4.11	False Positives detection	60
4.12	Violation rate	62
4.13	Accuracy comparison between Logit and BMA	70

4.14 The time required in seconds to fit a BMA model using MCMC and Deterministic methods.	72
4.15 NSGA-3 vs RANDOM	74

List of Tables

3.1	Semantic Space Variables used in the simulation.	44
4.1	Requirement 1	64
4.2	Requirement 2	64
4.3	Requirement 3	65
4.4	Requirement 4	65
4.5	Requirement 5	65
4.6	Requirement 6	66
4.7	Requirement 7	66
4.8	Requirement 8	67
4.9	Requirement 9	67
4.10	Requirement 10	68

List of Symbols

Acronyms	Description
BMA	Bayesian Model Averaging
MDP	Markov Decision Process
CPS	Cyber-Physical System
SAS	Self-Adaptive System
RUNE	RUNtime Equilibrium verification
TUNE	Taming model UNcErtainty
pMDP	parametric Markov Decision Process

Ringraziamenti

Questa tesi con ogni probabilità ha fermentato nella mia mente per quasi diec'anni, cioè fin da quando ero al liceo e riflettevo sul rapporto tra equilibrio e disequilibrio, e mi interrogavo sulla loro compatibilità al pari di chi osserva i disegni di M.C. Escher, fra i più concettualmente stimolanti di tutti i tempi, ispirati a paradossi, illusioni e doppi sensi. E non a caso i matematici furono tra i primi ammiratori tra i disegni di Escher, come mio fratello Giuseppe che in più dei suoi lavori si è ispirato al grafico olandese, facendomelo conoscere. E non a caso dagli studi classici sono approdato al Politecnico. La Cascata di Escher è stata l'ispirazione per questo lavoro. Si parte dall'equilibrio dell'acquedotto per poi giungere alla cascata che rappresenta un elemento distruttivo della precedente linearità; ma poi proseguendo nel corso d'acqua, ecco che ritorna l'equilibrio.

Quest'idea escheriana mi è stata proposta dalla Prof.ssa Rafaela Mirandola, alla quale debbo speciale gratitudine perché mi ha aiutato a sviluppare il mio pensiero, e i suoi preziosi suggerimenti si trovano sparsi in tutta la tesi. Con il suo prezioso aiuto, ho scoperto che non è solo nelle fiabe che si incontrano immensi spazi in equilibrio, ma è possibile anche nella realtà, non è solo materia di fiabe. Ho un grosso debito anche verso il Prof. Matteo Camilli, per avermi insegnato a dedurre e esporre gli argomenti, e anche per il suo costante incoraggiamento, le lodi implicite, talvolta le sue critiche.

Ringrazio gli amici del Poli, che hanno contribuito molto alla mia visione del mondo: Luca, Emanuele, Aba, Mariano e Andrea hanno influenzato il contenuto degli esempi sparsi in questa tesi. Per ricevere consigli su argomenti grandi e piccoli mi sono spesso rivolto a Nicolò, che più volte mi ha dato buone idee che io ho accolto con gioia. Vorrei anche ringraziare Ivan, amico che ha condiviso con me i momenti cruciali della mia vita universitaria e che perciò ha contribuito in parecchi modi diversi a questo mio lavoro.

Queste sono le persone che hanno contribuito al formarsi di questa tesi, e spero che il mio entusiasmo nella sua redazione traspaia e penetri nella mente almeno di alcuni lettori. Questo è il massimo che posso chiedere.

Acknowledgements

This thesis has most likely been fermenting in my mind for almost a decade - that is, ever since I was in high school and I reflected on the relationship between equilibrium and disequilibrium, pondering their compatibility just like those who observe the works of M.C. Escher, one of the most conceptually stimulating artists of all time, inspired by paradoxes, illusions, and double meanings. It is no coincidence that mathematicians were among the first admirers of Escher's works, such as my brother Giuseppe, who, in addition to his own works, drew inspiration from the Dutch artist, introducing me to him. And it is no coincidence that I went from classical studies to Polytechnic. Escher's Waterfall was the inspiration for this work: starting from the equilibrium of the aqueduct, then progressing to the waterfall, which represents a destructive element of the preceding linearity, but then returning to the equilibrium further *up* the stream.

This Escherian idea was proposed to me by Prof. Raffaella Mirandola, to whom I owe special gratitude for helping me develop my thoughts, with her valuable suggestions scattered throughout the thesis. With her invaluable help, I discovered that the immense spaces in equilibrium are not just found in fairy tales, but are possible even in reality. I also owe a great debt to Prof. Matteo Camilli for teaching me how to deduce and present arguments, and for his constant encouragement, sometimes implicit praise, at times criticism.

I thank my Poli's friends who have greatly contributed to shaping my worldview: Luca, Emanuele, Aba, Mariano, and Andrea influenced the content of the examples scattered throughout this thesis. To receive advice on both large and small matters, I often turned to Nicolò, who gave me good ideas that I joyfully accepted many times. I would also like to thank my friend Ivan, who shared with me the crucial moments of my university life, contributing to this work in many different ways.

These are the individuals who have contributed to the creation of this thesis, and I hope that my enthusiasm in its writing shines through and permeates the minds of at least a few readers. This is the most I could ask for.

