A UCB-Based Tree Search Approach to Joint Verification-Correction Strategy for Large-Scale Systems

Peng Xu, Xinwei Deng, and Alejandro Salado, Senior Member, IEEE

Abstract—Verification planning is a sequential decision-making problem that specifies a set of verification activities (VAs) and correction activities (CAs) at different phases of system development. While VAs are used to identify errors and defects, CAs also play important roles in system verification as they correct the identified errors and defects. However, current planning methods only consider VAs as decision choices. Because VAs and CAs have different activity spaces, planning a joint verification-correction strategy (JVCS) is challenging, especially for large-scale systems. Here, we introduce a UCB-based tree search approach to search for near-optimal JVCSs. First, verification planning is simplified as repeatable bandit problems and an upper confidence bound rule for repeatable bandits (UCBRBs) is presented with the optimal regret bound. Next, a tree search algorithm is proposed to search for feasible JVCSs. A tree-based ensemble learning model is also used to extend the tree search algorithm to handle local optimality issues. The proposed approach is evaluated on the notional case of a communication system.

Index Terms—Bayesian network (BN), multiarmed bandit problem, random forest regression (RFR), sequential decision-making, verification planning.

I. INTRODUCTION

SYSTEM verification is defined as the process that evaluates whether a system or its components fulfill their requirements [1]. System verification is planned and implemented as a verification strategy (VS) that specifies how to implement activities at different developmental phases and on different system configurations [2]. A VS consists of verification activities (VAs), each of which is used to identify errors and defects, and correction activities (CAs) that correct the identified errors and defects [1], [3]. A VS aims to maximize confidence on verification coverage, minimize risk of undetected problems, and minimize invested effort [4].

Several strategy planning methods have been proposed to design VSs, including decomposition approach [5], set-based design [6], parallel tempering method [7], and reinforcement learning method [8]. All these methods treat only VAs as dedicated decisions that are planned for a system configuration at a development phase (i.e., system state), and CAs are simplified as default actions or even ignored reactively [9]. This simplification may undermine the value of resulting VSs because of the potential suboptimality of CAs. Thus, an extended paradigm of verification planning is presented in our previous paper to include both VAs and CAs as the result of independent decisions. All dependency relationships of VAs and CAs are summarized in that extended paradigm. An order-based backward induction (OBI) method is also proposed to find the optimal joint verification-correction strategies (JVCS) by updating the values of all system states [9]. However, as there are more possible activities, the number of the resulting possible system states increases exponentially [9]. Thus, the OBI method fails to obtain optimal strategies when the system size is large, and an alternative planning method is lacking.

In this article, we proposed a UCB-based tree search approach to generate near-optimal JVCSs for large-scale systems. The proposed approach can fit well with the engineering requirements within certain computational resources. The general flowchart is shown in Fig. 1. Part of this article reports the results of the Xu’s dissertation [10]. The contributions of this article are as follows.

First, we presented a search rule based on the upper confidence bound for repeatable bandits (UCBRBs) with its regret bound. The UCBRB rule can be used to find near-optimal strategies for decision-making problems whose strategies are repeatable.

Manuscript received 7 February 2023; accepted 12 April 2023. Date of publication 4 May 2023; date of current version 18 August 2023. This work was supported by the National Science Foundation under Grant CMMI-1762883. This article was recommended by Associate Editor F. Chiclana. (Corresponding author: Alejandro Salado.)
Peng Xu is with the Department of Industrial and Systems Engineering, Virginia Tech, Blacksburg, VA 24061 USA (e-mail: xupeng@vt.edu).
Xinwei Deng is with the Department of Statistics, Virginia Tech, Blacksburg, VA 24061 USA (e-mail: xdeng@vt.edu).
Alejandro Salado is with the Department of Systems and Industrial Engineering, University of Arizona, Tucson, AZ 85721 USA (e-mail: alejandrosalado@arizona.edu).
This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TSMC.2023.3270446.
Digital Object Identifier 10.1109/TSMC.2023.3270446
Second, we designed a UCBRB1 tree search method to apply the UCBRB rule to verification planning. 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.

Third, we leveraged random forest regression (RFR) to handle the local optimality issue of the UCBRB1 tree search method. We trained RFR with collected samples of system states during the tree search process and used the lower confidence bound of RFR outputs to predict state values as prior information. These prediction values are used to narrow the gap of state values between different system states and help jump out of the local optimum spaces.

The remainder of this article is organized as follows. Section II reviews the literature about bandit-based methods, tree search methods for sequential decision making, and tree-based ensemble learning models. Section III introduces tree search methods for sequential decision making, which are discussed in this section.

The multiarmed 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 [11]. This problem provides a paradigm of the tradeoff between exploration (trying each arm to find the best one) and exploitation (playing the arm believed to give the best payoff). 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 [12], which is the expected loss due to not playing the best bandit.

In a seminal paper, Lai and Robbins [13] 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. [14] is considered the optimal [15], [16]. The UCB1 policy is to play a machine \( k \) that maximizes \( \hat{x}_k + \sqrt{\frac{2\ln n}{n_k}} \), where \( \hat{x} \) is the average reward obtained from machine \( k \), and \( n_k \) is the number of times machine \( k \) has been played so far. 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 [17], [18] proposed the use of UCB1 as a tree search policy, which is called the upper confidence bound for trees (UCT). Its formula is

\[
\text{UCB} = \hat{x}_k + D_1 \sqrt{\frac{2\ln n}{n_k}}
\]

where \( D_1 \) is constant. While the UCT follows the assumption of expected average rewards to determine upper confidence bounds, we find two variant policies of the UCT that do not use expected average rewards to choose moves. First, Schadd et al. [19] extended bandit problems to single-player games with perfect information. They proposed a single-player MCTS policy (SP-MCTS) that adds a third term to the UCT rule

\[
\text{UCB} = \hat{x} + D_2 \sqrt{\frac{2n}{n_k}} + \sqrt{\frac{x^2 - n_k \cdot \hat{x}^2 + D_3}{n_k}}
\]

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 [20]. Second, Galichet et al. [21] studied a risk-aware bandit problem that measures the machine’s performance with the conditional value at risk (CVaR), which is the average of the lowest quantiles of the reward distribution with level \( \alpha \). 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 \( \text{CVaR}_k - D_4 \sqrt{\left(\ln\left(\frac{\ln (\eta \alpha)}{n_k \alpha}\right)\right)} \), where \( D_4 > 0 \) is a constant that controls the exploitation versus exploration tradeoff. As \( \alpha \) 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 quantities.

B. Tree Search Methods for Sequential Decision Making

Sequential decision making is a procedural approach to decision making that aims at finding a policy that maximizes the expected return for all possible initial system states, where earlier decisions influence the later available choices [22]. We distinguish between two types of sequential decision-making problems according to whether all following possible states of a decision should be further planned. If only one following possible 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 possible states, the optimal policy can be represented as an AND/OR tree where all actions of a decision have OR relationship and all following possible states of an action have AND relationship. While the first type is widely studied in the fields related to pathfinding [23], reinforcement learning [24], the second type has been explored in a variety of research domains, such as system verification [2], [9], troubleshooting [25], and adaptive testing [26], [27].
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. Dynamic programming (DP) is the major solution of the second type because it can break the problem down into simpler subproblems in a recursive manner. Until now, two DP approaches have been proposed to find the policy of the second type. The first approach is the exact DP methods, such as backward induction and value iteration. Exact DP methods are used to solve exact planning problems [28], such as verification planning [9], [29], [30], circuit design [31], and fault location [32]. However, when the state space is large, exact DP methods fail in finding exact solutions due to the curse of dimensions [33].

The second approach is approximate DP methods based on the approximation of decision-making processes. 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 [33]. Because verification strategies have VAs and CAs, the structure of a policy suffers the dependency relationship between VAs and CAs. So, the fourth class is mostly related to verification planning. As far as we know, the only value function approximation method for the second type is AO* algorithms that approximate state values with an admissible heuristic function. Nilsson [34], [35] first described a version of AO* algorithm to find AND/OR trees. Martelli and Montanari [36] generalized this algorithm 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 [37]. Hansen and Zilberstein [37] proposed a generalization of AO*, called LAO*, to find solution graphs with loops. These AO* algorithms have been applied to troubleshooting [38] and diagnosis of autonomous systems [39]. However, the AO* algorithm cannot be applied when the admissible heuristic functions are not available.

C. 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 [40]. These ensemble techniques have the advantage of alleviating the small sample size problem by incorporating multiple learning models to reduce the potential for overfitting the training data [41], [42]. Decision trees are commonly used in ensemble learning models because decision trees are sensitive to small changes in the training set [40]. 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 [42], defect prediction [43], and remote sensing [44]. However, as far as we know, tree-based ensemble learning models have not been used in verification planning.

Some common types of ensemble learning include bagging, boosting, and stacking, which are realized as some basic models, such as random forest (RF) [45], XGBoost [46], LightGBM [47], and CatBoost [48]. Many studies have been conducted to test the performance of these models, and each model has its own merits. For example, Bentéjac et al. [49] found that CatBoost obtained the best results in generalization accuracy and AUC while LightGBM had the fastest training speed in the studied datasets. Ibrahim and Khoshgoftaar [50] recommended a CatBoost algorithm for better prediction of loan approvals and staff promotion. In another study about highly imbalanced Big Data [51], XGBoost was found to be better than CatBoost because of its shorter training time. In addition, the scope of tree-based ensemble learning models is not limited to these basic models. For example, Ibrahim et al. [52] proposed a weighted RF along with AdaBoost to predict the success rate of Kickstarter campaigns. Zhang et al. [53] combined RFs with XGBoost to establish the data-driven fault detection framework for wind turbines. Zeinulla et al. [54] proposed a fuzzy RF 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 widely used in nonstationary environments, where the underlying data distribution changes over time (i.e., concept drift). Elwell and Polikar [55] 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. [56], [57] 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 the verification planning that searches for optimal strategies.

III. Verification Planning Framework

A. System Verification With Bayesian Networks

We consider that a given system can be decomposed into a set of system elements and assume that the objective of system verification is to verify relevant requirements for these elements. We conceive system verification as a set of tuples of system parameters \( \theta_1, \ldots, \theta_I \) that provide information about such system parameters. Using the modeling framework presented in [58], we build a basic system verification model as a Bayesian network (BN). In the resulting BN, nodes representing VAs will be treated as observable nodes (those whose node states can be observed directly) and nodes representing system parameters will be treated as hidden nodes (those whose value states cannot be observed directly but are inferred from the values of the observable nodes). For example, consider a computer system that has two parameters, processor speed (denoted by \( \theta_1 \)) and computer speed (denoted by \( \theta_2 \)), and each parameter has its own VA (denoted by \( \mu_1 \) and \( \mu_2 \),...
The BN can be built accordingly, as shown in Fig. 2(a).

Because the interpretation of the information provided by VAs is subjective [59], we capture the information about system parameters as beliefs. Without loss of generality, all nodes are assumed to be binary (i.e., two node states, such as pass/fail). 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 and even continuous belief distributions [60]. The specific beliefs of a network node are presented as a conditional probability table (CPT) in this article. Each CPT summarizes the dependency relationships between a node and its parent nodes. After all CPTs are elicited as prior distributions of a BN, the impact of a VA on the beliefs is modeled as follows: 1) a verification result \( A(\mu_i) \) is collected after executing \( \mu_i \) (i.e., an observable node \( \mu_i \) is observed) and 2) the posterior distributions of the network nodes are updated by the Bayesian rule.

CAs are defined as those that correct errors or defects that are found during system development [61]. CAs impact the confidence of system parameters because they affect the system configuration. In our previous study [61], uncertain evidence is leveraged to model the effects of CAs on the BN. Three basic types of CAs are modeled with their uncertain evidence: rework, repair, and redesign. For example, when a repair activity is executed to modify a faulty element with parts, processes, or materials that were initially unplanned for that element, it is assumed that repairing the element impacts the beliefs of the corresponding parameter in the verified system. We can apply virtual evidence to represent the impact of repair on the beliefs. For example, a repair activity is conducted to improve the overall computer speed. After the repair is applied to the system, this piece of evidence is captured as virtual evidence \( \varphi_1 \) applied on \( \theta_2 \) directly, as shown in Fig. 2(b). When the uncertain evidence of a CA is collected, the posterior distributions of the network nodes are also updated by the Bayesian rule [61].

B. Verification Planning Problems

Verification planning is defined as a sequential process of repeating VAs and CAs at \( T \) time events [9]. For simplicity, we assume that only one VA and one CA is conducted at each time event and the CA follows the VA. The solution of verification planning is the assignment of VAs and CAs along a verification process, called a joint verification-correction strategy (JVC). Because each VA has multiple possible results (e.g., Pass/Fail), a JVC can be presented as an activity tree, as shown in Fig. 3. Each path from the root node to a terminal node is called a verification path. So, there are 5 verification paths in Fig. 3, and all verification paths share the same initial system state \( S_1 \). System states are generated along with the collection of activity results at each verification path. Each verification path terminates with a certain system state, which is called a terminal state (denoted by “Stop,” as shown in Fig. 3). 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” (i.e., No Activity) is selected when assigning a VA.

At each time event, verification planning consists of assigning a VA and a CA from their own activity spaces that includes all eligible activity actions, including the action NA. There are two constraints about the activity spaces of VAs and CAs. First, it is unnecessary to repeat any activity if this activity has been executed and its result remains valid. Second, implementing a CA can make the current results of a VA invalid if the VA depends on the corrected parameter. This is because once a CA changes a system parameter, all VAs that depend on such a parameter lose their credibility in deducing accurate posterior beliefs of the system [9]. Therefore, the activity spaces always changes along with the collection of evidence.

Three value factors are considered to calculate the value function. The first value factor is activity costs, a fixed amount of financial resources necessary to conduct an activity. It is denoted as \( C(\mu_j) \) for \( \mu_j \) and \( C(\varphi_k) \) for \( \varphi_k \). The second one is failure costs, \( C(A(\mu_j) = \text{Fail}) \), which is incurred when the result of a VA is found to be Fail. The third one is system revenue, \( B(\theta_i) \), which is obtained only when a verification process terminates (i.e., reaches a terminal state) and the system is deployed. We consider the system to be deployed when the confidence level \( P(\theta_i = \text{Pass}) \) of the target parameter \( \theta_i \) at a system state \( S_m \) reaches or surpasses certain thresholds, \( H_i \).

Consider a JVC \( \Psi \) that starts from a system state \( S_1 \) and has \( W \) verification paths \( Z_1, \ldots, Z_W \) and each verification path has a set of system states \( S_{1:w} \). For a given verification path \( Z_w \), the overall value is calculated as

\[
U(Z_w) = \sum_{i} B(\theta_i)P(\theta_i = \text{Pass})\delta(P(\theta_i = \text{Pass}) > H_i) - \sum_{j} C(\mu_j) - \sum_{j} C(A(\mu_j) = \text{Fail}) - \sum_{k} C(\varphi_k)
\]

where \( \delta(\cdot) \) is an indicator function whose value is 1 if the statement is true and 0 otherwise. Then, the performance of this JVC \( U(\Psi|S_1) \) is calculated as the weighted sum of the
overall values of all verification paths
\[ U(\Psi|S_1) = E(U(Z_w)) = \sum_w P(Z_w)U(Z_w) \] (4)

where \( P(Z_w) \) 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 a verification path
\[ P(Z_w) = \prod_m P(A|S_m). \] (5)
The verification planning problem is to solve for the optimal JVCS for a given initial system state \( S_1 \)
\[ \Psi_{opt} = \arg \max_{\Psi} U(\Psi|S_1). \] (6)

A summary of the notations used in this article is shown in the supplementary file.

IV. PROPOSED UCB-BASED TREE SEARCH APPROACH

A. Upper Confidence Bound for Repeatable Bandits

Verification planning consists of selecting activities at a set of system states. If it is hard to explore all system states, 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. Thus, 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 machine \( k \) is represented by a random variable \( X_{ks} \) for \( 1 \leq k \leq K \), where \( s \) is the index of successive plays. The reward \( X_{ks} \) is independent and identically distributed (i.i.d.) according to an unknown distribution \( F_k \) whose support is \([a_k, b_k]\). The distributions of all machines \( \{F_k\} \) are independent of each other. 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 an RBP in this article. The objective of an RBP is to maximize the sum of expected 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 maximum reward after \( n_k \) plays of machine \( k \). 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 \( u_s = \max_k(u_k) \). We follow the previous study [13] to define the regret of an RBP as:
\[ u_s \cdot n - u_k \sum_{k=1}^{K} E(n_k) \] (7)

where \( n_1 + n_2 + \cdots + n_k = n \) and \( n \) is the total plays of all machines. 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 [14]. Previous policies, such as the UCB1, are not appropriate choices for RBPs because they use the expectation of a machine as the target. Thus, we proposed a modified policy called the upper confidence bound for repeatable bandits (UCBRB).

1) Initialization: Play each machine once.
2) Loop: Play machine \( k \) that maximizes
\[ x_k^{max} + \left(\ln(n)\right)/D_0 \cdot n_k \], 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 shown in the supplementary file. 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 \((n + 1/n - 1)(x_{k}^{max} - x_{k}^{min})\). So, the estimate of \( D_0 \) is \( \min(n + 1/n)(x_{k}^{max} - x_{k}^{min})\).

The regret bound of this policy is found to be \( O(\ln(n)) \). The proof of this regret bound is provided in the supplementary file.

B. UCBRB1 Tree Search Method

As verification planning can be viewed as a sequential play of RBPs, we use the proposed UCBRB rule to calculate the UCB of the expected reward of a nonterminal system state \( S_m \)
\[ UCB(S_m) = x_{k}^{max} + D_6 \frac{\ln(n)}{n_k}, D_6 \geq 0 \] (8)

where \( x_{k}^{max} \) is the maximum expected reward of \( n_k \) JVCSs that starts from \( S_m \) (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 state \( S_{m-1} \). \( D_6 \) is a constant that depends on the unknown distribution 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 a system state is a terminal state,
its reward is deterministic no matter how many more times it is visited. So, the second item in (8) is 0, and the UCBs of terminal system states are their maximum reward value.

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, we proposed a UCBRB1 tree search method to make sequential decisions along a verification process. The method consists 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 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 III-B. 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

$$\mu_a = \arg \max_{\mu_j} \left(-C(\mu_j) - C(A(\mu_j) = \text{Fail}) P(A(\mu_j) = \text{Fail}) + \sum_a P(A(\mu_j) = a) \text{UCB}(S_{m+1}|a)\right),$$

$$\varphi_a = \arg \max_{\varphi_k} (-C(\varphi_k) + \text{UCB}(S_{m+1}|\varphi_k))$$

where $S_{m+1}$ means the following 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 its child nodes in this tree

$$U(S_m) = -C(\mu_j) - C(A(\mu_j) = \text{Fail}) P(A(\mu_j) = \text{Fail}) + \sum_a P(A(\mu_j) = a) U(S_{m+1}|\mu_a),$$

$$U(S_m) = -C(\varphi_k) + U(S_{m+1}|\varphi_k).$$

Then the expected rewards are compared with the previous record 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 [62] from three aspects to improve the efficiency of tree search processes. 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, we update the information of all nodes only when the whole AND/OR tree is expanded completely rather than whenever a node is expanded. Third, we use the UCBs rather than an admissible function to represent the heuristic value of a system state.

In addition, two adjustments are made to (8) in practice. First, while (8) is meaningless if either $n$ or $n_k$ is 0, the visit counts of all system states are 0 at the beginning. So, we assume that all system states have been visited once before the tree search and, for simplicity, the expected value is set as None. Then, (8) is equivalent to

$$\text{UCB}(S_m) = x_k^{\max} + \frac{D_0 \ln(n + 1)}{n_k + 1}.$$  

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, we add a penalty item in (8) to prevent the over-expansion of trees

$$\text{UCB}(S_m) = x_k^{\max} + \frac{D_0 \ln(n + 1)}{n_k + 1} - f(m|\Psi).$$

For simplicity, the penalty item $f(m|\Psi)$ is assumed to be a function of state index $m$ when the tree is expanded. For example, $f(m|\Psi) = D_k \times |m/D_0|$, $D_k = 1$, and $D_0 = 50$. That is, the UCB of a nonterminal state is reduced by 1 every 50 system states. Finally, the UCBRB1 tree search method is summarized in Algorithm 1.

### Algorithm 1 UCBRB1 Tree Search Method

1. **Inputs:**
   - $L = \{\emptyset\}$: lookup table; $\Psi$: sample tree.
2. Initialize $\Psi = \{S_1\}$, where $S_1$ is the initial state.
3. **while** $\Psi$ has some tip nodes **do**
4. Expand all tip nodes of $\Psi$ according to the UCBs of all following states calculated by Eq. 12.
5. Add expanded states to $\Psi$.
6. Denote all nonterminal expanded states as tip nodes.
7. **end while**
8. Update state values and visit counts of all nodes in $\Psi$ according to Eq. 10.
9. Update $L$ with the updated state values and visit counts.
10. **if** the expected value $U(\Psi)$ converges **then**
    11. Output $\Psi_{opt}$ as the solution.
    12. **else**
    13. Go to 2.
14. **end if**
process, 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). We use three types of variables as the input to predict state values as shown in Fig. 4. 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 different system states. These count variables are redundant because they are the sum of the absolute values of all statuses.

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

We add RFR approximation to extend the UCBRB1 tree search method in two places. First, while sample trees are generated continuously along the tree search process, we divide tree search processes into sampling periods and collect tree nodes of sample trees as training datasets at each sampling period. Because of the randomness of tree search processes, the dependency relationship of collected sample trees is random and constantly changes during the tree search process. That is, all training samples are not i.i.d. samples. So, we use RFR models to interpolate the datasets, and each decision tree is built with a zero mean squared error. Thus, the information of all visited system states are saved accurately in the RFR models. Second, we use the latest RFR model to predict state values as prior values because of the concept drift in the RFR models. Second, we use the latest RFR model to predict state values as prior values because of the concept drift in the RFR models. 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 article.

V. EXPERIMENTAL DESIGN

A. Problem Description

In this section, we implement the proposed framework to design a JVCS for an optical instrument in a satellite [58]. The notional instrument has been used to support prior research in verification [58]. The system parameters of this optical instrument and their possible VAs are modeled as the BN shown in Fig. 5. System parameters are represented as circle nodes and candidate VAs square nodes. The definitions of the nodes are given in [58], 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 [63], which takes into account the physical meaning of the different modes when estimating their mutual effects for the reasonability of the data. In the BN, there is a dependency between \( \theta_1 \) and all other network nodes. Once an activity is executed on any node of these 32 nodes and an activity result is collected, the confidence \( P(\theta_1 = \text{Pass}) \) will
change. While this instrument is verified through a set of these VAs, each parameter \( \theta_i \) 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 \( \theta_i \) is denoted as \( \varphi_k(\theta_i), k = 1, \ldots, 10 \).

This experiment assumes that system revenue is driven by the target system parameter \( \theta_i \). The threshold for the system deployment rule, \( H_1 \), is set as 0.90. Revenue and cost data have also been synthetically generated in thousand-dollar units ($1000). The revenue \( B(\theta_i) \) has been set to 20 000 to provide a balance when making a selection tradeoff between different VAs. The activity costs of the different activities and the failure costs of VAs, are provided in Table II in the supplementary file. Specific values have been generated according to the type of activities defined in [58].

### B. Experimental Method

With the provided problem and generated data, the whole experiment is realized with MATLAB R2018a, Python 3.6, scikit-learn 0.24.1, and Bayes Net Toolbox for MATLAB [64]. Two scenarios are conducted to study the performance of the proposed approach. In Scenario 1, the target network is the smaller one outlined by the dashed line in Fig. 5. With five system parameters and 9 VAs in this BN, there are \( 2 \cdot 2^5 = 1259712 \) total system states. To get some intuition about this scenario, we apply the OBI method first to solve for the exact JVCS. When the expected values of all system states are calculated, the optimal activities of all system states can be identified to constitute the exact JVCS, as shown in the supplementary file. The expected value of this JVCS is 7788, while the total running time is 99 925 s.

In Scenario 2, the target network is the whole network in Fig. 5. With ten system parameters and 22 VAs in this BN, there are \( 2 \cdot 2^{10} \cdot 2^{22} = 6.43 \cdot 10^{13} \) system states. As the total number of system states is too large, the OBI 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. It is discussed later whether a better JVCS can be found in the large network.

In each scenario, the proposed approach is compared with two types of benchmark methods. First, we compare the UCBRB rule with other UCB rules, including the UCT rule [i.e., (1)] and the SP-MCTS rule [i.e., (2)]. We combine Algorithm 1 with all these UCB rules to compare their effects. The constant values of each rule are selected through sensitivity analyses. We set \( D_7 = 20000, D_8 = 1 \), and \( D_6 = 50 \). The tree search process is conducted by generating 5000 sample trees. We record the expected value of the optimal JVCS for every 50 sample trees to compare the performance of different UCB rules.

Second, we compare the proposed methods with a Monte Carlo method. First, the UCBRB2 tree search method is 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. All RFR models are realized with the function “RandomForestRegressor” of scikit-learn. The hyperparameter “Bootstrap” is set as False. The hyperparameter “minimum number of samples required to be at a leaf node” is set as 1 to interpolate state samples. All other hyperparameters are set as their default values. Second, a Monte Carlo method is designed to search for a JVCS randomly. As the number of tree nodes can be large, this Monte Carlo method constrains 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 (4). Second, the program’s running time means wall-clock time, which is used to compare computational efficiency.

### C. Experimental Results

#### 1) Scenario 1:

In the UCBRB rule [given by (8)], the constant \( D_6 \) depends on the unknown distributions of the specific problem. So, it is necessary to first determine the constant \( D_6 \) 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 1 is conducted to test the performance, as shown in Fig. 6. The expected values and runtimes of all possible constants are listed in Table I. It is found that when \( D_6 \) is 0.5, the expected value reaches the maximum 7780.78 among the 5000 sample trees. So, we set \( D_6 \) as 0.50 for the UCBRB rule in this experiment.

From Fig. 6 and Table I, it can be observed that when \( D_6 \) decreases, the expected value converges faster. The reason is that more weights are allocated to exploiting existing strategies [i.e., the first item in (8)] 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 converges much slower.
value becomes stable after the 1950th sample tree. Thus, $D_6$ should be large enough given fixed computation resources.

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 constant values of UCT and SP-MCTS is 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. 7(a). 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 (8), which make the UCBRB more sensitive to the optimal activities. It is also found that SP-MCTS converges faster than UCT. We attribute it to the third item in (2) as it adds more weight to those states 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.

Finally, we also conducted the UCBRB2 tree search method and the Monte Carlo method to search this small strategy space. The UCBRB2 tree search method shares the same $D_6$ value with the UCBRB1 tree search method. The Monte Carlo method is conducted to generate 5000 sample trees. Their performances are shown in Fig. 7(b) and summarized in Table II. 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, we use the distribution of all activities at the initial state $S_1$ to study the local optimality issue. If the UCBRB1 tree search method is used, the first activity is fixed as $\mu_{19}$ after the 500th sample tree [Fig. 2(a) in the supplementary file]. This activity immovability problem is attributed to the concept drift that enlarges the value gap of the first item in (8). So, the UCB of other activities cannot surpass that of $\mu_{19}$ [Fig. 3(a) in the supplementary file]. The UCBRB2 tree search method solved this local optimality issue by narrowing the value gap with the prior information [Fig. 3(b) in the supplementary file]. The UCBRB2 tree search method solved this local optimality issue by narrowing the value gap with the prior information [Fig. 3(b) in the supplementary file]. So, more sample trees are allocated to other activities, and it is possible to jump out of the local optimum strategy space [Fig. 2(b) in the supplementary file].

---

**TABLE II**

Performance Comparison of All UCB Rules and Benchmark Methods (Including UCT, SP-MCTS, Monte Carlo, and DP)

| Method                | Scenario 1 Expected Value | Scenario 1 Runtime | Scenario 2 Expected Value | Scenario 2 Runtime |
|-----------------------|---------------------------|--------------------|---------------------------|--------------------|
| UCBRB                 | 7780.76                   | 7542.54            | 8478.43                   | 102109.83          |
| UCT                   | 7746.82                   | 8915.13            | 7011.08                   | 281919.13          |
| SP-MCTS               | 7526.81                   | 10183.85           | 6114.59                   | 227776.50          |
| UCBRB+RFR             | 7780.72                   | 12120.16           | 8487.27                   | 186804.14          |
| Monte Carlo            | 801.45                    | 6797.46            | 0                         | 73234.74           |
| Exact Solution        | 7788                      | 97925              | -                         | -                  |

---

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
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. We compare all UCB rules first in this scenario and show their results in Fig. 7(c) and Table II. 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 high-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 its 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 (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.

The performances of all three benchmark methods are shown in Fig. 7(d) and also summarized in Table II. 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 $\mu_{301}$ [Fig. 2(c) in the supplementary file]. It is caused by the same reason as in Scenario 1 that the value gap of the first item in (8) is too large. So, the UCBs of other activities can hardly surpass that of $\mu_{301}$. However, the UCBRB2 tree search method can improve the UCBs significantly so that other activities are allocated with more exploration times. Thus, the tree search process can jump out of local optimum spaces and find a better JVCS. As shown in Table II, the UCBRB2 method has the largest expected value 8487.27.

D. Discussion

The two scenarios in this experiment are designed by expanding the activity set of the same system network. With the proposed UCBRB1 and UCBRB2 tree search methods, a better JVCS is found in Scenario 2, which is attributed to those five extra CAs and 13 VAs (i.e., $\phi_5$ to $\phi_{10}$, and $\mu_{20}$ to $\mu_{32}$) in Scenario 2. However, this phenomenon does not always occur. We have verified the expected value of the exact JVCS (ref. Fig. 1 in the supplementary file) 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. 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 7788. Due to some low-cost activities, such as $\phi_{10}$ and $\mu_{23}$, 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.

We also found that activity immovability always happens in this verification planning problem during the tree search process, no matter which UCB rule is used. This problem cannot be solved by adjusting the constant $D_n$. It is attributed to the concept drift of value distributions. In addition, we assume there are two causes of concept drift. The first one is the lack 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 (8)], the first item is None and the second item is 0. Then, the optimal activity is “NA” because it has the minimal activity cost 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, this JVCS is fixed as a core set of activities, and the performance 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.

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 our 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 feasible JVCS with less runtime and storage space. However, suppose it is essential to explore other candidate activities. In that case, 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.
VI. CONCLUSION

This article has presented a UCB-based tree search approach to solve the verification planning problem. First, we simplify the verification planning problem as a RBP and propose a UCBRB rule. The upper regret bound of this UCBRB rule is also found and proved. Then we propose a UCBRB1 tree search method to apply the UCBRB rule to search for JVCSs. A tree-based ensemble learning model is also used to extend the UCBRB1 tree search method by using RFR models to predict state values. It is found that the proposed UCBRB rule can outperform other UCB rules in the experiment. This advantage is more evident when the size of the system network increases. The UCBRB2 tree search method can also effectively solve the local optimality issue and handle the exploration–exploitation dilemma better than the UCBRB1 one.

We would like to remark that there are three limitations to the proposed methodology. First, 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. Second, some other parameters and constants of tree search algorithms are not optimized, including discount constant, penalty item, and total sample tree number. Third, when training RFR models, we set most hyper-parameters with their default values and make some ad hoc adjustments, such as removing terminal states as samples and adding the same number of system states from lookup tables.

In addition, we suggest that this work opens two main future research directions. First, this work focuses on the tree search given a fixed system network. The selection of possible subnetworks could be explored to simplify tree search processes. The subnetworks may be generated by evaluating the impacts of activities on JVCSs. Second, only RFR is studied in this work. 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.

ACKNOWLEDGMENT

The authors thank Advanced Research Computing at Virginia Tech for providing high-performance computing resources.

REFERENCES

[1] A. Engel, Verification, Validation, and Testing of Engineered Systems. Hoboken, NJ, USA: Wiley, 2010.
[2] A. Salado and H. Kannan, “A mathematical model of verification strategies,” Syst. Eng., vol. 21, no. 6, pp. 593–608, 2018.
[3] Quality Management Systems-Fundamentals and Vocabulary (ISO 9000:2015), ISO, Geneva, Switzerland, 2015.
[4] A. Salado, “Defining better test strategies with tradespace exploration techniques and Pareto fronts: Application in an industrial project,” Syst. Eng., vol. 18, no. 6, pp. 639–658, 2015.
[5] M. Barad and A. Engel, “Optimizing VVT strategies: A decomposition approach,” J. Oper. Res. Soc., vol. 57, no. 8, pp. 965–974, 2006.
[6] P. Xu and A. Salado, “A concept for set-based design of verification strategies,” in Proc. INCOSE Int. Symp., vol. 29, 2019, pp. 356–370.
[7] P. Xu, A. Salado, and X. Deng, “A parallel tempering approach for efficient exploration of the verification tradespace in engineered systems,” 2021, arXiv:2109.11704.
[8] P. Xu, A. Salado, and G. Xie, “A reinforcement learning approach to design verification strategies of engineered systems,” in Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC), 2020, pp. 3543–3550.
[9] P. Xu and A. Salado, “A mathematical approach to design verification strategies that incorporate corrective activities as dedicated decisions,” IEEE Open J. Syst. Eng., vol. 1, pp. 1–11, 2023, doi: 10.1109/OJSIE.2022.3222731.
[10] P. Xu, “Design of joint verification-correction strategies for engineered systems,” Ph.D. dissertation, Ind. Syst. Eng., Virginia Polytech. Inst., State Univ., Blacksburg, VA, USA, 2022.
[11] D. Bergemann and J. Vălămăkii, “Bandit problems,” Cowles Found. Res. Econ., Yale Univ., New Haven, CT, USA, Rep., 2006.
[12] S. Bubeck and N. Cesa-Bianchi, “Regret analysis of stochastic and nonstochastic multi-armed bandit problems,” 2012, arXiv:1204.5721.
[13] T. L. Lai and H. Robbins, “Asymptotically efficient adaptive allocation rules,” Adv. Appl. Math., vol. 6, no. 1, pp. 4–22, 1985.
[14] P. Auer, N. Cesa-Bianchi, and P. Fischer, “Finite-time analysis of the multiarmed bandit problem,” Mach. Learn., vol. 47, no. 2, pp. 255–256, 2002.
[15] V. Kuleshov and D. Precup, “Algorithms for multi-armed bandit problems,” 2014, arXiv:1402.6028.
[16] X. Huo and F. Fu, “Risk-aware multi-armed bandit problem with application to portfolio selection,” Royal Soc. Open Sci., vol. 4, no. 11, 2017, Art. no. 171377.
[17] L. Kocsis and C. Szepesvári, “Bandit based Monte-Carlo planning,” in Proc. Eur. Conf. Mach. Learn., 2006, pp. 282–293.
[18] L. Kocsis, C. Szepesvári, and J. Willemsen, “Improved Monte-Carlo search,” Inst. Comput. Sci., Univ. Tartu, Tartu, Estonia, Rep., 2006.
[19] M. P. Schadd, M. H. Winands, H. Herik, G. M.-B. Chaslot, and J. W. Uiterwijk, “Single-player Monte-Carlo tree search,” in Proc. Int. Conf. Comput. Games, 2008, pp. 1–12.
[20] Y. Björnsson and H. Finnsson, “Cadiplayer: A simulation-based general game player,” IEEE Trans. Comput. Intell. AI Games, vol. 1, no. 1, pp. 4–15, Mar. 2009.
[21] N. Galichet, M. Sebag, and O. Teytaud, “Exploration vs exploitation vs safety: Risk-aware multi-armed bandits,” in Proc. Asian Conf. Mach. Learn., 2013, pp. 245–260.
[22] A. G. Barto, R. S. Sutton, and C. Watkins, Learning and Sequential Decision Making, Univ. Massachusetts Amherst, Amherst, MA, USA, 1989.
[23] Y. Wang, X. Li, and R. Ruiz, “An exact algorithm for the shortest path problem with position-based learning effects,” IEEE Trans. Syst., Man, Cybern., Syst., vol. 47, no. 11, pp. 3037–3049, Nov. 2017.
[24] L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement learning: A survey,” J. Artif. Intell. Res., vol. 4, pp. 237–285, May 1996.
[25] R. M. Khanafa et al., “Automated diagnosis for UMTS networks using Bayesian network approach,” IEEE Trans. Veh. Technol., vol. 57, no. 4, pp. 2451–2461, Jul. 2008.
[26] M. Plajner and J. Vomlel, “Bayesian network models for adaptive testing,” 2015, arXiv:1511.08488.
[27] P. Xiao, Y. Yin, B. Liu, B. Jiang, and Y. K. Malaiya, “Adaptive testing based on moment estimation,” IEEE Trans. Syst., Man, Cybern., Syst., vol. 50, no. 3, pp. 911–922, Mar. 2020.
[28] M. Sniedovich, Dynamic Programming: Foundations and Principles. Boca Raton, FL, USA: CRC Press, 2010.
[29] A. U. Kulkarni, A. Salado, C. Wernz, and P. Xu, “Is verifying frequently an optimal strategy? A belief-based model of verification,” in Proc. Int. Design Eng. Techn. Conf. Comput. Inf. Eng. Conf., 2020, Art. no. V009T09A061.
[30] A. U. Kulkarni, A. Salado, P. Xu, and C. Wernz, “An evaluation of the optimality of frequent verification for vertically integrated systems,” Syst. Eng., vol. 24, no. 1, pp. 17–33, 2021.
[31] R. D. Shachter and D. Bhattacharjya, “Dynamic programming in influence diagrams with decision circuits,” in Proc. 26th Annul. Conf. Uncertainty Artif. Intell. (UAI), 2010, pp. 509–516.
[32] L. Z. Velimirovic, A. Janicj, and J. D. Velimirovic, “Fault location and isolation in power distribution network using Markov decision processes,” in Proc. 14th Int. Conf. Adv. Technol., Syst. Services Telecommun. (TELSIKS), 2019, pp. 408–411.
[33] W. B. Powell, Approximate Dynamic Programming: Solving the Curses of Dimensionality. Hoboken, NJ, USA: Wiley, 2007, vol. 703.
[34] N. J. Nilsson, “Searching problem-solving and game-playing trees for maximal cost solutions,” in Proc. IFIP Congr., vol. 2, 1968, pp. 1556–1562.
[35] N. J. Nilsson, Principles of Artificial Intelligence. Cham, Switzerland: Springer, 1982.
[36] A. Martelli and U. Montanari, “Optimizing decision trees through heuristically guided search,” Commun. ACM, vol. 21, no. 12, pp. 1025–1039, 1978.

[37] E. A. Hansen and S. Zilberstein, “LAO*: A heuristic search algorithm that finds solutions with loops,” Artif. Intell., vol. 129, nos. 1–2, pp. 35–62, 2001.

[38] H. Wannquist, I. Kvarnström, and P. Doherty, “Planning as heuristic search for incremental fault diagnosis and repair,” in Proc. Int. Conf. Autom. Plan. Schedul. Plan. Appl. Workshop, 2009, pp. 1–7.

[39] E. Chantery, Y. Pencolé, and N. Bussac, “An AO*-like algorithm implementation for active diagnosis,” in Proc. 10th Int. Symp. Artif. Intell., Robot. Autom. Space, F-SAIRAS, 2010, pp. 75–76.

[40] T. G. Dietterich, “Ensemble methods in machine learning,” in Proc. Int. Workshop Multiple Classifier Syst., 2000, pp. 1–15.

[41] T. G. Dietterich, “An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization,” Mach. Learn., vol. 40, no. 2, pp. 139–157, 2000.

[42] P. Yang, Y. H. Yang, B. B. Zhou, and A. Y. Zomaya, “A review of ensemble methods in bioinformatics,” Current Bioinformatics, vol. 5, no. 4, pp. 296–308, 2010.

[43] F. Matloob et al., “Software defect prediction using ensemble learning: A systematic literature review,” IEEE Access, vol. 9, pp. 98754–98771, 2021.

[44] R. Saini and S. Ghosh, “Ensemble classifiers in remote sensing: A review,” in Proc. Int. Conf. Comput. Commun. Autom. (ICCCA), 2017, pp. 1148–1152.

[45] T. K. Ho, “Random decision forests,” in Proc. 3rd Int. Conf. Document Anal. Recognit., vol. 1, 1995, pp. 278–282.

[46] T. Chen and C. Guestrin, “XGBoost: A scalable tree boosting system,” in Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Disc. Data Min., 2016, pp. 786–794.

[47] G. Ke et al., “LightGBM: A highly efficient gradient boosting decision tree,” in Proc. Adv. Neural Inf. Process. Syst., vol. 30, 2017, pp. 1–9.

[48] A. V. Dorogush, V. Ershov, and A. Gulin, “CatBoost: Gradient boosting with categorical features support,” 2018, arXiv:1810.11363.

[49] C. Bentéjac, A. Csoögő, and G. Martínez-Muñoz, “A comparative analysis of gradient boosting algorithms,” Artif. Intell. Rev., vol. 54, no. 3, pp. 375–397, 2021.

[50] J. Hancock and T. M. Khoshgoftaar, “Performance of CatBoost and XGBoost in medicare fraud detection,” in Proc. 19th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA), 2020, pp. 572–579.

[51] S. Jhaiveri, I. Khedkar, Y. Kantharia, and S. Jaswal, “Success prediction using random forest, CatBoost, XGBoost and AdaBoost for kickstarter campaigns,” in Proc. 3rd Int. Conf. Comput. Methodol. Commun. (ICCMC), 2019, pp. 1170–1173.

[52] A. A. Ibrahim et al., “Comparison of the CatBoost classifier with other machine learning methods,” Int. J. Adv. Comput. Sci. Appl., vol. 11, no. 11, pp. 738–748, 2020.

[53] D. Zhang, L. Qian, B. Mao, C. Huang, B. Huang, and Y. Si, “A data-driven design for fault detection of wind turbines using random forests and XGBoost,” IEEE Access, vol. 6, pp. 20120–201303, 2018.

[54] E. Zeinulla, K. Bekbayeva, and A. Yazici, “Effective diagnosis of heart disease imposed by incomplete data based on fuzzy random forest,” in Proc. IEEE Int. Conf. Fuzzy Syst. (FUZZ-IEEE), 2020, pp. 1–9.

[55] R. Elwell and R. Polikar, “Incremental learning of concept drift in non-stationary environments,” IEEE Trans. Neural Netw., vol. 22, no. 10, pp. 1517–1531, Oct. 2011.

[56] X.-C. Yin, Y. Huang, and H.-W. Hao, “Dynamic ensemble of ensembles in nonstationary environments,” in Proc. Int. Conf. Neural Inf. Process., 2013, pp. 76–83.

[57] X.-C. Yin, Y. Huang, and H.-W. Hao, “DE2: Dynamic ensemble of ensembles for learning nonstationary data,” Neurocomputing, vol. 165, pp. 14–22, Oct. 2015.

[58] A. Salado and H. Kannan, “Elemental patterns of verification strategies,” Syst. Eng., vol. 22, no. 5, pp. 370–388, 2019.

[59] A. Salado and H. Kannan, “Properties of the utility of verification,” in Proc. IEEE Int. Syst. Eng. Symp. (ISSE), 2018, pp. 1–8.

[60] J. O. Berger. Statistical Decision Theory and Bayesian Analysis. New York, NY, USA: Springer, 2013.

[61] P. Xu and A. Salado, “Modeling correction activities in the context of verification strategies,” Syst. Eng., vol. 25, no. 2, pp. 173–188, 2021.

[62] M. Vomlelová and J. Vomel, “Troubleshooting: NP-hardness and solution methods,” Soft Comput., vol. 7, no. 5, pp. 357–368, 2003.

[63] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Amsterdam, The Netherlands: Elsevier, 2014.

[64] K. Murphy et al., “The Bayes net toolbox for MATLAB,” Comput. Sci. Stat., vol. 33, no. 2, pp. 1024–1034, 2001.