Design of Joint Verification-Correction Strategies for Engineered Systems

Peng Xu

Dissertation submitted to the faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Industrial and Systems Engineering

Alejandro Salado, Chair

Hanumanthrao Kannan

Jin-Hee Cho

Xinwei Deng

April 18, 2022

Blacksburg, Virginia

Keywords: system verification, correction activity, Bayesian network, sequential decision-making, tree search.

Design of Joint Verification-Correction Strategies for Engineered Systems

Peng Xu

ABSTRACT

System verification is a critical process in the development of engineered systems. Engineers gain confidence in the correct functionality of the system by executing system verification. Traditionally, system verification is implemented by conducting a verification strategy (VS) consisting of verification activities (VA). A VS can be generated using industry standards, expert experience, or quantitative-based methods. However, two limitations exist in these previous studies. First, as an essential part of system verification, correction activities (CA) are used to correct system errors or defects identified by VAs. However, CAs are usually simplified and treated as a component associated with VAs instead of independent decisions. Even though this simplification may accelerate the VS design, it results in inferior VSs because the optimization of correction decisions is ignored. Second, current methods have not handled the issue of complex engineered systems. As the number of activities increases, the magnitude of the possible VSs becomes so large that finding the optimal VS is impossible or impractical. Therefore, these limitations leave room for improving the VS design, especially for complex engineered systems.

This dissertation presents a joint verification-correction model (JVCM) to address these gaps. The basic idea of this model is to provide an engineering paradigm for complex engineered systems that simultaneously consider decisions about VAs and CAs. The accompanying research problem is to develop a modeling and analysis framework to solve for joint verification-correction strategies (JVCS). This dissertation aims to address them in three steps. First, verification processes (VP) are modeled mathematically to capture the impacts of VAs and CAs. Second, a JVCM with small strategy spaces is established with all conditions of a VP. A modified backward induction method is proposed to solve for an optimal JVCS in small strategy spaces. Third, a UCB-based tree search approach is designed to find near-optimal JVCSs in large strategy spaces. A case study is conducted and analyzed in each step to show the feasibility of the proposed models and methods.

Design of Joint Verification-Correction Strategies for Engineered Systems

Peng Xu

GENERAL AUDIENCE ABSTRACT

System verification is a critical step in the life cycle of system development. It is used to check that a system conforms to its design requirements. Traditionally, system verification is implemented by conducting a verification strategy (VS) consisting of verification activities (VA). A VS can be generated using industry standards, expert experience, or quantitative-based methods. However, two limitations exist in these methods. First, as an essential part of system verification, correction activities (CA) are used to correct system errors or defects identified by VAs. However, CAs are usually simplified and treated as remedial measures that depend on the results of VAs instead of independent decision choices. Even though this simplification may accelerate the VS design, it results in inferior VSs because the optimization of correction decisions is ignored. Second, current methods have not handled the issue of large systems. As the number of activities increases, the total number of possible VSs becomes so large that it is impossible to find the optimal solution. Therefore, these limitations leave room for improving the VS design, especially for large systems.

This dissertation presents a joint verification-correction model (JVCM) to address these gaps. The basic idea of this model is to provide a paradigm for large systems that simultaneously consider decisions about VAs and CAs. The accompanying research problem is to develop a modeling and analysis framework to solve for joint verification-correction strategies (JVCS). This dissertation aims to address them in three steps. First, verification processes (VP) are modeled mathematically to capture the impacts of VAs and CAs. Second, a JVCM with small strategy spaces is established with all conditions of a VP. A modified backward induction method is proposed to solve for an optimal JVCS in small strategy spaces. Third, a UCB-based tree search approach is designed to find near-optimal JVCSs in large strategy spaces. A case study is conducted and analyzed in each step to show the feasibility of the proposed models and methods.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my advisor Dr. Alejandro Salado for his patient guidance throughout my Ph.D. study. I have been fortunate to have an advisor as Dr. Salado, who has been providing so much valuable advice to me on my research work and my career life. I would like to thank my committee members, Dr. Hanumanthrao Kannan, Dr. Jin-Hee Cho, and Dr. Xinwei Deng, for their insightful suggestions, comments, and support during my graduate study. In particular, I thank Dr. Cho for helping me finish my first journal paper at Virginia Tech. And I thank Dr. Deng for providing me so many open office hours to discuss research questions.

I would like to express special thanks to my former advisor Dr. Jaime Camelio and other scholars at Virginia Tech, including but not limited to Dr. Zhenyu Kong, Dr. Ran Jin, Dr. Gregory Purdy, Dr. Scotland Leman, and Dr. Deborah Dickerson. Without their patience, support, and understanding, my Ph.D. journey would become harder than it is.

I would also like to thank all my friends for companying me and making my graduate life at Blacksburg more colorful. The moments I spent with them are the cherished memories of my future life.

Finally, I must thank my parents and other family members for the unconditional support and love, which encouraged me to overcome the difficulties throughout my Ph.D. life.

Table of Contents

Summary of Notations

Symbol	Description
t	Time event, $t = 1, \ldots, T$.
ε_i	System element, $i = 1, \ldots, I$.
θ_i	System parameter, $i = 1, \ldots, I$.
μ_j	Verification activity, $j = 1, \ldots, J$.
φ_k	Correction activity, $k = 1, \ldots, K$.
ξ_r	Causal factors of a conditional probability table.
λ	Likelihood ratio of uncertain evidence.
$\delta(\cdot)$	Indicator function whose value is 1 if the statement is true and 0 otherwise.
A or $A(\cdot)$	Activity results. $A(\mu_j)$ is the result of a VA (e.g., $A(\mu_j)$=*Pass* or *Fail*).
$B(\theta_i)$	Revenue that depends on system parameter θ_i.
$C(\cdot)$	Cost, including $C(\mu_j)$, $C(\varphi_k)$, and $C(A(\mu_j)$=Fail).
S_m	System state, $m = 1, \ldots, M$.
H_i	Threshold value for $B(\theta_i)$.
$Q(S_m)$	Order function of system state S_m.
$Y(\cdot)$	Type function.
Z_w	Verification path, $w = 1, \ldots, W$.
Ψ	JVCS, $\Psi = \{ Z_w: w = 1, \ldots, W\} = \{S_m: m = 1, \ldots, M\}$.
$U(\cdot)$	Value function. $U(S_m)$ is the state value of S_m. $U(Z_w)$ is the overall value of Z_w. $U(\Psi\|S_m)$ is the overall value of Ψ for a given initial system state S_m and $U(\Psi\|S_m) = U(S_m)$.
$x_{k,s}$	Reward function of the *s*th play of the *k*th machine in a bandit problem, which has a distribution F_k.
n_k	Number of plays of the *k*th machine in a bandit problem.
n	Total number of plays in a bandit problem.
u_k	Reward supremum of the *k*th machine.
u_*	Maximum reward supremum of all machines.
D	Constants, including D_0, D_1, D_2, D_3, D_4, D_6, D_7, D_8, D_9, D_{10}.
$UCB(S_m)$	Upper confidence bound function of system state S_m.
Ω	Lookup table that stores state values and visit counts.

CHAPTER 1: Introduction and Problem Setting

1.1 Background

Today, manufacturing industries are concerned with the development of products, services, or solutions that integrate a large mix of technologies and involve both people and machines [1]. For example, the semiconductor manufacturing in Taiwan Semiconductor Manufacturing Company is supported by more than 37,000 patents, 48,000 employees, and massive equipment and tools [2]. The production of a semiconductor chip relies on these complex entities rather than any single component. This kind of complex entities is named as an engineered system in this dissertation.

In addition, entities developing complex engineered systems also face the challenge of responding to various market demands. To be specific, manufacturing industries are required to satisfy consumer needs, reduce time to market, and produce qualified goods at reduced costs [1]. Therefore, controlling schedules, costs, and quality of products remains a significant challenge for product development in today's engineered systems [1].

In product development, *system verification* is defined as a set of activities that evaluate whether a system or its components fulfill their requirements [3, 4]. As a similar concept, system validation is referred to a set of activities used to check the compliance of a system or its components with its purpose (satisfaction of stakeholder needs) [3]. The research work in this dissertation is strictly restricted to system verification. System verification has a direct effect on the quality of products because it aims to identify errors and defects [1]. Specifically, a well-designed verification can contribute to the expected utility of a system in multiple ways, such as by shaping beliefs about a system exhibiting specific characteristics, consuming resources, or informing the need for specific design features [5].

A substantial part of system verification is performed in the test phase, a late and highly constrained period in the product life cycle, whereas it is often ignored in other phases, such as definition phase, model phase, maintenance phase, and disposal phase [1]. Due to this late execution of system verification, the expected cost associated with system correction, such as labor or monetary costs, exceeds 20% of engineering efforts [1]. Specifically, it is pointed out that when a product progresses through its life cycle, the expected costs associated with design changes (e.g., system correction) increase rapidly, as shown in Fig. 1.1 [6]. If the changes are implemented in later phases, the costs are more expensive. In addition, NASA reports that the success of a program or project is strongly related to the frequency and diligence of verification, especially in the early phases during which corrections might be made cost-effectively [7].

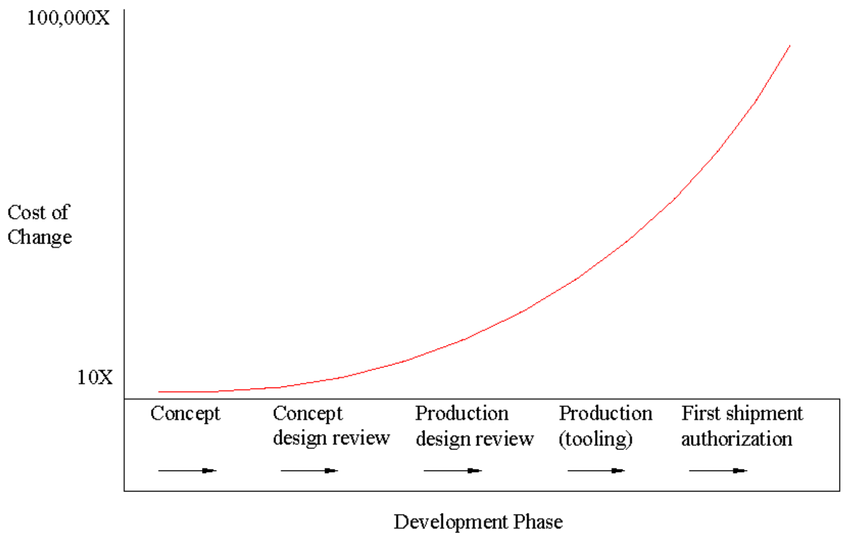


Fig. 1.1 Cost of changes in relation to product development phase [6]

1.2 Overview of the Problem Domain

To reduce the costs of product development, system verification has been extended to cover the entire product life cycle from the definition phase to the disposal phase [1]. This broad-range verification is defined as a verification process (VP) in this dissertation. This approach to verification has been applied in a variety of industrial applications, such as automotive [8], aviation [9], and motorcycle industries [10]. For example, in the life cycle of "Weapon System Development" for the Department of Defense acquisition [11], system development is structured in seven phases, as shown in Fig. 1.2. As can be seen, verification events are planned at different phases in this life cycle, which are outlined by red rectangles.

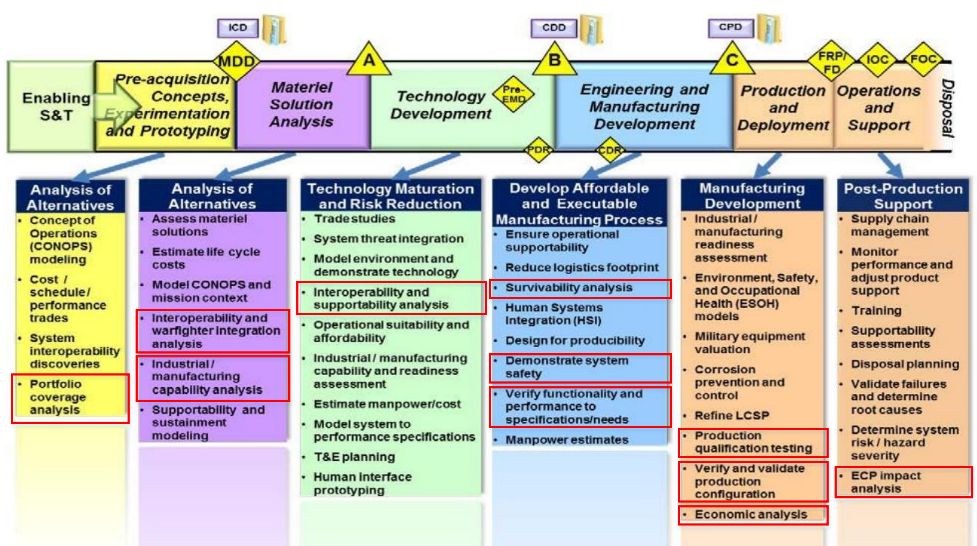


Fig. 1.2 The life cycle of "Weapon System Development" [11]

In a VP, the basic unit of evaluating a requirement is an activity used to perform a specific task. As to system verification, different activities may be performed to evaluate each specific requirement [12]. Their purpose is to ensure that engineered systems are delivered as error-free as possible, are functionally sound, and meet or exceed the user's needs [1]. This dissertation restricts its attention to two main types of activities: verification activities and correction activities. They are defined below:

- **Verification activities (VA):** activities that check the correctness of system components. They include inspection, analysis, analogy, demonstration, test, and sampling [16], described in Table 1.1.
- **Correction activities (CA):** activities that can eliminate errors that are found during the verification process. There are three common types of CAs: rework, repair, and redesign. Different definitions appear for these types of activities in the literature. For example, in [13], rework is defined as the repetition of an activity at the same scope and abstraction level. In [14], rework is defined as "the design change whose implementation alters work that was previously done upstream and downstream," and in [15], rework is defined more generally as a task that needs to be redone because of a change. Therefore, dedicated definitions have been adopted for this dissertation, presented in Table 1.1. Examples of these CAs are provided in Table 1.2.

Table 1.1. Various activities in system verification

Activity		**Description**
Verification Activity	Inspection	"Activity that examines an element visually or dimensionally. It typically includes the use of sight, hearing, smell, touch, and taste, simple physical manipulation, mechanical and electrical gauging, and measurement." [16]
	Analysis	"Activity that uses mathematical or probabilistic calculation, logical reasoning (including the theory of predicates), modeling and/or simulation under defined conditions to show theoretical compliance." [16]
	Analogy	"Activity that is based on evidence of similar elements to the target element or on experience feedback." [16]
	Demonstration	"Activity that demonstrates correct operation of the target element against operational and observable characteristics without using physical measurements (no or minimal instrumentation or test equipment). Demonstration is also called 'field testing'." [16]
	Test	"Activity that is performed onto the target element by which functional, measurable characteristics, operability, supportability, or performance capability is quantitatively verified when subjected to controlled conditions that are real or simulated." [16]
	Sampling	Activity that uses samples to verify the target element. [16]

Correction Activity	Rework	Activity that substitutes a faulty element with a working one to correct defects.
	Repair	Activity that directly corrects system defects by using processes, materials, or tools not initially considered in the design while the initial system elements are retained.
	Redesign	Activity that changes the design baseline of the system.

Table 1.2. Examples of CAs

CA	Example
Rework	The lens of a telescope does not offer the expected transmission performance. The cause for this degradation is a crack on its surface. It is removed from the telescope and a new lens, which is not damaged, is installed instead.
Repair	The lens of a telescope does not offer the expected transmission performance. The cause for this degradation is a scratch on its surface. The surface is polished and covered with a new coating.
Redesign	The lens of a telescope does not offer the expected transmission performance. The cause for this unexpected performance is a limit on the ability of the material that has been used. A new lens with a different shape and material is designed and installed in the telescope.

In current system verification, the number of activities always increases when the engineered system becomes complex. To give a sense of size, a system may well respond to over 100 requirements, and, for each requirement, over 10 types of inspections could be defined, 10 types of analyses, and 10 sets of tests. If there are only one inspection, one analysis, and one test for each requirement, the total possible combinations of these activities for this system is $(10^3)^{100}=10^{300}$, while the number of grains of sand on Earth is about 10^{19}. Therefore, handling size is not trivial when engineered systems are complex.

Considering these complex systems, VAs are often organized as an overall verification strategy (VS), which can be executed at different life cycle phases and on different system configurations, capturing all information related to the selection of activities [17]. Within a VS, dependencies exist between the VAs in terms of the expected utility. Specifically, the expected value yielded by a VA depends on the results of previous VAs and on the VAs that may be executed in the future. Thus, the interrelationship between VAs should not be ignored during the design of a VS [18, 19].

Currently, VSs are generally defined at the beginning of a system development program, following an industry standard or expert experience [1]. This approach heavily relies on qualitative opinions and is not supported by data or quantitative elicitation of expert opinions or beliefs. Thus, it is hard, if not impossible, to assess the optimality of resulting VSs. Quantitative-based methods are therefore necessary to improve the optimality of VSs, especially when the number of potential VAs is large. The advantages of quantitative-based methods can be summarized as maximizing the confidence in verification coverage, minimizing the risk of undetected problems, and minimizing invested effort [20].

It is also important to notice that CAs are a significant component of product development cycle time that can represent as much as two-thirds of project effort [21]. They occur when a prior decision is changed because the system was later found to be defective [22]. There are two different threads in current research related to CAs. The first one addresses the development of methods that help reduce or even prevent the need to execute CAs. These include, among others, the use of set-based practices [22], collective learning approaches [23], the use of design visualization models to facilitate early inspection [24], overlapping strategies to perform implementation tasks concurrently [25, 26], early testing to avoid cascading effects and late correction (which is generally more expensive) [27, 28], and the design structure matrix to isolate and mitigate rework by capturing interdependencies [29]. The second thread considers that some CAs are inevitable and focuses on reducing the impact (e.g., cost) of CAs. Because uncertainty seems to be the primary cause of the need for CAs [30, 31, 32], these methods primarily leverage modeling and optimization. For example, Browning and Eppinger [33] developed a stochastic model that uses a discrete event simulation to understand the impacts of process architecture on process cost, duration, and risk. Liao [34] proposed an economic production quantity model for maintenance, production, and free-repair warranty programs to minimize total production cost. Taleizadeh et al. [35] developed an imperfect, multi-product production system with rework to minimize the joint total cost of the system. While these studies provide valuable insights to manage CAs, they mainly focus on the negative impact of CAs on cost and time during system development, leaving aside their positive effect of regaining confidence in the correct state of the system. However, correction has become so commonplace that most development managers seem to consider it inevitable [36, 37, 38]. Multiple prototype build-and-test cycles [39] are even proposed to manage multiple layers of correction.

1.3 Problem Statement

To visualize the problem, a typical VP example is shown in Fig. 1.3, where solid lines represent deterministic transitions, dashed lines represent probabilistic transitions, and dotted lines are used to separate different time events (i.e., phase). At each time event, the system configuration is represented as a system state S_m that captures all necessary information for a decision. In this example, as the system parameters cannot be observed directly, a system state is represented by the time event and the confidence levels of the system parameters (i.e., the belief that the state of a system parameter is *Pass*). Given a system state S_1 at phase t, a verification activity μ_j is chosen from all possible choices $\{\mu_1, \mu_2, ..., \mu_J\}$ to check the fulfillment of a system requirement. If its result is *Pass*, the system state is updated to S_2, and there is no need for further verification at this time event. But if its result is *Fail*, the system state is updated to S_3, and there comes a choice whether a correction activity should be implemented. If a CA is implemented, some actions will be taken, and the system state will transit from S_3 to S_4. If a CA is not implemented, the confidence levels of system parameters will remain the same. But as the time event become t+1, the system state will transit from S_3 to S_5. This pattern is repeated throughout the whole verification process.

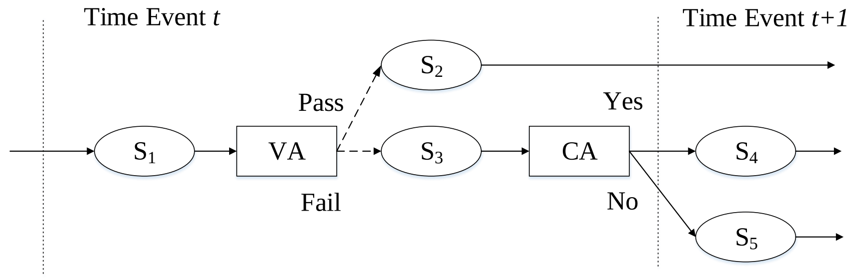


Fig. 1.3. A VP example

As it can be observed from Fig. 1.3, there are two aspects about how CAs fit the overall VP that are worth mentioning. First, CAs are not conducted simultaneously with VAs. Second, CAs result from dedicated choices (i.e., correction or no correction in this example) that are different from those of VAs. In current practice, implementing which choice of CA has not been captured as decisions that are independent of VAs. Instead, because the conduction of CAs depends on the occurrence of defects that are hard to predict beforehand, CAs are considered properties of the VAs, or default reactions to the results of VAs [40, 41, 42]. This limitation seeds the research problem in this dissertation that whether correction is conducted or not can, in itself, be a choice separately. This type of verification process that contains both VAs and CAs as decisions is named as an extended paradigm of verification processes (EPVP) in this dissertation. The main difference between traditional and proposed paradigms is that CAs are modeled as explicit decisions decoupled from VA decisions in this dissertation.

Compared with the traditional paradigm, the EPVP offers more decision choices that result from these two types of decisions (i.e., VAs and CAs). For example, CAs can be conducted in the early phases to eliminate some errors that could be more costly to eliminate in later phases. On the other hand, CAs can be postponed to later phases to save correction costs, which can be applied when CAs are not urgent. For example, in highly stochastic systems, letting some errors accumulate before correction may be cheaper than correcting errors individually as they are detected. Therefore, considering this flexibility, this dissertation models the system verification problem as verification processes with hybrid decisions.

Similar to the VS of a VP, a joint verification-correction strategy (JVCS) specifies which VA or CA should be implemented at each pair of a system configuration and its time event in our extended paradigm. The system verification problem for a given engineered system is how to design the JVCS given a corresponding VP. Following the example in Fig. 1.1, this design work is two-fold. First, it is required to determine whether a VA is conducted and which VA is chosen at each phase. Second, the JVCS should contain whether a correction is worth implementing if there are such correction opportunities. All these decisions about VAs and CAs constitute a JVCS. The goal of the JVCS design is to maximize the expected utility of system verification during the VP. The expected utility is determined according to the specific verification context. For example, the goal of a JVCS can be simply how to maximize the confidence levels of system parameters.

Designing JVCS faces three challenges. First, there are mutual effects between an engineered system and the JVCS that limit the ability to separate VA and CA decisions as decoupled problems.

The engineered system may change along the collection of results that are brought by VAs and CAs. The design of a JVCS at each phase should consider the updated engineered system. These mutual effects make decoupling of the activity choices difficult, if not impossible [17]. In other words, the implementation of any activity at each phase, as well as its results, may have temporal effects on each other during the whole VP. So these mutual effects need to be captured for designing effective JVCSs.

Second, as system verification contributes to the system's utility, the design of JVCS involves the maximization of a utility function. However, there is a tradeoff between short-term utility items at each phase and the long-term utility function of the whole VP. While the short-term utility items are obtained immediately after making a decision at each time event, the long-term utility cannot be determined until the end of the process. Because the JVCS is conducted along a process and the system changes stochastically at each time event, the long-term utility function suffers more uncertainty. Meanwhile, there is also a tradeoff between utility items that conflict with each other. For example, investing more correction cost may increase the deployment revenue. Then it is necessary to balance the correction cost and revenue. Therefore, these kinds of tradeoffs need to be considered during the design of JVCS.

Third, as mentioned in Section 1.2, the design of JVCS faces a high computational burden given the size of the strategy space, which consists of all possible strategies. The size of the strategy space is directly influenced by three factors, namely, activity choices, activity results, and time events. Taking the VP in Fig. 1.3 as the example, while there are two possible fixed activity results (i.e., *Pass* or *Fail*), the total number of possible system states (i.e., nodes in the process) increases exponentially along with the growth of either activity choices or time events. So, the corresponding shape size of a JVCS, as well as the number of all possible JVCSs, also increases exponentially. Therefore, given a complex engineered system, the resulting strategy space is high-dimensional, and it is hard to explore all possible strategies.

1.4 Research Question and Objectives

The overarching goal of this research is to develop quantitative approaches that enable the design of JVCSs that account for both VAs and CAs as decisions. To achieve this goal, this dissertation seeks to answer the research question: *what quantitative methods and mathematical models enable identifying near-optimal JVCSs with VAs and CAs as decisions in large verification spaces?* To answer this question, this research addresses the following objectives:

Objective 1. *Model a VP that includes VAs and CAs as independent decisions.*

As a first step, a mathematical model is built to capture all essential elements of a VP in the context of engineered systems. These essential elements include engineered systems, activities, mutual effects between system and activities, and utility function. The interrelationship between these essential elements should also be explained clearly. In particular, as a distinct feature of the VP in this dissertation, the effects of CAs on other essential elements need more investigation because the accurate description of these effects can contribute to the effective modeling of VPs.

Objective 2. *Solve for an optimal JVCS given a VP in small strategy spaces.*

Given a VP model that contains two sets of activity choices for VAs and CAs for all system states, it is essential to design a JVCS that can determine the optimal choices at each state. This dissertation has developed a method to generate an optimal JVCS that yields the maximum expected utility of system verification. In particular, the size of the activity choices is controlled as small verification spaces to ensure the optimal JVCS can be found. Once the method is successfully developed, some comparative analysis may be made to compare the optimal JVCSs under different conditions of a VP.

Objective 3. *Explore and find near-optimal JVCSs given a VP in large strategy spaces.*

The primary goal is to generalize the findings of Objective 2 to the VP problems of large strategy spaces. The design of a JVCS becomes complex when there is a large strategy space of a JVCS, as discussed in Section 1.3. In those cases where it is hard to obtain an optimal JVCS, it is a choice to obtain a near-optimal solution, that is, one that is not guaranteed to be optimal but is sufficient for system verification in practice. This research has developed methods to explore the large strategy space for a near-optimal JVCS.

1.5 Uniqueness of This Research

This work is unique in the sense that it views system verification problems as a decision-making process with two types of decisions. It also provides a comprehensive framework to solve this kind of decision-making process. Specifically, the main novelty of the proposed work is three-fold:

1) It leverages uncertain evidence to model the effects of CAs, which facilitates the understanding of CAs in a VP. Both VAs and CAs are unified as a Bayesian inference process, which is not investigated in previous works.

2) It integrates two kinds of decisions as a decision process, while the paradigms in previous works only treat VAs as decisions. A novel dynamic programming method is also presented to solve for the exact solution of the decision process, which proves that the proposed paradigm outperforms previous ones with higher expected values.

3) It explores large strategy spaces that consist of two types of decisions, which are not studied in previous works. Some novel tree search methods are proposed to solve complex verification planning problems in realistic engineering scenarios.

1.6 Dissertation Organization and Structure

This dissertation is structured as follows:

Chapter 2 describes the literature review and summarizes the main findings in the existing literature.

Chapter 3 describes the research methodology, including the tasks that will be carried out and the methods that will be employed.

Chapter 4 presents the hybrid verification and correction framework to model the impacts of both VAs and CAs on engineered systems.

Chapter 5 presents the proposed EPVP with correction decisions and a mathematical approach to design JVCSs in small strategy spaces.

Chapter 6 presents the proposed UCB-based tree search approach to design JVCSs in large strategy spaces.

Chapter 7 summarizes the dissertation’s contributions, limitations, and future work.

CHAPTER 2: Literature Review

2.1 Taxonomies

This chapter provides an overview of previous research related to the three research objectives in Section 1.4. First, the field of system verification is summarized as a core theme. Second, as the foundation of system verification, Bayesian network models and some related studies about uncertain evidence are discussed to identify work that can potentially be used to model the coordination of VA and CA decisions. Finally, some specific models and methods, including sequential decision-making, bandit problems, and ensemble learning models, are reviewed as a preparation to realize Objectives 2 and 3.

The taxonomies and correspondent keywords used to search and classify previous research are listed in Table 2.1. The areas listed in Table 2.1 are presented and discussed sequentially in the subsequent sections. All these studies are searched and collected from Google Scholar.

Table 2.1. Taxonomy elements and keywords

Taxonomy element	**Keywords**
System verification	System verification, verification process, verification planning, verification strategy, correction activity.
Bayesian network	Epistemic uncertainty, belief dependency, Bayesian network, probabilistic causal interaction model.
Uncertain evidence	Hard evidence, soft evidence, not-fixed probabilistic evidence, virtual evidence.
Sequential decision-making	Sequential decision-making under uncertainty, Bayesian network, troubleshooting, AND/OR tree, AO* algorithm, dynamic programming, backward induction, value function.
Bandit problems	Multi-armed bandit problem, upper confidence bound, UCB1, regret analysis, Monte Carlo tree search, single-player MCTS.
Ensemble learning models	Ensemble learning, decision tree, random forest, gradient boosting, concept drift, dynamic ensemble learning.

2.2 Mathematical Approaches for Verification Planning

System verification consumes a large proportion of development efforts, which can even be 40-50% of the total development effort [43]. Verification planning has traditionally relied on qualitative assessments performed by subject matter experts and industry standards derived from the collective experience of multiple experts [20]. Given the likely suboptimality of such approaches, methods underpinned by mathematical models have been proposed to support verification planning. Until now, three categories of mathematical methods have been proposed in the literature. The first category follows the experience of practitioners and project experts to propose verification strategies, which employ a conceptual model of verification planning [44, 45, 46]. The second one includes those studies in economics and management science that seek to manage the time and cost of testing strategies [47, 48]. The third one belongs to the field of systems engineering, where mathematics and optimization are applied to verification planning [20, 40, 49, 50]. The third category is mainly reviewed here because it is directly related to the life cycle of system development, which is the scope of this dissertation.

In the third category, fundamental work was done by Whiteside and Reimann [51]. They proposed a methodology for quantitatively assessing the effects of alternate verification, validation, and testing business practices on system life cycle costs. Engel and Barad extended this model as a decision problem where a verification strategy is defined as a policy, for a given system life cycle, under which, a subset of activities are fully performed, another subset of activities is partially performed, and the remaining activities are not performed at all [49]. Later, a decomposition method was proposed to find the optimal decision strategy [40]. The basic idea of this method is to decompose the decision process into separate decisions for each activity and use the simulation technique to generate strategies.

However, these models present three significant weaknesses. First, the dependence between systems and activities is not captured, even though it influences the selection of activities. A BN-based model was proposed in response to this gap to capture the dependencies between VAs [52]. Second, there is often a lack of knowledge about a system at the beginning of its development [53]. As a result, designing a fixed or static verification strategy early in the system development is likely suboptimal [42]. One solution to this problem is to quantify the impact of the designer's domain knowledge and problem framing on the information collection with a descriptive model [54]. Chaudhari et al. [55] also compared various descriptive models to describe information acquisition decisions when multiple information sources are present and the total budget is limited. Another solution is to use set-based design to collect information progressively during the process [56, 57, 58, 59, 60]. For example, a set-based design method was proposed in response to this gap to derive dynamic verification strategies that could adapt according to the progression of system development and verification [42, 61]. Third, previous verification planning models are built for single-firm cases, while large system development projects involve multiple firms and their coordination [62]. Thus, Kulkarni et al. [63] used game theory and incentive theory to model multiple-firm scenarios and investigated the alignment of verification strategies between multiple firms.

While system verification relies on CAs to eliminate errors, the methods discussed in the previous paragraphs simplify CAs as properties of VAs instead of as dedicated decisions during the verification planning. For example, the decomposition approach [40] modeled CAs as partial costs embedded in every verification activity. Kulkarni et al. [64] assumed that CAs always happen if the system is not in its ideal state. Xu et al. [42, 65] consider CAs to be a direct action based on the confidence threshold of a system parameter. In response to this gap, the BN-based verification model proposed in [52] was extended to model CAs [66]. This dissertation leverages such a model to integrate VAs and CAs as dedicated decisions in a verification planning problem.

2.3 Modeling Complex Systems with BNs

When developing new systems, the emergence of errors or undesirable effects that lead to dissatisfaction with the requirements is closely related to system development's uncertainty [67, 68, 69, 70]. Here, it is not distinguished if the source of the error is aleatory or epistemic, as both are significant in system development [71]. However, because verification activities primarily target epistemic uncertainty, that is, a lack of knowledge associated with whether the system meets its requirements [72], only epistemic uncertainty is studied in the context of system verification. Many approaches have been introduced to characterize epistemic uncertainty, including belief theory [73], fuzzy theory [74], possibility theory [75, 76], and generalized p-boxes [77]. Because belief theory has effectively been applied to build epistemic uncertainty models in system development [78, 79, 80], it is suggested that belief theory is a promising modeling paradigm to capture the subjective nature of how engineers interpret the information provided by VAs.

Furthermore, it is necessary to capture the belief dependencies between elements of an engineered system because the engineered systems of interest in this dissertation can be conceptualized as recursively formed by lower-level elements [81, 82]. That is, a system error or undesirable effect may be caused by an error in one of its elements, an error in the integration of the elements, or both. Various methods have been developed to capture this aspect, such as Bayesian networks (BN) [83], the transferable belief model [83, 84], the Fuzzy cognitive map [86], or the belief rule-based method [87]. Among them, BNs have been predominant in dealing with error diagnosis and have been effectively used in a broad range of fields, such as economics, accident analysis, risk analysis, and cognitive sciences [88]. BNs can directly capture uncertainty about the system state as beliefs and all dependencies between system parameters and between system parameters and VAs as a network, making them promising as a fundamental method to model VSs [52].

A BN is a probabilistic graphical model that represents a set of random variables and their conditional dependencies via a directed acyclic graph [89]. Structurally, a BN consists of nodes and arrows between the nodes. Nodes represent the random variables, which contain knowledge as an elicited distribution, and arrows between nodes represent the probabilistic dependencies between the nodes [90]. Due to its good model transparency [91], BNs have been widely adopted for statistical representation and Bayesian inference. The basic BN has also been extended to serve more sophisticated modeling needs, such as dynamic BN (which captures the change in beliefs as a time series model) [92, 93], object-oriented BN (which allows large-scale systems to be described in terms of inter-related objects) [94], and subjective BN (which generalizes BNs to the theory of subjective logic) [95, 96]. The flexibility of BNs is leveraged in this work to incorporate various types of CAs into the verification strategy models developed in [17, 52].

Even though BNs have been widely applied to update beliefs in complex systems, there is one limitation in the representation of BNs. As the conditional probabilities of a BN are generated by enumerating all possible state combinations of network parameters, it is hard to interpret sources of uncertainty about the parameters, especially when the parameter has many parent ones. This problem makes it hard to understand and update the prior probabilities directly. Therefore, as shown in Section 4.1.2, probabilistic causal interaction (PCI) models are leveraged to help interpret the sources of uncertainty in this research.

2.4 Uncertain Evidence in BNs

In BNs and Bayesian analysis in general, the term 'evidence' corresponds to new information about parameters of a distribution coming from the data [97]. This evidence plays a central role in Bayesian inference by triggering a belief update. In previous studies about system verification, evidence captures the information provided by the results of verification activities [52, 97, 98].

Two types of evidence may be employed: hard (certain) evidence and uncertain evidence. There are also three subtypes of uncertain evidence: soft evidence, not-fixed probabilistic evidence, and virtual evidence [99]. Hard evidence refers to the instantiation of a set of variables to their values without uncertainty [99]. Soft evidence [100] (also called fixed probabilistic evidence) specifies the probability distributions of some variables. These distributions cannot be changed by further information. Not-fixed probabilistic evidence also specifies the distributions of variables, which is similar to soft evidence. However, these distributions can be modified by the propagation of new evidence. Virtual evidence [83] (also called likelihood evidence) is represented by a likelihood ratio (LR) that represents the strength of confidence toward the observed event [101]. While the two subtypes of probabilistic evidence directly represent uncertainty as distributions, the uncertainty in virtual evidence comes from the unreliability or imprecision of the source of information [101].

Hard evidence is arguably the most common type in BN applications, but applications of the other three subtypes of *uncertain* evidence have also been attempted. For example, Gowadia et al. [102] used soft evidence to develop a framework for intrusion detection. Subramanya and Bilmes [103] used virtual evidence to reduce the amount of supervisory training data for automatic speech recognition systems. Corradi et al. [104] used soft evidence to classify individuals for forensic purposes. However, to our knowledge, *uncertain* evidence has not been used in systems engineering research, particularly concerning the design of verification strategies. In this dissertation, *uncertain* evidence will be leveraged to capture different CAs' effects on BNs, as shown in Section 4.2.2.

2.5 Sequential Decision-making under Uncertainty

Sequential decision-making is a procedural approach to decision-making that aims to find a policy that maximizes the expected return for all possible initial system states, where earlier decisions influence the later available choices [105, 106]. Sequential decision-making under uncertainty is referred to those problems where the results of a decision follow a random distribution, such as bandit problems, path planning under uncertainty, and system verification. For example, while the inference of BNs depends on the collected evidence of network nodes, how to observe network nodes and collect evidence is a decision-making problem under uncertainty [107]. Two types of sequential decision-making problems under uncertainty are distinguished according to whether all following states of a decision should be further planned. If only one following state should be planned after a decision, finding the policy requires determining the set of optimal actions that are connected as a path. In contrast, if it is necessary to plan all following states, the optimal policy can be represented as an AND/OR tree where all actions of a decision have OR relationship and all results of an action have AND relationship. While the first type is widely studied in the fields related to reinforcement learning [108], the second type has been explored in a variety of research domains, such as system verification [17], disease diagnosis [110], adaptive testing [111], and troubleshooting [112].

The critical difference between the methods of these two types lies in the calculation of rewards. That is, the reward of a state depends on all following states in the second type. Until now, two major approaches have been proposed to solve for the second type's policy. The first approach is heuristic search methods originating from AO* algorithms for AND/OR trees. Nilsson [113, 114, 115] first described a version of AO* for searching AND/OR trees to find a solution in the form of a tree. Martelli and Montanari [116] generalized this algorithm for searching AND/OR graphs to find a solution in the form of an acyclic graph. The graph-search version is more efficient than the tree-search version when the same state can be reached along different paths because it avoids performing duplicate searches [117]. Hansen and Zilbersterin [117] proposed a generalization of AO*, called LAO*, to find solution graphs with loops. As the solution of verification planning can be presented as directed acyclic graphs, only the AO* algorithm is presented for system verification, as shown in Algorithm 2.1. All these AO* algorithms use admissible heuristic functions to find exact solutions. For example, Vomlelova´ and Vomlel [118] applied the depth-first search algorithm with pruning and the AO* algorithm to find a troubleshooting strategy. Warnquist et al. applied AO* to solve the troubleshooting problem of heavy vehicles [119]. Even though these heuristic search algorithms can improve search efficiency by pruning unnecessary states through value bounds, an admissible heuristic function should be determined first. Because of the addition of CAs in verification planning, it is hard to find an admissible heuristic function that can be used to prune unnecessary states. Therefore, this approach cannot be applied in this dissertation.

Another approach is dynamic programming (DP) which solves the problem by breaking it down into simpler sub-problems in a recursive manner. DP methods can be classified into exact DP methods and approximate DP methods according to whether the solution policies are exact or approximate. Exact DP methods can be classified into two classes: direct methods and successive approximation methods [120]. Direct methods, such as backward induction, can obtain the solution strategy directly according to the recursive relationship (i.e., Bellman equation), which has been applied to solve various problems, such as verification planning [121, 122], petroleum exploration [123], circuit design [124], and speech/music discrimination [125]. However, it is necessary to interpret the recursive relationship with explicit decision graphs, such as decision trees or influence diagrams. Successive approximation methods, such as value iteration and policy iteration, begin with a rough approximation of the solution strategy and solve for the optimal one iteratively. They are usually applied under the framework of Markov decision processes [126, 127]. For example, Petousis et al. applied the QMDP algorithm to solve the sequential decision-making problem in lung cancer screening [128]. Velimirovic et al. used the policy iteration method of dynamic programming to locate faults in power distribution networks [129]. Even though successive approximation methods do not need explicit decision graphs, they are less efficient than direct methods because they require lots of iterations over all states to obtain the optimal strategy. This dissertation proposes an order-based method to improve the efficiency of successive approximation methods, as shown in Section 5.1.3.

Algorithm 2.1. AO* Algorithm [117]

---

1. The explicit graph Ψ initially consists of the start state S_1

2. While Ψ has some nonterminal tip states:

2.1 Expand some tip state S_m of Ψ and add any new successor states to Ψ.

2.2 Assign each new nonterminal state S_{m+i} with the heuristic value $h(m)$ and each new terminal state S_{m+j} with 0.

2.2 Update state values and mark best actions:

2.2.1 Create a set G that contains the expanded state and all of its ancestors in the explicit graph along marked action arcs.

2.2.2 Repeat the following steps until G is empty:

2.2.2.1 Remove from G a state $S_{m'}$ if no descendent of $S_{m'}$ occurs in G.

2.2.2.2 Set $U(S_m) = \min_{a \in A(S_m)}[C_i(a) + \sum_{S_{m'}} p_{S_m,S_{m'}}(a)U(S_{m'})]$ and mark the best action for S_m.

5.3 Return the optimal solution graph.

When the state space is large, exact DP methods fail in finding exact solutions due to the curse of dimensions [130]. Instead, approximate DP methods can be used to approximate the decision-making process. These approximate methods can be broken down into four classes according to the types of approximate policies: myopic policies, lookahead policies, policy function approximations, and value function approximations [130]. While myopic and lookahead policies minimize costs for one or several finite time periods, ignoring the effect of a decision now on the later time periods, they are limited if the long-term effects are important. Policy function approximations are applied when the structure of a policy is fairly apparent [131]. Because verification strategies have VAs and CAs in this dissertation, the structure of a policy suffers the dependency relationship between VAs and CAs. So, the fourth type of value function approximations is mainly related to verification planning. Many different approximation models of value functions have been proposed, such as lookup tables of Q learning [132], linear regression [133, 134], support vector regression [135], Gaussian process regression [136], and neural network [137]. However, as far as I know, there are no value function approximation methods for the second type of sequential decision-making under uncertainty. So, this dissertation proposes a function approximation method based on random forest regression to design an approximate DP method, as shown in Section 6.1.3.

2.6 Bandit-based Methods

The multi-armed bandit problem is a sequential decision model in which an agent needs to decide which arm of K different slot machines to maximize their reward while improving their information simultaneously [138, 139]. This problem provides a paradigm of the tradeoff between exploration (trying out each arm to find the best one) and exploitation (playing the arm believed to give the best payoff). The reward of each arm is a random variable x_k with an unknown distribution, and the distributions of all arms are independent from each other. This bandit problem aims at finding a policy that determines which bandit to play based on previous trials. The performance of a policy is commonly measured by the agent's regret [140, 141], which is the expected loss due to not playing the best bandit.

In a seminal paper, Lai and Robbins [142] made a regret analysis to find an asymptotic lower bound on the growth rate of total regret for a large class of reward distributions. The regret bound is O(ln*n*), where n is the overall number of plays. Since then, various online policies have been proposed, among which the UCB1 policy developed by Auer et al. [143] is considered the optimal [144, 141]. The UCB1 policy is to play a machine k that maximizes $\overline{\mathrm{x}}_k + \sqrt{\frac{2\ln n}{n_k}}$, where $\overline{\mathrm{x}}_k$ is the average reward obtained from machine k, and n_k is the number of times machine k has been played so far. That is, this policy can balance the exploitation of the bandit currently believed to be optimal with the exploration of other bandits that currently appear suboptimal but may turn out to be superior in the long run [145]. This UCB1 policy achieves logarithmic regret uniformly over time (not asymptotically) without any prior knowledge regarding the reward distributions. However, the UCB1 is limited by its assumption that the optimal machine is determined by comparing the expected average rewards of all machines. That is, the UCB1 policy would lose its rationality if the expected average reward is not the measure of a machine's performance.

To solve the tree search problem, Kocsis and Szepesvári [146, 147] proposed the use of UCB1 as a tree search policy, which is called the upper confidence bound for trees (UCT). Its formula is

$$\overline{\mathrm{x}}_k + D_1\sqrt{\frac{2\ln n}{n_k}}, \tag{2.1}$$

where D_1 is constant. While the UCT follows the assumption of expected average rewards to determine upper confidence bounds, some variant policies have been proposed as extensions of the UCT to adapt to their domains where the assumption is invalid. Two domains are found to be related to system verification. First, Schadd et al. [148] extended bandit problems to single-player games with perfect information where the expected maximum value rather than the expected average value is used to choose moves. They proposed a single-player MCTS policy (SP-MCTS) that added a third term to the UCT rule:

$$\overline{\mathrm{x}}_k + D_2\sqrt{\frac{2\ln n}{n_k}} + \sqrt{\frac{\sum x^2 - n_k \bullet \overline{x}^2 + D_3}{n_k}}, \tag{2.2}$$

where D_3 is a constant that artificially inflates the standard deviation for infrequently visited nodes. However, as the averaged values are used as the first item to represent the performance, the phenomenon that a good move is hidden by previous records of bad moves may still occur [149]. Second, Galichet et al. [150] studied a risk-aware bandit problem that measures a machine's performance with the conditional value at risk α (CVaR), which is the average of the lowest quantiles of the reward distribution with level α. As the goal is to find the machine with maximal CVaR, machines are selected with the best lower confidence bound on their CVaRs, which is called MARAB policy $CVaR_k - D_4\sqrt{\frac{\ln(\lceil n\alpha \rceil)}{\lceil n_k\alpha \rceil}}$, where $D_4 > 0$ is a constant that controls the exploration vs. exploitation tradeoff. As α decreases, the risk-aware bandit problem boils down to a standard max-min optimization problem. However, the calculation of the lowest quantiles results in the storage issue when the bandit problem is large because all previous trials must be recorded to determine the quantiles. To solve these problems, this dissertation proposes an upper confidence bound rule for repeatable bandits, as shown in Section 6.1.1.

2.7 Tree-based Ensemble Learning Models

Ensemble learning is an effective technique that has increasingly been adopted to combine multiple learning models to improve overall prediction accuracy [152]. These ensemble techniques can alleviate the small sample size problem by incorporating multiple learning models to reduce the potential for overfitting the training data [153, 154]. Decision trees are commonly used in ensemble learning models because decision trees are sensitive to small changes in the training set [152]. So, the scope of ensemble learning models is narrowed down to tree-based models. These ensemble learning models have been applied to many fields, such as bioinformatics [154], defect prediction [155], and remote sensing [156]. However, as far as I know, tree-based ensemble learning models have not been used in verification planning. So, a tree search method enabled by ensemble learning is presented in this dissertation, as shown in Section 6.1.3.

Some common types of ensemble learning include bagging, boosting, and stocking, which are realized as some basic models, such as random forest (RF) [157], gradient boosting model [158], XGBoost [159], LightGBM [160], and CatBoost [161]. Several studies [162, 163, 164] have been conducted to test the performance of these models, and each of them has its own merits. For example, Bentéjac and Csörgő [163] found that CatBoost obtained the best results in generalization accuracy and AUC, while LightGBM had the fastest training speed in the studied datasets. Ibrahim et al. [165] recommended CatBoost algorithm for better prediction of loan approvals and staff promotion. In another study about highly imbalanced Big Data [164], XGBoost was better than CatBoost because of its shorter training time. In addition, the applications of tree-based ensemble learning models are not limited to these basic models. For example, Jhaveri et al. [166] proposed a weighted random forest along with AdaBoost to predict the success rate of Kickstarter campaigns. Zhang et al. [167] combined RFs with XGBoost to establish the fault detection framework of wind turbines. Zeinulla et al. [168] proposed a fuzzy random forest model to diagnose heart disease with incomplete and dirty datasets. So, the selection of ensemble learning models depends on the characteristics of the research task and datasets.

It is noticeable that ensemble learning has also been used in nonstationary environments, where the underlying data distribution changes over time (i.e., concept drift). Elwell and Polikar [169] proposed an ensemble of classifiers-based approach for incremental learning of concept drift. The proposed algorithm collects consecutive batches of data, trains one new classifier for each batch of data, and combines these classifiers using a dynamically weighted majority voting. Later, Yin et al. [170, 171] introduced a comprehensive hierarchical approach called dynamic ensemble of ensembles. It includes two stages. First, component classifiers and interim ensembles are dynamically trained. Second, the final ensemble is learned by exponentially weighted averaging with available experts. However, all these methods use the weighted averaging method to fuse the information of all components. Thus, they cannot be directly applied in verification planning that searches for optimal strategies. Instead, the proposed tree search method in this dissertation only uses the latest component to solve the concept drift issue.

CHAPTER 3: Research Methodology

This chapter presents the methodology to achieve the three objectives in Section 1.4. First, this chapter gives an overview of the research methodology and the structure of all tasks. Then each of them is described in the following sections.

3.1 Overview of the Research Methodology

This section presents an overview of the research methodology. A three-phase research methodology is adopted, as shown in Fig. 3.1, where the three phases aim to achieve the three objectives in Section 1.4, respectively. In each phase, several tasks have been presented to achieve each objective in steps, as shown in Fig. 3.2. If all tasks are completed, the outputs or conclusions of each phase are the solutions to the research objective. It is also noticeable that dependence relationships exist between the three phases. The engineering context in the first phase is the foundation of the tasks in the rest two phases. The proposed paradigm and approaches in the second phase can be used as a benchmark to guide the strategy search for large networks in the third phase. Thus, the three phases are connected with arrows according to these dependence relationships.

Each task is presented with its purpose to simplify the realization of objectives. In the first phase, there are two tasks, including modeling the engineering context of system verification and designing the data synthesization method for engineering contexts. An engineering context means the set of elements or variables required for system verification. An engineering context serves as a platform that provides decision sets of VAs and CAs. In addition, the data synthesization method should be provided to facilitate the analytical study of decision-making. In the second phase, the primary goal is to solve verification planning problems in small strategy spaces. This phase has three tasks, including mathematical modeling of EPVPs, design of exact JVCSs, and comparative analysis. First, an EPVP model is presented to build the process model for verification planning. Then a task is assigned to propose a method to solve for the exact JVCS given an engineering context. As an extension, the impacts of CAs on the exact JVCS will be explored with a comparative analysis.

In the third phase, the goal is to solve for JVCSs in large strategy spaces. Due to the difference in action spaces between VAs and CAs, a heuristic tree search approach has been proposed to solve for a near-optimal JVCS. This approach is presented as a set of three tasks that have a hierarchical structure. First, the verification planning problem is simplified as a bandit problem and a bandit-based rule is proposed to solve the bandit problem. Second, as the number of system states is so huge in large systems that it is impossible to examine all of them, it is necessary to reduce, compress, and generalize the information of system states. A function approximation method is presented to approximate the collected information. Finally, a heuristic tree search algorithm is presented to search for the near-optimal JVCS using the proposed methods in the previous two tasks.

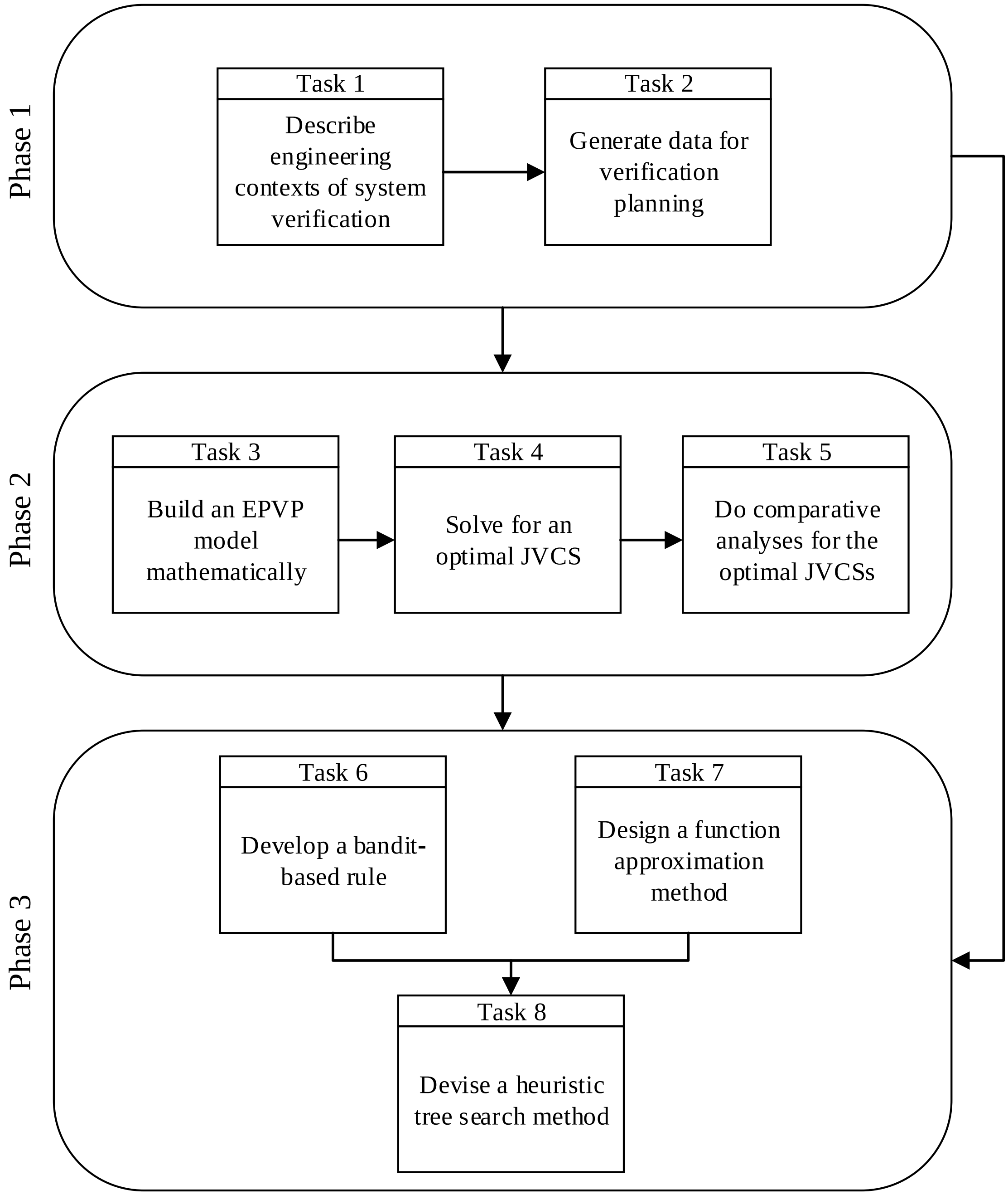


Fig. 3.1 Structure plot of all tasks

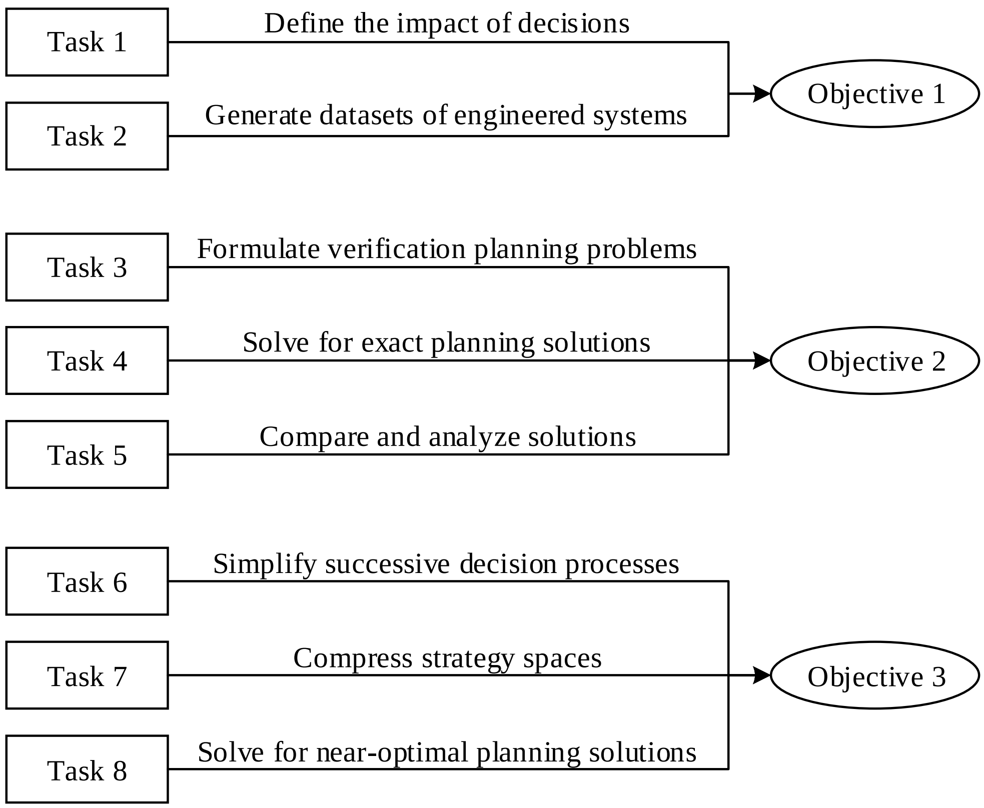


Fig. 3.2 Relationship between tasks and objectives

3.2 Task 1: Describe Engineering Contexts of System Verification

As discussed in Section 1.2, system verification aims to maximize confidence in verification coverage, minimize the risk of undetected problems, and minimize invested effort [20]. As an engineering context is defined to contain essential elements of system verification, three types of basic elements have been found to represent a system verification model, including system components' interrelationship, correction effects, and utility model, which are the basis of Objective 1. Thus, this task focused on describing engineering contexts from these three aspects.

First, as discussed in Section 2.3, a BN model is used to represent the interrelationship between system components as it can directly capture uncertainty about the system components as beliefs and all dependencies between system components as a network [52]. There exist two types of network nodes for two types of system components. The first one is hidden nodes representing system parameters whose states cannot be observed directly. The second one is observable nodes for VAs, which can be tested to obtain results. These two types of nodes are connected according to their dependency relationships, shown as a list of conditional probability tables (CPT). From the aspect of system verification, the implementation of VAs for an engineered system is equivalent to testing the observable nodes of a corresponding BN to discover their results. Once the test results of a VA are obtained, the effects of such VA can be reasoned by the Bayesian inference.

Second, as correction decisions are a distinct feature of this dissertation, the effects of CAs should be incorporated into the BN model. As discussed in Section 2.2, the effects of CAs are simplified as properties of VAs in current practice. However, this practice fails to capture the effects of CAs from three aspects. First, CAs can make some evidence collected by VAs invalid. Because a CA updates some system parameters, the evidence related to these parameters may not be applied anymore or at least need some extra verification to check whether the evidence is still effective. Second, CAs can update the beliefs of the system parameters (i.e., the joint distribution of a BN). For example, if a repair activity is executed to repair a system component, the confidence that this system component functions well can be improved, even though this change of confidence is uncertain beforehand. Thus, uncertain evidence is used in this dissertation to capture the belief changes. While CAs have three types, including rework, repair, and redesign, how to represent belief changes of all these types is the challenge. Third, CAs can also change the network structure of an engineered system. For example, if a system component is removed after repair, the corresponding parameter can be removed from the BN. However, BNs are always used with a fixed network structure because the network nodes are restricted by node dependency and the joint distribution of all nodes. So, a general method was needed to update the network structure while simplifying the elicitation work of BNs.

Third, while the uncertainty and dependency of system verification are captured in the first two aspects, the expected utility is also crucial in system verification as it plays a decisive role in determining VSs in practice [5]. So, the third task extended the previous study [42] to build the expected utility that measures the performance of system verification on an engineered system. The expected utility is a function of three utility terms in this task. The first term is the system revenue generated when a verified system is deployed. The second term is the activity cost for VAs and CAs, which is a fixed amount of financial resources necessary to conduct each activity on the engineered system during the VP. The third term is the failure cost, which estimates the financial resources to compensate for the disastrous effects caused by some verification results, such as the failure in trial flights. While the previous research only considered the first two terms, this task added the failure cost as it also exists as part of utility changes in practice.

3.3 Task 2: Design a Data Synthesization Method for Engineering Contexts

The experiment examples of engineered systems were taken from historical research projects, and all data of these experiment examples were generated synthetically. The purpose of this task was two-fold. First, this task aimed to understand engineered systems' data structures and summarize them as a data synthesization method. Second, the designed data synthesization method was used to provide specific datasets that describe all activities and their effects on engineered systems. Only when this task was completed, it was possible to solve verification planning problems and do comparative analysis mathematically. This task consisted of three steps: network elicitation, belief generation, and utility estimation.

First, the set of system components and their interrelationship were elicited to present the structure of engineered systems. For example, to search for a near-optimal JVCS in large strategy spaces, the optical satellite instrument presented in [52] can be used as a study case. The structure of the BN is shown in Fig. 3.3. There are 50 nodes, and each node has its physical meaning, as described in [52]. Hidden nodes are represented as circles and observable nodes squares. To facilitate a scalability study, it was also required to scale down the whole network by pruning nodes not directly related to the target node θ_3. This pruning procedure is not arbitrary, as the node set of the pruned network should still be meaningful in practice. One possible pruned network is outlined by the dashed line in Fig. 3.3. Then the strategy search may start with the pruned network and then increase to the whole network. A comparative analysis can be made by comparing the strategies of pruned and whole networks.

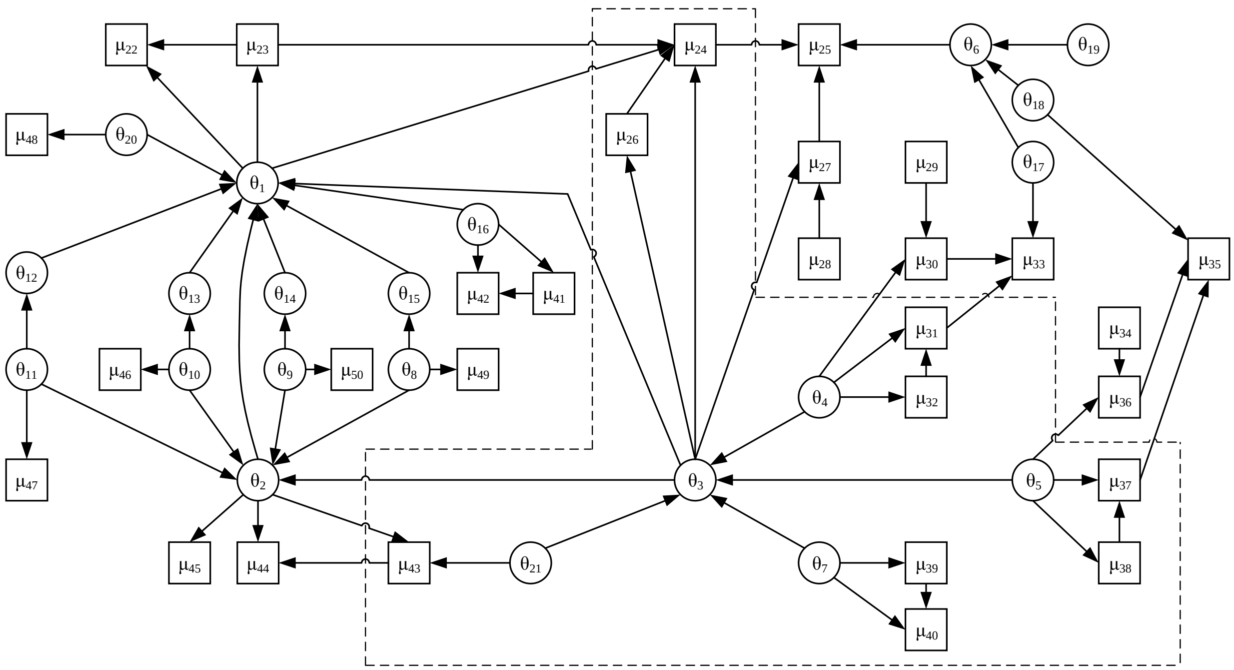


Fig. 3.3 The full BN for a case study

Second, belief information was generated for all system components following their physical meanings in an engineered system. In this dissertation, the probabilistic causal interaction model [83, 172, 173, 174] was used to analyze the inherent structure of CPTs of system components. That is, each CPT of a node was decomposed as a set of cause factors, and each cause factor represented a dependency relation between two nodes. For each cause factor, its distribution of each cause factor was assumed to follow Gaussian distributions whose parameters were estimated according to the physical meanings of each parent node [52]. Then, a belief value was sampled from the Gaussian distribution of this cause factor. All CPTs can be generated by fusing all belief values to simulate a real system. In addition, the correctness of the CPTs was verified by running Bayesian inference for all network nodes to ensure their posterior probabilities are acceptable in practice [52].

Third, all revenue and cost items were also generated synthetically as the utility data. The revenue item was estimated according to the overall income of verification planning. All cost items were estimated through a general elicitation process in three steps. First, all activities were divided into several classes according to their definitions in [52]. The specific classes were determined according to the context of engineered systems. Then in each class, there was an activity cost for each activity and a failure cost for each VA. To simplify the problem, it was assumed that each CA has only one correction choice (i.e., either rework, repair, or redesign) in this task. So, there was only one correction cost for each system parameter. These cost items of each class were assumed to follow their independent Gaussian distributions, each of which followed its physical meaning. Finally, specific cost values were sampled randomly from these Gaussian distributions as activity costs and failure costs separately.

3.4 Task 3: Establish EPVPs Mathematically with All Process Constraints

The quantitative analysis of verification planning cannot be separated from the mathematical modeling of EPVPs. So, it is necessary to list all basic conditions of an EPVP which is also the problem formulation of Objective 2. In this dissertation, all activities were conducted discretely within certain time events. Here each time event represented a phase of the product development. Without loss of generality, it was assumed that at most one VA and CA can be conducted at each time event. Then, a set of activities within T time events can form a path that is a specialization of an EPVP. Each activity is conducted at the beginning of its corresponding time event, i.e., $t = 1, ..., T$. Once the EPVP is completed, the system is deployed at the end of the T^{th} time event. In addition, the path of an EPVP is subject to some process constraints from the practice. For example, each CA must be conducted after collecting a verification result at each time event. The reason is that CAs are triggered by the false results of VAs. So there is a sequential constraint between each pair of them.

Even though the basic conditions and process constraints were summarized according to the practice, it did not present all assumptions explicitly for an EPVP. The first assumption is that there is only one decision-maker in verification planning. While verification planning can be solved by a team or a group of managers, they can be viewed as one entity making decisions for both VAs and CAs. This assumption is embodied in two ways. First, all information collected during the VP is saved without any loss and can be observed for decision-making at any time. Second, no matter how the decisions are elicited from a team, the strategy is organized as a deterministic policy for all possible system states, each of which has only one decision choice.

The second assumption is that system states are partially observable to the decision-maker. As the goal of system verification is to diagnose some hidden system components, the EPVP model can be applied to only those systems whose components are partially observable or hidden. However, with the BN method and belief theory, the belief of a system parameter is represented by a continuous variable *confidence level*. Then the EPVP model can be viewed as a fully observable model whose state is a belief vector. The belief update can be realized by Bayesian inference. It contributes to understanding the EPVP model by clarifying all these assumptions.

3.5 Task 4: Design a Modified Backward Induction Method

As discussed in Section 2.5, dynamic programming can be used to solve problems that have suboptimal structures. The EPVP model shares this characteristic of suboptimal structure because the optimization of a state value can be recursively decomposed into suboptimal ones. So, this task started from this characteristic to propose a modified backward induction (BI) method to search for optimal JVCSs (i.e., Objective 2). Because optimal JVCSs can always be found with other benchmark methods, such as the value iteration method, these benchmark methods were considered to evaluate the proposed method.

There were three points worthy of discussion for the design of the modified BI method. First, as the update unit of backward induction is a system state, it was necessary to provide a state representation method for DP methods. While all activities had been provided in engineering contexts and the process constraints between them had been defined in the EPVP model, the state representation method was not given yet. So, a vector of the statuses of all activities was used to represent a system state. All beliefs of a system can be determined as long as activity results are provided. With the combination of all statuses, all possible states can be enumerated. It is noticeable that the prior beliefs of the system were assumed to be consistent during a VP. If the prior belief changed, it was insufficient to use the statuses of activities as they were only collected evidence.

Second, the design work adopted the basic idea of the BI that all following states are evaluated in a backward direction, and their optimal results constitute the strategy. The optimal JVCS was found by searching for optimal activities of all system states. It is noticeable that state transitions caused by VAs are stochastic rather than deterministic in an EPVP. As a result, even though a high utility can be obtained after reaching some system states in the long term, this high utility is discounted by the transition probabilities of the VP. So these transition probabilities should be considered during the backward induction process.

Third, because a JVCS is a set of activities at different system states, the same system state may reappear multiple times in a JVCS. Thus, reusing the induction results of system states can be considered to accelerate the backward induction process. The challenge was how to determine the dependency relationship of all system states. An analysis was made in Section 5.1.3 to determine it according to the process constraints of EPVPs. Then an order function was used to rank all possible system states. Then wherever a system state is in a JVCS, its value only needs one backward induction.

3.6 Task 5: Do Comparative Analyses for Optimal JVCSs

In Section 3.2, the effects of CAs were discussed as part of the engineering contexts. However, their impact on the JVCS still needed more exploration. With the modified BI method above, this task continued to perform comparative analyses on optimal JVCSs. The goal of this task was to deepen the understanding of the relationship between CAs and optimal JVCSs.

This task focused on two aspects. First, a sensitivity analysis was performed to understand the relationship between correction costs and optimal JVCSs. A set of various correction costs was created to test and compare the generated JVCS. As a result, for a particular JVCS solution, some cost range of each correction cost may be found for corresponding CAs. That is, within this cost range, the JVCS always remains the same for all choice nodes. On the other hand, as CAs influence the joint distributions of a system network, it was necessary to observe and calibrate such influence. That is, with the virtual evidence in Section 3.2, a set of different likelihood ratios was used to test the impact of the uncertain effects of CAs.

Second, a general CA framework was established for multiple types of CAs. In previous tasks, only two correction alternatives were considered (i.e., correction or no correction), which were insufficient to describe all CAs. So the proposed framework combined uncertain evidence and correction costs to define multiple CAs, such as rework, repair, and redesign, each of which has its likelihood ratio and correction cost that may be determined according to the practical context. The set of these CAs is used as the correction choices for the actual design of JVCSs. Based on this framework, the modified BI method can also be applied to obtain the optimal JVCS.

3.7 Task 6: Develop a Bandit-based Rule for Verification Planning

As a JVCS is a set of optimal activities at all possible system states, the verification planning problem can be decomposed into a set of decision-making sub-problems at different system states. So, to simplify the verification planning problem, this task only addressed the activity planning at one system state. As the expected reward of each activity is unknown, it can be viewed as a multi-armed bandit problem. Then the task was to develop a decision-making rule for this simplified bandit problem which attempts to find the optimal activity at a given system state. With this simplification, this task was organized into three steps.

First, while verification planning was simplified as a bandit problem, it was necessary to ensure all properties of verification planning were retained in the simplified bandit problem. So, the properties of activities and their results should be analyzed by comparing them with multi-armed bandit problems. For example, the expected value of an activity result is unknown if all following system states are not explored. So, it was reasonable to assume the results of a bandit followed an unknown distribution. However, with this assumption of randomness, there is also a constraint for each unknown distribution that its distribution has finite upper and lower bounds. The reason is that revenue and cost items are predefined as finite values, and the expected value is bounded by the maximum revenue and the sum of cost items. Thus, the distribution of each bandit has its support range.

Second, as bandit problems are solved by developing bandit-based rules, a related question is how to measure the performance of those rules. Regret theory is used to measure the value of difference for decision-making under uncertainty [175]. This value of difference is called regret to mimic human emotion regret. It has been applied to measure the rules and reduce regret for multi-armed bandit problems [176]. Thus, the regret analysis was adopted as the criteria for the simplified bandit problem.

Third, this task aimed to identify a bandit rule for the simplified bandit problem. The basic idea was to resolve the exploration and exploitation tradeoff between different bandits. That is, while those bandits with larger values deserve more exploitation, it is also important to explore other bandits to increase knowledge about them as it can help avoid locally optimal choices. One example of bandit rules was UCB1 [146, 147] that balanced such tradeoff by calculating both values and visit counts as the criteria of bandit selection. However, UCB1 and its extension rules used the average rewards of a bandit as its expected reward, while verification planning only concerned the maximum value of an activity when comparing different activities. Thus, the challenge was to develop a special rule for the simplified bandit problem to find the bandit with maximum values. In addition, the regret proof of this rule was also needed.

3.8 Task 7: Design a Function Approximation Method

In verification planning, the large strategy space is inevitable as long as the number of system parameters, activities, or time events is very large. So the challenge of large strategy spaces is how to compress a large number of system states. Two opportunities were found to reduce the difficulty caused by the large size. First, a VP is restricted by its process constraints, which means some system states are invalid during the strategy design. So, the first opportunity was to reduce the scalability by reducing the number of basic strategy elements, such as activities or time events. Second, as only a small proportion of system states is obtained as samples, most system states are unvisited, and the exploration always starts from scratch if there is no prior knowledge about these unvisited system states. So, if the knowledge of visited states can be reused to guide the search of unvisited ones, the search efficiency is expected to be boosted. Thus, this task focused on how to extract knowledge from sampled system states and predict state information to compress large strategy spaces.

This task followed the idea of function approximation that selects a function model to approximate the relationship between system states and some relevant factors of VPs, which was organized in three steps. First, it was necessary to analyze the properties of the collected datasets. For example, in a JVCS, there are dependency relationships between all activities. So, the collected system states are not independent from each other. With this finding, those function models may not be effective if they rely on the independence of samples. In addition, if system states are collected randomly, the target relationship may be a stochastic function, which is also a constraint for the function approximation models.

Second, the target relationship was established as a mapping from collected datasets to state information. There are three kinds of information sources in verification planning, including belief values of network nodes, statuses of activities, and cost terms. To simplify the mapping relationship, only the status of activities and posterior beliefs of target parameters may be selected as the input variables because they represent the real-time information of a system state. In contrast, prior beliefs and cost terms are constant along with VPs. Each VA has three status values {0, -1, 1} for not verified, verified with the Fail result, and verified with the *Pass* result, respectively. Each CA has two status values {0, 1} for not corrected or corrected, respectively.

The state information consists of expected values and visit counts. It is noticeable that node information also depends on the search rule (e.g., UCB1 rule). However, as search rules cannot be quantified as variables, they are not considered in this function approximation model.

Third, random forest regression (RFR) was used to realize the function approximation. One reason why RFR is selected is the variable types of the mapping function. That is, activity statuses are categorical and posterior beliefs are continuous. Random forest is one of the few models that accept this mixed type of input variables. Because the state information, including expected value and visit counts, is continuous, it also belongs to regression problems. So, RFR is appropriate for verification planning. Next, the standard training procedure of RFR was followed to approximate the mapping function, including data preparation, model learning, and cross-validation. In particular, the training procedure was conducted with some parameter analyses as RFR has its hyperparameters, such as tree numbers, tree depth, and training criteria. The goal of this training procedure was to minimize the risk function (e.g., mean squared error) of the training datasets.

3.9 Task 8: Devise a Heuristic Tree Search Algorithm

Given the bandit-based rule and function approximation models above, this task aimed to provide a heuristic tree search algorithm that can generate a near-optimal JVCS in large strategy spaces (i.e., Objective 3). Because the total number of possible states is huge in large-network systems, the modified backward induction method in Section 3.5 cannot be applied to update the state value of all system states. Instead, the idea of graph traversal algorithms [177] was followed to search for a near-optimal strategy tree in a forward fashion. This forward search method started the search from the nearest time events and expanded the strategy tree progressively along with the VPs. So, it enjoyed the advantage of prioritizing the closest states and reducing unnecessary efforts on states far from the starting states. Thus, the strategy space within the nearest horizon can be explored more often, and the resulting activities at the nearest system states are well-tested in forwarding search methods.

Three points were worthy of further exploration of heuristic tree search algorithms. First, A*-like tree search algorithms rely on the design of admissible heuristic function at all states [177]. That is, it is necessary to determine an exact upper bound value of each state. Then those states with larger admissible values are prioritized when comparing different states of a graph. However, if these admissible functions are not close to the true values, the search efficiency cannot be guaranteed. In verification planning, the effects of CAs on JVCSs are hard to be captured under all possible states because the expected value of a state may be significantly changed with trivial costs. So, the upper bound values cannot be determined accurately. Instead, the upper confidence bounds in bandit-based methods are a choice to approximate the exact upper bound value. Therefore, applying the bandit-based rule for tree search needed more effort in the algorithm design.

Second, the heuristic tree search also shows the local optimality issue. That is, the heuristic tree search process may get stuck in local optimums of strategy spaces. Two methods may be taken to reduce the risk of local optimum. The first one is to change the parameters of bandit-based rules as they directly control the convergence speed of tree search. The second one is to use function approximation methods to provide extra information for the exploration. As the state information of unvisited states can be predicted from knowledge of visited states, it is expected that the gaps between true values and record values of a state can be narrowed with the compressed knowledge of visited states.

Third, there were many design factors in the algorithm design. For example, a strategy tree can be explored by expanding all nonterminal tip nodes or a subset of nonterminal tip nodes. In addition, the node information of a strategy tree can either be updated whenever a new node is added to the tree or after the whole strategy tree is built. While a strategy tree has multiple branches, the parallel computing method can also be considered to accelerate the search efficiency, which is also a design factor. All these design factors required more testing in the algorithm design.

CHAPTER 4: Modeling CAs in the Context of Verification Strategies

This chapter presents a hybrid verification and correction framework that integrates CAs in verification strategies as stand-alone activities. The content of this chapter has been published in [66]. First, Bayesian networks (BN) and probabilistic causal interaction (PCI) models are used to capture the mutual dependencies between system parameters and verification activities. Second, the prior modeling framework is augmented by incorporating different types of evidence modeling. This incorporation is used to capture the impact that different types of CAs have on the system parameters and the confidence of their state, as well as on the information expected to be acquired through the execution of VAs. Third, a basic interaction loop (BIL) is used to integrate all VAs and CAs so that a generalized Bayesian inference process can be applied.

This chapter is organized as follows: Section 4.1 introduces BN and PCI models. Section 4.2 describes the proposed hybrid verification and correction framework to integrate VAs and CAs. Section 4.3 presents a demonstrative case to illustrate the application of the framework. Section 4.4 offers the discussion of this chapter.

4.1 Preliminaries

4.1.1 Basic System Verification Model

Consider that a given system can be decomposed into a set of elements ε_i, $i = 1, \ldots, I$, and assume that the objective of system verification is to verify relevant requirements for these elements[1]. Then a basic system verification model is conceived as a set of tuples of system parameters θ_i, $i = 1, \ldots, I$, and VAs μ_j, $j = 1, \ldots, J$ that provide information about such system parameters [17]. With the modeling framework presented in [52], a basic system verification model is built as a directed acyclic graph $\Omega = (\Theta \cup M, E)$, where Θ is the set of nodes, each of which represents one system parameter, M is the set of nodes that represent verification activities, and E is a set of directed edges that connect the different nodes. There are three types of directed edges: $\Theta \Rightarrow \Theta$, $\Theta \Rightarrow M$, and $M \Rightarrow M$, which capture the information dependencies between system parameters, between system parameters and VAs, and between VAs respectively. There are no directed edges from VAs to system parameters because an actual system parameter is not caused by the VAs; however, the results of the VAs causally depend on the system parameter. In the resulting BN, nodes representing VAs are treated as observable nodes (those whose value states can be observed directly), and nodes representing system parameters are treated as hidden nodes (those whose value states cannot be observed directly but are inferred from the values of the observable nodes).

Because the interpretation of the information provided by VAs is subjective [5], the information about system parameters is captured as beliefs. Without loss of generality, all nodes are assumed to be binary (i.e., two node states, such as Pass/Fail, True/False, or Compliant/Non-compliant). The nature of Bayesian analysis, and of BNs by extension, allows for easy removal of this restriction and use of any number of discrete values (e.g., also adding an inconclusive state to pass

[1] It is noted that verification may also be used to inform design choices and support performance optimization, as indicated in the introduction. The models presented here can be used for any of those purposes.

and fail) or even continuous belief distributions [178], which can be used to model VAs aimed at informing design choices or supporting performance optimization. The specific beliefs of a network node are presented as a conditional probability table (CPT). Each CPT summarizes the dependency relationships between a node and all its parent nodes.

4.1.2 Probabilistic Causal Interaction Model

Probabilistic causal interaction (PCI) models, such as the noisy OR-gate model [83], were initially proposed to alleviate the elicitation burden of CPTs [174, 179, 180]. In PCI models, a CPT is decomposed into a set of cause factors and the interaction between these cause factors is represented by combination rules. This mechanism aligns well with the verification strategy models used in this dissertation, given the causal interactions in system verification. While several PCI models exist, two types of PCI models are selected for system verification, one to derive belief distributions for system parameters and one for the information yielded by VAs separately, without loss of generality.

First, the generalized noisy OR-gate model [172, 173, 174] is applied to interpret the conditional distributions of system parameters. For a system parameter θ_i, the interaction of all its cause factors is assumed to be the noisy OR-type [83]. Applying this model consists of three steps:

- For each system parameter θ_i, if all its parent nodes have no faults, the belief in whether θ_i passes or not is estimated as a cause factor (i.e., a random variable) ξ_0 with a distribution $\mathscr{F}(\xi_0)$. This cause factor is called *independent factor* in this study. In nature, an independent factor represents the belief that an error will occur in the absence of all parent nodes modeled explicitly.
- For each system parameter θ_i, the dependency relationships of all its parent nodes are represented by a set of cause factors $\{\xi_1, \xi_2, \ldots \xi_R\}$, each of which has a distribution $\mathscr{F}(\xi_r)$, where θ_i has R parent nodes and the index r follows the index sequence of the parent nodes. Each $\mathscr{F}(\xi_r)$ represents the belief in whether θ_i passes or not if the rth parent node fails. They are called *dependent factors* in this study;
- With the noisy OR-type assumption between those cause factors, the conditional distributions of θ_i are given by the formula below:

$$\mathscr{F}(\theta_i|\{\xi_r = Fail\}) = \begin{cases} 1-(1-\mathrm{P}(\xi_0 = Fail))\bullet(\prod_{\xi_r \in \{\xi_r\}}(1-\mathrm{P}(\xi_r = Fail))), \text{ for } \theta_i = Fail \\ (1-\mathrm{P}(\xi_0 = Fail))\bullet(\prod_{\xi_r \in \{\xi_r\}}(1-\mathrm{P}(\xi_r = Fail))), \text{ for } \theta_i = Pass \end{cases}, \quad (4.1)$$

  where $\{\xi_r\}$ is the set of all cause factors that are *False*. The sum of each conditional distribution is 1.

Second, for a VA μ_j, its results are *completely* determined by its parent nodes. For simplicity, it is assumed that the causal interaction of cause factors is of noisy AND-type. So the noisy AND-gate model [83] is used for the conditional distributions of VAs in this study. Its application consists of four steps:

- When all parent nodes pass, the belief in whether μ_j passes or not is estimated as a cause factor ξ_- with a distribution $\mathscr{F}(\xi_-)$. This cause factor is called *negative factor* in this study.
- When all parent nodes fail, the belief in whether μ_j fails or not is estimated as a cause factor ξ_+ with a distribution $\mathscr{F}(\xi_+)$. This cause factor is called *positive factor* in this study. The dependency of both negative and positive factors is only on the parent nodes because the results of a VA are determined by its parent nodes.
- Otherwise, the belief in whether μ_j passes or not is assumed to follow a uniform distribution *unif{Fail, Pass}* for simplicity. That is, the belief of each possible state is 0.5. In reality, the specific distribution may be so complex that it is a hybrid function of all parental states, which is left for future work.
- The conditional distribution of μ_j is given by the formula below:

$$\mathscr{F}(\mu_j|\mathbf{N}_K) = \begin{cases} \mathscr{F}(\xi_-), \text{if } N_k = Pass, \forall N_k \in \mathbf{N}_K; \\ \mathscr{F}(\xi_+), \text{if } N_k = Fail, \forall N_k \in \mathbf{N}_K; \\ unif\{Fail, Pass\}, \text{ otherwise,} \end{cases} \tag{4.2}$$

  where $\mathbf{N}_k$ is the set of all parent nodes of μ_j and N_k is a parent node that may be either a system parameter or a VA.

4.2 Hybrid Verification and Correction Framework

4.2.1 Modeling of VAs

With the basic system verification model presented in Section 4.1.1, the planning and execution of VAs are modeled as follows: (1) A BN is built based on an elicited network structure and estimated prior distributions; (2) A set of hard evidence is collected after executing some VAs (i.e., observable nodes); and (3) The posterior distributions of the network nodes are updated by the Bayesian rule.

To illustrate this process, a basic system verification model about the speed of a computer product is provided as an example. The basic system verification model consists of four nodes, as shown in Fig. 4.1. The speed of the computer's processor and the overall speed of the computer are denoted by θ_1 and θ_2. If the speed values reach a certain threshold for each of these two parameters, these nodes are assigned with the value *compliant*. Otherwise, they would be assigned the value *non-compliant*. While these two parameters cannot be observed directly, the processor's speed and the overall computer speed may be inferred from the results of VAs (denoted by μ_1 and μ_2, respectively). Similar to the system parameters, the VAs are also assumed to be binary with their thresholds. If the processor speed test indicates that it is larger than such a threshold, μ_1 is assigned the value of *Pass*; it is assigned the value of *Fail* otherwise. The same reasoning applies to μ_2.

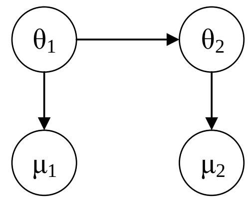


Fig. 4.1 An example BN

Once the results of a VA become available (that is, an observable node is observed), the Bayesian inference method can be applied to update the belief of the system parameters (that is, in the state of the hidden nodes). For example, if the benchmark processor speed test turns out $\theta_2=Pass$, the posterior marginal probability of θ_2 can be obtained with the Bayes rule:

$$P(\theta_2 | \mu_2 = Pass) = \frac{P(\theta_2)P(\mu_2 = Pass | \theta_2)}{P(\mu_2 = Pass)}. \tag{4.3}$$

In practice, when the results of multiple nodes are collected simultaneously, the posterior probabilities of the nodes can be obtained with off-the-shelf methods such as the variable elimination method and the junction tree algorithm. The specific inference process can be referred to in [107].

4.2.2 Modeling of CAs

4.2.2.1 Basic Types of CAs

The purpose of CAs is to correct the defects and errors that are found by VAs. This section focuses on three basic types of CAs that are commonly used in system development: rework, repair, and redesign. Once a CA is implemented according to its correction procedure, some corresponding system elements may be changed to meet system requirements or improve system performance. As a result, the prior beliefs of these elements are influenced after these correction procedures.

In general, VAs only are assumed to *observe* or measure attributes of the system. That is, they shape the confidence of the state of the system, but the system remains unchanged after a VA has been executed[2]. However, the execution of CA not only shapes the confidence of the state of the system but also, by definition, affects the state of the system. These effects of CAs challenge the estimation of posterior beliefs because a BN represents a joint distribution and some marginal distributions need to be reserved after a CA. Thus, the basic verification strategy model presented in the previous section needs to be extended to capture these effects. These extensions are presented in the following three subsections.

These modeling constructs are applicable to both systems being newly developed (where iterative design changes and corrections informed by early prototypes often occur due to the availability of little or ill-defined data) and systems that are recurrently produced (where knowledge and data

[2] In practice, some VAs can actually affect the state of a system, such as subjecting the system to vibration testing. However, these effects are generally expected and infrequent, so the assumption can be considered valid for the purpose of this discussion.

exist but the performance of a specific unit is poor). Only two procedural differences are noted. First, while determining priors for models of recurring systems likely rely on historical data, models of newly developed systems need to resort to subject matter experts [181] or fusion techniques [96]. Second, while all kinds of CAs may be necessary for a newly developed system, redesign activities are typically unnecessary for recurring systems.

4.2.2.2 Impact of Rework

According to the definition of rework in Table 1.1, when a rework activity is executed, an existing system element is directly replaced with a replica. Three assumptions are made with respect to the impact of rework. First, the two elements share the same cause factors as well as their distributions. This assumption is sensible because both elements are built using the same design, materials, and processes. Second, the new element is assumed to be in a correct state where all other elements have no errors. This is sensible because rework only applies in cases in which a unit is faulty, that is, there are no inherent design, process, or material problems with the unit. Third, while it is assumed that the new element does not present the problem identified in the faulty unit, it may still have other problems (both related to design and production) that have not been identified yet. Therefore, the impact of rework can be interpreted as including additional sources of information to the original BN.

To illustrate this, the previous computer BN is used as a rework example. Assume that μ_1 fails and it is determined that the processor is faulty. A new processor replaces the faulty one. The speed of the new processor shares the same parameter θ_1 with the faulty one. This is because the new processor is a replica of the faulty one (except for the fault). Thus, the distribution of θ_1 remains the same as the faulty one. The impact of rework can be interpreted from two aspects. First, as the faulty element is removed after the rework, the evidence about the faulty element becomes uninformative. (Here, while the faulty processor can be considered as new information that should update the belief distributions for future processors, it is assumed that such an effect is negligible, being a single event.) Second, the new element is known to be working after rework. This piece of new evidence is interpreted as a belief update on the independent factor of θ_1. That is, rework does not influence other parameters and their dependency relationships.

Assume that the rework information is an LR $\lambda_{Rework} = L(\xi_0(\theta_1)=Pass|Rework):L(\xi_0(\theta_1)=Fail|Rework) = 99:1$, and virtual evidence can be used to capture this repair information in three steps. First, the LR is represented as a virtual node φ_1 attached to the independent factor $\xi_0(\theta_1)$, as shown in Fig. 4.2 (a). In particular, virtual nodes are usually set as child nodes of the target node to ensure that the prior distribution of all existing nodes is not affected. Second, since rework information is presented as an LR λ_{Rework} on the distribution of $\xi_0(\theta_1)$, the CPT of φ_1 should be restricted by λ_{Rework}. In the example, there is a constraint on the CPT of φ_0:

$$\lambda_{\text{Re}work} = \frac{L(\xi_0(\theta_1) = Pass \mid \text{Rework})}{L(\xi_0(\theta_1) = Fail \mid \text{Rework})} = \frac{P(\varphi_0 = Pass \mid \xi_0(\theta_1) = Pass)}{P(\varphi_0 = Pass \mid \xi_0(\theta_1) = Fail)} = \frac{99}{1}\,. \quad (4.4)$$

In particular, as the virtual evidence only specifies an LR, there is no requirement on the specific values of the CPT of φ_1. Third, because the new processor is known to be working, the virtual node needs to be instantiated with φ_1 *=Pass* to capture this evidence. When the virtual evidence is added, the posterior distribution of $\xi_0(\theta_1)$ can be updated with the Bayes rule, as shown in Eq. 4.5:

$$P(\xi_0(\theta_1)\,|\,\text{Rework}) = \frac{P(\xi_0(\theta_1))P(\varphi_0 = Pass\,|\,\xi_0(\theta_1))}{P(\varphi_0 = Pass)} = \frac{P(\xi_0(\theta_1))L(\xi_0(\theta_1)\,|\,\varphi_0 = Pass)}{\sum_{\xi_0(\theta_1)} P(\xi_0(\theta_1))L(\xi_0(\theta_1)\,|\,\varphi_0 = Pass)}\,. \quad (4.5)$$

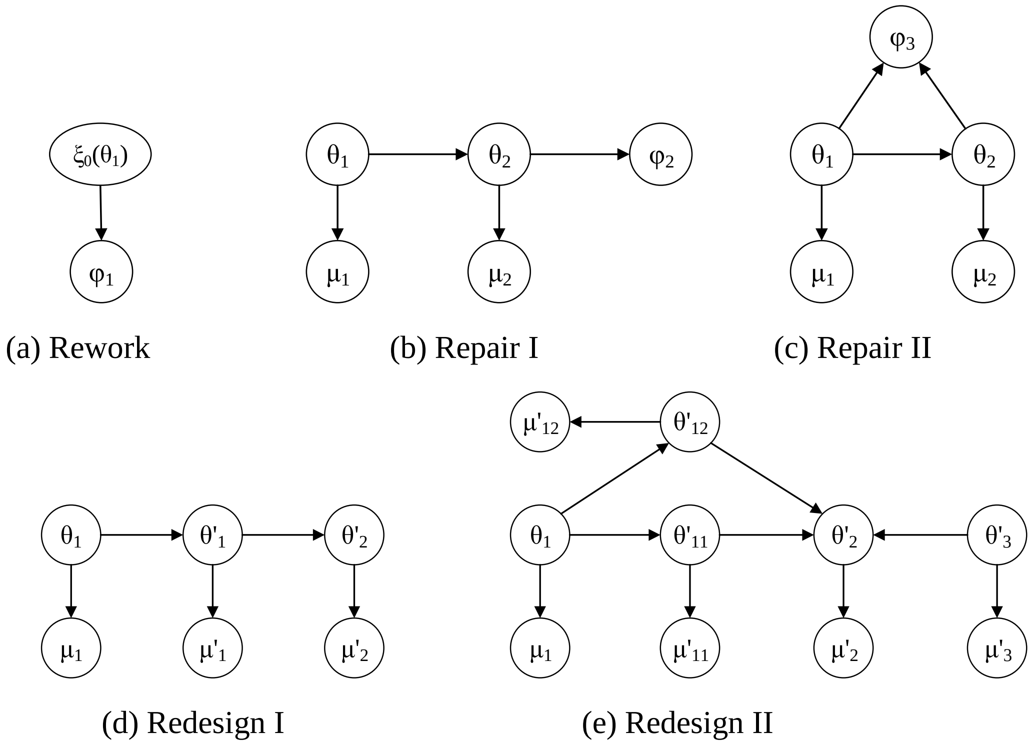


Fig. 4.2 Illustration of CAs

4.2.2.3 Impact of Repair

According to the definition of repair in Table 1.1, when a repair activity is executed, an existing system element is modified with parts, processes, or materials that were initially unplanned for that element. Four assumptions are made with respect to the impact of repair. First, all initial system elements are retained after repair. This implies that all prior beliefs of the initial elements remain useful for system verification and the network structure remains the same. Second, similar to rework actions, a repair action also provides information that cannot be deduced from the prior beliefs. Third, when repairing a system element, all its cause factors may be changed simultaneously to correct the fault. Fourth, a repair activity may inadvertently degrade the system. With these characteristics, the impact of repair is treated as a belief update of the repaired elements with extra repair information.

Even though all cause factors of an element may be influenced by repair, it is impractical to build a direct relationship between a repair activity and all cause factors. Instead, for simplicity, the belief update is assumed to be applied to the entire conditional distribution of the repaired element.

The repair information can be captured in two ways, depending on whether the likelihood of repair success derives from historical records of a repair activity or direct elicitation about the repaired elements. If information about the success of the repair activity is available, the repair information can be captured as virtual evidence. Continuing with the computer BN as an example, consider that μ_2 is executed and its results indicate that the computer has not achieved the desired speed. Further inspection indicates that the computer does not achieve the desired speed because part of the shielding is damaged. A repair activity is conducted directly on the computer by fixing the shield with metallic tape, leading to the computer exhibiting the desired speed. This repair activity has been conducted in the past, providing a track record of 80% success.

This piece of information is captured by defining an LR $\lambda_{Repair} = L(\theta_2{=}Pass|Repair) : L(\theta_2{=}Fail|Repair) = 80{:}20$ and conducting a belief update in the same three steps as that of rework. First, the virtual evidence is modeled as a virtual node that is attached to the BN. So if repair is applied to θ_2, a virtual node φ_2 can be inserted into the BN, as shown in Fig. 4.2 (b). Second, the CPT of the virtual node is restricted by λ_{Repair}. That is, there is nothing but an LR constraint on the CPT of φ_2:

$$\lambda_{\mathrm{Repair}} = \frac{P(\varphi_2 = Pass \mid \theta_2 = Pass)}{P(\varphi_2 = Pass \mid \theta_2 = Fail)} = \frac{80}{20} . \tag{4.6}$$

Third, the virtual node needs to be instantiated with $\varphi_2 = Pass$ to ensure that only the LR, rather than the specific conditional probabilities of φ_2, are leveraged to update beliefs. When the virtual evidence is added, the confidence of other parameters can be updated with Pearl's method [83]. For example, the updated marginal confidence of θ_2 can be calculated as:

$$P(\theta_2 \mid \mathrm{Repair}) = \frac{P(\theta_1)P(\theta_2 \mid \theta_1)L(\theta_2 \mid \varphi_2 = Pass)}{\sum_{\theta_1} P(\theta_1)P(\theta_2 \mid \theta_1)L(\theta_2 \mid \varphi_2 = Pass)} . \tag{4.7}$$

This procedure can be extended to the case in which multiple system elements are repaired simultaneously. It is realized by extending the current use of *virtual evidence* to cover multiple parameters in a similar way. First, a virtual node is added as a mutual child node of all repaired nodes. For example, if the repair is conducted for θ_1 and θ_2, a virtual node φ_3 can be added to the BN, as shown in Fig. 4.2 (c). Second, the repair information is estimated as an LR for the virtual node: $L(\theta_1 = Pass, \theta_2 = Pass \mid Repair) : L(\theta_1 = Pass, \theta_2 = Fail \mid Repair) : L(\theta_1 = Fail, \theta_2 = Pass \mid Repair) : L(\theta_1 = Fail, \theta_2 = Fail \mid Repair) = 45 : 15 : 21 : 17$. Third, the virtual node is instantiated with $\varphi_3 = Pass$. After adding this virtual node, Pearl's method [83] can be applied again to update the BN beliefs.

If direct elicitation of the repaired element is employed, the repair information can be captured as not-fixed probabilistic evidence. In this case, a belief is characterized as a probability of the system parameter to be correct after repair, which captures the success in repairing the defect and the possibility of inadvertently deteriorating the system. Then, this not-fixed probabilistic evidence is converted to virtual evidence, after which the same procedure as described for the case in which historical data is leveraged can be applied. Continuing with the computer example, assume that the experts' belief on whether θ_2 passes after the repair activity is characterized as 0.8, that is, the marginal probability of the overall speed $P(\theta_2=Pass|Repair)$ is 0.8 after repair. This not-fixed probabilistic evidence can then be converted to virtual evidence with the formula [182]:

$$\lambda_{\text{Repair}} = \frac{P(\theta_2 = Pass \mid \text{Repair})}{P(\theta_2 = Pass)} : \frac{P(\theta_2 = Fail \mid \text{Repair})}{P(\theta_2 = Fail)}, \tag{4.8}$$

where $P(\theta_2=Pass)$ and $P(\theta_2=Fail)$ are the prior marginal probabilities before repair. Then the belief update can be realized in the same way as was presented for virtual evidence.

4.2.2.4 Impact of Redesign

According to the definition of redesign in Table 1.1, when a redesign activity is executed, a new system is produced according to a new design. Two assumptions are made with respect to the impact of redesign. First, since the system itself changes, all redesigned elements may present errors or defects. Second, because redesign may only affect parts of the system, certain original system elements may be retained. However, the number of possible redesigns is likely vast, and, as a result, the possible impacts are also enormous. For example, some redesign activities could simply affect a minor element in the system, which removes one system parameter node in the BN. In contrast, other redesign activities may change the complete system, effectively substituting the BN with a completely different one. Therefore, a general procedure is presented here, which may need to be tailored to the specific redesign case. The general procedure consists of four steps:

- All system parameters that remain after the redesign activity are connected as a new network according to their dependency relationships.
- A set of cause factors $\{\xi_r\}_{new}$ that contains all factors that are changed after the redesign activity is built. As belief elicitation is necessary to characterize the new cause factors, a set of soft evidence is estimated for each cause factor in $\{\xi_r\}_{new}$.
- Another set of cause factors $\{\xi_r\}_{old}$ is derived and reused from the old network. This set includes all cause factors from the old network that are not affected by the redesign activity. The priors for these cause factors are directly reused from the old network.[3]
- The CPTs of all network nodes are generated using Eq. 4.1-4.2 and with these two sets.

Two aspects of this general procedure are worth mentioning. First, selecting which cause factors are reused from the old network (i.e., design) and which ones need to be newly modeled depends on both the redesign activity itself and the engineer's experience. Specifically, suppose an engineer assesses that their prior belief on a cause factor has changed because of the redesign activity. In that case, this cause factor will be chosen as a new cause factor. Second, even when the old distribution is replaced with soft evidence, past information is not ignored. Rather, eliciting the updated CPTs inherently includes past information in the expert's experience.

Two examples are provided to describe this procedure. Consider the computer case presented earlier and assume that μ_1 fails. An investigation shows that the processor is inherently unable to achieve the required speed. A redesign activity is conducted, resulting in a new processor. The

[3] It is noted that this task is not trivial and may be significant. It includes assessing that such parameters have indeed not been influenced by the redesign activity.

speed of the *new* computer (since it now uses a different processor) is denoted by θ'_2 and the speeds of the new processors by θ'_1. New VAs are defined for each of the attributes of the new system, as shown in Fig. 4.2 (d). A key aspect is that, because past information is used, it is necessary to model a dependency between θ_1 and θ'_1, since the knowledge obtained on the original processor with μ_1 shapes the confidence of the performance of the new processor.

Now consider the computer case presented earlier. In the new case, however, instead of only designing a new processor, the overall processing concept is redesigned. Specifically, instead of using a single processor, the computer uses two processors that are coordinated by a central unit. The speed of this *new* computer is denoted by θ'_2, the speeds of each of the new processors by θ'_{11} and θ'_{12}, respectively, and the ability of the coordination unit to coordinate the two processors by θ'_3. New VAs are defined for each of the attributes of the new system, as shown in Fig. 4.2 (e). A key aspect is that, because past information is used, it can be modeled as a dependency between θ_1 and θ'_{11} and θ'_{12}, since the knowledge obtained on the original processor with μ_1 shapes the confidence of the performance of the new processors.

4.2.3 Integrating VAs and CAs under a Hybrid Framework

4.2.3.1 Two Decisions

Our framework captures VAs and CAs as dedicated decisions, informed by each other but independent from each other as well. In one type of decision (which is referred to as verification decisions, VDs), whether a VA is executed or not is chosen; in the other type of decision (which is referred to as correction decisions, CDs), whether a CA is executed or not is chosen.[4] The confidence achieved as per the evidence provided by a VA or after the results of a CA inform both types of decisions.

As shown in the basic system verification model, the decision set of a VD contains all VAs that could be executed at a given time (that is, all observable nodes in the BN). Thus, a VD specifies a set of VAs first. If there are no observable nodes selected, the result of the VD is 'NA' (i.e., no activity). The decision set of a CD is based on two components. First, a set of system parameters is selected as the target of the CD. Second, for each system parameter, there are three possible types of CAs, as explained earlier (i.e., rework, repair, and redesign). If no CA is chosen, the result of the CD is 'NA'.

In this chapter, VDs and CDs are based on confidence and past activities. First, for each system element, if the posterior belief of its parameter is sufficiently high, there is no need to conduct VAs or CAs for it. That is, it is considered that, if $P(\theta_i|Evidence)$ is larger than some threshold H_i, the system element ε_i has low uncertainty of errors and no additional VAs or CAs are necessary. The values of current posterior beliefs can be deduced from the BN. Second, past activities may also influence the decision. For simplicity, this study only considers one case that past CAs may influence VDs. That is, once a CA is executed, some previously collected results about the

[4] The selection of which VAs and CAs to evaluate or assess derives or is informed by the evidence that VAs can provide, the corrective power of CAs, and the resources necessary to implement them. A decision-making framework for such decision tuple is outside of the scope of this dissertation.

corrected elements may become meaningless because their corresponding system parameters change. So, if necessary, some VAs previously conducted may have to be repeated to ensure all system parameters meet their desired targets.

4.2.3.2 Basic Interaction Loop

The hybrid verification and correction framework is constructed by composing several basic interaction loops (BIL). The BIL captures the VD and CD to be made within a given time event in the system development process. For simplicity and without loss of generality, two primary assumptions are made in this respect. First, only activities within a given time event are considered to inform VDs and CDs. This implies, for example, that VAs and CAs within a given time event can be treated independently of activities in other time events. Second, because CAs aim to eliminate system errors that are identified by VAs, there are sequential constraints between each pair of them. In other words, all repair, rework, and redesign activities are only considered when some errors are identified by the results of VAs, or, in general, when a desired result or performance is not achieved. Under these conditions, the BIL has been designed as a workflow:

1. At the beginning of each time event, a designed system is provided with its BN.
2. List all VAs, including ‘NA’, and choose a set of VAs.
3. Conduct the selected VAs, collect their results, and update the BN.
4. List all alternative CAs, including ‘NA’, and choose a CA from them.
5. Execute the selected CA, evaluate its impact, and update the BN.
6. If the uncertainty of error $P(\theta_i)$ is larger than H_i, go to 7. Otherwise, go to 2.
7. If there is a next time event, go to 1 and repeat these decisions and activities above. Otherwise, the whole development process is completed.

The basic relation between the different elements of the BIL is shown in Fig. 4.3. In contrast to existing methods, as described in Section 2.2, VAs and CAs are chosen based on active decisions, which are implemented interactively with real-time results and informed by posterior beliefs of system elements updated by the effects of previous decisions.

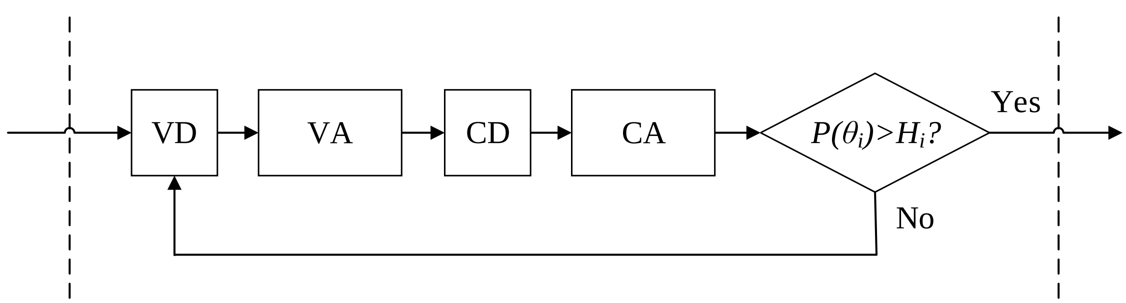


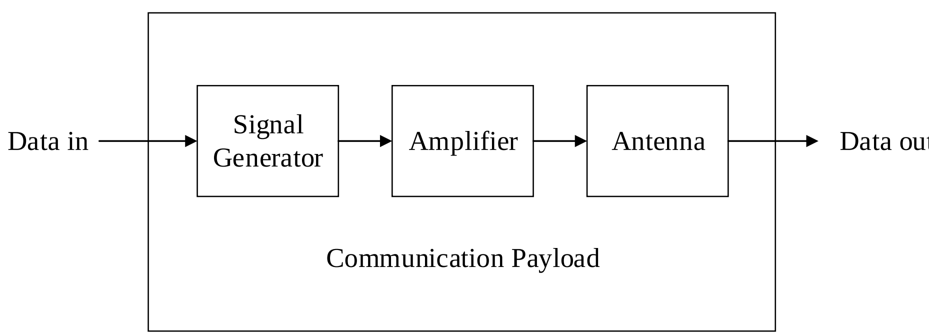
Fig. 4.3 Basic interaction loop

4.3 Case Demonstration

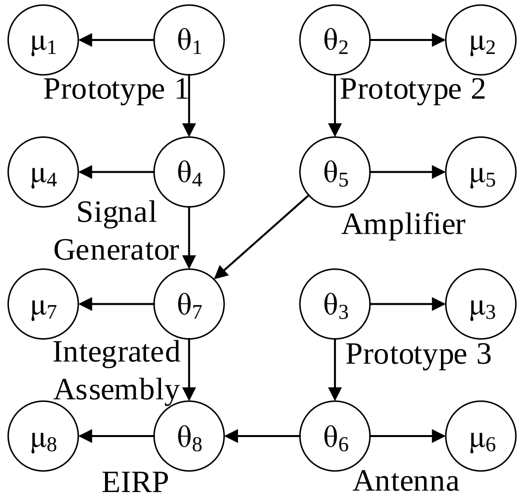
To show how CAs can be modeled and integrated as part of a hybrid verification and correction framework, this case demonstrates a partial system and corresponding verification strategy.

4.3.1 Case Description

The hybrid verification and correction framework is illustrated in a notional satellite communication payload. Consider that the payload is formed by a Signal Generator, an Amplifier, and an Antenna, as shown in Fig. 4.4 (a). The scope of this case is restricted to the following performance parameters. The primary target is the Effective Isotropic Radiated Power (EIRP) of the payload (denoted by θ_8), which is characterized as a function of the Signal Generator output power (denoted by θ_4), the Amplifier gain (denoted by θ_5), and the Antenna gain (denoted by θ_6). Furthermore, the output power of the integrated assembly formed by the Signal Generator and the Amplifier is considered as an intermediate system parameter of potential interest for the verification campaign, which is denoted by θ_7. In addition, the system includes prototypes for the Signal Generator, for the Amplifier, and for the Antenna as potentially interesting parameters for the verification campaign, whose output power and gain are denoted by θ_1, θ_2, and θ_3, respectively. Eight VAs are considered for evaluation of VDs, denoted by μ_1, ..., μ_8, where μ_j provides information about θ_i for i=1, ..., 8. The resulting verification model represented as a BN is shown in Fig. 4.4 (b).



(a) The element structure



(b) The Bayes network

Fig. 4.4 A satellite communication payload example

For clarity, it should be noted that the state of the performance of the integrated assembly (θ_7) depends on the performance of the Signal Generator (θ_4), the performance of the Amplifier (θ_5), and the cabling connecting them (embedded in θ_7). Similarly, the performance of the overall communication payload (θ_8) depends on the performance of the integrated assembly (θ_7), the performance of the Antenna (θ_6), and the cabling between them (embedded in θ_8).

Table 4.1 Distributions of all cause factors of the pay load

	$\xi_0(\theta_1)$	$\xi_0(\theta_2)$	$\xi_0(\theta_3)$	$\xi_0(\theta_4)$	$\xi_1(\theta_4)$	$\xi_0(\theta_5)$	$\xi_1(\theta_5)$	$\xi_0(\theta_6)$
Pass	0.95	0.95	0.95	0.92	0.15	0.92	0.15	0.92
Fail	0.05	0.05	0.05	0.08	0.85	0.08	0.85	0.08
	$\xi_1(\theta_6)$	$\xi_0(\theta_7)$	$\xi_1(\theta_7)$	$\xi_2(\theta_7)$	$\xi_0(\theta_8)$	$\xi_1(\theta_8)$	$\xi_2(\theta_8)$	
Pass	0.15	0.92	0.25	0.25	0.92	0.25	0.25	
Fail	0.85	0.08	0.75	0.75	0.08	0.75	0.75	
	$\xi_-(\mu_1)$	$\xi_+(\mu_1)$	$\xi_-(\mu_2)$	$\xi_+(\mu_2)$	$\xi_-(\mu_3)$	$\xi_+(\mu_3)$	$\xi_-(\mu_4)$	$\xi_+(\mu_4)$
Pass	0.80	0.05	0.80	0.05	0.80	0.05	0.75	0.10
Fail	0.20	0.95	0.20	0.95	0.20	0.95	0.25	0.90
	$\xi_-(\mu_5)$	$\xi_+(\mu_5)$	$\xi_-(\mu_6)$	$\xi_+(\mu_6)$	$\xi_-(\mu_7)$	$\xi_+(\mu_7)$	$\xi_-(\mu_8)$	$\xi_+(\mu_8)$
Pass	0.75	0.10	0.75	0.10	0.70	0.10	0.70	0.10
Fail	0.25	0.90	0.25	0.90	0.30	0.90	0.30	0.90

A simulation is performed to evaluate VDs and CDs for the verification of the communication payload. Considering a starting point in which verification activities have not been conducted, a set of cause factors is used to construct the CPTs of all nodes. Their distributions are listed in Table 4.1. The threshold *H* is notionally set as 0.90. All values in this section have been synthetically generated, albeit not arbitrarily. The values reflect the physical meanings of the relationships between the different elements, as well as the relative prediction power of different verification activities. The use of synthetic data is considered adequate because actual predictions are not relevant to showcase the application of the modeling framework. This is an accepted practice in this type of research [49, 63, 122]. This BN captures the prior beliefs of all elements. The prior marginal probabilities of the eight parameters are shown in the third row of Table 4.2. While the probabilities of all independent factors $\{\xi_0(\theta_i)\}$, i=1,...,8 reach the threshold H=0.90, the marginal probabilities of $\{\theta_4, \theta_5, \theta_6, \theta_7, \theta_8\}$ are lower than H=0.90 because of the dependency relationships among parameters. In particular, $P(\theta_8 = Pass)$ represents the confidence in the payload-level performance required to deploy the communication payload. The BIL model is applied to capture the effect of verification and correction in updating such prior beliefs. Three BILs are used to demonstrate the hybrid verification and correction framework proposed in Section 4.2.

4.3.2 BIL 1: VA and Redesign Activity

The first BIL starts with a VD. As there is no evidence about the state of the payload yet, the verification action starts with the VAs of all prototypes in the BN (i.e., μ_1, μ_2, μ_3). Assume that μ_1 fails while the other two pass. Given these three verification results, the posterior marginal beliefs of system parameters (i.e., $P(\theta_i = Pass)$) can be updated using Bayesian inference, as shown in the fifth row of Table 4.2. It can be found that θ_8 has a confidence value lower than $H=0.90$. Assume that the failure in μ_1 implies that it is hard to achieve the desired confidence on θ_8 even if other VAs are executed in the future. So no more VAs are executed for the moment.

An investigation is performed to find the causes of the possible errors, and it is determined that the Signal Generator is unable to provide the desired output voltage due to its operating voltage, which is insufficient. Consider that a CA is worth doing (given rework costs, expected confidence increase, etc.), and it is decided to redesign the prototype of the Signal Generator to operate with a higher voltage. This correction leads to a change in the structure of the original BN, following the modeling paradigm described in Section 4.2.2.4. This change is explained in three steps, which are depicted in Fig. 4.5. First, a new Signal Generator is obtained with a new prototype. So, they are characterized as two new nodes θ'_4 and θ'_1, respectively. As a result, all higher-level assemblies (in this case, the Integrated Assembly and the whole Communication Payload) also become new versions of the system; note that the original Communication Payload used a Signal Generator that will no longer be used, thus, effectively, a different Communication Payload is being developed after the correction. The resulting Integrated Assembly and Communication Payload are characterized as new nodes θ'_7 and θ'_8, respectively. These changes are shown in Fig. 4.5 (a). Second, system elements that are not affected by this change remain unchanged with respect to the original BN. They are used as causal factors of the new nodes defined earlier. In this example, notably, neither the Antenna nor the Amplifier is affected by the change in the Signal Generator. Yet, the performance of the new Integrated Assembly still depends on the performance of the Amplifier and the performance of the Communication Payload still depends on the performance of the Antenna. Therefore, θ_5 and θ_6 in the original BN are connected to θ'_7 and θ'_8, respectively. Furthermore, since the learning that occurs in past activities is used in future activities, the confidence in the state of the new Signal Generator is a function of the results obtained from the prototype of the original Signal Generator. This use of past information is captured by the connection from θ_1 to θ'_1. These changes are shown in Fig. 4.5 (b). Third, because the original system is no longer used as a solution (note that the development target is θ'_8 instead of θ_8), nodes representing such solutions can be removed from the BN to simplify the model with no impact on its fidelity. Specifically, θ_4, θ_7, θ_8 and their corresponding VAs (μ_4, μ_7, μ_8) are removed from the network leading to the one shown in Fig. 4.5 (c).

In this section, only two cause factors of θ'_1 are re-estimated after the redesign (i.e., the independent factor $\xi_0(\theta'_1)$ and the dependent factor $\xi_1(\theta'_1)$ between θ_1 and θ'_1). Other affected elements reuse the original cause factors. Two corresponding pieces of soft evidence are estimated for the two cause factors. That is, $P(\xi_0(\theta'_1)=Fail) = 0.05$ and $P(\xi_1(\theta'_1)=Fail) = 0.40$. With this soft evidence, the posterior marginal probabilities can be calculated using the Bayesian inference, as shown in the seventh row of Table 4.2. In particular, $P(\theta'_1=Pass|\ Redesign)$ is *0.991*, which suggests that the confidence of the prototype of the Signal Generator is larger than the original one (i.e., $P(\theta_1=Pass) = 0.800$).

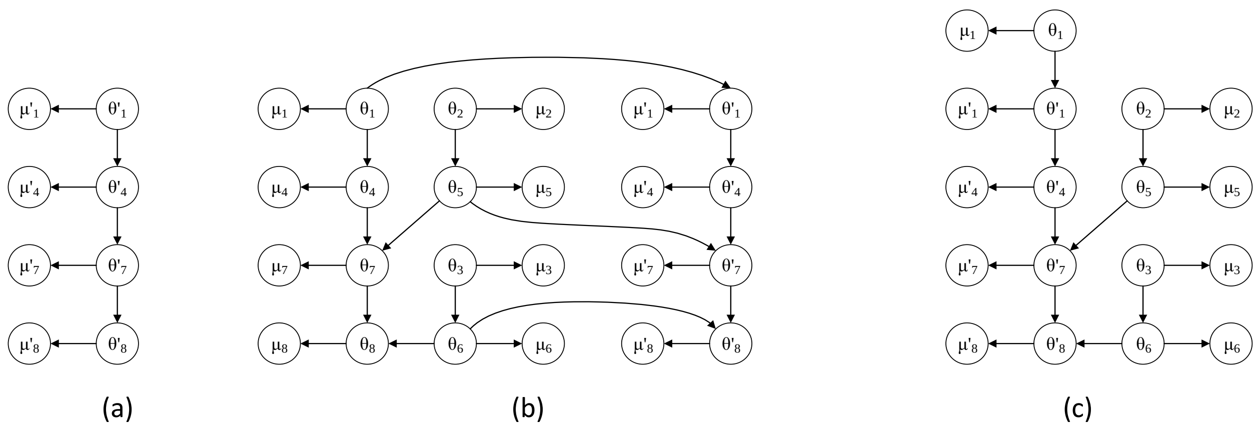
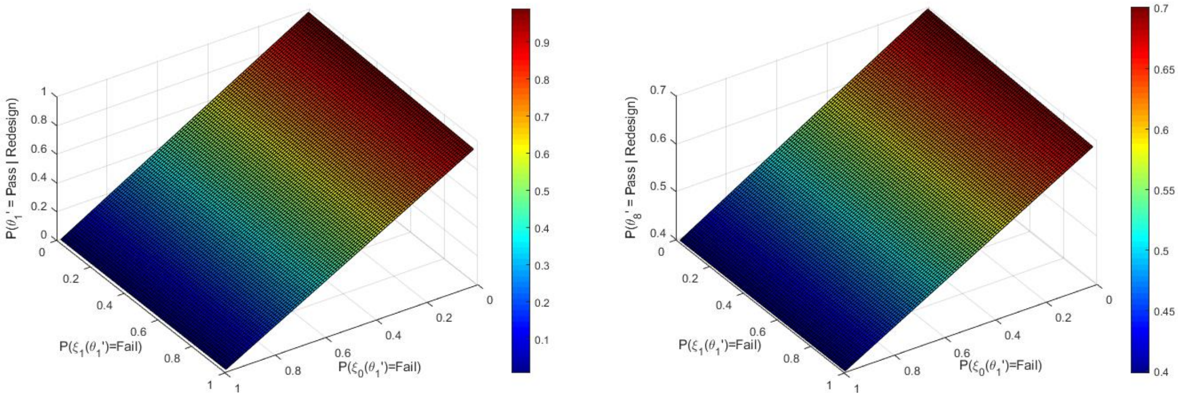


Figure 4.5 Construction of the new BN for the redesign activity

To observe how the impact of redesign stands up under uncertainties in the evidence, the relationship is explored between the soft evidence $\{\xi_0(\theta'_1), \xi_1(\theta'_1)\}$ and the posterior confidences of two target nodes $\{\theta'_1, \theta'_8\}$. The legend of the surface shows the trend of the posterior confidences given different values of the two causal factors. First, the surface between $P(\xi_0(\theta'_1)=Fail)$, $P(\xi_1(\theta'_1)=Fail)$, and $P(\theta'_1=Pass|\ Redesign)$ is shown in Fig. 4.6 (a). It is found that the maximum of $P(\theta'_1=Pass|\ Redesign)$ is 0.990 when all values of $\xi_0(\theta'_1)=Fail$ and $\xi_1(\theta'_1)=Fail$ are set at 0.01, and the minimum is 0.010 when all these values are set at 0.99. Second, the surface between $P(\xi_0(\theta'_1)=Fail)$, $P(\xi_1(\theta'_1)=Fail)$, and $P(\theta'_8=Pass|\ Redesign)$ is shown in Fig. 4.6 (b). It is found that the two surfaces reach the maximum/minimum with the same values of $P(\xi_0(\theta'_1))$ and $P(\xi_1(\theta'_1))$. However, the range of $P(\theta'_8=Pass|\ Redesign)$ is much smaller than that of $P(\theta'_1=Pass|\ Redesign)$. This is attributed to the distance between the redesigned node and the target nodes.



(a) Impact on $P(\theta'_1$=*Pass*| *Redesign)* (b) Impact on $P(\theta'_8$=*Pass*| *Redesign)*

Fig. 4.6 Sensitivity analysis of redesign

4.3.3 BIL 2: VA and Rework Activity

Since $P(\theta'_8=Pass|\ Redesign)=0.716$ is still smaller than H = 0.90 after the first BIL, three additional VAs (i.e., μ'_4, μ_5, μ_6) are executed at higher integration levels. Assume that μ_5 fails while the other two pass. Given the verification results, the posterior marginal beliefs of the system parameters can be updated with the Bayesian inference, and their belief values are shown in the ninth row of Table 4.2. It is assumed that the failure evidence of μ_5 implies that it is difficult to achieve the desired confidence on θ'_8 even if all remaining VAs are executed. Thus, no more VAs are executed for the moment.

An investigation is performed, and it is determined that θ_5 is unable to provide the desired gain due to a workmanship problem. It is assumed that a CA is worth doing, under similar conditions to the previous case, and it is decided to rework the faulty part of the Amplifier. Another Amplifier is built to conform to the design (i.e., until the test results of the reworked Amplifier are *Pass*). Since no design difference exists between the new Amplifier and the original one, the structure of the BN does not change. Instead, a notional LR $\lambda_{Rework} = L(\xi_0(\theta_5)=Pass|\ Rework): L(\xi_0(\theta_5)=Fail|\ Rework) = 99:1$ is estimated for the rework activity and it is used to update the independent factor $\xi_0(\theta_5)$ for the reworked Amplifier. The evidence $\mu_5=Fail$ becomes uninformative because the rework activity has removed the faulty part of the Amplifier; only the evidence $\mu'_4=Pass$ and $\mu_6=Pass$ are applicable in this case. With this LR, the updated independent factor $X_0(\theta_5)$ becomes $P(\xi_0(\theta_5)=Fail)=0.08*1/(0.92*99+0.08*1)=0.0009$. With this updated independent factor, the posterior marginal probability $P(\theta'_8|Rework)$ increases to 0.842 using the Bayesian inference, as shown in the tenth row of Table 4.2.

For the reworked payload, the relationship between λ_{Rework} and $P(\theta'_8=Pass|\ Rework)$ is plotted in Fig. 4.8 (a). The list of LRs is $\{2^i\}$, i = -10, -9, … , 10. Three aspects are worth mentioning. First, $P(\theta'_8=Pass|\ Rework)$ is a monotone function of λ_{Rework} and it is monotone-increasing in this case. Especially, when λ_{Rework} is larger than 1, the marginal probability is larger than 0.806, which is the marginal probability when the rework has no impact (i.e., only the evidence μ_5=*Fail* becomes uninformative after rework). Second, the upper bound is 0.843, approximating the case θ_5=*Pass*, and the lower bound is 0.380 for the case θ_5=*Fail*. Third, $P(\theta'_8=Pass|\ Rework)$ increases sharply within the interval $[2^{-8}, 2^{1}]$ and becomes stable when λ_{Rework} is outside of it.

4.3.4 BIL 3: VA and Repair Activity

Since $P(\theta'_8=Pass|\ Rework)=0.842$ is still lower than the desired confidence of H = 0.90, it is assumed that it is decided to conduct μ'_7 and it fails. The posterior marginal beliefs of system parameters can be obtained by the Bayesian inference, which are shown at the twelfth row of Table 4.2. Assume that it will be challenging to achieve the desired confidence on θ'_8 after all remaining VAs are executed in the future. So no additional VAs are conducted for the moment.

After an investigation, it is determined that θ'_7 is unable to provide the desired output power because of a cable defect between the Signal Generator, the Amplifier, and the cabling between them. So, under similar assumptions as for the previous cases, it is decided to conduct a CA and, instead of producing a new cable, cut and connect the defective wires in the cable. It is assumed that the repair activity is successful and that the repaired Integrated Assembly is reported to operate correctly. It is also assumed that historical records indicate that 95% of this type of repair yields error-free Integrated Assemblies. In this case, the BN evolves as shown in Fig. 4.7. Using virtual evidence, an LR $\lambda_{Repair} =L(\theta'_7=Pass|\ Repair)$: $L(\theta'_7= Fail\ |\ Repair) = 19:1$ is added to the parameter θ'_7, shown as a virtual node φ_7 in the BN. It is also noticeable that the evidence μ'_7=*Fail* becomes uninformative in this case because the repair activity has changed the Integrated Assembly and the virtual evidence embeds such past information. Therefore, the posterior marginal probability $P(\theta'_8=Pass|\ Repair)$ becomes *0.908*, which shows the positive impact of the repair activity on the BN. Because $P(\theta'_8=Pass|\ Repair)$ is larger than H=0.90 now, it is unnecessary to conduct more VAs or CAs. The development is considered completed.

For completeness, in the case where the direct estimation of the repaired element would have been used, instead of historical records to evaluate the performance of the repair activity, the impact of the repair would have been estimated as the posterior marginal distribution $P'(\theta'_7=Pass|Repair)$. For example, assuming an estimation of 99.4% confidence on the repaired Integrated Assembly correctly operating, the posterior marginal distribution $P'(\theta'_7=Pass|Repair)$ would be 0.994. This evidence could be converted to virtual evidence:

$$\lambda_{\text{Repair}} = \frac{P'(\theta'_7 = Pass\,|\,\text{Repair})}{P(\theta'_7 = Pass)} : \frac{P'(\theta'_7 = Fail\,|\,\text{Repair})}{P(\theta'_7 = Fail)} = \frac{0.994}{0.898} : \frac{0.006}{0.102} \approx 19:1,$$

where $P'(\theta'_7=Pass)=0.898$ and $P'(\theta'_7= Fail)=0.102$ are the marginal probabilities of $P(\theta'_7)$ before the repair activity. Then the Bayesian inference could be used in the same way as the virtual evidence method, as described in the previous paragraph.

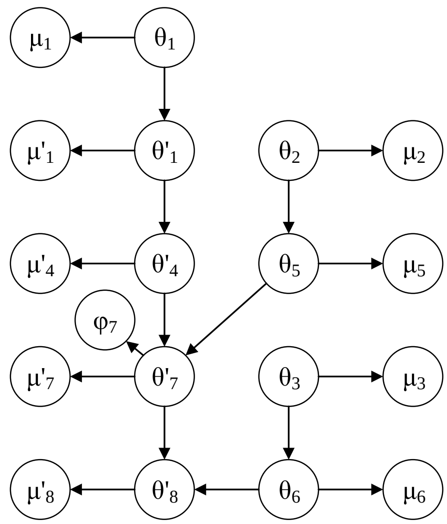
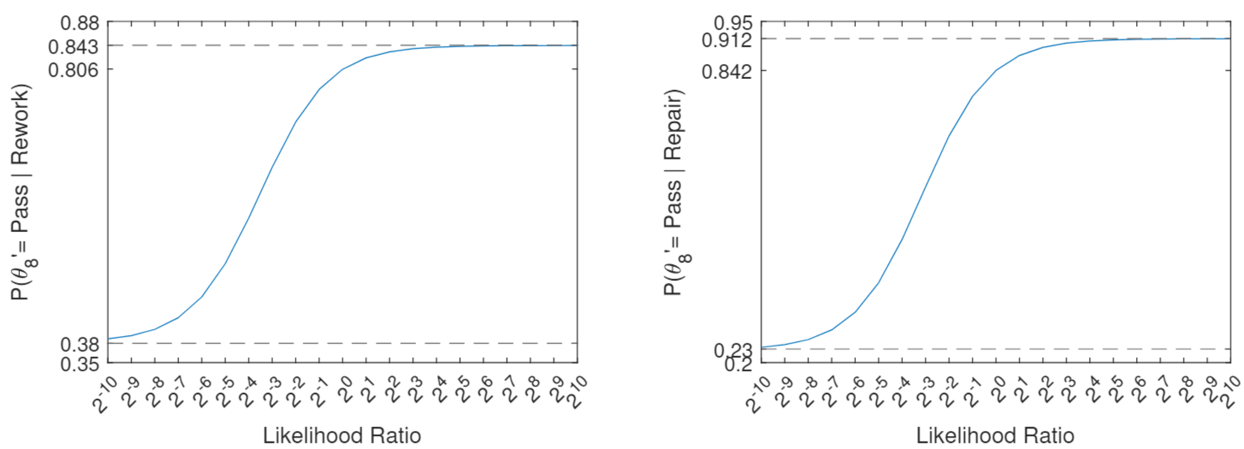


Fig. 4.7 Adding a virtual node φ_7 to the BN

To investigate the sensitivity of the impact of repair on *P(θ'₈=Pass| Repair)*, λ_{Repair} is assigned with 21 values of LRs $\{2^i\}$, i=-10, -9, … , 10 to update the BN. The 21 results are plotted as a blue line in Fig. 4.8 (b). The curve has the same three characteristics as that of the rework in BIL 2. First, $P(\theta'_8=Pass|\ Repair)$ is a monotone function of λ_{Repair}. When λ_{Repair} is larger than 1, the marginal probability is larger than 0.842, which is the marginal probability when the repair has no impact. Second, the upper bound is 0.912, which approximates the case $\theta'_7=Pass$, and the lower bound is 0.230 for the case $\theta'_7=Fail$. Third, $P(\theta'_8=Pass|\ Repair)$ increases sharply within the interval $[2^{-8}, 2^1]$ and becomes stable when λ_{Repair} is outside of it.



(a) Impact of λ_{Rework} (b) Impact of λ_{Repair}

Fig. 4.8 Sensitivity analysis of repair and rework

Table 4.2 Evidence and confidence results of 3 BILs

		Evidence	Confidence								
Prior			θ_1	-	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7	θ_8
			0.950	-	0.950	0.950	0.881	0.881	0.881	0.763	0.689
BIL 1			θ_1	-	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7	θ_8
	VA	$\mu_1=\mathit{Fail}$, $\mu_2=\mathit{Pass}$, $\mu_3=\mathit{Pass}$.	0.800		0.997	0.997	0.764	0.917	0.917	0.710	0.675
			θ_1	θ'_1	θ_2	θ_3	θ'_4	θ_5	θ_6	θ'_7	θ'_8
	CA	$\mu_1=\mathit{Fail}$, $\mu_2=\mathit{Pass}$, $\mu_3=\mathit{Pass}$, $P(\xi_0(\theta'_1)=\mathit{Fail}) = 0.05$, $P(\xi_1(\theta'_1)=\mathit{Fail}) = 0.40$	0.865	0.991	0.997	0.950	0.913	0.917	0.881	0.807	0.716
BIL 2			θ_1	θ'_1	θ_2	θ_3	θ'_4	θ_5	θ_6	θ'_7	θ'_8
	VA	$\mu_1=\mathit{Fail}$, $\mu_2=\mathit{Pass}$, $\mu_3=\mathit{Pass}$, $P(\xi_0(\theta'_1)=\mathit{Fail}) = 0.05$, $P(\xi_1(\theta'_1)=\mathit{Fail}) = 0.40$, $\mu'_4= \mathit{Pass}$, $\mu_5= \mathit{Fail}$, $\mu_6=\mathit{Pass}$.	0.868	0.998	0.991	0.986	0.988	0.755	0.982	0.744	0.734
	CA	$\mu_1=\mathit{Fail}$, $\mu_2=\mathit{Pass}$, $\mu_3=\mathit{Pass}$, $P(\xi_0(\theta'_1)=\mathit{Fail}) = 0.05$, $P(\xi_1(\theta'_1)=\mathit{Fail}) = 0.40$, $\mu'_4= \mathit{Pass}$, $\mu_6=\mathit{Pass}$, $\frac{L(\xi_0(\theta_5)=Pass\|Rework)}{L(\xi_0(\theta_5)=Fail\|Rework)}=\frac{99}{1}$.	0.849	0.962	0.997	0.999	0.972	0.996	0.988	0.898	0.842
BIL 3			θ_1	θ'_1	θ_2	θ_3	θ'_4	θ_5	θ_6	θ'_7	θ'_8
	VA	$\mu_1=\mathit{Fail}$, $\mu_2=\mathit{Pass}$, $\mu_3=\mathit{Pass}$, $P(\xi_0(\theta'_1)=\mathit{Fail}) = 0.05$, $P(\xi_1(\theta'_1)=\mathit{Fail}) = 0.40$, $\mu'_4= \mathit{Pass}$, $\mu_6=\mathit{Pass}$, $\frac{L(\xi_0(\theta_5)=Pass\|Rework)}{L(\xi_0(\theta_5)=Fail\|Rework)}=\frac{99}{1}$, $\mu'_7= \mathit{Fail}$.	0.839	0.944	0.994	0.999	0.941	0.992	0.988	0.746	0.738
	CA	$\mu_1=\mathit{Fail}$, $\mu_2=\mathit{Pass}$, $\mu_3=\mathit{Pass}$, $P(\xi_0(\theta'_1)=\mathit{Fail}) = 0.05$, $P(\xi_1(\theta'_1)=\mathit{Fail}) = 0.40$, $\mu'_4= \mathit{Pass}$, $\mu_6=\mathit{Pass}$, $\frac{L(\xi_0(\theta_5)=Pass\|Rework)}{L(\xi_0(\theta_5)=Fail\|Rework)}=\frac{99}{1}$, $\frac{L(\theta_7=Pass\|Repair)}{L(\theta_7=Fail\|Repair)}=\frac{19}{1}$.	0.855	0.974	0.999	0.999	0.992	0.999	0.988	0.994	0.908

4.4 Discussion

This chapter has presented modeling constructs to capture the effects of correction activities when modeling verification strategies using BN. These constructs allow for capturing correction activities resulting from dedicated decisions instead of properties that are inherent to verification activities. In this way, Bayesian inference can be used as a general process to assess the joint effect of verification and correction activities quantitatively.

A vital tenet of the presented modeling constructs is recognizing that the system being verified changes when a correction activity occurs. Consequently, the belief structure that stems from a given verification strategy on a system design must necessarily change to reflect the *new* system. Specific modeling constructs have been presented for redesign, rework, and repair. For redesign, new nodes are incorporated into the BN: those representing the redesigned elements and those at higher levels of integration that embed such redesigned elements. The elements in the original BN that had to be redesigned then act as causal factors to the redesigned elements, since past information is used to inform the confidence of the redesigned elements. Higher-level assemblies in the original network are omitted, since they represent planned development paths that have been discarded. Elements in the original network that are not causally related to the redesigned system remain. For rework and repair, it suffices to add statistical features to the BN in the form of uncertain evidence or LR to capture the effects of the corrections, effectively leading to a new BN.

In all cases, beliefs must usually be re-elicited for all system elements. However, the presented constructs can be effective in limiting this effort. First, the application of uncertain evidence narrows the elicitation scope from joint distribution to likelihood information. This allows the reuse of several of the original beliefs. Second, using the PCI model makes it possible to decompose the CPTs into cause factors, allowing the reuse of all old cause factors that are affected by CAs in the new network. While the size of a CPT exponentially increases with the number of parent nodes, the number of cause factors is linearly proportional to that of the parent nodes. Third, the independence of the cause factors in the PCI models makes the change of the BN structure more easily than usual, given the joint distributions in the BN.

Overall, it is suggested that these constructs should improve the fidelity of the models used to design verification strategies, resulting in *more valuable* verification strategies. By adopting the presented modeling constructs, it is anticipated that a systems engineer will be able to (1) select a more valuable set of verification activities and verification paths, (2) anticipate valuable points of potential correction, and (3) reduce the amount of reactive work during verification. These potential improvements are enabled by the mathematical nature of modeling constructs, to which systems engineers do not have access from existing methods.

CHAPTER 5: Designing JVCSs in Small Strategy Spaces

This chapter presents an extended paradigm of verification processes (EPVP) that incorporates CAs as dedicated decisions and an order-based backward induction method to design JVCSs in small strategy spaces. This chapter has been reproduced from my paper [183]. As a basis of the EPVP, the belief models of VAs and CAs are skipped in this chapter, which can be found in Section 4.2. The EPVP is described to specify all constraints about verification processes and activity spaces. A value-based model is used to characterize the goodness of the verification process and is used to measure the performance of verification strategies. An order-based backward induction method is proposed based on the process constraints in EPVP to solve for exact JVCSs.

This chapter is organized as follows: Section 5.1 describes the proposed JVCM, including the EPVP and the value-based model. An order-based backward induction approach is also presented to solve for exact JVCSs. Finally, the use of the JVCM is illustrated in a case study in Section 5.2, followed by a discussion of this chapter in Section 5.3.

5.1 Proposed Joint Verification-Correction Model (JVCM)

5.1.1 Extended Paradigm of Verification Processes

This chapter extends the paradigm of verification processes by considering both VAs and CAs as independent decisions. While VAs and CAs are two kinds of activities, they are tightly connected to each other because errors and defects are identified by VAs and corrected by CAs. Thus, a VP can be viewed as the process of identifying and eliminating errors. As part of the system development, two key process constraints are considered in a VP. First, VPs can be decomposed into a set of time events. This is because system verification and correction are planned and carried out throughout a life cycle that consists of relatively independent phases with different requirements [129]. A time event is used to represent the life cycle phase. Second, there is a sequential constraint between VAs and CAs to make system verification valid and sound. That is, CAs must be conducted after collecting certain VA results. This is caused by their inherent logical relationship in handling errors and defects. That is, without the explicit evidence provided by VAs, which indicates the presence of errors, it is unreasonable to execute CAs to eliminate these errors. For example, if the temperature of a machining tool is tested and found to be expected, there is no apparent need to change the tool parameters.

With two process constraints, the VP is modeled as a sequential process with time events $t = 1, \ldots, T$. For simplicity, two assumptions are made about the sequential process. First, the number of time events is not predetermined and the process terminates only when there is no need for more activities. That is, the number of phases is not restricted externally in this study. Notably, this sequential process does not have an infinite horizon because the number of activities in a BN is finite. Second, only one VA and one CA are conducted at each time event and the VA is followed by the CA. That is, each time event has two time points and only one activity is conducted at each time point. This assumption can be relaxed when conducting more than one activity in parallel, which is reserved for future work. Thus, the EPVP is defined as a sequential process of repeating VAs and CAs at T time events, as shown in Fig. 5.1.

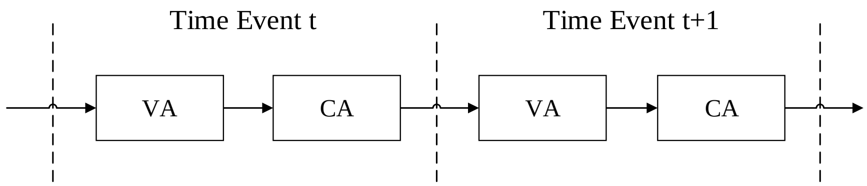


Fig. 5.1 Illustration of two time events of an EPVP

At each time event, verification planning consists of assigning a VA and a CA from their own activity spaces, each of which is a set of all eligible activity actions, including the action 'NA'. It is notable that there are two constraints about the activity spaces of VAs and CAs. First, implementing a CA can make the current results of a VA invalid if the result of the VA depends on the corrected parameter. This is because once a CA changes a system parameter, all existing verification results that depend on such a parameter lose their credibility in deducing accurate posterior beliefs of the system. Second, it is unnecessary to repeat any activity if the previous result of such an activity remains valid. That is, if an activity has been executed and its result can still be used to update beliefs, such an activity is not included in the activity space. According to these two constraints, it can be inferred that each VA may be executed multiple times if some CAs influence the relevant parameters of such a VA. Therefore, the activity spaces of VAs and CAs depend on the system state when the decision is made, and the specific set of activity actions constantly changes along the entire verification process.

With the extended paradigm, the solution of verification planning is the assignment of VAs and CAs along a verification process, which is referred to as the joint verification-correction strategy (JVCS). Because each VA has multiple possible results (e.g., *Pass/Fail*), all possible activities of a VP can be presented as an activity tree. One example of an activity tree is shown in Fig. 5.2. Each path from the root node to a terminal node is called a verification path in this study. In the example in Fig. 5.2, there are 5 verification paths, and all verification paths share the same initial activity S_1. System states are generated along with the collection of activity results at each verification path. Each verification path terminates with a particular system state, which is called the terminal state (denoted by 'Stop', as shown in Fig. 5.2).

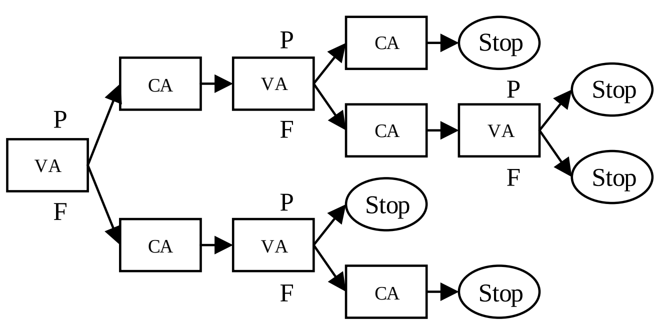


Fig. 5.2 One JVCS example (P: pass; F: fail)

5.1.2 Performance Measurement of JVCSs

While the paradigm of verification processes is extended with CAs, this extended paradigm shares the same purpose as previous paradigms: maximizing confidence in verification coverage, minimizing the risk of undetected problems, and minimizing invested effort [20]. These three aspects can be assessed in terms of their inherent value with respect to satisfying the objective of the project. Thus, a value-based model can be used to blend the three objectives as a unified value function [184]. This value function is used to measure and compare the performance of JVCSs.

In this dissertation, three value factors are considered to calculate the value function of a JVCS. The first value factor is activity costs, a fixed amount of financial resources necessary to conduct either a VA or a CA. It is denoted as $C\left(\mu_j\right)$ for a VA μ_j and $C\left(\varphi_\mathrm{k}\right)$ for a CA φ_k. For example, if a rework activity is executed to replace a faulty element, corresponding activity costs could include the purchasing cost of new elements and labor fees to replace the elements. The second value factor is failure costs, $C\left(A(\mu_j)=Fail\right)$, which are incurred when the result of a VA is found to be *Fail*. The third value factor is system revenue, $B\left(\theta_i\right)$, which is obtained once the system is deployed and operates correctly. $B\left(\theta_i\right)$ depends on the evolution in confidence that the system is operating correctly as VAs are performed. The system is considered to be deployed only when the confidence level $P\left(\theta_i = Pass\right)$ of the target parameter θ_i at a system state S_m reaches or surpasses the certain threshold, H_i.

During a verification process, these value factors arise at different points in time. Activity costs are incurred when the activity is executed. Failure costs are incurred if the *Fail* results of VAs are collected. The system revenue is obtained only when a verification process terminates (i.e., reaches a terminal state) and the system is deployed. For simplicity, all of these value factors are summarized at each terminal state. Each verification path could be stopped in two situations. First, the confidence levels of all target parameters reach their thresholds H_i. Second, the action 'NA' is selected when assigning a VA.

Consider a JVCS Ψ that starts from a system state S_1 and has W verification paths $Z_1,...,Z_W$ and each verification path has a set of system states $\{S_m\}_w$. For each verification path Z_w, the overall value is calculated as:

$$\begin{aligned}\mathrm{U}\left(\mathrm{Z}_w\right) = \sum_i B\left(\theta_i\right)\mathrm{P}\left(\theta_i{=}Pass|\,\mathrm{Z_w}\right)\delta\left(\mathrm{P}\left(\theta_\mathrm{i}{=}Pass|\,\mathrm{Z_w}\right) > H_i\right) - \sum_j C\left(\mu_j\right) \\ - \sum_j C\left(A(\mu_j)=Fail\right)\delta(A(\mu_j)=Fail) - \sum_k C\left(\varphi_k\right),\end{aligned} \tag{5.1}$$

where $\delta\left(\cdot\right)$ is an indicator function whose value is 1 if the statement is true and 0 otherwise. Since a JVCS consists of a set of verification paths, the overall values of all verification paths should be considered to inform verification planning. Therefore, the performance of this JVCS $U\left(\Psi\,|\,S_1\right)$

can be calculated as the expected value of a verification process $E\left[U(Z_w)\right]$, which is the weighted sum of the overall values of all verification paths:

$$U\left(\Psi \mid S_1\right) = E\left[U(Z_w)\right] = \sum_w P\left(\mathrm{Z}_w\right)U\left(\mathrm{Z}_w\right), \tag{5.2}$$

where $P\left(\mathrm{Z}_w\right)$ is the probability of a verification path Z_w that is calculated by multiplying state transition probabilities (i.e., the probability of an activity result) along the verification path:

$$P\left(\mathrm{Z}_w\right) = \prod_m \mathrm{P}\left(A \middle| S_m\right). \tag{5.3}$$

The verification planning problem is to solve for the optimal JVCS for a given initial system state S_1:

$$\Psi_{opt} = \arg\max_{\Psi} U\left(\Psi \mid S_1\right). \tag{5.4}$$

5.1.3 Order-based Backward Induction Method

As shown in Eq. 5.4, the verification planning problem can be conceptualized as a search for an optimal JVCS. A method that efficiently copes with this sequential decision-making problem in terms of computational resources is lacking. While the specific search method is independent of the JVCM presented in this chapter, one such method is suggested to respond to this need. Because the belief distributions of all activity results can be obtained from the BN model and collected evidence, this planning problem satisfies the assumption about the perfect model of dynamic programming (DP) methods [185]. Thus, the procedure of DP methods is followed to analyze this verification planning problem using the properties of verification processes. Our analysis is organized into three steps.

First, the state concept in DP methods (called system state in this dissertation) is used to guide the search for optimal JVCSs. However, as activity spaces changed along the verification process, it was insufficient to represent each system state only by their time events. Instead, the history of all previous activities and the activity type of the current activity, needed to be used to determine the current system state before implementing the activity. In this study, a vector is used to represent each system state S_m in the *w*th verification path at the *t*th time point, and defined it as $S_m = \left(Y(S_m), Y(\varphi_1), Y(\varphi_2), \ldots, Y(\varphi_K), Y(\mu_1), Y(\mu_2), \ldots, Y(\mu_J)\right)$, with $w = 0, \ldots, W$ and $m = 1, \ldots, M$, where:

- $Y(S_m)$ denoted the type of the current activity (i.e., 0 for VA and 1 for CA).
- $Y(\varphi_k)$ denoted the type of the result of a CA on its system parameter θ_i (e.g., its values are $\{0,1\}$ for not corrected or corrected, respectively).

- $Y(\mu_j)$ denoted the type of the result of a VA μ_i (e.g., its values are $\{0,-1,1\}$ for not verified, verified with a *Fail* result, and verified with a *Pass* result, respectively).

According to the extended paradigm in Section 3.1, there are four constraints for this representation. (a) if $Y(S_m)=0$, the activity space is the set of VAs whose result type is 0 (i.e., $Y(\mu_j)=0$) and an action 'NA' for no activity. (b) if $Y(S_m)=1$ and the result of the previous VA is *Pass*, the activity space of CAs has only one action, 'NA'. (c) if $Y(S_m)$ is 1 and the result of the previous VA is *Fail*, the activity space of CAs is the action 'NA' and the set of CAs whose result type is 0 (i.e., $Y(\varphi_k)=0$). (d) whenever the value of $Y(\varphi_k)$ for θ_i is changed from 0 to 1, the values of all $Y(\mu_j)$ must be set to 0 if μ_j is the child node of θ_i in the BN.

Second, because this representation has included all necessary information of a system state, S_m, the strategy that starts from S_m only depends on this system state S_m. In this dissertation, the expected value of S_m is represented by the expected value of the strategy that starts from S_m, which is denoted as $U(S_m)$. Therefore, the Bellman equation in dynamic programming [185] can be used to deduce the expected value $U(S_m)$ according to the recursive relationship between the expected value of system states. That is, if the expected values of all following system states are found, the expected value of the current system state can be found by comparing all following ones and using the optimal following system states to calculate the current expected value. Furthermore, if the optimal expected values of all system states are found, the optimal activity for each system state can be generated by identifying the following system state with the optimal expected value. The set of all optimal activities along a verification process constitutes the optimal JVCS.

Third, it should be noted that, since VAs cannot be repeated when successful, all system states are irreversible. That is, whenever a system state S_m occurs at a time point, this system state will never occur again in all its verification paths. Even though a CA may reset the results of VAs, this reset action is also recorded in the vector representation. According to this property, there exists a decision graph between all system states even though this decision graph is not explicit. Therefore, an order-based backward induction method is proposed to search for the optimal JVCS.

The proposed method is implemented in two steps to solve the verification planning problem. The first stage aims to determine the decision graph among all system states, called the order iteration stage. All system states of a verification process are listed and initialized with an order function $Q(S_m)=0$. For each process state S_m, the order functions of its next process state $\{Q(S_{m'})\}$ are examined by following their sequential relationship. If the value of each order function $Q(S_{m'})$ is not larger than that of $Q(S_m)$, then the value of the order function $Q(S_{m'})$ will be assigned with the new value $Q(S_m)+1$. This assignment ensures that all order functions of a JVCS are always increasing. All system states are iteratively examined until there is no more change in their order functions. The second stage follows the traditional backward induction method [186] to evaluate the state values of all system states, which is called the backward induction stage. All process states are arranged in decreasing order according to their order functions. Then the value of each system state is updated by comparing all following states and selecting the best ones from them, which is summarized as the equations:

$$\mathrm{U}(S_m)=\begin{cases} B(\theta_i)\mathrm{P}(\theta_i|S_m),\ \text{if } \mathrm{P}(\theta_i|S_m)>H_i; \\ 0,\ \text{else if 'NA' is selected as a VA}; \\ \max\limits_j\left(-C(\mu_j)-\mathrm{P}(\mathrm{A}(\mu_j)=Fail)C(\mathrm{A}(\mu_j)=Fail)+\sum\limits_a \mathrm{P}(\mathrm{A}(\mu_j)=a|S_m)\mathrm{U}(S_{m'}|a)\right),\ \text{else if } Y(S_m)=0; \\ \max\limits_k\left(-C(\varphi_k)+\mathrm{U}(S_m|\varphi_k)\right),\ \text{else if } Y(S_m)=1. \end{cases} \tag{5.4}$$

Algorithm 5.1. Order-based Backward Induction Algorithm

Input: $\{S_m\}$ —— all system states;

$\{Q(S_m)\}$ —— order functions of all system states;

1. Initialize order functions $\{Q(S_m)\}$ with 0.

2. While all order functions of system states do not change:

 2.1 For each system state S_m in $\{S_m\}$:

  2.1.1 List all next system states $\{S_{m'}\}$.

  2.1.2 If $Q(S_{m'})\le Q(S_m)$:

  2.1.3 $Q(S_{m'})=Q(S_m)+1$.

3. Rank all system states into a list $\{S_m\}_{\text{ordered}}$ in descending order according to $Q(S_m)$.

4. For each system state S_m in $\{S_m\}_{\text{ordered}}$:

 4.1 Update the state value according to Eq. 5.4.

5. Generate the set of all system states $\{S_m\}_{\text{opt}}$ along the VP from S_1.

6. Identify the optimal activities from $\{S_m\}_{\text{opt}}$ as the optimal JVCS Ψ_{opt}.

7. Return the optimal JVCS Ψ_{opt} as the exact solution.

When the values of all system states are determined, the optimal JVCS is generated by identifying the optimal activities for all possible system states along the verification process. The corresponding algorithm is shown in Algorithm 5.1.

5.2 Case Study

5.2.1 Problem Description

This section leverages a notional satellite communication instrument in a satellite as a demonstrative case to validate our framework. The notional instrument has been used to support prior research in verification [66]. Consider that the instrument is formed by a Signal Generator, an Amplifier, and an Antenna, as depicted in Fig. 5.3. The scope is restricted to the following performance parameters: The Effective Isotropic Radiated Power (EIRP) of the communication instrument (denoted by θ_6) is used as the primary target, which is characterized as a function of the Signal Generator output power (denoted by θ_1), the Amplifier gain (denoted by θ_2), and the Antenna gain (denoted by θ_3). Furthermore, the output power of the integrated assembly formed by the Signal Generator and the Amplifier is considered an intermediate system parameter of potential interest for the verification campaign, which is denoted by θ_4. In addition, a prototype for this communication instrument is included as a potentially interesting parameter for the verification campaign, whose EIRP is denoted by θ_5. Each parameter is subjected to a dedicated VA, denoted by $\mu_1,...,\mu_6$, where μ_j provides information about θ_i for $i=1,...,6$. While this instrument is verified through a set of VAs, each parameter has CAs to correct potential errors and defects. Without loss of generality, this case study takes repair activities as an example of CAs. Therefore, the term 'CA' always refers to repair activities in this case study.

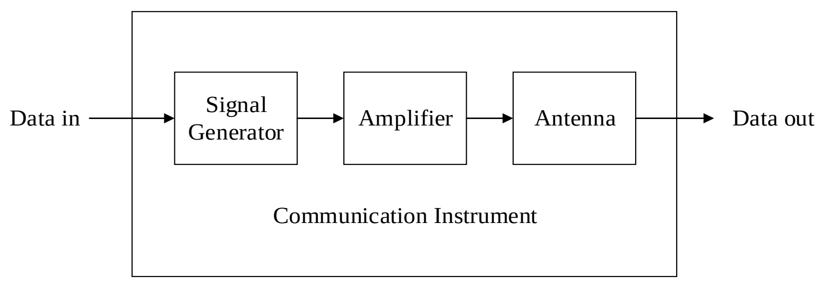


Fig. 5.3 The instrument structure

To illustrate the advantage of the proposed JVCM, I compare the resulting verification strategy against those obtained using benchmark rules from the literature. Specifically, the rule-based corrections in current planning methods are employed as identified in Section 2.1, which is the only method to solve verification planning problems, as far as I know. These benchmark methods are based on two primary principles. First, if a VA $A(\mu_j)$ fails, a CA is triggered automatically as long as the confidence $P(\theta_6)$ drops below a predefined threshold, H_{repair}. Second, it is assumed that a CA raises confidence in the correct operation of the system to the level that would have been attained if the last VA before the CA had been successful.

5.2.2 Models and Data

The system parameters of this optical instrument and their possible VAs are modeled as the BN shown in Fig. 5.4. System parameters are denoted by θ_i and candidate VAs are denoted by μ_j. For clarity, it should be noted that the state of the performance of the integrated assembly (θ_4) depends on the performance of the Signal Generator (θ_1), the performance of the Amplifier (θ_2), and the performance of the cabling connecting them (embedded in θ_4). Similarly, the performance of the overall communication payload (θ_6) depends on the performance of the prototype (θ_5), the performance of the integrated assembly (θ_3), the performance of the Antenna (θ_4), and the performance of the cabling between them (embedded in θ_6). The CPTs of all nodes are synthetic and have been generated using probabilistic causal interaction models (i.e., generalized noisy-OR and noisy-AND models) [83, 172, 173, 174], which take into account the physical meaning of the different nodes when estimating their mutual effects for reasonability of the data. This approach has been used in prior verification research [66, 96].

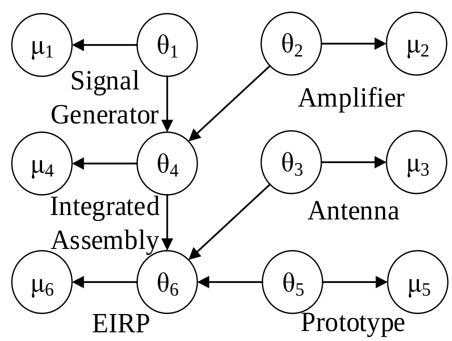


Fig. 5.4 BN for the case study

In this experiment, it is assumed that system revenue is driven by system parameter θ_6. Hence, θ_6 is set as the single target parameter. Its marginal prior confidence is $P(\theta_6) = 0.676$. The threshold for the system deployment rule is set as $H_6 = 0.90$. All value factors have also been synthetically generated in thousand-dollar units, \$1,000. The activity costs, as well as the failure costs of VAs, are provided in Table 5.1. The likelihood ratios of all repair activities are also provided in the fourth row of Table 5.1. The revenue $B(\theta_i)$ has been set to 20,000, which is not larger than the summation of all costs. The purpose is to embody the selection tradeoff between different activities in practice. That is, the expected value of a suboptimal strategy can be negative. Two metrics are used to evaluate the performance of the proposed methods. First, the expected value of a JVCS is shown in Eq. 5.2. Second, the running time of the program means wall-clock time, which is used to compare computational efficiency.

Table 5.1. Costs and likelihood ratios of activities

Activity	$\varphi_1(\theta_1)$	$\varphi_2(\theta_2)$	$\varphi_3(\theta_3)$	$\varphi_4(\theta_4)$	$\varphi_5(\theta_5)$	$\varphi_6(\theta_6)$	μ_1	μ_2	μ_3	μ_4	μ_5	μ_6
Activity Cost	1000	1000	1000	2000	500	8000	200	200	200	400	100	500
Failure Cost	-	-	-	-	-	-	1000	1000	1000	3000	0	15000
Likelihood Ratios of Repair Activities	18:1	18:1	18:1	8:1	3.5:1	8:1	-	-	-	-	-	-
Benchmark Cost	-	-	-	-	-	-	1000	1000	1000	2000	500	8000

5.2.3 Results and Analysis

JVCM. With this experimental case, the order-based backward induction algorithm has been conducted to solve for an optimal JVCS. The proposed JVCM and algorithm are realized with Python 3.6 and Bayes Net Toolbox for Matlab [187]. With 6 system parameters and 6 VAs in this BN, there are $2\times 2^6 \times 3^6 = 93312$ total system states. After the order iteration, the order functions of all of these system states are updated with 20 iterations for 1,059 sec. The resulting order functions of all system states range from 1 to 38. When the backward induction concludes, the optimal activities at all states are identified to constitute the optimal JVCS. With 20 verification paths (i.e., the number of terminal 'Stop' states), the optimal JVCS has 193 tree nodes (excluding terminal 'Stop' states). The depth of this JVCS is 14 nodes (i.e., 7 time events). Due to limited space, only the first 5 time events are plotted in Fig. 5.5. Each node of the JVCS suggests optimal activity at the corresponding state. For example, at the initial state, it is suggested to verify μ_5 first. If its result is *Fail*, $P(\theta_6)$ drops to 0.609 and a CA on θ_5 is suggested. However, if its result is *Pass*, $P(\theta_6)$ increases to 0.710 and no CA is necessary. Then the verification process goes to the next time event. The expected value of this optimal JVCS is 11,523, according to Eq. 5.2 and 5.3. The total running time is 20,472 sec.

Benchmark. Some verification strategies are also solved with current correction rules, as introduced in Section 4.1. In this experiment, three different thresholds, H_{repair}, are evaluated, specifically 0.2, 0.5, and 0.9. The largest value of H_{repair} is set to 0.9 because the system is deployed once $P(\theta_6)$ reaches $H_6 = 0.90$. The CA cost followed by a VA μ_j is set the same as that repair cost on θ_i, which is the nearest parent node of μ_i. They are shown in the fifth row of Table 5.1. Because there are no decisions for CAs in these rules, the system state only depends on 6 VAs each, which has four states including 'Not Verified', 'Verified with the Fail Result', 'Verified with the *Pass* Result', and 'Corrected'. Thus, there are $4^6 = 4096$ system states. The order-based backward induction method described in Section 5.1.3 was employed and implemented using the same tools (Python 3.6 and Bayes Net Toolbox for Matlab) [187]. The optimal strategies are shown in Fig. 5.6. The expected values of the strategy are {4584, 8819, 9816} for H_{repair} = 0.2, 0.5, and 0.9, respectively. The total running times are {3926, 3689, 3552} sec for H_{repair} = 0.2, 0.5, 0.9, respectively.

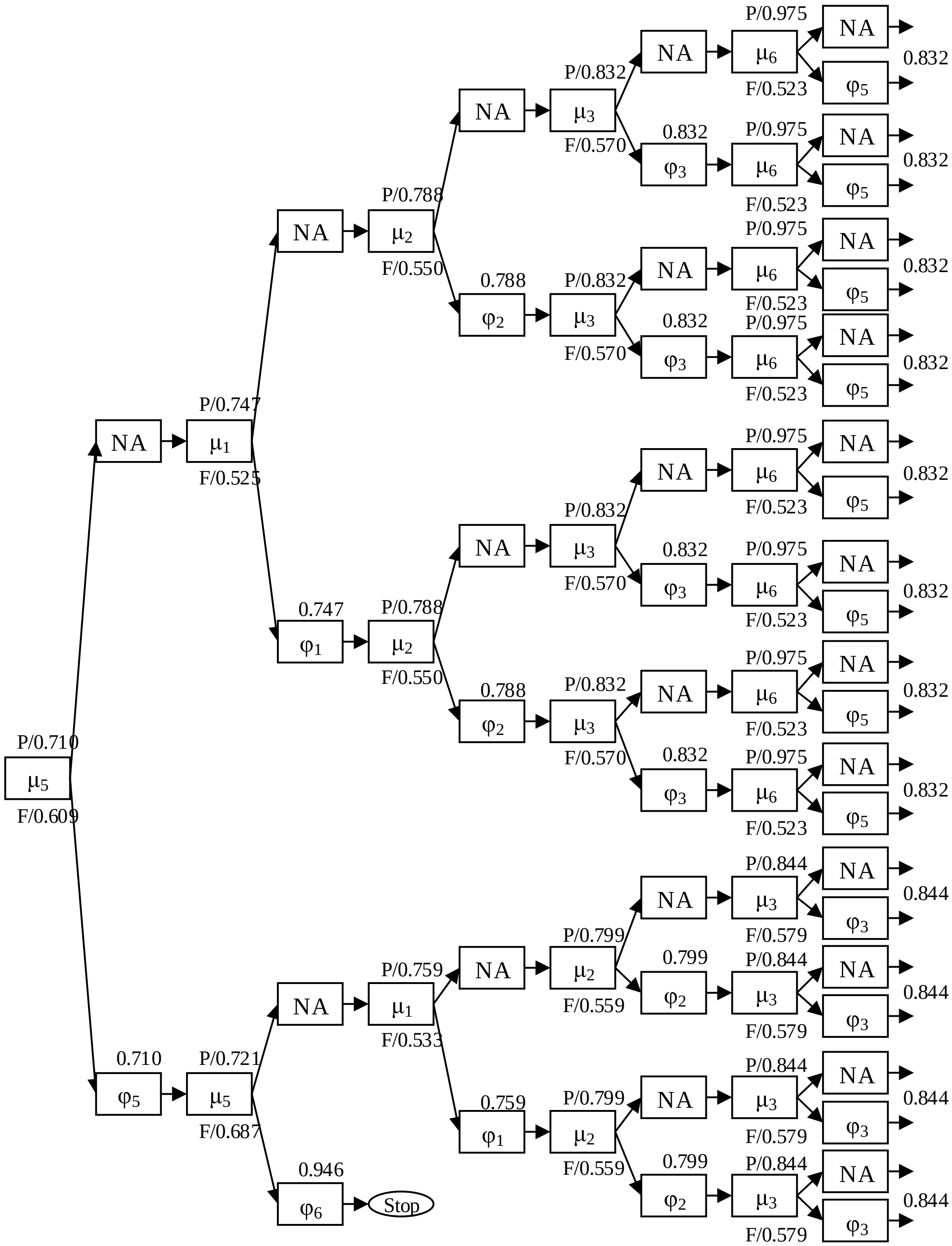
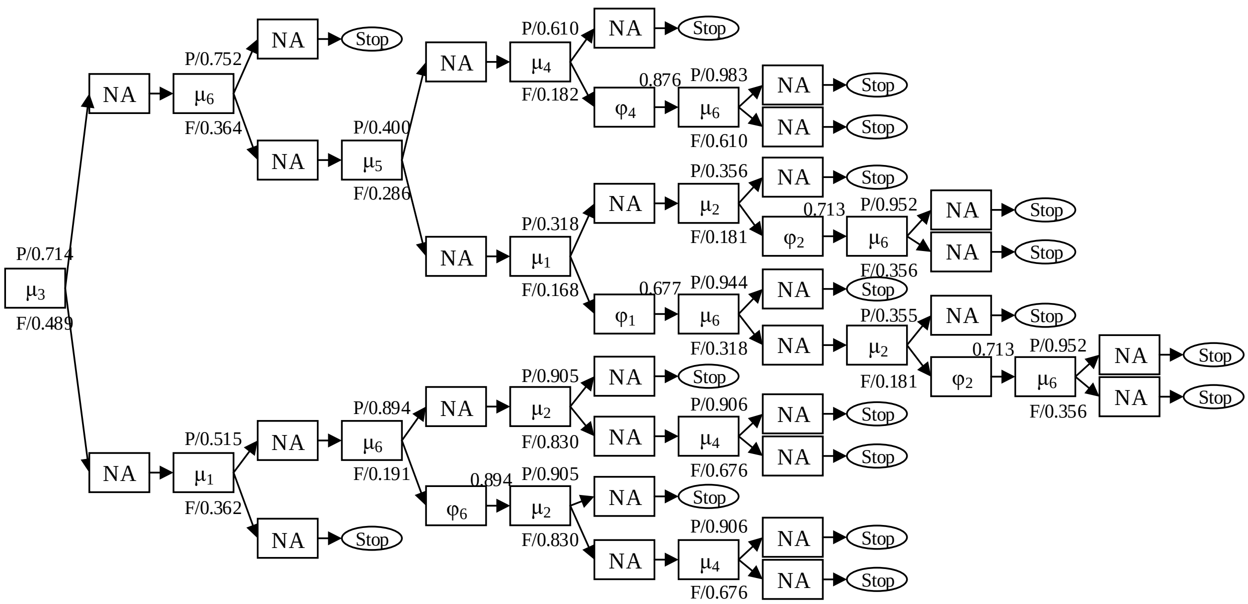
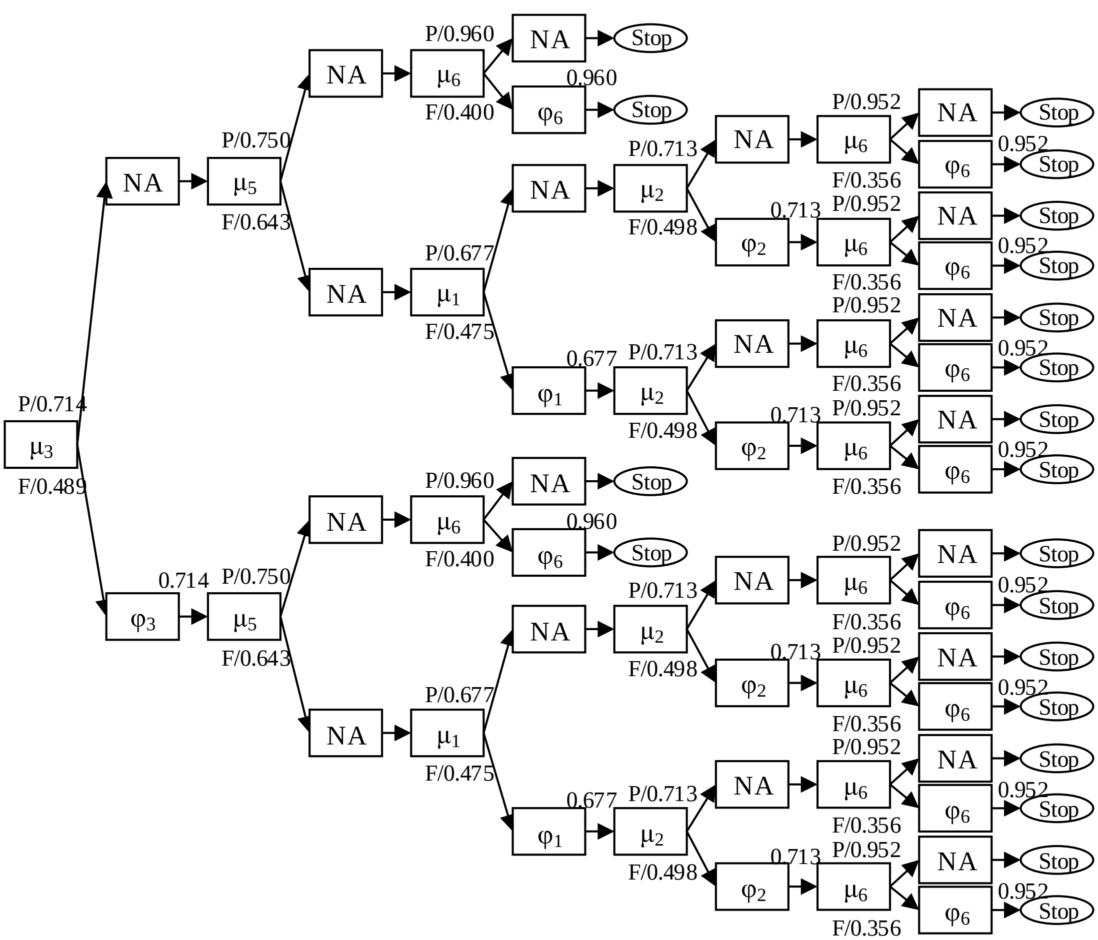


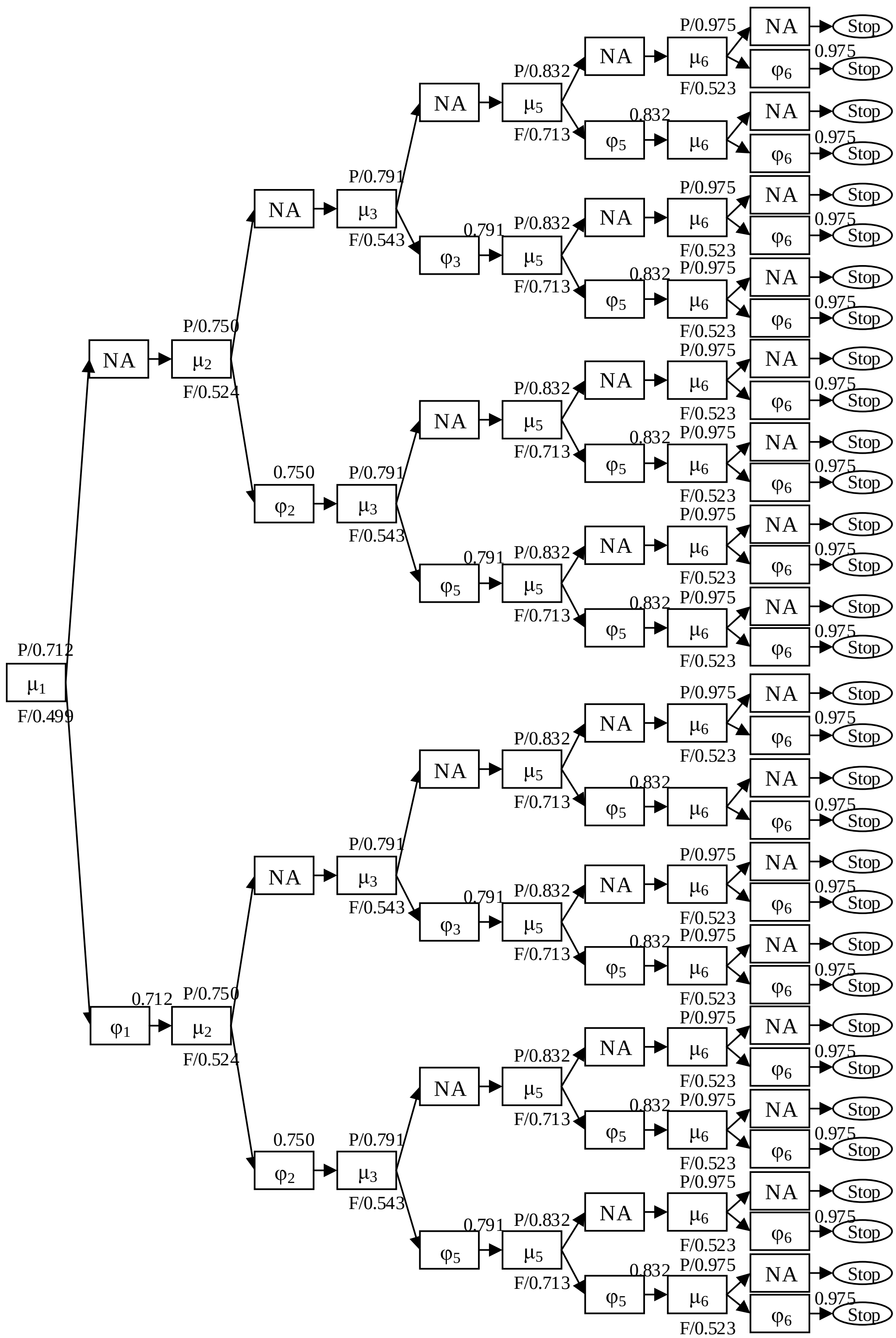
Fig.5.5 Partial graph of the optimal JVCS (P: pass; F: fail)



(a) H_{repair} = 0.2



(b) H_{repair} = 0.5



(c) $H_{repair} = 0.9$

Fig. 5.6 Strategies of the benchmark rules (P: pass; F: fail)

Findings. As shown in Table 5.2, using the proposed JVCM improves the performance of system verification by at least (11523-9816)/9816= 17.4%, which shows the advantage of our model in absolute terms. This is attributed to the explicit decisions of CAs. As more CA choices are provided at each system state in the JVCM, CAs can be selected in a larger activity space and choose the one with the best expected value. That is, for each parent system state, the child system state with better performance can be found under the JVCM. If this deduction is repeated for all system states, the performance of the initial system state (i.e., the performance of the JVCS) is higher than those of the benchmark rules. However, the computational time required to use the proposed JVCM is much larger than when using the benchmark rules, specifically by a factor between 20472/3926 = 5.21 and 20472/3552 = 5.76. This is also explained by the gap in the number of activities. With more CAs provided, the total number of system states increases from 4,096 to 93,312. As all system states have to be examined, the JVCM has a higher overhead. This time difference does not account for the time required to construct the model, which grows significantly when including CAs as decision nodes.

Table 5.2. Comparison between JVCM and benchmark rules

	Expected Value	Running Time (sec)
JVCM	11,523	20,472
Benchmark Rule $H_{repair} = 0.2$	4,584	3,926
Benchmark Rule $H_{repair} = 0.5$	8,819	3,689
Benchmark Rule $H_{repair} = 0.9$	9,816	3,552

These results provide interesting insights into the selection of paradigms. If the total number of system states is small and the computational resources are sufficient, the proposed JVCM may be worth using in verification planning. Otherwise, the benchmark rules may also be considered as alternative methods to obtain approximate strategies with less running time. In particular, these benchmark rules can be analyzed from two perspectives. First, CAs in the benchmark rules are tied with the parameter that has been verified, which leads to smaller activity spaces. More efforts can be spent on how to expand the activity sets of benchmark rules effectively in future work. Second, the predefined thresholds (i.e., H_{repair}) play the same role as the correction decisions of the JVCM that decide which CA is selected. It is found from this case study that among these three benchmark rules, H_{repair} = 0.2, 0.5, 0.9, the benchmark rules with higher thresholds always have better performance for verification planning. Especially if CAs are executed whenever a *Fail* result is collected (i.e., H_{repair} = 0.9), the generated strategy has the best performance. While there are other possible lower threshold values and each time event can be assigned with a different threshold, investigating the optimality of different threshold rules is reserved for future work. A systems engineer could use these findings to decide if using the JVCM is worth the additional modeling effort concerning the benchmark rules.

The time analysis for calculating the optimal JVCS also shows why our proposed method can outperform successive approximation methods. While the 20 iterations of all order functions cost 1,059 sec during the order iteration stage, it costs 20472-1059 = 19413 sec to update all state values during the backward induction stage. So, updating the order functions is 19413/(1059/20) = 367 times more efficient than updating the state values. After updating order functions between system states, the state value of each system state only needs to be updated once. In contrast, successive approximation methods solve for the optimal strategy by iterating all system states two or more times. If all states were evaluated twice in the best scenario, the total running time was approximately 19413*2 = 38826, much larger than that of our proposed method. Thus, the order-based backward induction method is the best choice to solve for exact optimal verification strategies.

5.3 Discussion

This chapter has presented a joint verification-correction model that jointly incorporates VAs and CAs as dedicated decisions. The JVCM uses an extended paradigm to establish the dependency relationship between VAs and CAs with all necessary constraints along with a VP. A value-based model is also incorporated with three value factors to measure the performance of the verification strategies. It is noticeable that only these necessary constraints and value factors are considered in the proposed JVCM to simplify the VP model. This simplification is assumed to be sufficient for research analysis in this dissertation. However, VPs in practice may have other constraints and value factors. For example, multiple activities may be executed simultaneously while only one activity is allowed in the proposed JVCM. So, in those cases, some modeling modifications are needed before applying the JVCM to real applications.

According to the dynamic nature of the verification planning, an order-based backward induction method is provided to seek the exact optimal strategies that specify the optimal activities for all possible system states. This order-based backward induction method is robust to the modeling of VPs, because it has only one requirement that sequential dependency between system states can be determined. That is, if other types of process constraints and value factors are adopted in the EPVP, this method still works. However, if the strategy space is large, this method becomes infeasible as it relies on the value update of all system states. Thus, this method can solve for JVCSs in small strategy spaces.

The use of the JVCM was illustrated with a case study. A notional satellite communication instrument was used to evaluate the performance of the proposed JVCM against three benchmark rules that represent current correction rules in the literature. The instrument system and all activities were modeled according to their physical meaning in verification planning, and all data were generated synthetically. The results of the case study show that the proposed JVCM outperforms benchmark rules with a larger expected value, but it costs more computational time. This difference is attributed to the explicit decisions of CAs provided by the extended paradigm. The advantage of the proposed search algorithm was also shown by time analysis that the proposed search algorithm saves computational time by leveraging the order functions of system states.

CHAPTER 6: Designing JVCSs in Large Strategy Spaces

This chapter presents a UCB-based tree search approach for large-scale engineered systems. The proposed approach can generate near-optimal JVCSs that fit the engineering requirements within finite computational resources. The content of this chapter has been submitted in [188]. First, verification planning is simplified as repeatable bandit problems, and an upper confidence bound rule for repeatable bandits (UCBRB) is presented with the optimal regret bound. Next, a tree search algorithm is proposed to search for near-optimal JVCSs. A tree-based ensemble learning model is also used to extend the tree search algorithm to handle local optimality issues.

The rest of this chapter is organized as follows. Section 6.1 presents the proposed UCB-based tree search approach, including the UCBRB rule, UCBRB1 tree search method, and function approximation models. Section 6.2 uses a demonstrative case to illustrate the performance of the approach. A discussion of this chapter is made in Section 6.3.

6.1 Proposed UCB-based Tree Search Approach

6.1.1 Upper Confidence Bound for Repeatable Bandits (UCBRB)

As each verification path has a set of its system states, all system states of a JVCS can be represented as an AND/OR tree. In any system state whose following system states are not fully explored, verification planning shares the uncertain characteristic with bandit problems. That is, when an activity is selected, the expected reward of this activity follows an unknown distribution. However, verification planning differs from a sequential play of bandit problems. As long as the prior knowledge about the target system is determined, implementing a given JVCS always generates the same expected reward by enumerating all possible verification paths, which means a JVCS is repeatable. So, a repeatable bandit problem (RBP) is investigated in this section first as a simplification of verification planning.

Consider a K-armed bandit problem where K machines are played sequentially, and only one machine is played each time. The reward of a level pull of each machine is represented by a random variable $X_{k,s}$ for $1 \leq k \leq K$, where k is the index of a machine. Successive plays of machine k yield rewards $X_{k,1}$, $X_{k,2}$, …, which are independent and identically distributed (i.i.d.) according to an unknown distribution F_k. The support of F_k is $[a_k, b_k]$. The distributions of all machines are independent from each other; i.e., $X_{k,s}$ and $X_{k',s'}$ are independent (and usually not identically distributed) for each $1 \leq k < k' \leq K$ and each $s, s' \geq 1$. Assume that whenever a reward is collected from machine k, the player can remember the tricks about reproducing such a reward by controlling the pulling factors, such as reaction time point, pull speed, and pull length. So, when the player has collected some pull results of all machines, they can select the machine of the optimal reward and repeat some previous level pull to obtain the same reward. This bandit problem is defined as RBP in this dissertation. The objective of an RBP is to maximize the sum of rewards earned through a sequence of pulls.

The main difference between traditional bandit problems and the RBP is the expected reward of each machine. In traditional bandit problems, the reward of a level pull is a random observation of machine k. So, the expected reward of machine k is the expectation of F_k. However, in RBPs, only the maximum reward of machine k will be considered for further repetition. So, the expected reward of machine k can be represented by the observation result with the maximum reward after n_k plays of machine k, where $n_1 + n_2 + \ldots + n_k = n$ and n is the total plays of all machines. Other results with lower rewards are not considered for future repetition anymore. As each machine of an RBP has its distribution with a supremum $u_k \leq b_k$, the maximum among the supremum rewards of all bandits is defined as the overall supremum $u_* = \max_k(u_k)$. The previous study [142] is followed to define the regret of an RBP as below:

$$u_* \bullet n - u_k \sum_{k=1}^{K} E(\mathrm{n}_k)\,, \tag{6.1}$$

That is, the regret of an RBP is the expected loss due to the fact that the player does not always repeat the optimal play.

For bandit problems, a policy is a strategy that chooses the next machine to play based on the sequence of past plays and obtained rewards [143]. Previous policies, such as the UCB1 [143], provide upper confidence bound for each machine to optimize the accumulated regret. However, as these policies use the expectation of a machine as the target, they are not appropriate choices for RBPs. Thus, a modified policy called upper confidence bound for repeatable bandits is proposed in this chapter:

> **Initialization**: play each machine once;
>
> **Loop**: play machine k that maximizes $x_k^{\max} + \dfrac{4\ln(\mathrm{n})}{D_0 \bullet n_k}$, where $x_k^{\max} = \max(x_{k,1}, x_{k,2}, \ldots, x_{k,n_k})$ is the maximum reward obtained from machine k, n_k is the number of times machine k has been played so far, D_0 is a constant that is determined by the distributions of all machines, and n is the overall number of plays.

It is noticeable that D_0 is the minimum of $\{D_{0,k}\}$ and each $D_{0,k}$ is the bound constant of the distribution of each machine. As the distribution of each machine is unknown, the constant $D_{0,k}$ can be estimated according to presumptive distributions and collected samples. For example, assume the distribution of any machine F_k is a uniform distribution $[a_k, b_k]$. The range b_k-a_k can be estimated as $\dfrac{n+1}{n-1}(x_k^{\max} - x_k^{\min})$. So, the estimate of D_o is $\min_k \dfrac{n-1}{(n+1)(x_k^{\max} - x_k^{\min})}$.

The regret bound of this policy is summarized as the theorem below. The proof of this theorem is provided in Appendix A.

Theorem 1: For all K > 1, if the UCBRB rule is run on K machines having arbitrary reward distributions with support in [0, 1], then its expected regret after any number n of plays is at most O(ln(n)).

6.1.2 UCBRB1 Tree Search Method

As verification planning can be viewed as a sequential play of RBPs, the proposed UCBRB rule is used to calculate the UCB of the expected reward of a nonterminal system state S_m:

$$\mathrm{UCB}(\mathrm{S}_m)=x_k^{\max}+\mathrm{D}_6\frac{\ln(\mathrm{n})}{n_k}, \tag{6.2}$$

where $x_k^{\max}$ is the maximum expected reward of n_k JVCSs that starts from a state S_m (i.e., $\max_{\Psi}(U_{\Psi}(S_m))$), n_k is the visit count of S_m, n is the overall visit count of its preceding system state S_{m-1}, and D_6 is a constant that depends on the unknown distributions of the state value, which may be obtained through sensitivity analysis in practice. If the range of $x_k^{\max}$ is larger than 1, the first item may be replaced by $x_k^{\max}/D_7$, and D_7 is a discount constant to normalize the range. With this UCBRB rule, system states are evaluated with the balance between exploration and exploitation. In addition, if the confidence of target parameters reaches the threshold H_i or a 'NA' is selected as a VA, the corresponding system state is a terminal state, and the reward of this state is deterministic no matter how many more times it is visited. So, the UCBs of these terminal states are simply their maximum reward value, and the second item in Eq. 6.2 is 0.

As a verification process consists of multiple time events, the UCBRB rule is insufficient because it only solves the comparison between the activities at a given system state. Therefore, a UCBRB1 tree search method is proposed to make sequential decisions along a verification process. The method consists of a set of loops, with each loop having two stages. First, an AND/OR tree is generated in a forward way where each nonterminal state is expanded as a tip node and an activity is selected to generate its following states. Then, the set of nonterminal states is updated by removing the expanded state and adding the following nonterminal states. This expansion step is repeated until the set of nonterminal states is empty. In particular, all feasible activities are determined first according to the activity constraints in Section 5.1.1. Then each activity is evaluated with the UCB values of all its following states. The activity with the largest value is chosen, as shown in Eq. 6.3:

$$\begin{aligned}&\mu_*=\arg\max_{\mu_j}(-C(\mu_j)-C(A(\mu_j)=Fail)P(A(\mu_j)=Fail)+\sum_a P(A(\mu_j)=a)\,\mathrm{UCB}(\mathrm{S}_{m+1}\,|\,a)\text{, or}\\&\varphi_*=\arg\max_{\varphi_k}(-C(\varphi_k)+\mathrm{UCB}(S_{m+1}|\varphi_k)),\end{aligned} \tag{6.3}$$

where S_{m+1} means the next system state after conducting the activity, and a represents *Pass* or *Fail*. Second, the node information of all tree nodes, including expected reward and visit counts, are updated backwardly from terminal nodes to the root node. That is, the expected value of each node is updated according to the expected values of their child nodes in this tree:

$$\begin{aligned}&\mathrm{U}(\mathrm{S}_m)=-C(\mu_j)-C(A(\mu_j)=Fail)P(A(\mu_j)=Fail)+\sum_a P(A(\mu_j)=a)\,\mathrm{U}(\mathrm{S}_{m+1}\,|\,\mu_*)\text{, or}\\&\mathrm{U}(\mathrm{S}_m)=-C(\varphi_k)+\mathrm{U}(\mathrm{S}_{m+1}\,|\,\varphi_*).\end{aligned} \tag{6.4}$$

Then the expected rewards are compared with the previous record of this state and saved in the lookup table. If its value is larger than that in the lookup table, the record is updated with the larger one. The visit counts of all tree nodes are added by 1 in the lookup table.

The UCBRB1 tree search method is designed by extending the previous work [118] from three aspects to improve tree search efficiency. First, as all following states of an AND branch must be considered in an AND/OR tree, all nonterminal states of an AND branch are expanded simultaneously in the method. Second, the information of all nodes is updated only when the whole AND/OR tree is expanded completely rather than whenever a node is expanded. Third, the UCBs function as admissible functions to represent the heuristic value of a system state.

In addition, two adjustments are made to Eq. 6.1 in practice. First, while Eq. 6.1 is meaningless if either n or n_k is 0, the visit counts of all system states are 0 at the beginning. So, it is assumed that all system states have been visited once before the tree search, and, for simplicity, the expected value is set as *None*. Then, Eq. 6.1 is equivalent to:

$$\mathrm{UCB}(\mathrm{S}_m)=x_k^{\max}+\mathrm{D}_6\frac{\ln(\mathrm{n}+1)}{n_k+1}. \tag{6.5}$$

Second, because all following system states are expanded, the number of tree nodes increases exponentially along with the tree depth, making the strategy space very complex. To simplify the strategy space, a penalty item is added in Eq. 6.5 to prevent the over-expansion of trees:

$$\mathrm{UCB}(\mathrm{S}_m)=x_k^{\max}+\mathrm{D}_6\frac{\ln(\mathrm{n}+1)}{n_k+1}-f(m\,|\,\Psi). \tag{6.6}$$

For simplicity, the penalty item $f(m\,|\,\Psi)$ is assumed to be a function of the system state index when the tree is expanded. For example, $f(m\,|\,\Psi)=D_8\bullet\lfloor m/D_9\rfloor$, $D_8=1$ and $D_9=50$. That is, the UCB of a nonterminal state is reduced by 1 every 50 tree nodes. Finally, the UCBRB1 tree search method is summarized in Algorithm 6.1.

Algorithm 6.1. UCBRB1 Tree Search Method

Initialization: $\Omega = \{\emptyset\}$—— a lookup table stores the state values and visit counts of all visited system states;

$\Psi = \{\emptyset\}$—— an explicit graph consists of selected activities

1. Initialize Ψ with the initial system state S_1

2. While Ψ has some tip nodes:

2.1 Expand all tip nodes of Ψ according to the UCBs of the following states calculated by Eq. 6.6.

2.2 Add expanded states to Ψ.

2.3 Denote all nonterminal expanded states as tip nodes.

3. Update state values and visit counts of all nodes in Ψ according to Eq. 6.4.

4. Update Ω with the updated state values and visit counts.

5. If the expected value $U(\Psi|S_1)$ converges:

5.1 Output Ψ_{opt} as the solution.

5.2 Otherwise:

5.3 Go to 1.

6.1.3 Function Approximation of State Values

Even though the proposed UCBRB1 tree search method can solve for a near-optimal JVCS, the tree search process has local optimality issues. That is, the activities of a strategy become fixed at a system state after certain rounds of tree search, and it is impossible to explore other possible activities. A function approximation model is used to approximate state values by generalizing the values of collected system states to handle this issue. First, when a new nonterminal state is expanded, the selection of activities suffers from a lack of state information because all following states have not been visited before. However, if a prior state value is added to the calculation of UCBs, some promising action will be selected, and the tree search efficiency will be improved. Second, because the tree search process continuously collects state information incrementally, concept drifts occur in terms of the distribution of $U_\Psi(S_m)$. This concept drift can result in the immovability of activity selection, as shown in the experiment section. So, the function approximation model is expected to provide some heuristic information to narrow the gap of distributions between different system states. Thus, the tree search process can jump out of local optimum spaces.

To approximate state values, it is necessary to identify all factors contributing to the state value distribution. During a verification process, there are four types of independent information, including prior confidence of system parameters and activities, collected evidence of activities, value factors of a verification process, and policy rules (e.g., the UCBRB rule). Three kinds of variables are used as the input of the function approximation model to predict state values, as shown in Fig. 6.1. First, the posterior confidence values of all system parameters are used because they directly determine whether one system can be deployed. Second, the statuses of all activities, including {*Not Corrected*, *Corrected*} for CAs and {*Not Verified*, *Verified with a Fail Result*, and *Verified with a Pass Result*} for VAs, are denoted as a list of categorical variables. Third, the counts of all executed VAs and CAs are calculated as two count variables to distinguish node information. These count variables are redundant because they are the sum of all status values.

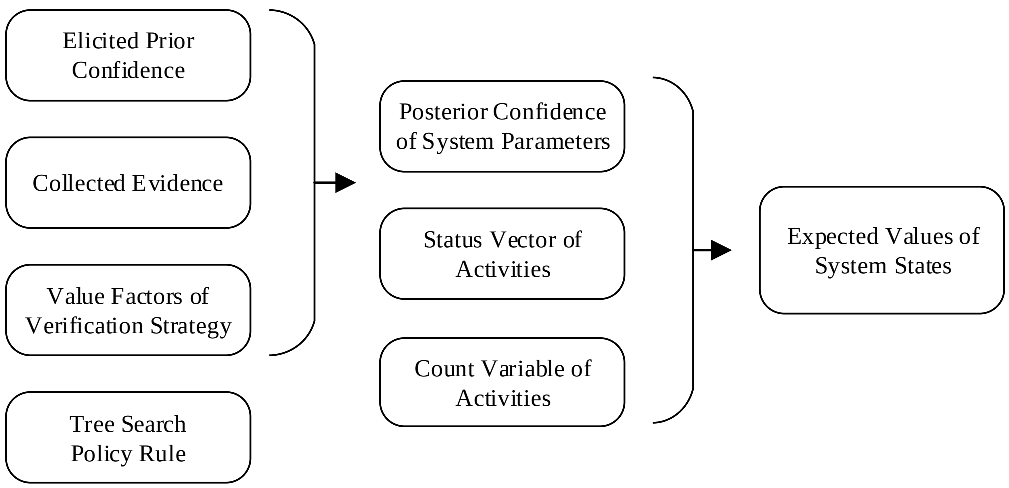


Fig. 6.1 Dependency between verification information and state values

With the analysis above, random forest regression (RFR) is leveraged to approximate the dependency relationship. The rationales for using RFR as a function approximation are as follows. First, the input variables are a mixture of continuous, categorical, and count ones. RFR models can accept such mixed variables as inputs. Second, all input variables are correlated because they depend on collected evidence. RFR models are robust to such redundant inputs. This is important because if an improper activity is selected, all following activities may be vain attempts.

RFR is added to extend the UCBRB1 tree search method in two places. First, while sample trees are generated continuously along the tree search process, RFR models are trained periodically along the tree search process. That is, tree search processes are divided into sampling periods and collect tree nodes of sample trees as training datasets at each sampling period. Because of the randomness of a tree search process, the dependency relationship of collected sample trees is random and always changes during the tree search process. That is, all training samples are not i.i.d. samples. So, RFR models are used to interpolate the datasets, and each decision tree is built with a zero mean squared error. That is, the information of all visited system states is saved accurately in the RFR models. Second, because of the concept drift of state value distributions, only the latest RFR model is used to predict state values as prior values. In particular, the *k*-th percentile of the outputs of all decision trees is used as the prior state value $\hat{x}_k$. Each prior state value is compared with the state value record in the lookup table. Thus, the UCB formula in Eq. 6.6 is extended as:

$$\mathrm{UCB}(\mathrm{S}_m)=\max(x_k^{\max},\hat{x}_k)+\mathrm{D}_6\frac{\ln(\mathrm{n}+1)}{n_k+1}-f(m\,|\,\Psi), \tag{6.7}$$

where the first item is the maximum between the prior value and the maximum record value.

Two adjustments are also made to the data processing to improve the dataset quality of training datasets. First, when each sample tree is generated, all terminal nodes of a sample tree are not included as training samples. If ‘NA’ is selected as a VA, this type of terminal node suffers larger variance of state values than nonterminal nodes because their following states have not been visited. If parameter confidence reaches the threshold, it is unnecessary to predict state values. Second, besides the tree nodes in the latest sampling period, another set of system states is uniformly sampled from the lookup table with replacement and added to the training datasets. This adjustment is made to reduce the bias caused by the dependency between sample trees. For simplicity, the number of sampled system states is the same as that of sample tree nodes. The extended algorithm is called the UCBRB2 tree search method in this dissertation. Finally, the UCBRB2 method is shown in Algorithm 6.2.

6.2 Experimental Design

6.2.1 Problem Description

This section implements the proposed framework to design a JVCS for an optical instrument in a satellite [52]. The notional instrument has been used to support prior research in verification [52]. This optical instrument's system parameters and possible VAs are modeled as the BN shown in Fig. 6.2. System parameters are represented as circle nodes and candidate VAs square nodes. The definitions of the nodes are given in [52], hence not presented here. Each node is characterized by its CPT. Their specific values are synthetic and have been generated using the generalized Noisy-OR and Noisy-AND model [83], which considers the physical meaning of the different modes when estimating their mutual effects for the reasonability of the data. While this instrument is verified through a set of these VAs, each parameter θ_i has CAs to correct potential errors and defects. Without loss of generality, this experiment takes only repair activities as an example of CAs. Each repair activity on parameter θ_i is denoted as $\varphi_k(\theta_i)$, $k = i = 1, \ldots, 10$.

Algorithm 6.2. UCBRB2 Tree Search Method

Initialization:	$\Omega = \{\emptyset\}$—— a lookup table stores the maximum expected values and visit counts of all visited states;
	$\Psi = \{\emptyset\}$—— an explicit graph consists of selected activities
	$\Phi = \{\emptyset\}$—— a set of state samples for RFR models

1. While the number of state samples in Φ is less than some threshold:

1.1 Initialize Ψ with the initial system state S_1.

1.2 While Ψ has some tip nodes:

1.2.1 Expand all tip nodes of Ψ according to UCBs calculated by Eq. 6.7.

1.2.2 Add all expanded states to Ψ.

1.2.3 Denote all nonterminal expanded states as tip nodes.

1.3. Update state values and visit counts of all nodes in Ψ according to Eq. 6.4.

1.4 Updated Φ with all tree node samples in Ψ.

2. Updated Φ with a set of system states sampled from Ω.

3. Train an RFR model with Φ.

4. Update Ω with Φ and then set Φ as an empty set.

5. If the expected value $U(\Psi|S_1)$ converges:

5.1. Output Ψ_{opt} as the solution

5.2 Otherwise:

5.3 Go to 1.

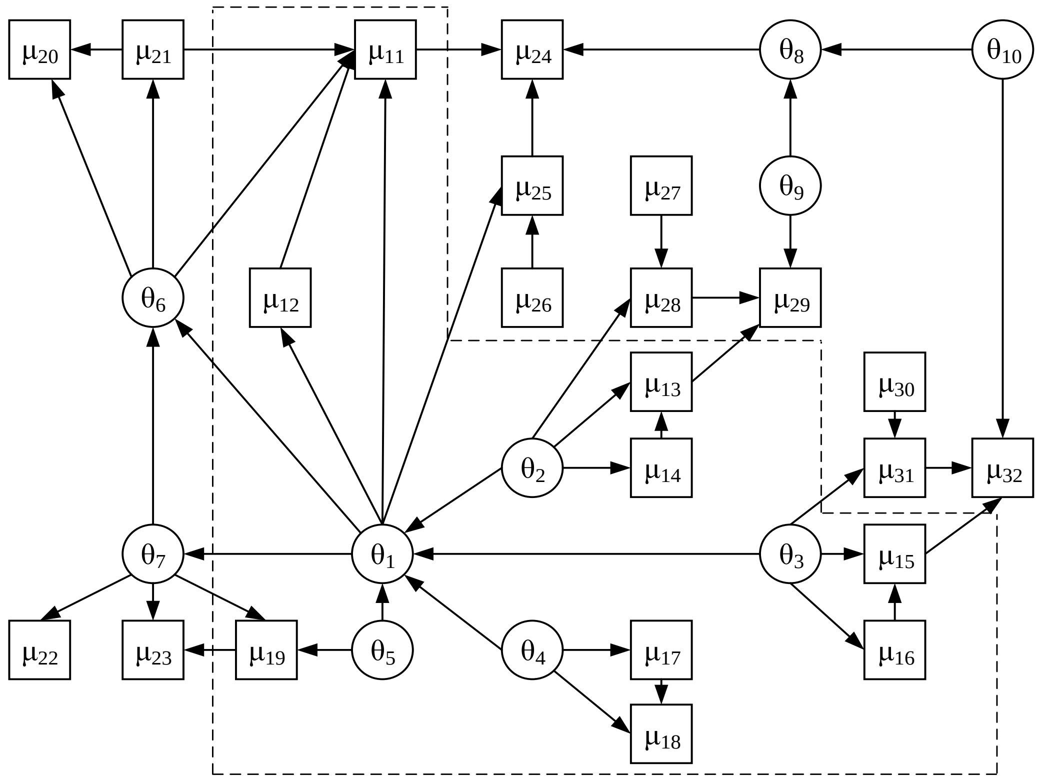


Fig. 6.2 BN of the Optical Instrument

In this experiment, it is assumed that system revenue is driven by system parameter θ_1. Hence, θ_1 is set as the single target parameter. The threshold for the system deployment rule, H_1, is set as 0.90. Cost data have also been synthetically generated in thousand-dollar units ($1,000). The revenue $B(\theta_1)$ has been set to 20,000 to provide a balance when making a selection tradeoff between different VAs. In the BN, there is a dependency between θ_1 and all other network nodes. That is, once an activity is executed on any node of these 32 nodes and an activity result is collected, the confidence $P(\theta_1=Pass)$ will change. The activity costs of the different activities and the failure costs of VAs are provided in Table 6.1. Specific values have been generated according to the types of activities defined in [52].

Table 6.1. Cost Table (Activities in Scenario 1 are marked in bold.)

Activity	**$\varphi_1(\theta_1)$**	**$\varphi_2(\theta_2)$**	**$\varphi_3(\theta_3)$**	**$\varphi_4(\theta_4)$**	**$\varphi_5(\theta_5)$**	$\varphi_6(\theta_6)$	$\varphi_7(\theta_7)$	$\varphi_8(\theta_8)$
Activity Cost	**8500**	**5200**	**2000**	**2300**	**1000**	8500	4200	6000
Failure Cost	-	-	-	-	-	-	-	-
Likelihood Ratio	**3:1**	**3.5:1**	**3.5:1**	**3.5:1**	**7.5:1**	3.5:1	8.5:1	3:1

Activity	$\varphi_9(\theta_9)$	$\varphi_{10}(\theta_{10})$	**μ_{11}**	**μ_{12}**	**μ_{13}**	**μ_{14}**	**μ_{15}**	**μ_{16}**
Activity Cost	2300	2000	**3300**	**500**	**3300**	**3400**	**300**	**300**
Failure Cost	-	-	**15000**	**8000**	**0**	**0**	**0**	**0**
Likelihood Ratio	10:1	8:1	-	-	-	-	-	-

Activity	**μ_{17}**	**μ_{18}**	**μ_{19}**	μ_{20}	μ_{21}	μ_{22}	μ_{23}	μ_{24}
Activity Cost	**300**	**500**	**100**	3300	3400	3400	300	2400
Failure Cost	**0**	**0**	**0**	12000	8000	2000	0	10000
Likelihood Ratio	-	-	-	-	-	-	-	-

Activity	μ_{25}	μ_{26}	μ_{27}	μ_{28}	μ_{29}	μ_{30}	μ_{31}	μ_{32}
Activity Cost	3500	2300	2400	3300	2300	400	300	400
Failure Cost	12000	5000	0	0	0	0	0	0
Likelihood Ratio	-	-	-	-	-	-	-	-

6.2.2 Experimental Method

With the provided problem and generated data, the whole experiment is realized with Python 3.6 and Bayes Net Toolbox for Matlab [187]. Two scenarios are conducted to study the performance of the proposed approach. In Scenario 1, the target network is outlined by the dashed line in Fig. 6.2. With 5 system parameters and 9 VAs in this BN, there are $2\times 2^5 \times 3^9 = 1,259,712$ total system states. The cost items of all activities are shown in bold in Table 6.1. To get some intuition about this scenario, the order-based backward induction method [183] is applied first to solve for the exact JVCS. When the backward induction is conducted to calculate the expected values of all states, the optimal activities of all system states are identified to constitute the exact JVCS, as shown in Fig. 6.3. With 24 verification paths (i.e., the number of terminal states 'Stop'), the exact

JVCS has 93 tree nodes (excluding terminal states 'Stop'). The depth of this JVCS is 13 nodes (i.e., 7 time events). The expected value of this JVCS is 7,788, while the total running time is 97,925 sec.

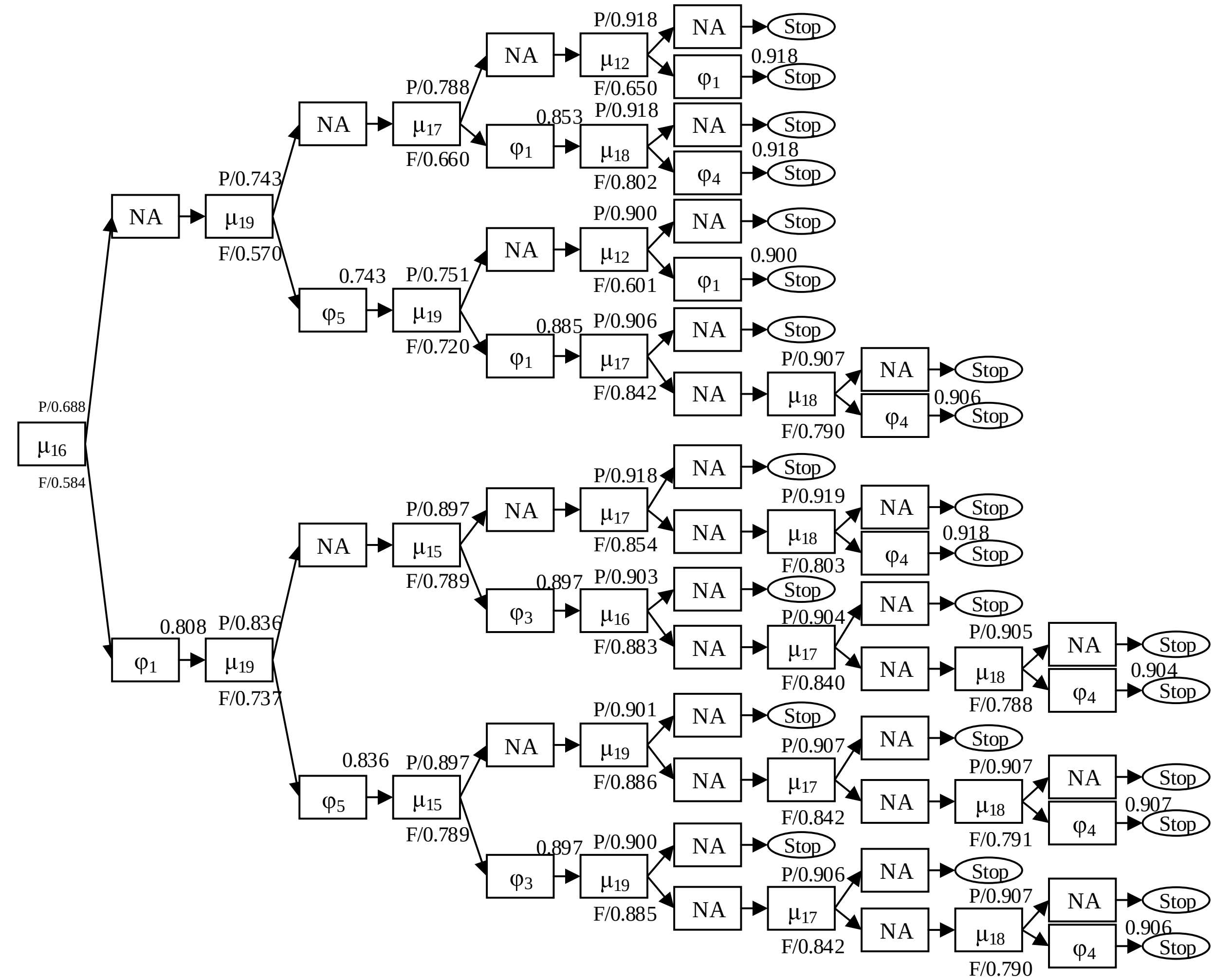


Fig. 6.3 The exact JVCS solution

In Scenario 2, the target network is the whole network in Fig. 6.2. With 10 system parameters and 22 VAs in this BN, there are $2\times 2^{10}\times 3^{22}=6.43\times 10^{13}$ total system states. As the number of total system states is too large, the order-based backward induction method is infeasible for this whole network, and the exact solution is unknown. However, because the small network is a subset of the whole one, the exact JVCS in Scenario 1 can be used as a reference. So, it needs more exploration about whether a better JVCS can be found in the large network, as shown in the discussion section.

In each scenario, the proposed approach is compared with two types of benchmark methods. First, the effects of different UCB rules, including UCBRB, UCT, and SP-MCTS (defined in Eq. 6.1, Eq. 2.1, and Eq. 2.2), are compared concerning the expected values of JVCS. As verification planning is such a particular problem that I cannot find a proper admissible heuristic function of all states, the AO* algorithm cannot be applied to solve for AND/OR trees. Algorithm 6.1 in Section 6.1.2 is the only feasible method to find a JVCS as far as I know. So, Algorithm 6.1 is combined with all these UCB rules to compare their effects. For each rule, their constant values are selected through their own sensitivity analyses. The constants are set as $D_7=20000$, $D_8=1$, and $D_7=50$. The two modifications in Section 6.1.2 are applied to all UCB rules. The tree search process is conducted by generating 5000 sample trees. The optimal state value for every 50 sample trees is recorded at the initial state to compare the performance of different UCB rules.

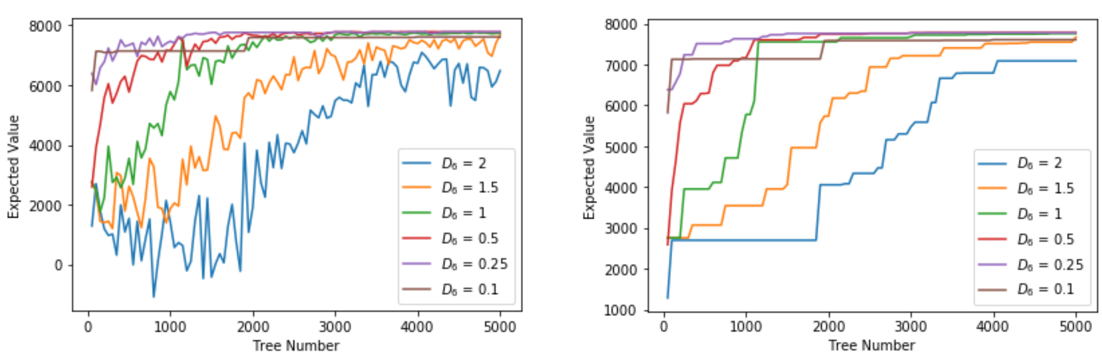
Second, the proposed methods are also compared with a Monte Carlo method. The UCBRB2 tree search method is also tested as an extension of the UCBRB1 one. An RFR model is trained every 3000 tree nodes, and another 3000 system states are sampled from the lookup table. Each RFR model consists of 100 decision trees, and the 5th percentile of the 100 outputs is used as predicted state values. The hyperparameter 'Bootstrap' is set as *False*, and the hyperparameter 'minimum number of samples required to be at a leaf node' is set as 1 to interpolate state samples. Other hyperparameters are set as their default values. Finally, a Monte Carlo method is designed to search for a JVCS randomly. As the number of tree nodes can be large if all activities are chosen randomly. This Monte Carlo method adds a constraint about the total number of tree nodes. That is, the total node number is less than $D_{10} = 50$.

Only two metrics are used to compare different UCB rules and benchmark methods in this experiment. First, the expected value of a JVCS is shown in Eq. 5.2. Second, the running time means wall-clock time, which is used to compare computational efficiency.

6.2.3 Experimental Results

6.2.3.1 Scenario 1

In the UCBRB rule (given by Eq. 6.1), the constant D_6 depends on the unknown distributions of the specific problem. So, it is necessary to determine the constant D_6 first for the proposed approach. For simplicity, a set of six possible D_6 constants [0.1, 0.25, 0.5, 1, 1.5, 2] is used to find the optimal one. For each constant value, the tree search algorithm in Algorithm 6.1 is conducted to test the performance, and their trends are shown in Fig. 6.4. It is found that when D_6 is 0.5, the expected value reaches the maximum 7780.78 among the 5000 sample trees. The expected values and runtimes of all possible constants are listed in Table 6.2.



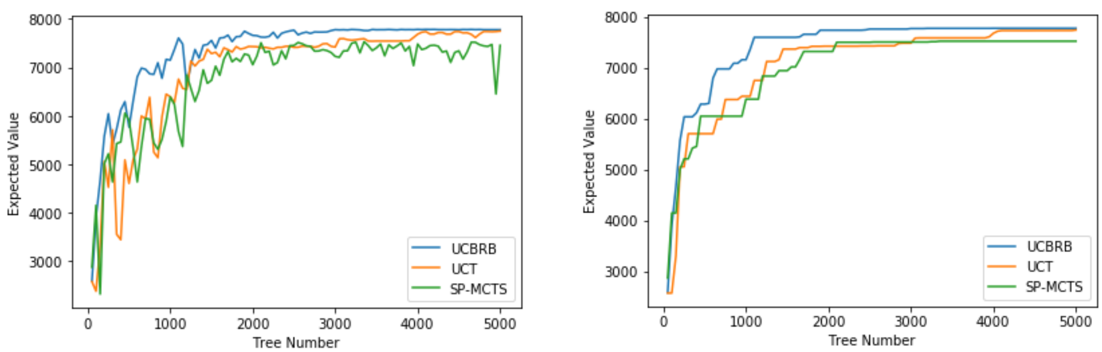
(a) Optimal expected value (b) Cumulated optimal expected value

Fig. 6.4. Constant selection for the UCBRB rule

Table 6.2 Expected value and runtime of all possible D_6 constants

	D_6=0.1	D_6=0.25	D_6=0.5	D_6=1	D_6=1.5	D_6=2
Expected Value	7608.14	7777.40	7780.78	7767.16	7670.18	7088.53
Runtime	979.50	4510.50	7452.68	15095.41	19899.54	22555.04

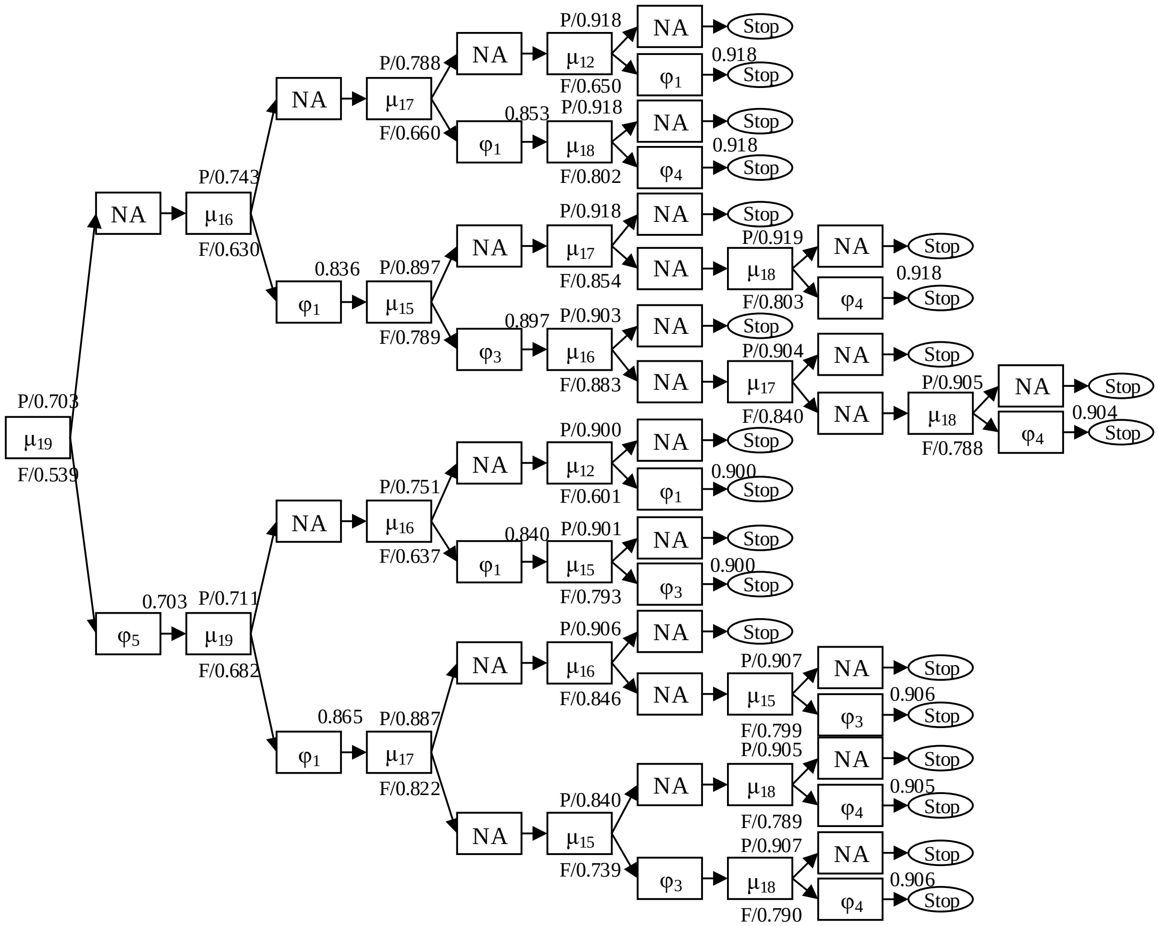
From Fig. 6.4 and Table 6.2, it can be observed that when D_6 decreases, the expected value converges faster. This is because more weights are allocated to exploiting existing strategies (i.e., the first item in Eq. 6.1) when D_6 decreases. The runtime also decreases become similar trees are exploited. However, if D_6 is too small, the tree search process may get stuck in local optimal spaces. For example, when D_6 = 0.1, the expected value becomes stable after the 1950th sample tree. Thus, D_6 should be large enough given fixed computation resources. In this case, as the tree number limit is 5000, D_6 is set as 0.50 for the UCBRB rule in this experiment. The corresponding JVCS is shown in Fig. 6.6 (a).



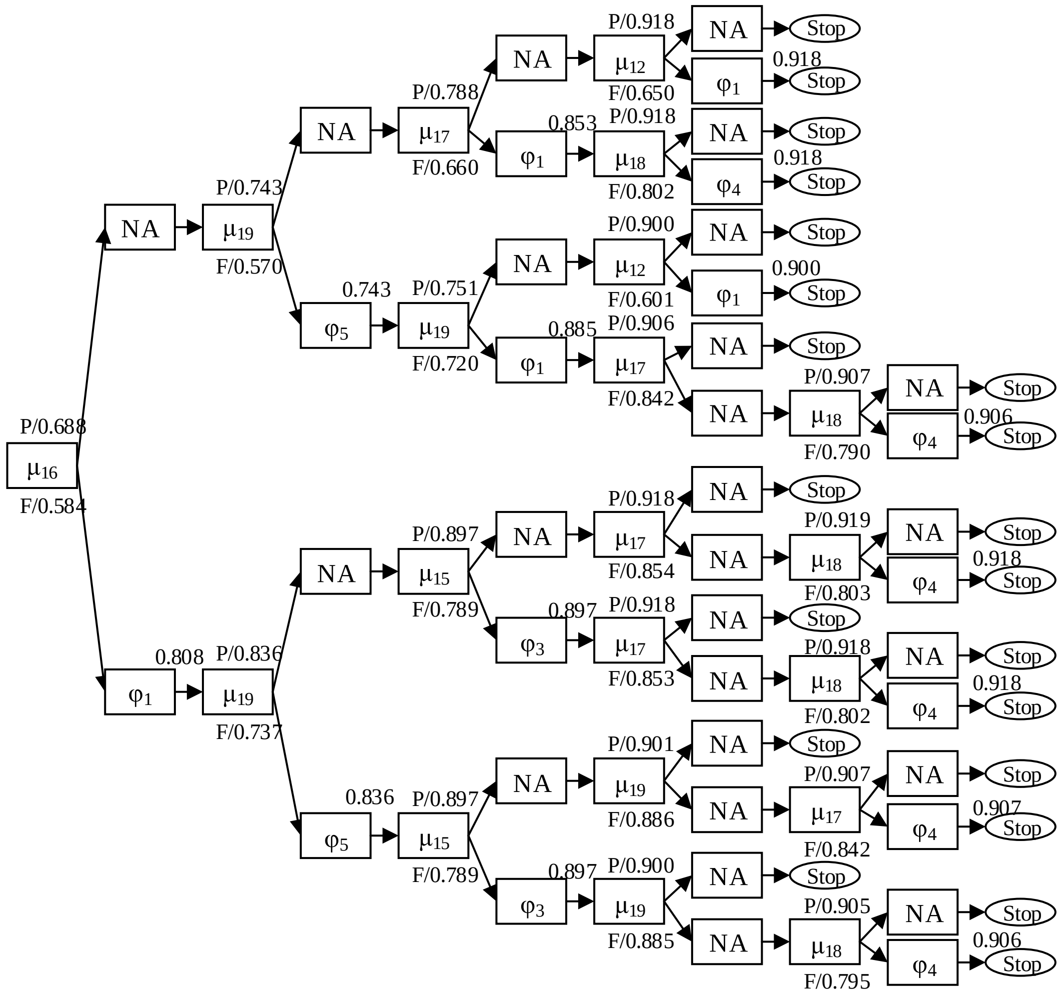
(a) Optimal expected value (b) Cumulated optimal expected value

Fig. 6.5. Performance of all UCB rules in Scenario 1

Next, three kinds of UCB rules are compared to solve this small network problem. With a similar approach for the D_6 value of the UCBRB, the optimal constants of UCT and SP-MCTS rules are found to be D_1 = 0.50, D_2 = 0.50, and D_3 = 10000. The performance of all UCB rules is shown in Fig. 6.5. It is found that even though both UCT and SP-MCTS rules can find their near-optimal strategies, the UCBRB rule outperforms them slightly after the 4000th sample tree. This is explained by the usage of maximum value as the first item in Eq. 6.1, which make the UCBRB more sensitive to the optimal activities. It is also found that SP-MCTS converges faster than UCT. It is attributed to the third item in Eq. 2.2 as it adds more weight to those nodes visited less frequently. Because there is no significant difference between the optimal expected values of all UCB rules, all UCB rules can be considered for the activity selection in this scenario.



(a) Solution of UCBRB1 tree search method

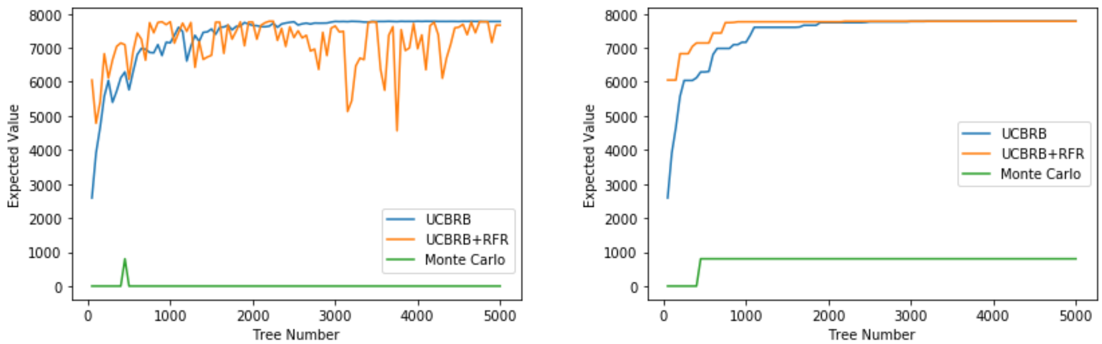


(b) Solution of UCBRB2 tree search method

Fig. 6.6. Near-optimal JVCSs in Scenario 1

Finally, the Monte Carlo method and the UCBRB2 tree search method are also conducted to search this small strategy space. The Monte Carlo method is conducted to generate 5000 sample trees. Then the UCBRB2 tree search method shares the same D_6 value as the UCBRB1 tree search method. Its JVCS is shown in Fig. 6.6 (b). Their performances are shown in Fig. 6.7 and summarized in Table 6.4. It is found that the Monte Carlo method cannot compete with the proposed two methods in terms of the expected values of JVCS, even though it costs the least runtime among all methods. The UCBRB1 tree search method can find the optimal strategy with the highest expected value, and its runtime is close to that of the Monte Carlo method.

Even though the UCBRB2 tree search method costs more runtime and its expected value is not the highest, it can solve the local optimality issue in the UCBRB1 tree search method. For simplicity, the distribution of all activities at the initial state S_1 is used to study the local optimality issue, as shown in Fig. 6.8. If the UCBRB1 tree search method is used, the first activity is always fixed as the one μ_{19} after the 500^{th} sample tree, as shown in Fig. 6.8 (b) (i.e., activity immovability). This activity immovability problem is attributed to the concept drift that enlarges the value gap of the first item in Eq. 6.1. So, the UCB of other activities cannot surpass that of μ_{19}, as shown in Fig. 6.9 (a). The UCBRB2 tree search method solved this local optimality issue by narrowing the value gap with the prior information, as shown in Fig. 6.9 (b). So, more sample trees are allocated to other activities, and it is possible to jump out of the local optimum strategy space (i.e., Fig. 6.8 (c)).



(a) Optimal expected value (b) Cumulated optimal expected value

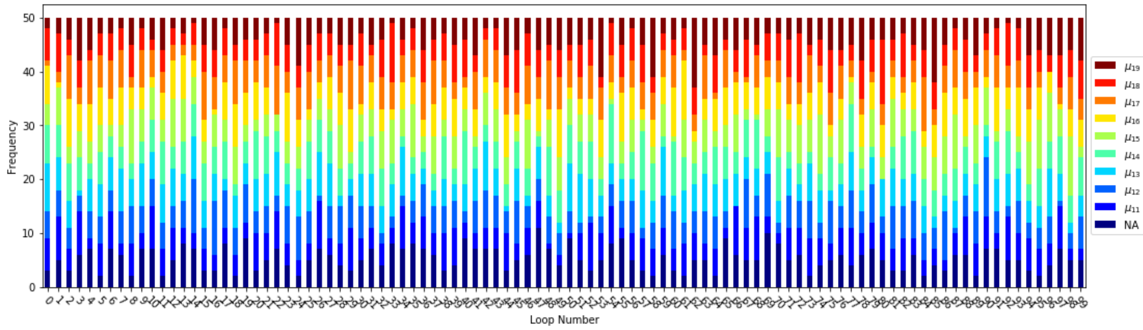
Fig. 6.7. Performance of all benchmark methods in Scenario 1

Table 6.3. Performance comparison of all UCB rules

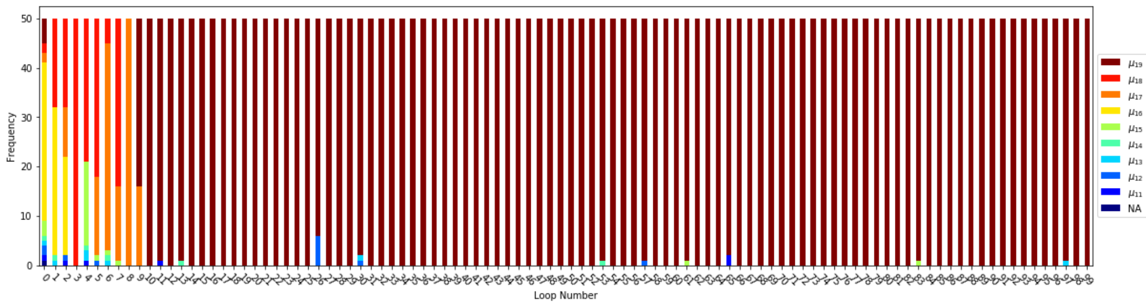
	Scenario 1		Scenario 2	
	Expected Value	Runtime (second)	Expected Value	Runtime (second)
UCBRB rule	7780.78	7452.68	8478.43	102109.83
UCT rule	7746.82	8915.13	7011.08	281919.13
SP-MCTS rule	7526.81	10183.85	6114.59	227776.50

Table 6.4. Performance comparison of all benchmark methods (including Monte Carlo and order-based backward induction)

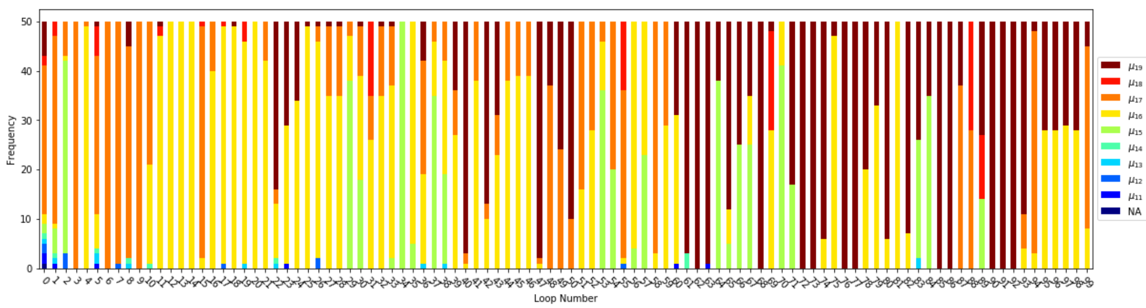
	Scenario 1		Scenario 2	
	Expected Value	Runtime (second)	Expected Value	Runtime (second)
UCBRB1	7780.78	7452.68	8478.43	102109.83
UCBRB2	7780.52	12120.16	8487.27	186804.14
Monte Carlo	801.45	6797.46	0	73234.74
Exact Solution	7788	97925	-	-



(a) Monte Carlo method

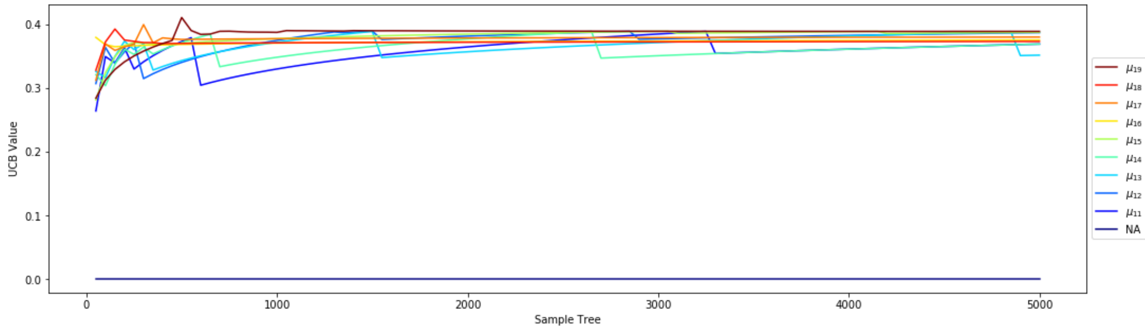


(b) UCBRB1 tree search method

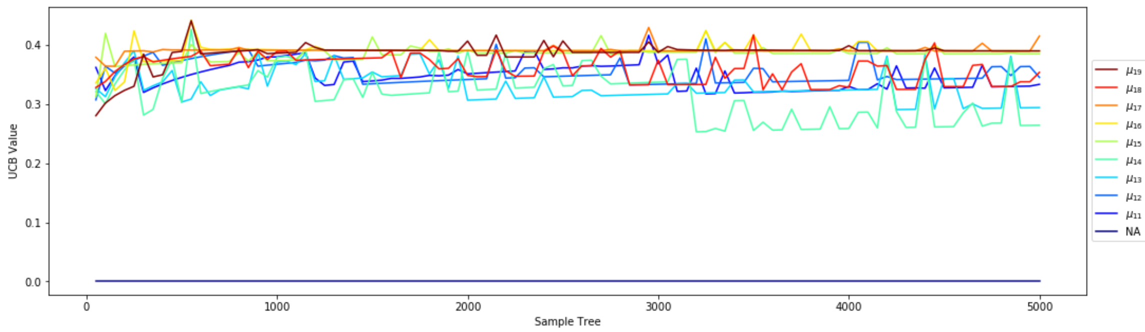


(c) UCBRB2 tree search method

Fig. 6.8 Distributions of all activities at the initial state S_1



(a) UCBRB1 tree search method

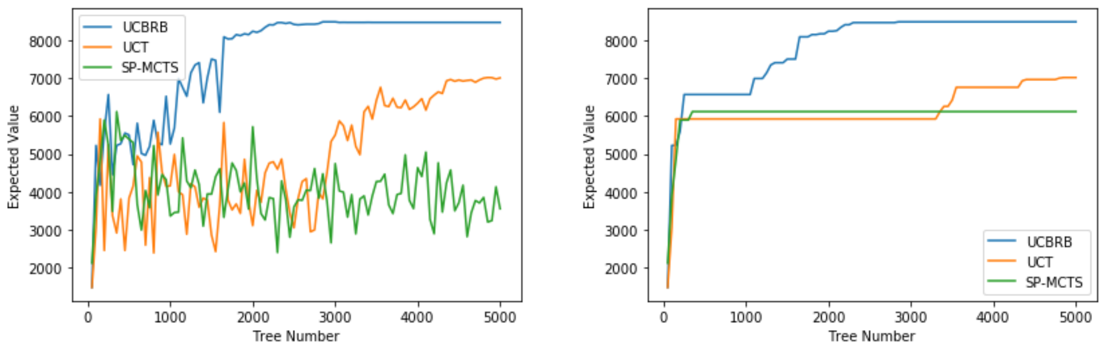


(b) UCBRB2 tree search method

Fig. 6.9 UCBs of all activities at the initial state S_1

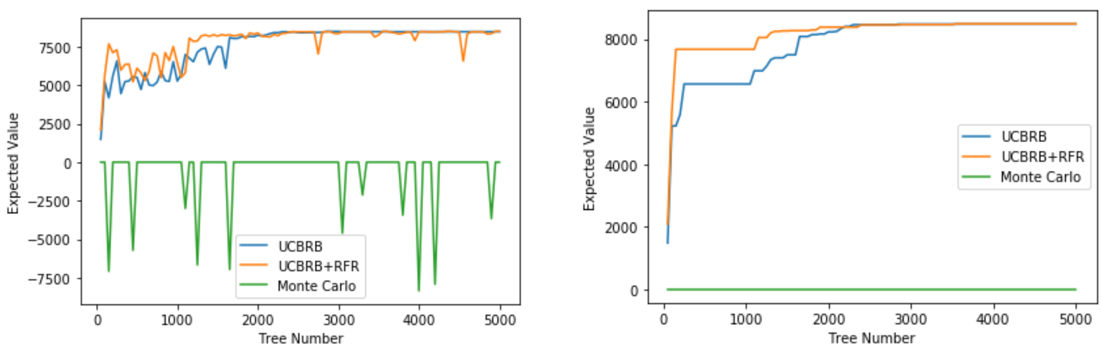
6.2.3.2 Scenario 2

In Scenario 2, there are 10 CAs and 22 VAs. As only some extra network nodes are added to the BN, all methods in Scenario 1 can be applied directly. For simplicity, all constants are assigned the same values as those in Scenario 1. All UCB rules are compared first in this scenario and show their results in Fig. 6.10 and Table 6.3. It is found that the JVCS generated by the UCBRB rule has the maximum expected value 8478.43, while the UCT and SP-MCTS only found their strategies with 7011.08 and 6114.59. In particular, the expected values of the UCT rule do not surpass 6000 until the 3350th sample tree. Our analysis is that this large network has more large-cost activities than the small network. So, the mean value is a biased estimate of a state value that makes the estimated UCB less accurate. However, the mean value still has some effect in the long term as it can be found that the expected value gradually reaches 7000. The SP-MCTS rule can find a JVCS with the value 6114.59 at the 300th sample tree. However, once this rule finds the JVCS with the 6114.59, it fails to provide a better JVCS as the value gradually decreases after the 300th sample tree. This is explained by the third item in Eq. 2.2 as it is not sensitive to the maximum value when the number of samples is large. Therefore, the UCBRB rule can find a better strategy than those in Scenario 2 because it uses the maximum function as a more accurate estimate of a state value.



(a) Optimal expected value (b) Cumulated optimal expected value

Fig. 6.10. Performance of all UCB rules in Scenario 2

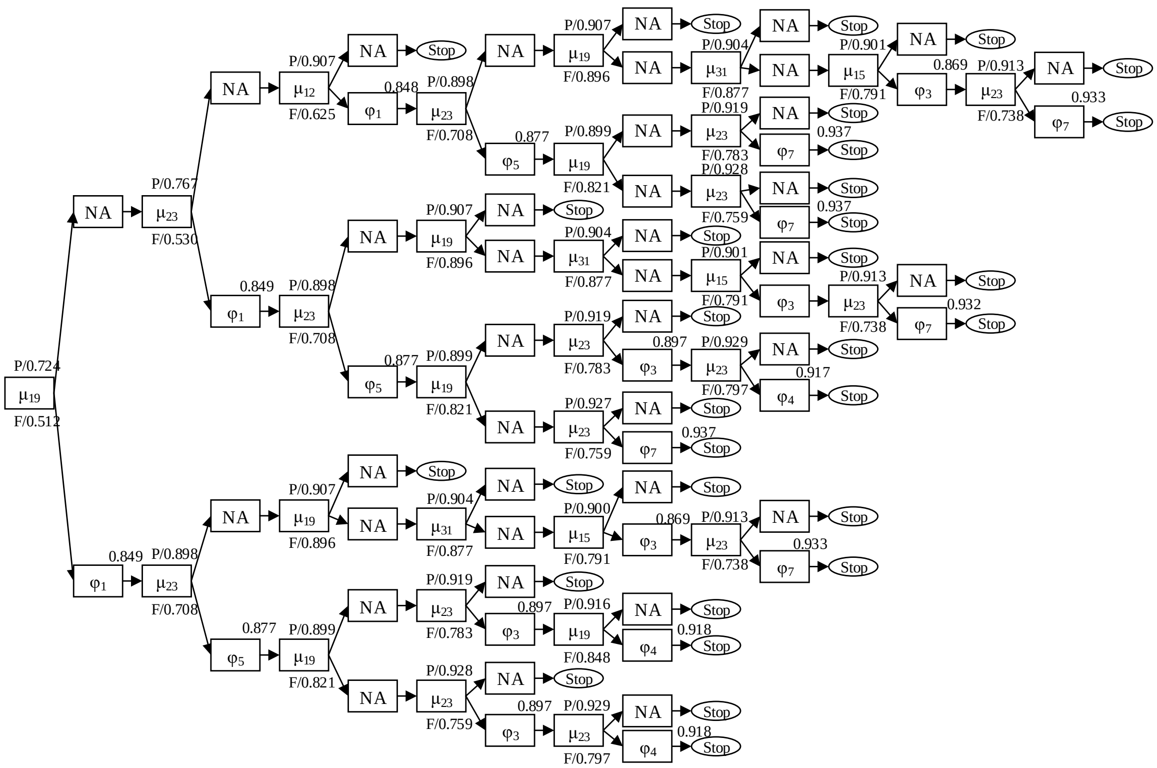


(a) Optimal expected value (b) Cumulated optimal expected value

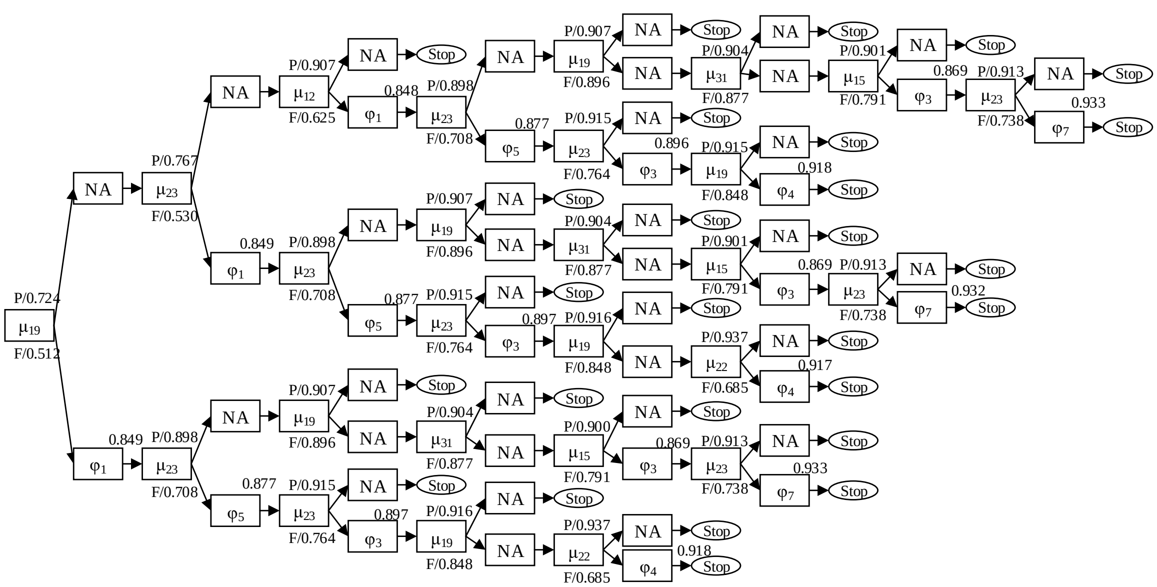
Fig. 6.11. Performance of all benchmark methods in Scenario 2

The performances of all three benchmark methods are shown in Fig. 6.11 and also summarized in Table 6.4. The JVCSs of UCBRB1 and UCBRB2 methods are shown in Fig. 6.12. It is found that the proposed two methods have found near-optimal JVCSs while the Monte Carlo method fails to provide a JVCS with a positive expected value. The reason is attributed to the large network that has more high-cost activities. So, it is harder to search for a strategy with positive expected values randomly. Instead, estimating the expected value with UCBs can avoid repeating the over-exploration of high-cost activities and yield better strategies with limited tree samples. However, these two proposed methods also cost much more time than the Monte Carlo method. It is attributed to the UCB calculation and the network size. The UCBRB2 tree search method converges faster than the UCBRB1 one in the first 2000 sample trees, even though there is no significant difference between their optimal expected values.

The activity immovability problem still occurs when the UCBRB1 tree search method is used. The optimal activity at the initial state is fixed as μ_{19}, as shown in Fig. 6.13 (a). It is caused by the same reason as in Scenario 1 that the value gap of the first item in Eq. 6.1 is too large. So, the UCBs of other activities can hardly surpass that of μ_{19}, as shown in Fig. 6.14 (a). However, the UCBRB2 tree search method can improve the UCBs significantly so that other activities are allocated with more exploration times, as shown in Fig. 6.13 (b). Thus, the tree search process can jump out of local optimum spaces and find a better JVCS. This explains why the JVCS of the UCBRB2 method has the largest expected value 8487.27.

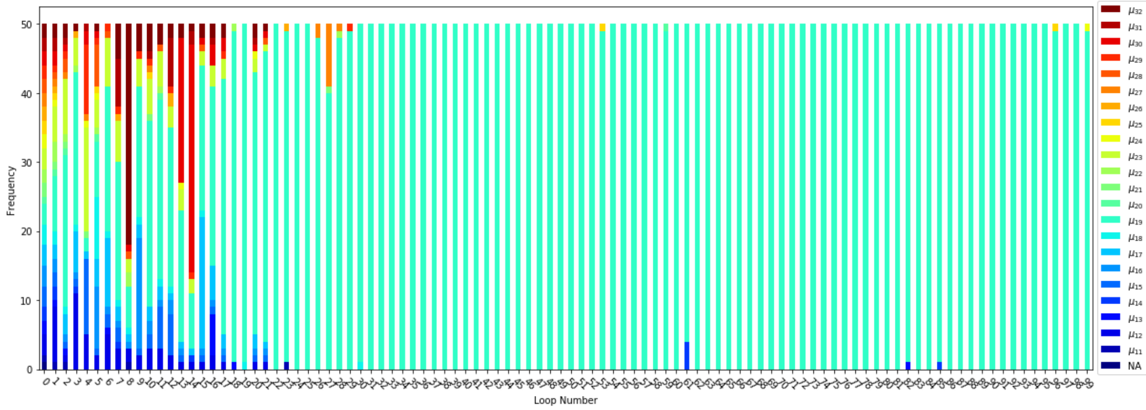


(a) Solution of UCBRB1 tree search method

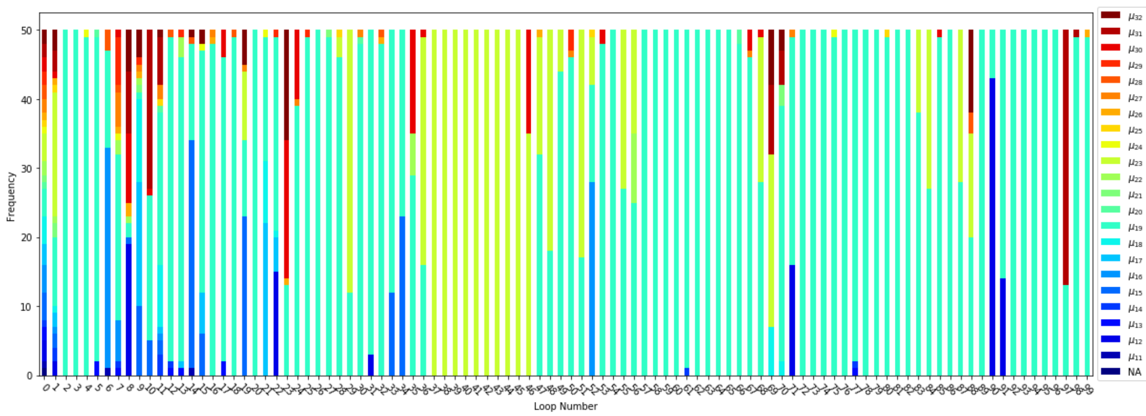


(b) Solution of UCBRB2 tree search method

Fig. 6.12. Near-optimal JVCSs in Scenario 2

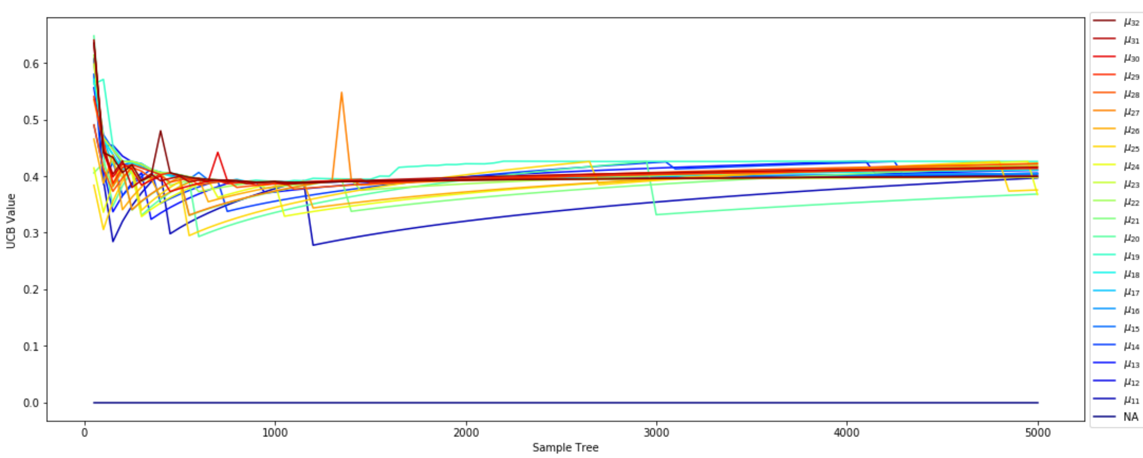


(a) UCBRB1 tree search method

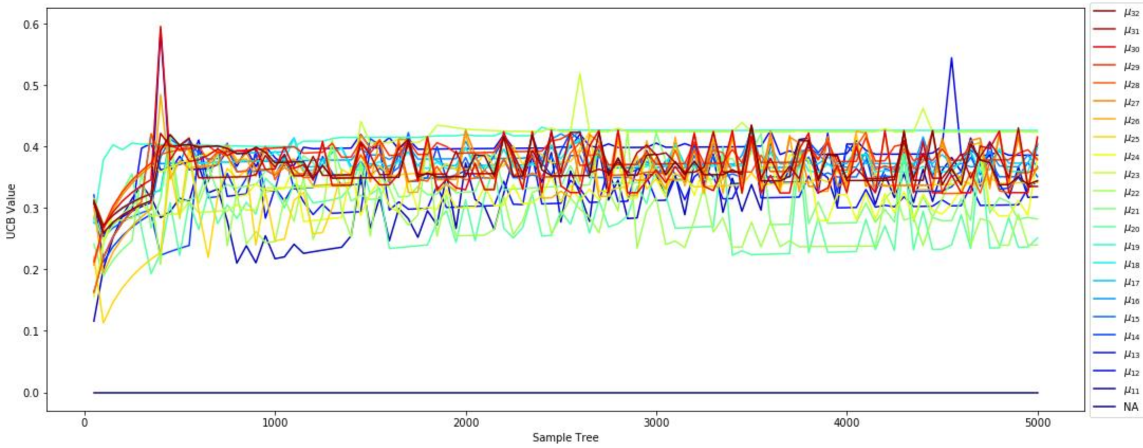


(b) UCBRB2 tree search method

Fig. 6.13 Distributions of all activities at the initial state S_1



(a) UCBRB1 tree search method



(b) UCBRB2 tree search method

Fig. 6.14 UCBs of all activities at the initial state S_1

6.3 Discussion

The two scenarios in this chapter are designed by expanding the activity set of the same system network. With the proposed approach, a better JVCS is found in Scenario 2, which is attributed to those extra 5 CAs and 13 VAs (i.e., φ_6 to φ_{10}, and μ_{20} to μ_{32}) in Scenario 2. However, this phenomenon does not always occur. I have verified the expected value of the exact JVCS (ref. Fig. 6.3) in the context of the large network and found its expected value is 76 because the confidence of the target node drops below 0.9 at most terminal states of verification paths in the large network, as shown in Fig. 6.15. Thus, if the costs of all extra activities are considerable, it is highly likely that the expected values of near-optimal JVCSs in Scenario 2 are lower than 7,788. Due to some low-cost activities, such as φ_{10} and μ_{23}, there are still some opportunities to improve the strategy performance in Scenario 2. Therefore, while finding JVCSs is the task of this research, the effects of all activity costs on JVCSs are not fully explored, which is left as future work.

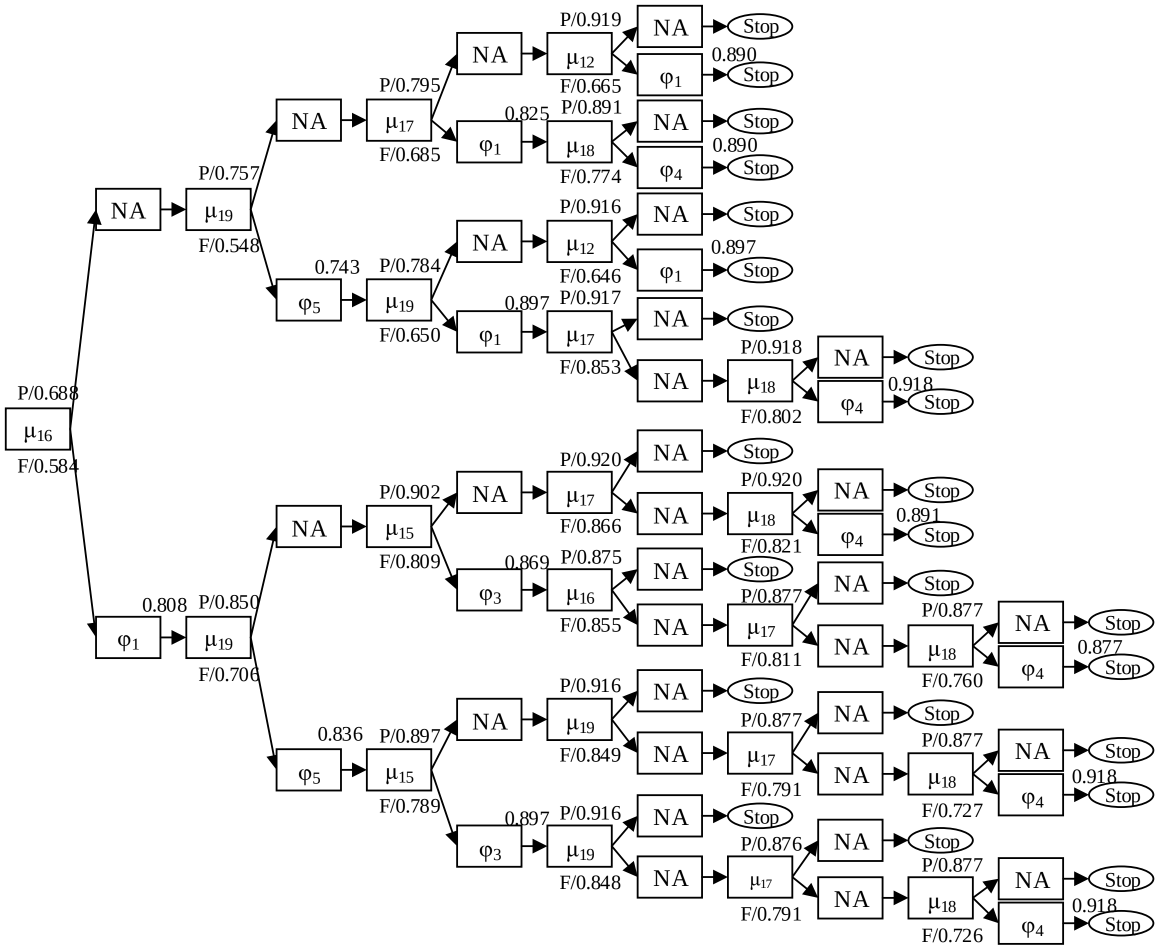
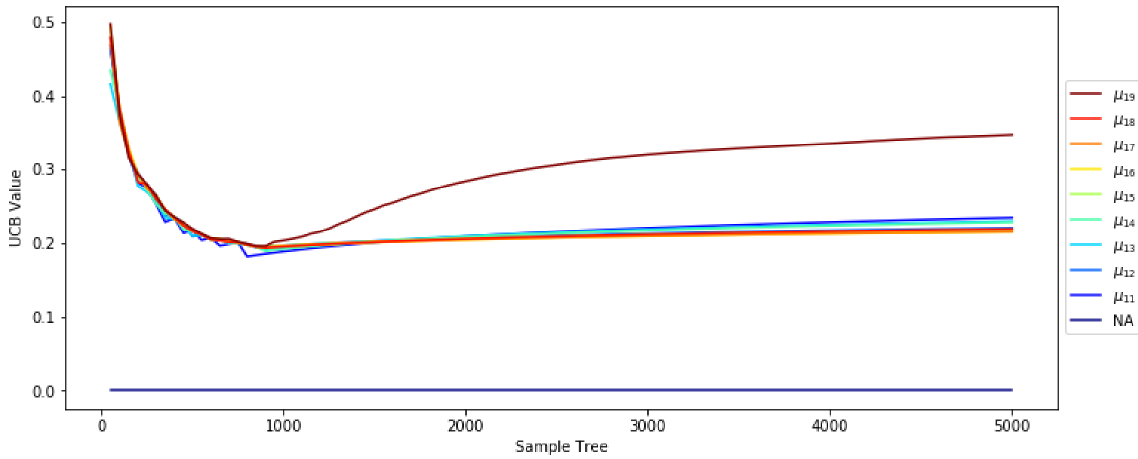
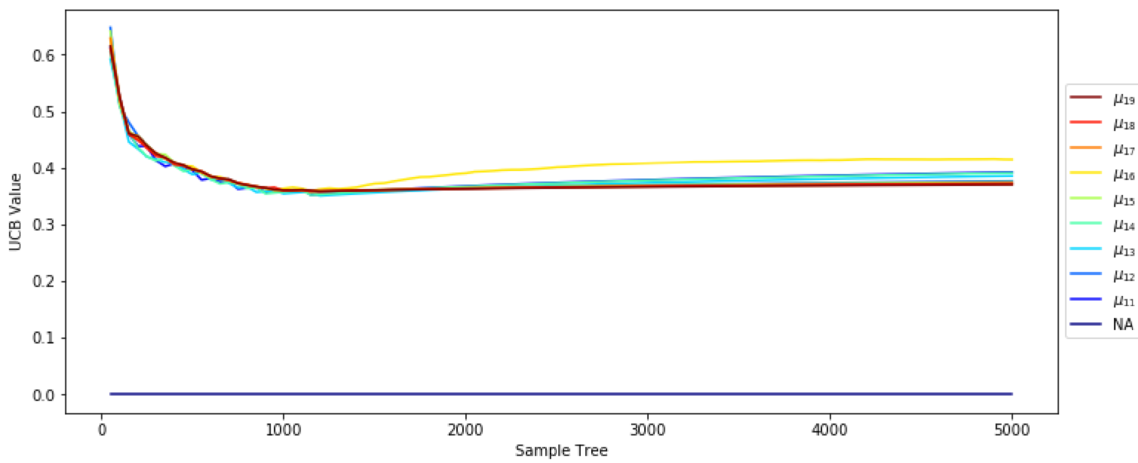


Fig. 6.15. Performance of the exact JVCS in Scenario 2

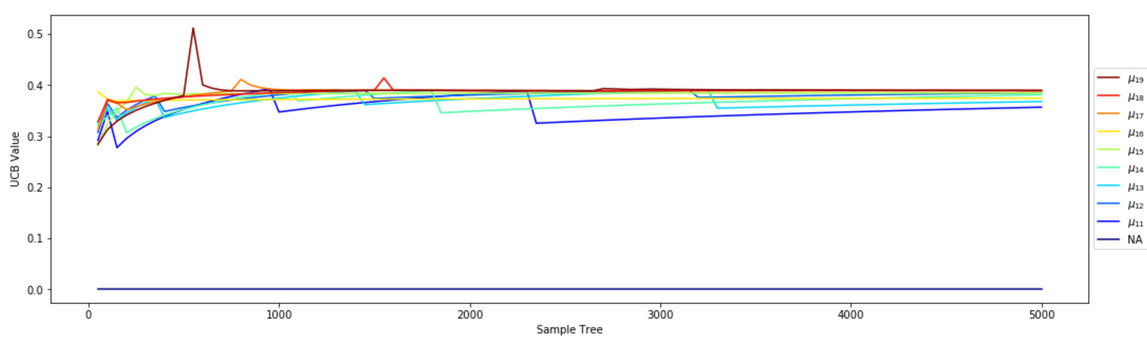
During the tree search process, it is also found that activity immovability always happens in this verification planning problem, no matter which UCB rule is used, as shown in Fig. 6.16 (a)(b). This problem cannot be solved by adjusting the constant D_6, as shown in Fig. 6.16 (c)(d). It is attributed to the concept drift of value distributions. In addition, it is assumed there are two causes of concept drift. The first one is the lacking of completeness in tree search processes. If it requires a large number of *Pass* activity results to reach the deployment threshold of the target parameter, the tree of a near-optimal JVCS has a large depth, and it cannot be found easily. Instead, only some partial information is collected, which results in concept drift. The second cause is the second item of UCB rules. When the parent visit count is 1 (i.e., n=0 in Eq. 6.1), the first item is *None* and the second item is 0. Then, the optimal activity is 'NA' because its activity cost is the minimum (i.e., 0). So, the optimal activity of an unvisited state is always 'NA', which makes the expected values of its parent tree nodes inaccurate. However, it is noticeable that activity immovability also has a positive effect on the search of JVCSs. After the tree search processes find a simple JVCS with a positive expected value, the activities of this JVCS are fixed as a core set, and its value can be improved by exploitation. Thus, it is unnecessary to eliminate this effect. Instead, it is suggested to improve the UCBs of other promising activities.



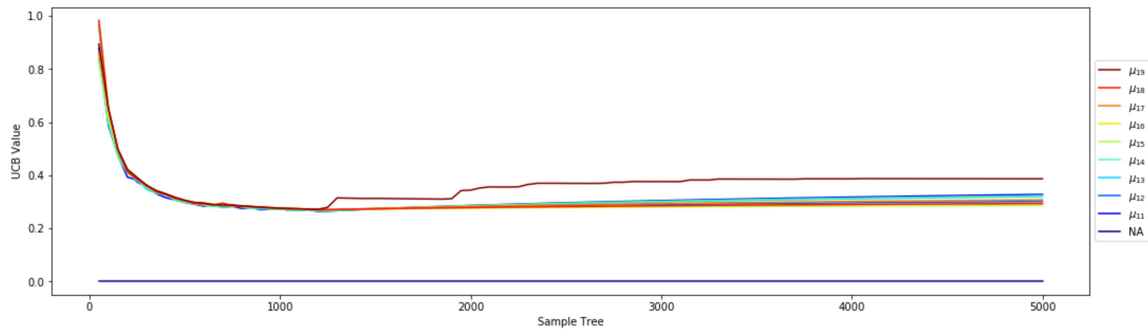
(a) UCBs of all activities when $D_1 = 0.5$, UCT rule



(b) UCBs of all activities when $D_2 = 0.5$, SP-MCTS rule



(c) UCBs of all activities when $D_0 = 0.25$, UCBRB rule



(d) UCBs of all activities when D_0 = 1.5, UCBRB rule

Fig. 6.16 Frequencies and UCBs of all activities every 50 sample trees

While the ensemble learning model is applied to extend the UCBRB1 tree search method, the optimal expected value is not improved significantly. It is mainly attributed to CAs in my view. Because CAs can eliminate errors and defects and reset the statuses of VAs, they can fundamentally improve the confidence of system parameters and reduce the impact of negative activity results. So, if a JVCS can eliminate system errors, the selection of activities at early time events does not matter too much on expected values in this experiment. From the aspect of practice, the UCBRB1 tree search method should be considered first, as it can provide a near-optimal JVCS with less runtime and storage space. However, if it is essential to explore other candidate activities, the UCBRB2 method is a better choice because it can handle local optimality issues and solve the exploration-exploitation dilemma much better than the UCBRB1 one.

CHAPTER 7: Conclusions

7.1 Contributions

This dissertation contributes to the field of system verification with respect to modeling of correction activities:

1) It has facilitated BN-based system diagnosis with PCI models. In the diagnosis applications via Bayes networks, the prior beliefs or the joint distribution of a BN are used as consistent distributions, which makes the knowledge elicitation and network reconstruction difficult. However, with the PCI model, a BN can be decomposed into independent cause factors. So, all belief changes can be realized by changing cause factors directly, which is also the foundation of the effects of CAs.
2) It has deepened the understanding of CAs in system verification. While CAs are commonly used to correct errors and defects in engineered systems, their impacts on beliefs of system components have never been explored before. It is the first time that uncertain evidence is used to calibrate the effects of activities on belief updates in system verification. Three kinds of CAs, including rework, repair, and redesign, have been instantiated with uncertain evidence as typical examples.
3) It has generalized VPs as a sequential process of Bayesian inference. While system verification involves VAs and CAs, they are treated as separate research targets, which poses an obstacle to the optimization of VSs. This obstacle is removed through the proposed framework as both VAs and CAs can be viewed as a process of collecting evidence. So, VP is a sequential process that updates prior beliefs with collected evidence.

This dissertation results in two contributions with respect to modeling joint verification-correction strategies:

1) It has formalized the extended paradigm of verification processes. All necessary process constraints and value factors have been presented in the EPVP. It is the first time that an EPVP is built mathematically as a flow of sets of activities with constraints. The process constraints specify the changes in activity sets. In addition, three necessary value factors are also presented to show the economic effects of system verification. Thus, this EPVP model defines all conditions of verification planning problems and these conditions are about the constraints, and costs of evidence collection.
2) It has proposed an order-based backward induction method to solve for exact JVCS in small strategy spaces. This method leverages the concepts *state* and *state value* in dynamic programming and solves the verification planning problem by updating the values of all system states. This method can significantly reduce the runtime of value update as it clarifies the relationship of all system states as an explicit diagram with an order function. Thus, as long as the total number of system states is small, the proposed method can be applied to solve for exact solutions.

This dissertation also makes three contributions with respect to applying this work at scale.

1) It has presented a search rule based on the upper confidence bound for repeatable bandits (UCBRB). To simplify the verification planning problem, a repeatable bandit model is presented as an extension of the traditional multi-armed bandit problem. Then the UCBRB rule is proposed with the regret theory. The UCBRB rule can be used to find near-optimal strategies of those decision-making problems whose strategies are repeatable.
2) It has designed a UCBRB1 tree search method to apply the UCBRB rule to verification planning. This method originates from traditional AO* algorithms that require accurate admissible heuristic functions, which is hard to define in verification planning. So, the heuristic functions are approximated with the proposed UCBRB rule. The characteristics of system verification, including confidence information, two types of activities, and a value-based model, are also considered to design this method for the search of JVCSs.
3) It has leveraged a tree-based ensemble learning model to handle the local optimality issue of the UCBRB1 tree search method. As verification planning is a sequential decision-making problem with dependency, the distributions of state values are not stationary during tree search processes (i.e., concept drift). These concept drifts usually result in the local optimality of JVCSs. RFR models are trained with collected samples of system states during the tree search process, and the lower confidence bound of RFR outputs is used to predict state values as prior information. These prediction values are used to narrow the gap of concept drifts between different system states and help jump out of the local optimum spaces.

Therefore, it can be concluded that the research works in this dissertation have fulfilled the three objectives in Chapter 1.

7.2 Limitations

Some limitations still exist in the research of this dissertation. These limitations result from the simplification of research assumptions that are acceptable for this dissertation:

1) The cause factors in the PCI models are assumed to be independent from each other, while their interactions may exist in reality.
2) Each type of CA is assumed to result in one type of uncertain evidence. Other possible combinations between CAs and uncertain evidence are not explored yet.
3) In the EPVP model, the prior beliefs are assumed to be consistent along with the VP.
4) Only one VA and one CA is allowed at each time event of a VP.
5) The constants of all UCB rules are selected from several possible values in the experiment. The selected value is used as a deterministic value at all system states in the two scenarios.
6) Some other parameters of the UCBRB1 tree search method are not optimized, including discount constant, penalty item, and total sample tree number.
7) When training RFR models, some ad hoc adjustments are made, such as removing terminal states as samples and adding the same number of system states from lookup tables, which are not fully optimized to improve the tree search performance.

7.3 Future Work

Although this dissertation has addressed the three objectives in Chapter 1, the research works have opened more questions for research and applications. That is, this dissertation should be understood as a building step toward a fully implementable tool for product lifecycle management. Therefore, future research is suggested in this direction from three aspects.

1) In terms of modeling CAs, there are three future opportunities. First, the CPTs of a BN are decomposed in this chapter with basic PCI models (i.e., generalized noisy OR and noisy AND). Other models may need to be explored to cope with cause factors of complex interactions in practice. Second, the proposed constructs need to be tested in decision and/or optimization methods that consider past information, current beliefs, and future alternative actions, as well as other development related attributes, such as costs. Third, it is recognized that the proposed constructs and modeling approach will likely require higher financial and temporal efforts than existing methods, particularly as the BN becomes larger and the frequency of correction increases. These aspects need to be studied to assess the conditions for adopting the presented constructs in practice.
2) As to the modeling of JVCS, there are three future opportunities. First, the JVCM embeds some major assumptions that were necessary for simplicity. These include a restriction to an unrestricted number of time events. It is suggested future work to explore the effects of refining the temporal aspects of our model. Second, the case study has been limited to one type of CA. Future work that illustrates other types of CAs is suggested to contribute to the generalization of the model. Third, the threshold values in the benchmark method seem to have a substantial effect on the potential value of the proposed JVCM. It is suggested to investigate the existence of optimal thresholds in the context of verification planning.
3) In terms of the large-scale strategy design, it is suggested that this dissertation opens three main future research directions. First, the horizon of verification processes is not fully explored as a penalty item is simply added to restrict the horizon. More mechanisms may be developed to explore the effect of the horizon on JVCSs. Second, this dissertation focuses on the tree search given a fixed system network. The selection of possible sub-networks could be explored to simplify tree search processes. The sub-networks may be generated by evaluating the impacts of activities on JVCSs. Third, only RFR models are studied in this dissertation. They are also trained with some intuitive features to approximate system values. Other machine learning methods need to be explored as benchmarks in the future.

REFERENCES

[1] Engel, A. (2010). *Verification, Validation, and Testing of Engineered Systems* (Vol. 73). John Wiley & Sons.

[2] TSMC. (2019). TSMC 2018 corporate social responsibility report: https://esg.tsmc.com/download/file/2018_tsmc_csr_report_published_May_2019/english/pdf/e_all.pdf.

[3] INCOSE, B. (2015). Systems engineering handbook: A guide for system life cycle processes and activities. *San Diego, US-CA: International Council on Systems Engineering*.

[4] Buede, D. M., & Miller, W. D. (2016). *The Engineering Design of Systems: Models and Methods*. John Wiley & Sons.

[5] Salado, A., & Kannan, H. (2018, October). Properties of the utility of verification. In *2018 IEEE International Systems Engineering Symposium (ISSE)* (pp. 1-8). IEEE.

[6] Folkestad, J. E., & Johnson, R. L. (2001). Resolving the conflict between design and manufacturing: integrated rapid prototyping and rapid tooling (IRPRT). *Journal of Industrial Technology*, *17*(4), 1-7.

[7] Hirshorn, S. R., Voss, L. D., & Bromley, L. K. (2017). NASA systems engineering handbook.

[8] Liu, B., Zhang, H., & Zhu, S. (2016, December). An incremental V-model process for automotive development. In *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)* (pp. 225-232). IEEE.

[9] Kreitz, T., & Thielecke, F. (2016). Virtual integration of an all-electric flight control system architecture and the aircraft electrical power distribution network (No. 2016-01-2034). *SAE Technical Paper*.

[10] Schönwald, J. R., Forsteneichner, C., Vahrenhorst, D., & Paetzold, K. (2019, July). Improvement of collaboration between testing and simulation departments on the example of a motorcycle manufacturer. In *Proceedings of the Design Society: International Conference on Engineering Design* (Vol. 1, No. 1, pp. 149-158). Cambridge University Press.

[11] Zimmerman, P. (2014). Model-based systems engineering (MBSE) in government: Leveraging the 'M' for DoD Acquisition. In *2014 INCOSE MBSE Workshop January* (Vol. 25).

[12] ISO/IEC/IEEE 15288. (2015). Systems and software engineering–System life cycle processes.

[13] Costa, R., & Sobek, D. K. (2003, January). Iteration in engineering design: inherent and unavoidable or product of choices made? In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference* (Vol. 37017, pp. 669-674).

[14] Mitchell, V. L., & Nault, B. R. (2007). Cooperative planning, uncertainty, and managerial control in concurrent design. *Management Science*, 53(3), 375-389.

[15] Taylor, T., & Ford, D. N. (2006). Tipping point failure and robustness in single development projects. *System Dynamics Review: The Journal of the System Dynamics Society*, 22(1), 51-71.

[16] System Verification. (15 May 2020). *SEBoK*, . Retrieved 17:03, July 16, 2020 from https://www.sebokwiki.org/w/index.php?title=System_Verification&oldid=59221.

[17] Salado, A., & Kannan, H. (2018). A mathematical model of verification strategies. *Systems Engineering*, *21*(6), 593-608.

[18] Salado, A. (2016). Applying tradespace exploration to verification engineering: From practice to theory and back again. In *Conference on Systems Engineering Research (CSER)*.

[19] Salado, A., Kannan, H., & Farkhondehmaal, F. (2019). Capturing the information dependencies of verification activities with Bayesian networks. In *Systems Engineering in Context* (pp. 581-591). Springer, Cham.

[20] Salado, A. (2015). Defining better test strategies with tradespace exploration techniques and Pareto fronts: Application in an industrial project. *Systems Engineering*, *18*(6), 639-658.

[21] Osborne, S. M. (1993). *Product Development Cycle Time Characterization through Modeling of Process Iteration* (Doctoral dissertation, Massachusetts Institute of Technology).

[22] Kennedy, B. M., Sobek, D. K., & Kennedy, M. N. (2014). Reducing rework by applying set-based practices early in the systems engineering process. *Systems Engineering*, *17*(3), 278-296.

[23] Love, P. E., Ackermann, F., Teo, P., & Morrison, J. (2015). From individual to collective learning: A conceptual learning framework for enacting rework prevention. *Journal of Construction Engineering and Management*, 141(11), 05015009.

[24] Li, H., & Love, P. E. (1998, July). Visualization of building interior design to reduce rework. In *Proceedings. 1998 IEEE Conference on Information Visualization. An International Conference on Computer Visualization and Graphics* (Cat. No. 98TB100246) (pp. 187-191). IEEE.

[25] Krishnan, V., Eppinger, S. D., & Whitney, D. E. (1997). A model-based framework to overlap product development activities. *Management Science*, *43*(4), 437-451.

[26] Roemer, T. A., & Ahmadi, R. (2004). Concurrent crashing and overlapping in product development. *Operations Research*, *52*(4), 606-622.

[27] Thomke, S., & Bell, D. E. (2001). Sequential testing in product development. *Management Science*, *47*(2), 308-323.

[28] Tahera, K., Earl, C., & Eckert, C. (2017). A method for improving overlapping of testing and design. *IEEE Transactions on Engineering Management*, *64*(2), 179-192.

[29] Yassine, A., Joglekar, N., Braha, D., Eppinger, S., & Whitney, D. (2003). Information hiding in product development: the design churn effect. *Research in Engineering Design*, *14*(3), 145-161.

[30] Love, P. E., Mandal, P., & Li, H. (1999). Determining the causal structure of rework influences in construction. *Construction Management & Economics*, 17(4), 505-517.

[31] Mitchell, V. L., & Nault, B. R. (2007). Cooperative planning, uncertainty, and managerial control in concurrent design. *Management Science*, 53(3), 375-389.

[32] Denardo, E. V., & Lee, T. Y. (1996). Managing uncertainty in a serial production line. *Operations Research*, 44(2), 382-392.

[33] Browning, T. R., & Eppinger, S. D. (2002). Modeling impacts of process architecture on cost and schedule risk in product development. *IEEE Transactions on Engineering Management*, 49(4), 428-442.

[34] Liao, G. L. (2015). Production and maintenance policies for an EPQ model with perfect repair, rework, free-repair warranty, and preventive maintenance. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 46(8), 1129-1139.

[35] Taleizadeh, A. A., Jalali-Naini, S. G., Wee, H. M., & Kuo, T. C. (2013). An imperfect multi-product production system with rework. *Scientia Iranica*, 20(3), 811-823.

[36] Fujimoto, T., & Clark, K. (1991). Product development performance. *Strategy, Organisation and Management in the World Auto Industry*.

[37] Fricke, E., Gebhard, B., Negele, H., & Igenbergs, E. (2000). Coping with changes: causes, findings, and strategies. *Systems Engineering*, *3*(4), 169-179.

[38] Forsberg, K., Mooz, H., & Cotterman, H. (2005). *Visualizing project management: models and frameworks for mastering complex systems*. John Wiley & Sons.

[39] Jarratt, T. A. W., Eckert, C. M., Caldwell, N. H., & Clarkson, P. J. (2011). Engineering change: an overview and perspective on the literature. *Research in Engineering Design*, *22*(2), 103-124.

[40] Barad, M., & Engel, A. (2006). Optimizing VVT strategies: a decomposition approach. *Journal of the Operational Research Society*, *57*(8), 965-974.

[41] Kulkarni, A. U. (2018). *Incentive Mechanism Design for Systems with Many Agents: A Multiscale Decision Theory Approach* (Doctoral dissertation, Virginia Tech).

[42] Xu, P., & Salado, A. (2019, July). A concept for set-based design of verification strategies. In *INCOSE International Symposium* (Vol. 29, No. 1, pp. 356-370).

[43] Boehm, B., & Basili, V. R. (2005). Software defect reduction top 10 list. *Foundations of empirical software engineering: the legacy of Victor R. Basili*, 426(37), 426-431.

[44] Oberkampf, W. L., Trucano, T. G., & Hirsch, C. (2004). Verification, validation, and predictive capability in computational engineering and physics. *Appl. Mech. Rev*., 57(5), 345-384.

[45] Camburn, B., et al. (2017). Design prototyping methods: state of the art in strategies, techniques, and guidelines. *Design Science*, 3.

[46] Liu, J., & Williams, R. K. (2020). Data-driven models with expert influence: A hybrid approach to spatiotemporal process estimation. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 2467-2473). IEEE.

[47] Ha, A. Y., & Porteus, E. L. (1995). Optimal timing of reviews in concurrent design for manufacturability. *Management Science*, 41(9), 1431-1447.

[48] Loch, C. H., & Terwiesch, C. (1998). Communication and uncertainty in concurrent engineering. *Management Science*, 44(8), 1032-1048.

[49] Engel, A., & Barad, M. (2003). A methodology for modeling VVT risks and costs. *Systems Engineering*, 6(3), 135-151.

[50] Engel, A., & Last, M. (2007). Modeling software testing costs and risks using fuzzy logic paradigm. *Journal of Systems and Software*, 80(6), 817-835.

[51] Whiteside, M., & Reimann, M. (1994). A methodology for quantitative assessment of the impact of alternate business practices on systems life-cycle costs. *Int J Flexible Automat Integrated Manuf*, 2(3), 231-343.

[52] Salado, A., & Kannan, H. (2019). Elemental patterns of verification strategies. *Systems Engineering*, *22*(5), 370-388.

[53] Blanchard, B. S., Fabrycky, W. J., & Fabrycky, W. J. (1990). *Systems Engineering and Analysis* (Vol. 4). Englewood Cliffs, NJ: Prentice Hall.

[54] Shergadwala, M., Bilionis, I., Kannan, K. N., & Panchal, J. H. (2018). Quantifying the impact of domain knowledge and problem framing on sequential decisions in engineering design. *Journal of Mechanical Design*, 140(10).

[55] Chaudhari, A. M., Bilionis, I., & Panchal, J. H. (2020). Descriptive models of sequential decisions in engineering design: An experimental study. *Journal of Mechanical Design*, 142(8), 081704.

[56] Ward, A., Liker, J. K., Cristiano, J. J., & Sobek, D. K. (1995). The second Toyota paradox: How delaying decisions can make better cars faster. *Sloan Management Review*, 36, 43-43.

[57] Ward, A., Durward Sobek, I. I., John, J. C., & Jeffrey, K. L. (1995). Toyota, concurrent engineering, and set-based design. *Ch, 8*, 192-216.

[58] Liu, J., & Williams, R. K. (2019). Submodular optimization for coupled task allocation and intermittent deployment problems. *IEEE Robotics and Automation Letters*, 4(4), 3169-3176.

[59] Liu, J., Zhou, L., Tokekar, P., & Williams, R. K. (2021). Distributed resilient submodular action selection in adversarial environments. *IEEE Robotics and Automation Letters*, 6(3), 5832-5839.

[60] Liu, J., & Williams, R. K. (2020, May). Monitoring over the long term: Intermittent deployment and sensing strategies for multi-robot teams. In *2020 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 7733-7739). IEEE.

[61] Xu, P., Salado, A., & Deng, X. (2022). A parallel tempering approach for efficient exploration of the verification tradespace in engineered systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems.*

[62] Nagano, S. (2008). Space systems verification program and management process: Importance of implementing a distributed - verification program with standardized modular - management process. *Systems Engineering*, 11(1), 27-38.

[63] Kulkarni, A. U., Wernz, C., & Salado, A. (2021). Coordination of verification activities with incentives: a two-firm model. *Research in Engineering Design*, 32(1), 31-47.

[64] Kulkarni, A. U. (2018). *Incentive Mechanism Design for Systems with Many Agents: A Multiscale Decision Theory Approach* (Doctoral dissertation, Virginia Tech).

[65] Xu, P., Salado, A., & Xie, G. (2020, October). A reinforcement learning approach to design verification strategies of engineered systems. *In 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (pp. 3543-3550). IEEE.

[66] Xu, P., & Salado, A. (2021). Modeling correction activities in the context of verification strategies. *Systems Engineering*.

[67] Thunnissen, D. P. (2003, June). Uncertainty classification for the design and development of complex systems. In *3rd Annual Predictive Methods Conference* (pp. 1-16).

[68] Stockstrom, C., & Herstatt, C. (2008). Planning and uncertainty in new product development. *R & d Management*, 38(5), 480-490.

[69] Wang, K., Wang, Y., Yue, X., & Cai, W. (2021). Multiphysics modeling and uncertainty quantification of tribocorrosion in aluminum alloys. *Corrosion Science*, 178, 109095.

[70] Wang, Y., Wang, K., Cai, W., & Yue, X. (2022). NP-ODE: Neural process aided ordinary differential equations for uncertainty quantification of finite element analysis. *IISE Transactions*, 54(3), 211-226.
[71] Gralla, E., & Szajnfarber, Z. (2016). Characterizing representational uncertainty in system design and operations. *Systems Engineering*, 19(6), 535-548.
[72] Sentz, K., & Ferson, S. (2002). *Combination of Evidence in Dempster-Shafer Theory* (Vol. 4015). Albuquerque: Sandia National Laboratories.
[73] Shafer, G. (1976). *A Mathematical Theory of Evidence*. Princeton University Press.
[74] Zadeh, L. A. (1965). *Fuzzy sets*. Information and Control, 8(3), 338-353.
[75] Dubois, D., & Prade, H. (1998). Possibility theory: qualitative and quantitative aspects. In *Quantified Representation of Uncertainty and Imprecision* (pp. 169-226). Springer, Dordrecht.
[76] Dubois, D. (2006). Possibility theory and statistical reasoning. *Computational Statistics & Data Analysis*, 51(1), 47-69.
[77] Destercke, S., Dubois, D., & Chojnacki, E. (2008). Unifying practical uncertainty representations–I: Generalized p-boxes. *International Journal of Approximate Reasoning*, 49(3), 649-663.
[78] Hester, P. (2012). Epistemic uncertainty analysis: an approach using expert judgment and evidential credibility. *Journal of Quality and Reliability Engineering*, 2012.
[79] Martinez, F. A., Sallak, M., & Schön, W. (2015). An efficient method for reliability analysis of systems under epistemic uncertainty using belief function theory. *IEEE Transactions on Reliability*, 64(3), 893-909.
[80] Liang, D., Lau, N., Baker, S. A., & Antin, J. F. (2020). Examining senior drivers' attitudes toward advanced driver assistance systems after naturalistic exposure. *Innovation in Aging*, 4(3), igaa017.
[81] Cox, L., Delugach, H. S., & Skipper, D. (2001, July). Dependency analysis using conceptual graphs. In *Proceedings of the 9th International Conference on Conceptual Structures*, ICCS (pp. 117-130).
[82] Sharma, A., Grover, P. S., & Kumar, R. (2009). Dependency analysis for component-based software systems. *ACM SIGSOFT Software Engineering Notes*, 34(4), 1-6.
[83] Pearl, J. (2014). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Elsevier.
[84] Smets, P., & Kennes, R. (1994). The transferable belief model. *Artificial Intelligence*, 66(2), 191-234.
[85] Sallak, M., Schön, W., & Aguirre, F. (2010). Transferable belief model for reliability analysis of systems with data uncertainties and failure dependencies. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, 224(4), 266-278.
[86] Stylios, C. D., & Groumpos, P. P. (2004). Modeling complex systems using fuzzy cognitive maps. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 34(1), 155-162.
[87] Yang, J. B., Liu, J., Wang, J., Sii, H. S., & Wang, H. W. (2006). Belief rule-base inference methodology using the evidential reasoning approach-RIMER. *IEEE Transactions on systems, Man, and Cybernetics-part A: Systems and Humans*, 36(2), 266-285.

[88] Grover, J. (2013). A literature review of Bayes' theorem and Bayesian belief networks (BBN). *Strategic Economic Decision-making*, 11-27.

[89] Cai, B., Huang, L., & Xie, M. (2017). Bayesian networks in fault diagnosis. *IEEE Transactions on Industrial Informatics*, 13(5), 2227-2240.

[90] Neapolitan, R. E. (2004). *Learning Bayesian Networks* (Vol. 38). Upper Saddle River, NJ: Pearson Prentice Hall.

[91] Isermann, R. (2005). *Fault-diagnosis Systems: an Introduction from Fault Detection to Fault Tolerance*. Springer Science & Business Media.

[92] Dagum, P., Galper, A., & Horvitz, E. (1992, January). Dynamic network models for forecasting. In *Uncertainty in Artificial Intelligence* (pp. 41-48). Morgan Kaufmann.

[93] Dagum, P., Galper, A., Horvitz, E., & Seiver, A. (1995). Uncertain reasoning and forecasting. *International Journal of Forecasting*, 11(1), 73-87.

[94] Koller, D., & Pfeffer, A. (2013). Object-oriented Bayesian networks. *arXiv preprint arXiv*:1302.1554.

[95] Ivanovska, M., Jøsang, A., Kaplan, L., & Sambo, F. (2015, July). Subjective networks: Perspectives and challenges. In *International Workshop on Graph Structures for Knowledge Representation and Reasoning* (pp. 107-124). Springer, Cham.

[96] Xu, P., Cho, J. H., & Salado, A. (2020). Expert opinion fusion framework using subjective logic for fault diagnosis. *IEEE Transactions on Cybernetics*.

[97] Box, G. E., & Tiao, G. C. (2011). *Bayesian inference in statistical analysis* (Vol. 40). John Wiley & Sons.

[98] Liang, D., Antin, J. F., Lau, N. K., Stulce, K. E., & through Disruption, S. (2021). Examining seniors' adaptation to mixed function automated vehicles: analysis of naturalistic driving data (No. 03-040). *Safety through Disruption (Safe-D) University Transportation Center (UTC)*.

[99] Mrad, A. B., Delcroix, V., Piechowiak, S., Leicester, P., & Abid, M. (2015). An explication of uncertain evidence in Bayesian networks: likelihood evidence and probabilistic evidence. *Applied Intelligence*, 43(4), 802-824.

[100] Valtorta, M., Kim, Y. G., & Vomlel, J. (2002). Soft evidential update for probabilistic multiagent systems. *International Journal of Approximate Reasoning*, 29(1), 71-106.

[101] Mrad, A. B., Delcroix, V., Piechowiak, S., Maalej, M. A., & Abid, M. (2013, April). Understanding soft evidence as probabilistic evidence: Illustration with several use cases. In *2013 5th International Conference on Modeling, Simulation and Applied Optimization (ICMSAO)* (pp. 1-6). IEEE.

[102] Gowadia, V., Farkas, C., & Valtorta, M. (2003, May). Agent based intrusion detection with soft evidence. In *Proceedings of the 14th IRMA International Conference* (pp. 345-350).

[103] Subramanya, A., & Bilmes, J. (2007, April). Virtual evidence for training speech recognizers using partially labeled data. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers* (pp. 165-168).

[104] Corradi, F., Pinchi, V., Barsanti, I., & Garatti, S. (2013). Probabilistic classification of age by third molar development: the use of soft evidence. *Journal of forensic sciences*, 58(1), 51-59.

[105] Barto, A. G., Sutton, R. S., & Watkins, C. J. C. H. (1989). *Learning and Sequential Decision Making*. Amherst, MA: University of Massachusetts.

[106] Amir, Eyal (2014). Reasoning and decision making. In Frankish, Keith; Ramsey, William M. (eds.). *The Cambridge handbook of artificial intelligence*.

[107] Koller, D., & Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

[108] Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4, 237-285.

[109] Amir, E. (2014). Reasoning and decision making. *The Cambridge Handbook of Artificial Intelligence*. Cambridge University Press. 191-212.

[110] Mitra, S. K., Lee, T. W., & Goldbaum, M. (2005). A Bayesian network based sequential inference for diagnosis of diseases from retinal images. *Pattern Recognition Letters*, 26(4), 459-470.

[111] Plajner, M., & Vomlel, J. (2015). Bayesian network models for adaptive testing. *arXiv preprint arXiv*:1511.08488.

[112] Khanafer, R. M., Solana, B., Triola, J., Barco, R., Moltsen, L., Altman, Z., & Lazaro, P. (2008). Automated diagnosis for UMTS networks using Bayesian network approach. *IEEE Transactions on Vehicular Technology*, 57(4), 2451-2461.

[113] Nilsson, N. J. (1968). Searching problem-solving and game-playing trees for minimal cost solutions. In *Proceedings IFIP Congress*, 1968 (Vol. 2, pp. 1556-1562).

[114] Chang, C. L., & Lee, R. C. T. (1972). Problem-solving methods in artificial intelligence (Nils J. Nilsson). *SIAM Review, SIREV.*

[115] Nilsson, N. J. (1982). *Principles of Artificial Intelligence*. Springer Science & Business Media.

[116] Martelli, A., & Montanari, U. (1978). Optimizing decision trees through heuristically guided search. *Communications of the ACM*, 21(12), 1025-1039.

[117] Hansen, E. A., & Zilberstein, S. (2001). LAO∗: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1-2), 35-62.

[118] Vomlelová, M., & Vomlel, J. (2003). Troubleshooting: NP-hardness and solution methods. *Soft Computing*, 7(5), 357-368.

[119] Warnquist, H., Kvarnström, J., & Doherty, P. (2009, September). Planning as heuristic search for incremental fault diagnosis and repair. In *Scheduling and Planning Applications Workshop (SPARK) at the International Conference on Automated Planning and Scheduling (ICAPS)*.

[120] Sniedovich, M. (2010). *Dynamic Programming: Foundations and Principles*. CRC Press.

[121] Kulkarni, A. U., Salado, A., Wernz, C., & Xu, P. (2020, August). Is verifying frequently an optimal strategy? A belief-based model of verification. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference* (Vol. 83983, p. V009T09A061). American Society of Mechanical Engineers.

[122] Kulkarni, A. U., Salado, A., Xu, P., & Wernz, C. (2021). An evaluation of the optimality of frequent verification for vertically integrated systems. *Systems Engineering*, 24(1), 17-33.

[123] Martinelli, G., Eidsvik, J., Hokstad, K., & Hauge, R. (2014). Strategies for petroleum exploration on the basis of Bayesian networks: A case study. *SPE Journal*, 19(04), 564-575.

[124] Shachter, R. D., & Bhattacharjya, D. (2010, July). Dynamic programming in influence diagrams with decision circuits. In *Proceedings of the Twentysixth Conference on Uncertainty in Artificial Intelligence (UAI)* (pp. 509-516).

[125] Pikrakis, A., Giannakopoulos, T., & Theodoridis, S. (2007, September). A dynamic programming approach to speech/music discrimination of radio recordings. In *2007 15th European Signal Processing Conference* (pp. 1226-1230). IEEE.

[126] Boucherie, R. J., & Van Dijk, N. M. (Eds.). (2017). *Markov Decision Processes in Practice*. Springer International Publishing.

[127] Liu, J., & Williams, R. K. (2018, May). Optimal intermittent deployment and sensor selection for environmental sensing with multi-robot teams. In *2018 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 1078-1083). IEEE.

[128] Petousis, P., Winter, A., Speier, W., Aberle, D. R., Hsu, W., & Bui, A. A. (2019). Using sequential decision making to improve lung cancer screening performance. *IEEE Access*, 7, 119403-119419.

[129] Velimirovic, L. Z., Janjic, A., & Velimirovic, J. D. (2019, October). Fault location and isolation in power distribution network using Markov decision process. In *2019 14th International Conference on Advanced Technologies, Systems and Services in Telecommunications (TELSIKS)* (pp. 408-411). IEEE.

[130] Powell, W. B. (2007). *Approximate Dynamic Programming: Solving the curses of dimensionality* (Vol. 703). John Wiley & Sons.

[131] Powell, W. B. (2016). Perspectives of approximate dynamic programming. *Annals of Operations Research*, 241(1), 319-356.

[132] Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3), 279-292.

[133] De Farias, D. P., & Van Roy, B. (2003). The linear programming approach to approximate dynamic programming. *Operations Research*, 51(6), 850-865.

[134] Li, L., & Littman, M. L. (2008). Efficient value-function approximation via online linear regression. In *ISAIM*.

[135] Bethke, B., How, J. P., & Ozdaglar, A. (2008, December). Approximate dynamic programming using support vector regression. In *2008 47th IEEE Conference on Decision and Control* (pp. 3811-3816). IEEE.

[136] Bethke, B., & How, J. P. (2009, June). Approximate dynamic programming using Bellman residual elimination and Gaussian process regression. In *2009 American Control Conference* (pp. 745-750). IEEE.

[137] Bertsekas, D. P., & Tsitsiklis, J. N. (1995, December). Neuro-dynamic programming: an overview. In *Proceedings of 1995 34th IEEE Conference on Decision and Control* (Vol. 1, pp. 560-564). IEEE.

[138] Robbins, H. (1952). Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58(5), 527-535.

[139] Bergemann, D., & Välimäki, J. (2006). Bandit problems (No. 1551). *Cowles Foundation for Research in Economics*, Yale University.

[140] Bubeck, S., & Cesa-Bianchi, N. (2012). Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *arXiv preprint arXiv*:1204.5721.

[141] Huo, X., & Fu, F. (2017). Risk-aware multi-armed bandit problem with application to portfolio selection. *Royal Society Open Science*, 4(11), 171377.

[142] Lai, T. L., & Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6(1), 4-22.

[143] Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2), 235-256.

[144] Kuleshov, V., & Precup, D. (2014). Algorithms for multi-armed bandit problems. *arXiv preprint arXiv*:1402.6028.

[145] Browne, C. B., et al. (2012). A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1), 1-43.

[146] Kocsis, L., & Szepesvári, C. (2006, September). Bandit based Monte-Carlo planning. In *European Conference on Machine Learning* (pp. 282-293). Springer, Berlin, Heidelberg.

[147] Kocsis, L., Szepesvári, C., & Willemson, J. (2006). Improved Monte-Carlo search. *Univ. Tartu, Estonia, Tech. Rep*, 1.

[148] Schadd, M. P., Winands, M. H., Herik, H. J. V. D., Chaslot, G. M. B., & Uiterwijk, J. W. (2008, September). Single-player Monte-Carlo tree search. In *International Conference on Computers and Games* (pp. 1-12). Springer, Berlin, Heidelberg.

[149] Bjornsson, Y., & Finnsson, H. (2009). Cadiaplayer: A simulation-based general game player. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(1), 4-15.

[150] Galichet, N., Sebag, M., & Teytaud, O. (2013, October). Exploration vs exploitation vs safety: Risk-aware multi-armed bandits. In *Asian Conference on Machine Learning* (pp. 245-260). PMLR.

[151] Chanthery, E., Pencolé, Y., & Bussac, N. (2010). An AO*-like algorithm implementation for active diagnosis. In *10th International Symposium on Artificial Intelligence, Robotics and Automation in Space, i-SAIRAS* (pp. 75-76).

[152] Dietterich, T. G. (2000, June). Ensemble methods in machine learning. In *International Workshop on Multiple Classifier Systems* (pp. 1-15). Springer, Berlin, Heidelberg.

[153] Dietterich, T. G. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(2), 139-157.

[154] Yang, P., Hwa Yang, Y., B Zhou, B., & Y Zomaya, A. (2010). A review of ensemble methods in bioinformatics. *Current Bioinformatics*, 5(4), 296-308.

[155] Matloob, F., et al. (2021). Software defect prediction using ensemble learning: A systematic literature review. *IEEE Access*.

[156] Saini, R., & Ghosh, S. K. (2017, May). Ensemble classifiers in remote sensing: A review. In *2017 International Conference on Computing, Communication and Automation (ICCCA)* (pp. 1148-1152). IEEE.

[157] Ho, T. K. (1995, August). Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition* (Vol. 1, pp. 278-282). IEEE.

[158] Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 1189-1232.

[159] Chen, T., & Guestrin, C. (2016, August). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 785-794).

[160] Ke, G., et al. (2017). LightGBM: A highly efficient gradient boosting decision tree. *Advances in Neural Information Processing Systems*, 30.

[161] Dorogush, A. V., Ershov, V., & Gulin, A. (2018). CatBoost: gradient boosting with categorical features support. *arXiv preprint arXiv*:1810.11363.

[162] Kinnander, M. (2020). Predicting profitability of new customers using gradient boosting tree models: Evaluating the predictive capabilities of the XGBoost, LightGBM and CatBoost algorithms.

[163] Bentéjac, C., Csörgő, A., & Martínez-Muñoz, G. (2021). A comparative analysis of gradient boosting algorithms. *Artificial Intelligence Review*, 54(3), 1937-1967.

[164] Jhaveri, S., Khedkar, I., Kantharia, Y., & Jaswal, S. (2019, March). Success prediction using random forest, CatBoost, XGBoost and AdaBoost for kickstarter campaigns. In *2019 3rd International Conference on Computing Methodologies and Communication (ICCMC)* (pp. 1170-1173). IEEE.

[165] Hancock, J., & Khoshgoftaar, T. M. (2020, December). Performance of CatBoost and XGBoost in medicare fraud detection. In *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)* (pp. 572-579). IEEE.

[166] Ibrahim, A. A., Ridwan, R. L., & Muhamme, M. M. (2020). Comparison of the CatBoost classifier with other machine learning methods. *International Journal of Advanced Computer Science and Applications*, 11(11), 738-748.

[167] Zhang, D., Qian, L., Mao, B., Huang, C., Huang, B., & Si, Y. (2018). A data-driven design for fault detection of wind turbines using random forests and XGBoost. *IEEE Access*, 6, 21020-21031.

[168] Zeinulla, E., Bekbayeva, K., & Yazici, A. (2020, July). Effective diagnosis of heart disease imposed by incomplete data based on fuzzy random forest. In *2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)* (pp. 1-9). IEEE.

[169] Elwell, R., & Polikar, R. (2011). Incremental learning of concept drift in nonstationary environments. *IEEE Transactions on Neural Networks*, 22(10), 1517-1531.

[170] Yin, X. C., Huang, K., & Hao, H. W. (2013, November). Dynamic ensemble of ensembles in nonstationary environments. In *International Conference on Neural Information Processing* (pp. 76-83). Springer, Berlin, Heidelberg.

[171] Yin, X. C., Huang, K., & Hao, H. W. (2015). DE2: Dynamic ensemble of ensembles for learning nonstationary data. *Neurocomputing*, 165, 14-22.

[172] Henrion, M. (2013). Practical issues in constructing a Bayes' belief network. *arXiv preprint arXiv*:1304.2725.

[173] Diez, F. J. (1993, January). Parameter adjustment in Bayes networks. The generalized noisy OR–gate. In *Uncertainty in Artificial Intelligence* (pp. 99-105). Morgan Kaufmann.

[174] Oniśko, A., Druzdzel, M. J., & Wasyluk, H. (2001). Learning Bayesian network parameters from small data sets: Application of Noisy-OR gates. *International Journal of Approximate Reasoning*, 27(2), 165-182.

[175] Loomes, G.; Sugden, R. (1982). Regret theory: An alternative theory of rational choice under uncertainty. *Economic Journal*. 92 (4): 805–824. doi:10.2307/2232669. JSTOR 2232669.

[176] Kuleshov, V., & Precup, D. (2014). Algorithms for multi-armed bandit problems. *arXiv preprint arXiv*:1402.6028.

[177] Russell, S., & Norvig, P. (2002). *Artificial intelligence: A Modern Approach*.

[178] Berger, J. O. (2013). *Statistical Decision Theory and Bayesian Analysis*. Springer Science & Business Media.

[179] Bolt, J. H., & van der Gaag, L. C. (2010, June). An empirical study of the use of the noisy-OR model in a real-life Bayesian network. In *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems* (pp. 11-20). Springer, Berlin, Heidelberg.

[180] Woudenberg, S. P., & Van Der Gaag, L. C. (2011, June). Using the noisy-OR model can be harmful… but it often is not. In *European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty* (pp. 122-133). Springer, Berlin, Heidelberg.

[181] Abbas, A. E., & Howard, R. A. (2015). *Foundations of Decision Analysis*. Pearson Higher Ed.

[182] Chan, H., & Darwiche, A. (2005). On the revision of probabilistic beliefs using uncertain evidence. *Artificial Intelligence*, 163(1), 67-90.

[183] Xu, P., Salado, A, A mathematical approach to design verification strategies that incorporate correction activities as dedicated decisions. Submitted.

[184] Collopy, P. D., & Hollingsworth, P. M. (2011). Value-driven design. *Journal of Aircraft*, 48(3), 749-759.

[185] Bellman, R. (1966). *Dynamic Programming*. Science, *153*(3731), 34-37.

[186] Rust, J. (2008). *Dynamic Programming*.

[187] Murphy, K. (2001). The Bayes net toolbox for matlab. *Computing Science and Statistics*, 33(2), 1024-1034.

[188] Xu, P., Deng, X., & Salado, A. (2022). A UCB-based tree search approach to joint verification-correction strategy for large scale systems. *arXiv preprint arXiv:*2204.00925.

APPENDICES

Appendix A: Proof of the Regret Bound of the UCBRB Rule

Lemma [150]: Let v be a bounded distribution with support in [0, 1], with *u* its supremum, and let us assume that v is lower bounded in the neighborhood of a:

$$\exists \mathrm{D} > 0, \forall \varepsilon > 0, P(\mathrm{X} \geq \mathrm{a} - \varepsilon) > \mathrm{D}\varepsilon, \text{ with X r.v.} \sim \mathrm{v}$$

Let x_1, … x_t be t samples independently drawn from X. The maximum value over x_i, i =1,…, t, goes exponentially fast to *u:*

$$P(\max_{1 \leq i \leq t} x_i \leq u - \varepsilon) > \exp(-\mathrm{tD}\,\varepsilon)\ .$$

Proof:

As the samples x_i are i.i.d., it comes:

$$P(\max_{1 \leq i \leq t} x_i \leq u - \varepsilon) = \mathrm{P}(\forall \mathrm{i} \in \{1, ..., \mathrm{t}\}, \mathrm{x}_i \leq u - \varepsilon)$$
$$= \prod_{i=1}^{t} P(x_i \leq u - \varepsilon) \leq (1 - \mathrm{D}\,\varepsilon)^t \leq \exp(-\mathrm{tD}\,\varepsilon)$$

■

Theorem 1: For all K > 1, if the UCBRB rule is run on K machines having arbitrary reward distributions with support in [0, 1], then its expected regret after any number n of plays is at most O(ln(n)).

Proof [143]:

For each machine k, the upper bound of n_k is determined first as $T_k(n)$. Let h be some positive integer, I_t=k is an indicator function, and $X_k^{\max}$ be the maximum value over $x_{k,i}$, i =1,…, t. The supremum of $X_k^{\max}$ is u_k.

$$T_k(\mathrm{n}) = 1 + \sum_{t=K+1}^{n} \{\mathrm{I}_t = \mathrm{k}\}$$
$$\leq h + \sum_{t=K+1}^{n} \{\mathrm{I}_t = \mathrm{k}, T_k(\mathrm{t\text{-}1}) \geq \mathrm{h}\}$$
$$\leq h + \sum_{t=K+1}^{n} \{X_*^{\max} + \frac{4\ln(t-1)}{D \bullet T_*(\mathrm{t\text{-}1})} \leq X_k^{\max} + \frac{4\ln(t-1)}{D \bullet T_k(\mathrm{t\text{-}1})}, T_k(\mathrm{t\text{-}1}) \geq \mathrm{h}\}$$
$$\leq h + \sum_{t=K+1}^{n} \{\min_{0<s<t} X_{*,s}^{\max} + \frac{4\ln(t-1)}{D \bullet s} \leq \max_{h \leq s_k < t} X_{k,s_k}^{\max} + \frac{4\ln(t-1)}{D \bullet s_k}\}$$
$$\leq h + \sum_{t=1}^{\infty} \sum_{s=1}^{t-1} \sum_{s_k=h}^{t-1} \{X_{*,s}^{\max} + \frac{4\ln(t)}{D \bullet s} \leq X_{k,s_k}^{\max} + \frac{4\ln(t)}{D \bullet s_k}\}$$

The inequation $X_{*,\mathrm{s}}^{\max}+\frac{4\ln(t)}{D\bullet s}\le X_{k,\mathrm{s}_k}^{\max}+\frac{4\ln(t)}{D\bullet s_k}$ implies at least one of the following inequations must hold:

$$X_{*,\mathrm{s}}^{\max}\le u_*-\frac{4\ln(t)}{D\bullet s},$$

$$X_{k,\mathrm{s}_k}^{\max}\ge \mathrm{u}_k+\frac{4\ln(t)}{D\bullet s_k},$$

$$\mathrm{u}_*<\mathrm{u}_k+2\frac{4\ln(t)}{D\bullet s_k}.$$

Because $X_{k,\mathrm{s}_k}^{\max}$ is always less than u_k, the second inequation cannot be true.

When $\mathrm{s}_k\ge\frac{8\ln(t)}{D\bullet(\mathrm{u}_*\text{-}\mathrm{u}_k)}$, the third inequation is false because

$$\mathrm{u}_*\text{-}\mathrm{u}_k-2\frac{4\ln(t)}{D\bullet s_k}\ge \mathrm{u}_*\text{-}\mathrm{u}_k-(\mathrm{u}_*\text{-}\mathrm{u}_k)=0.$$

With the lemma above, we have

$$\mathrm{P}(X_{*,\mathrm{s}}^{\max}\le \mathrm{u}_*-\frac{4\ln(t)}{D\bullet s})\le\exp(-sD\frac{4\ln(t)}{D\bullet s})=\exp(-4\ln(t))=\mathrm{t}^{-4}\cdot$$

So, we have

$$\begin{aligned}\mathrm{E}(T_k(\mathrm{n}))&\le\left\lceil\frac{8\ln(t)}{D\bullet(\mathrm{u}_*\text{-}\mathrm{u}_k)}\right\rceil+\sum_{t=1}^{\infty}\sum_{s=1}^{t-1}\sum_{s_k=\lceil 8\ln(t)/D\bullet(\mathrm{u}_*\text{-}\mathrm{u}_k)\rceil}^{t-1}\mathrm{P}(X_{*,\mathrm{s}}^{\max}\le\mathrm{u}_*-\frac{4\ln(t)}{D\bullet s})\\&\le\left\lceil\frac{8\ln(t)}{D\bullet(\mathrm{u}_*\text{-}\mathrm{u}_k)}\right\rceil+\sum_{t=1}^{\infty}\sum_{s=1}^{t}\sum_{s_k=1}^{t}t^{-4}\\&\le\frac{8\ln(t)}{D\bullet(\mathrm{u}_*\text{-}\mathrm{u}_k)}+1+\frac{\pi^2}{6}\end{aligned}.$$

Thus,

$$\mathrm{u}_*\bullet n-\mathrm{u}_k\sum_{k=1}^{K}E(\mathrm{n}_k)=\sum_{k=1}^{K}(\mathrm{u}_*\text{-}\mathrm{u}_k)E(\mathrm{n}_k)\le\sum_{k=1}^{K}((\mathrm{u}_*\text{-}\mathrm{u}_k)(\frac{8\ln(t)}{D\bullet(\mathrm{u}_*\text{-}\mathrm{u}_k)}+1+\frac{\pi^2}{6}))=\mathrm{O}(\ln(\mathrm{t})).\ \blacksquare$$

Appendix B: List of Publications during the Ph.D. Period

Journals:

1. **Xu, P.**, Deng, X., & Salado, A. (2022). A UCB-based tree search approach to joint verification-correction strategy for large scale systems. *arXiv preprint arXiv:*2204.00925.
2. **Xu, P.**, Salado, A, A mathematical approach to design verification strategies that incorporate correction activities as dedicated decisions. Submitted.
3. **Xu, P.**, Salado, A., & Deng, X. (2022). A parallel tempering approach for efficient exploration of the verification tradespace in engineered systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems.*
4. **Xu, P.**, & Salado, A. (2021). Modeling correction activities in the context of verification strategies. *Systems Engineering*.
5. Kulkarni, A. U., Salado, A., **Xu, P.**, & Wernz, C. (2021). An evaluation of the optimality of frequent verification for vertically integrated systems. *Systems Engineering*, 24(1), 17-33.
6. **Xu, P.**, Cho, J. H., & Salado, A. (2020). Expert opinion fusion framework using subjective logic for fault diagnosis. *IEEE Transactions on Cybernetics*.

Conferences:

1. **Xu, P.**, Salado, A., & Xie, G. (2020, October). A reinforcement learning approach to design verification strategies of engineered systems. *In 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (pp. 3543-3550). IEEE.
2. Kulkarni, A. U., Salado, A., Wernz, C., & **Xu, P.** (2020, August). Is verifying frequently an optimal strategy? A belief-based model of verification. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference* (Vol. 83983, p. V009T09A061). American Society of Mechanical Engineers.
3. **Xu, P.**, & Salado, A. (2019, July). A concept for set-based design of verification strategies. In *INCOSE International Symposium* (Vol. 29, No. 1, pp. 356-370).