Online Maintenance Prioritization via Monte Carlo Tree Search and Case-based Reasoning

Michael Hoffman
Department of Industrial and
Manufacturing Engineering
Pennsylvania State University
University Park, PA 16802
e-mail: hoffman@psu.edu

Eunhye Song
Department of Industrial and
Manufacturing Engineering
Pennsylvania State University
University Park, PA 16802
e-mail: eus358@psu.edu

Michael Brundage
National Institute of Standards
and Technology
Gaithersburg, MD 20899
e-mail: mpb1@nist.gov

Soundar Kumara
Department of Industrial and
Manufacturing Engineering
Pennsylvania State University
University Park, PA 16802
e-mail: skumara@psu.edu

ABSTRACT

When maintenance resources in a manufacturing system are limited, a challenge arises in determining how to allocate these resources among multiple competing maintenance jobs. This work formulates an online prioritization problem to tackle this challenge using a Markov decision process (MDP) to model the system behavior and Monte Carlo tree search (MCTS) to seek optimal maintenance actions in various states of the system. Further, Case-based Reasoning (CBR) is adopted to retain and reuse search experience gathered from MCTS to reduce the computational effort needed over time and to improve decision-making efficiency. The proposed method results in increased system throughput when compared to existing methods of maintenance prioritization while also reducing the computation time needed to identify optimal maintenance actions as more information is gathered. This is especially beneficial in manufacturing settings where maintenance decisions must be made quickly to minimize the negative performance impact of machine *downtime.*

1 INTRODUCTION

In this work, we examine the problem of online maintenance prioritization in a manufacturing setting with capacity-constrained maintenance resources. Under this setting, periodic maintenance conflicts may occur where the number of machines that require maintenance exceeds the currently available maintenance capacity. The goal is then to determine where to allocate the limited maintenance resources in these instances in order to maximize the expected system performance. This maintenance prioritization problem may arise in any system where maintenance resources are limited, regardless of the maintenance strategy that is in place.

When a machine fails in a manufacturing system, it is unable to continue production until it is restored to a healthy state by a maintenance action. Downtime of a machine results in lost production time and may also force other machines in the system to become starved or blocked [1]. In complex systems with many machines, there can be a large number of alternative maintenance actions to consider. Thoroughly evaluating the resulting performance of each alternative course of action is computationally demanding and may require a substantial amount of time. Meanwhile, as downtime accumulates in the system the negative impact to performance increases. Therefore, selecting proper maintenance actions is a crucial task that should be done as quickly as possible so as to minimize machine downtime.

We formulate the maintenance prioritization problem as a Markov decision process (MDP) and seek the best action using Monte Carlo tree search (MCTS). A search for the optimal decision in the form of which immediate maintenance action to perform is conducted each time a maintenance conflict occurs. Additionally, we retain the information from each search for the best action to learn an effective policy and improve the decision making process over time. As more information is gathered, the reasoner will be able to predict the best action in some states of the system, reducing the rate at which simulation effort is expended.

The following criteria are given by [**?**] to classify a maintenance scheduling problem as “online”: (1) jobs arrive as an input stream over time and scheduling decisions must be made without knowledge of future jobs, (2) duration of jobs are unknown until the job is completed, and (3) machine failure and maintenance intervals are unknown [?]. Since our problem setting of interest meets each of these criteria, we consider the proposed maintenance prioritization method to be online.

The rest of this document is organized as follows: Section 2 contains a review of relevant existing work in both maintenance prioritization and simulation-based planning. Section 3 gives a more detailed description of the problem setting and the assumed behavior of the target system. The methodology is described in Section 4 and experimental results of the proposed method are presented in Section 5. Lastly, conclusions are given in Section 6.

2 BACKGROUND

Maintenance priority management addresses the problem of allocating limited resources among competing maintenance needs. These resources can include labor, time, material and spare parts, or funds. We focus mainly on a limited number of maintenance workers which imposes a limit on the number of maintenance jobs that may be carried out simultaneously.

A review of recent work in maintenance optimization is given by [2] which identifies several references that consider optimization under constrained maintenance capacity. However, in each case there are additional simplifying assumptions such as negligible time to repair or identical machines. Such assumptions are often made for mathematical convenience and are not representative of many real-world manufacturing systems [3]. Maintenance prioritization subject to limited resources is also examined by [4] in a variety of industries, including manufacturing. They identify the following four prioritization methods as the most common among recent literature: analytical hierarchy process, priority criterion, matrix-based priority, and failure mode and effect analysis.

Analytical hierarchy process (AHP) is a method of evaluating alternative decisions using pair-wise comparisons of weighted criteria and sub-criteria [5]. Each criterion is compared against all others and assigned a relative numeric importance. Then each candidate decision is evaluated against the criteria and the alternative with the highest score is selected as the best. The criteria and weights are typically selected by expert opinion. [6, 7] apply AHP in a maintenance setting using production disruption, mean time between failure, mean time to repair, and resource availability as selection criteria. Each maintainable asset is evaluated using these criteria resulting in a fixed priority ranking of assets. This approach does not account for the changing dynamics of a production system, where the criteria score may change depending on the state of the system.

For example, as shown in [8, 9], the production disruption that results from machine downtime will depend on the distribution of buffer contents throughout the system which will change over time.

Similar to AHP, both priority criterion and matrix-based priority use expert opinion to establish relevant criteria and evaluate competing maintenance needs [10]. Several case studies using priority criterion for maintenance are presented by [11]. Priority criterion for both long-term strategic planning and short-term operational decision support is considered. In complex production systems, however, it is not feasible to establish maintenance priorities for all operating states of the system even if expert input is available.

Failure mode and effects analysis (FMEA) aims to identify potential machine failures and their impact on the system [12]. It involves measuring the risk priority number (RPN) of each possible failure which is based on the likelihood of occurrence of the failure, the severity of the failure, and the ability to detect the failure. Generally, maintenance of failure modes with a greater risk of occurrence and severe impact are prioritized over lower-risk failures. Maintenance of a manufacturing system using FMEA is presented by [13]. This approach again relies heavily on expert opinion to evaluate maintenance actions against the RPN criteria. Doing so can be difficult or impossible for complex systems where the consequence of downtime is not easily inferred.

In each method of maintenance priority management discussed so far, priority criteria are developed and evaluated for a specific instance of a system. If the system is modified, the criteria will need to be reevaluated within the context of the new system. This can be costly and time-consuming, especially if inputs from experts and stakeholders are required. Additionally, an expert may struggle to identify optimal maintenance actions in a complex system with a high degree of interactions among components. The system may also encounter rare or unforeseen conditions with which an expert has no experience.

To alleviate the need for expert input when prioritizing maintenance, traditional queueing service disciplines have also been applied to real-time operational maintenance decision making. For instance, several common scheduling rules are compared by [14] including first-in, first-out (FIFO), shortest processing time first (SPTF), longest processing time first (LPTF), and an expert-derived static heuristic. Birnbaum importance is another heuristic metric that prioritizes machines according to their structural importance [15]. When used as a maintenance rule, Birnbaum importance prefers maintaining machines that have a greater likelihood of disrupting the system-level production, as demonstrated in [16, 17]. While these scheduling rules may perform well in some scenarios, they do not consider the evolving state of the system. [14] further proposes a dynamic priority that is found using an evolutionary algorithm for a single state of the system. Another dynamic scheduling priority measure based on the concept of a maintenance opportunity window [8] is proposed by [18]. The objective is to minimize the throughput disruption caused by machine downtime due to maintenance, though no substantial improvement is shown over a static FIFO rule.

While simple to implement, it is unlikely that a static priority will yield the optimal maintenance action in all states of a complex system. Monte Carlo planning algorithms aim to improve upon static rules for the action selection problem in systems with a large state space. They use simulation to evaluate alternative sequences of actions in order to identify the optimal action in the current state. These algorithms typically require a generative model, or simulator, of the target system and strategically sample this model to estimate the expected performance of alternatives. The behavior of the generative model as well as the Markov decision process (MDP) model of the manufacturing system is formalized for our system of interest in Section 3.2.

Several methods exists to find optimal actions for an MDP. Value iteration, policy iteration, and Q-learning are common methods that involve exploring the state space and improving an agent's ability to choose rewarding actions as it gathers more experience [19]. However, the space complexity of each of these algorithms is a function of the system state space size, which limits their applicability to complex settings. To overcome this limitation, Kearns et al. propose a method of seeking optimal actions for large or infinite MDPs using an online search in the current state of the system. Their approach involves constructing a search tree by sampling each possible action a fixed number of times at each level of the tree until a specified depth is reached [20]. While this method provides useful theoretical guarantees on the optimality of the result, it is possible that significant simulation effort could be expended on suboptimal action trajectories. [21] proposed the upper confidence bound for trees (UCT) algorithm to improve the efficiency at which simulation effort is expended during the tree search. UCT offers an action selection criterion that balances the exploitation of promising actions with the exploration of those that have been sampled less often. Monte Carlo tree search (MCTS) combines the UCT selection criteria with progressive tree building to effectively seek actions in large state space MDP settings and is the principal planning algorithm used throughout this work [22].

In [23] we have applied MCTS to the maintenance prioritization problem for a system subject to condition-based maintenance. Each time a maintenance conflict occurs, that is, in each instance where the number of maintenance requests exceeds the currently available maintenance capacity, we apply MCTS to seek the optimal maintenance action. This approach has demonstrated an improvement in system throughput for a variety of system configurations. In this paper, we aim to improve upon this method by retaining and reusing experience that is gathered during each search.

Case-based reasoning (CBR) is a framework for using past experience to understand and solve newly encountered problems and is founded on the idea that similar problems have similar solutions [24–26]. CBR reflects the everyday human reasoning process that is often used when we attempt to solve problems. For example, from [24]:

> When we order a meal in a restaurant, we often base decisions about what might be good on our other experiences in that restaurant and those like it. As we plan our household activities, we remember what worked and didn't work previously, and use that to create our new plans. A childcare provider mediating an argument between two children remembers what worked and didn't work previously in calming such situations, and bases her suggestion on that.

In a maintenance setting, when we encounter a state for which a maintenance decision is needed, we use CBR to identify past states that are sufficiently similar to the current state. Two system states might be considered similar if, for example, they share a similar distribution of buffer contents or if the same set of machines requires maintenance. We then examine the maintenance decisions that were made in those previously encountered similar states and determine if there is an action that can be retrieved and applied in the current state. Over time, as we gather more experience, we can retrieve existing actions more frequently which reduces the computational effort that is spent conducting an online search for the best action. A CBR agent, therefore, acts as a virtual expert that can provide the expert judgement needed in some maintenance priority management methods.

CBR has been applied to some areas of maintenance in past work. For example, in fault diagnosis for manufacturing equipment and identification of the proper corrective action, as in [27–29]. Planning under limited maintenance resources is not considered, however. [30] develops a maintenance decision support system that uses CBR, but focuses on using cases to represent human expert knowledge rather than the results of an autonomous heuristic search.

CBR has also been effectively applied in some reinforcement learning (RL) settings which share a similar formulation as our maintenance planning problem. Many RL problems are also formulated as an MDP with the goal of choosing the best action in each state of the system. For example, [31] uses CBR to estimate the expected reward of actions in an episodic RL task. They address several challenges of using CBR for real-time decision support that are also important considerations in this work, such as storage and management of information in the case base. [32] also examines the real-time decision support problem and proposes a general CBR framework for such settings which we adapt to our maintenance planning problem.

3 PROBLEM DESCRIPTION

In this section, we elaborate on the manufacturing system behavior and the maintenance planning problem formulation.

3.1 System Description

The system studied in this work is a discrete manufacturing system consisting of multiple machines and buffers. While in a working state, each machine will retrieve a part from an upstream buffer (if one is available), process that part for some amount of time, and then place the completed part in a downstream buffer (if there is available space). The system has a single source of incoming unprocessed parts as well as a single sink to collect finished parts that leave the system. The overall system-level production is measured as the number of parts that traverse from source to sink over a specified time frame. The system can be viewed as a directed graph where each node represents either a machine or buffer and edges define possible routings through the system. Several real-world case studies that adopt this model of a manufacturing system are presented by [33].

Each machine is also subject to random failures, at which point the machine can no longer process parts and a corrective maintenance action is needed to restore the machine to a healthy state. It is assumed that failures are immediately observable and that a request for maintenance is generated as soon as a failure occurs.

The set of current maintenance requests forms a virtual “maintenance queue” of machines that are waiting to be repaired. When maintenance resources are limited, periodically situations may arise where the number of machines in the maintenance queue exceeds the available maintenance capacity. That is, at time t for the current maintenance queue $L(t)$ and available maintenance capacity $c_{\text{idle}}(t)$, the system encounters a maintenance conflict if

$$1 \leq c_{\text{idle}}(t) < |L(t)|. \tag{1}$$

In these instances, we must decide where to allocate the limited maintenance resources among the currently pending jobs. When a machine is chosen for maintenance, it is removed from the queue and seizes an available maintenance resource until the job is complete.

In the following we consider only binary state machines (a machine is either working or failed) under a corrective maintenance strategy. This is to emphasize the contributions of the proposed planning methodology, which is easily extendable to any maintenance policy by using the maintenance queue formulation. The proposed method is agnostic to how or why maintenance jobs are placed in the queue so long as the other stated assumptions are met.

3.2 Markov Decision Processes

We model the underlying behavior of the manufacturing system as a Markov decision process (MDP), defined by the tuple $M = \langle S, A, T, R, \gamma \rangle$ where

- S is the state space
- A is the action space and $A(s)$ is the set of actions available in state s
- T is the transition probability function such that

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

- R is the reward function where $R(s, a, s')$ is the immediate reward obtained after transitioning from s to s' as a result of taking action a
- $\gamma \in [0, 1]$ is the discount factor over the time horizon

MDP behavior is defined by a policy π that specifies the action to take in each state, $\pi = \{\pi_t | \pi_t : S \mapsto A, t \geq 0\}$. From an initial state s_0, the expected reward of following policy π at each step is given by

$$Q(s_0, \pi(s_0)) = \sum_{s' \in S} T(s_0, \pi(s_0), s') \cdot (R(s, \pi(s_0), s') + \gamma Q(s', \pi(s'))) \tag{2}$$

where $Q(s, a)$ is the action-value function. It represents the expected discounted return when starting in a state s and following π indefinitely. The optimal policy π^* will maximize the reward in each state, defined as

$$\pi^* = \max_{\pi} \ Q(s, \pi(s)) \ \forall \ s \in S. \tag{3}$$

For our problem and many others, T and R are unknown but a generative model, or simulator, $\mathcal{G}$ is available. When given a state and an action, the generative model is able to sample a resulting future state via $\mathcal{G}(s, a) \rightarrow s'$. The immediate reward is observable upon this transition to s'. This "single step" simulation can be chained together to obtain a trajectory of states and actions that emulates the system behavior over some (potentially infinite) time horizon. The problem then becomes to determine how to best allocate calls to the simulator in order to find a good sequence of actions.

The online prioritization problem is formulated with s_0 representing the current state of the system when Eq. 1 becomes true. We are interested in finding the action a that maximizes the expected return in the current state, $Q(s_0, a)$, given that our only knowledge of the system dynamics comes from $\mathcal{G}$.

The overall objective in selecting maintenance actions is to maximize the rate at which parts are produced by the system over an indefinite time horizon. We therefore model the reward function to represent the number of finished parts that leave the system. For example, $R(s, a, s')$ would be equal to the difference in production in state s' and s. By choosing $\gamma < 1$, we place preference on obtaining rewards earlier instead of deferring rewards.

4 METHODOLOGY

The state space of the system grows exponentially with the number of machines and buffers, which precludes the application of classical dynamic programming algorithms for MDPs such as value iteration and policy iteration. The complexity of these algorithms is a function of the size of the state space, making them intractable for even moderately-sized systems. Instead, we turn to simulation-based decision tree search algorithms, namely Monte Carlo tree search (MCTS), to strategically sample the state space using the generative function $\mathcal{G}$ and evaluate alternative actions.

In addition to using MCTS to seek an action from a given state, we would like to be able to learn from past experience to improve our decision making while reducing the computational effort needed over time. The proposed approach uses MCTS along with Case-based Reasoning (CBR) to retain and reuse the information that is gained over time. This approach is illustrated in Fig. 1, where the problem space consists of system states that are mapped to a point in the solution space representing the results of a search, including the estimated best action for that state. When encountering a new state for which we need to determine the best action, we query the gathered information to determine if we have previously found a good solution for similar states and attempt to adapt that solution to the current state.

A high-level overview of the combined MCTS and CBR approach is as follows:

0. Operate the system until a state is encountered for which we want to find the optimal action.
1. Determine if the case base contains sufficient experience related to the current state. Sufficient experience is the ability to predict the optimal action in the current system state with an acceptable statistical confidence, as will be explained in Section 4.2.1.
2. If sufficient experience exists, retrieve the solution from the existing relevant cases, go to 4.
3. Formulate MCTS to seek a solution.

    (a) Update the case base with experience gained during this search.

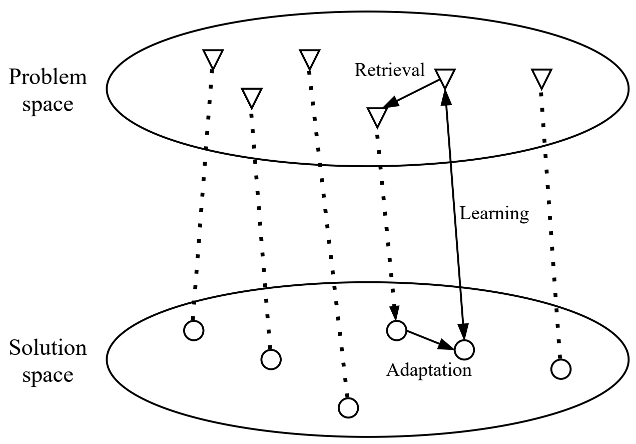

Fig. 1. Experience storage and retrieval.

4. Return the estimated optimal action as the best action as for the current state, go to 0.

This procedure can be run indefinitely for a specified system so long as a simulator $\mathcal{G}$ is available and accurately reflects the system's behavior. The following sections describe the steps of this approach in more detail.

4.1 Monte Carlo Tree Search

The general MCTS algorithm is initialized by creating a root node v_0 representing the current state of the physical system, s_0. At each successive iteration, a leaf node v_ℓ is selected according to a "tree policy." A simulation result is obtained from the state of this node, $s(v_\ell)$, using the "default policy" and then backed up through the tree from v_ℓ to v_0. Once the specified search budget is exhausted, we choose the best estimated action from the current state.

Upper confidence bound for trees (UCT) [21] is the typical node selection criterion for the tree policy of MCTS. According to UCT, at each node v_i the action a should be chosen that maximizes the quantity

$$UCT(v_i, a, C_p) = \hat{Q}_i(s(v_i), a) + 2C_p\sqrt{\frac{\ln \bar{n}}{N_{a,i}^{s(v_i)}}} \tag{4}$$

where $\hat{Q}_i(s(v_i), a)$ is the estimated expected reward from choosing action a in state $s(v_i)$ as defined by Eq. 2, C_p is the exploration constant, $\bar{n}$ is the number of times node v_i has been visited so far, and $N_{a,i}^{s(v_i)}$ is the number of times action a has been chosen from node v_i.

At termination, we choose the action that maximizes Eq. 4 with $C_p = 0$ as the best action. For each candidate action, MCTS also returns the mean reward, reward variance, and number of times that action was sampled from the initial state. These statistics are used during the CBR procedure to conduct a statistical comparison of action rewards.

4.2 Case-based Reasoning

In general, experience is stored in a case base CB which is a set of cases describing problem instances and their associated solutions. Initially, $CB = \varnothing$ and new cases are added over time as new problem instances are solved. As additional experience is gathered, the case base can be used to predict the optimal action in newly encountered states with greater confidence.

There are four main steps in the CBR process when a new problem instance is encountered:

1. Retrieve: Identify the existing cases most relevant to the target problem.
2. Reuse: Adapt a solution from the existing cases to be applied to the target problem.
3. Revise: Adjust the implemented solution based on observation of the results.
4. Retain: Add the new problem-solution pair to the case base as a new case.

We use the numeric system state vector described in [23] for problem descriptions in the case base under a manufacturing setting. To summarize, the state of a manufacturing system is determined by the level of each buffer in the system, the elapsed processing time of each machine that currently has a part in progress, the degradation state of each machine, and the elapsed repair time for machines currently under repair. We encode each of these attributes with numeric values to create a numeric state-space vector. The solutions are stored as the results of each search in the form of the mean, variance, and sample size of each candidate action in the initial state.

Given input as a vector of state variables representing the current system state, the case-based reasoner will return an output in the form of a prediction of the best action for that state. The following section will describe the formulation of this prediction problem in more detail.

4.2.1 Action Label Prediction Task

In this section, we formulate the problem of predicting the best action for a queried state given the set of experience that has been gathered so far. We assume that each state can be represented by a numeric vector of state variables.

This task can be viewed as a classification problem where our goal is to classify each state according to its optimal action. The action space $A = \{a_1, a_2, \ldots, a_m\}$ is the set of possible labels and states that share the same optimal action, therefore, belong to the same class. In order to train a classifier, we first need to gather experience by applying MCTS in various states to seek the best action. Using a traditional statistical classification approach, the best action as determined by MCTS is used as a label for the initial state of the search and this state-solution pair becomes a case in CB and one instance in the classifier training set. Each training instance, or case, for the classification problem is defined as

$$c^i = (s_i, a_i) \tag{5}$$

where s_i is the vector of state variables defining a particular state and a_i is the best action label assigned to that state. Once a sufficiently large training set is established, a classifier can be trained on these instances and used to generate predictions for newly observed states.

A significant limitation of this approach is that it does not account for the uncertainty of MCTS. In some cases, MCTS may be unable to distinguish which action is best if multiple actions from the current state yield a similar estimated reward. Conversely, MCTS may very easily distinguish an action that is clearly better than the others, so our predictions should ideally reflect these cases as well. To account for this uncertainty, we can instead aim to predict the likelihood that any particular action is optimal in the current state. This requires assigning some probabilistic measure to the labels for each state. The cases are then represented by

$$c^i = \big(s_i, \langle a_1, q_1 \rangle, \ldots \langle a_m, q_m \rangle\big) \tag{6}$$

where $q_i \in [0,1]$ is a measure of our belief that action label a_i is optimal. Since we are now aiming to predict the continuous q_i values, we are faced with a regression problem. This problem setting is referred to by [34] as "learning classification with auxiliary probabilistic information."

Predicting q_i for each action a_i at a queried state instance requires $|A| = m$ regressors. To perform the regression, we use instance-based learning methods which avoids generalizing the training data until a query is made to the predictor. Instance-based learning algorithms (also sometimes called "lazy learning") do not require a training period which is especially beneficial for case-based reasoning since we do not need to explicitly retrain each regressor when the experience stored in the case base changes [35]. In this work, we use k-nearest neighbor regression, which fits these criteria and has been successful in many previous applications of CBR.

Once the predicted action label confidence $\hat{q}_i$ is calculated for each action of a queried state, we choose that with the greatest confidence exceeding some specified threshold ζ as the predicted optimal action. If no predicted action label confidence exceeds this threshold, we formulate and solve MCTS to seek the best action. The experience obtained from this MCTS instance is then potentially added to the case base to aid future predictions.

4.2.2 Measuring Action Label Confidence

For a set of m alternative actions, we want to find the optimal action a_i^* where

$$a_i^* := \arg\max_{1 \le i \le m} \mathrm{E}[X_i] \tag{7}$$

where X_i is the return of taking action a_i in the current state. However, with the uncertainty of MCTS we instead aim to determine the probability that an action a_i is optimal given the observations gathered from a search. This probability will serve as the auxiliary probabilistic information for the action labels of each state in the case base.

We introduce a stochastic process η_i to model the uncertainty regarding $E[X_i]$. By adopting a normal-normal model for η_i as described by [36], we assume

$$X_i \sim N(\eta_i, \lambda_i^2),\ \eta_i \sim N(\mu_i, \sigma_i^2) \tag{8}$$

where λ_i^2 is the simulation error variance, and μ_i and σ_i^2 are the mean and variance of the prior distribution of η_i, respectively.

Each instance of MCTS returns a set of statistics from sampling each candidate action a_i from the initial state including mean reward $\bar{X}_i$, sample size n_i, and sample variance S_i^2. We choose the noninformative prior $\sigma_i^2 = \infty$ and use S_i^2 as a plug-in estimate of λ_i^2. We then update the posterior mean and variance of η_i in

$$\mu_i = \bar{X}_i,\ \sigma_i^2 = S_i^2/n_i. \tag{9}$$

The probability that an action a_i is optimal in the current state is therefore given by

$$\begin{aligned} q_i &= \mathsf{Pr}(\eta_i \geq \max_{j\neq i} \eta_j) \\ &= \int \mathsf{Pr}(x \geq \max_{j\neq i} \eta_j) \cdot f_i(x)\ dx \\ &= \int \Big[\prod_{j\neq i} \mathsf{Pr}(\eta_j \leq x)\Big] \cdot f_i(x)\ dx \end{aligned} \tag{10}$$

where $f_i(\cdot)$ is the probability density function of the distribution $N(\mu_i, \sigma_i^2)$ as defined in Eq. 8. We then set q_i to indicate our confidence that action a_i is optimal in the current state.

Additionally, we may be interested in finding the probability that the estimated reward of an action is within δ of the best alternative for some user-specified $\delta > 0$. We refer to δ as the optimality indifference parameter reflecting that the decision maker is indifferent about the optimality gap as long as it is within δ. This probability is given by

$$\begin{aligned} q_i &= \mathsf{Pr}\big(|\eta_i - \max_{1\leq j\leq m} \eta_j| \leq \delta\big) \\ &= \mathsf{Pr}\big(\eta_i + \delta \geq \max_{1\leq j\leq m} \eta_j\big) \\ &= \mathsf{Pr}\big(i \text{ is best}\big) + \mathsf{Pr}\big(\eta_i + \delta \geq \max_{1\leq j\leq m} \eta_j,\ i \text{ is not best}\big) \end{aligned} \tag{11}$$

The term Pr(i is best) is given in Eq. 10. The second term of Eq. 11 is

$$\begin{aligned}\mathsf{Pr}\big(\eta_i \le \max_{j\neq i} \eta_j \le \eta_i + \delta\big) &= \int \mathsf{Pr}\big(x \le \max_{j\neq i} \eta_j \le x + \delta\big) \cdot f_i(x)\, dx \\ &= \int \Big[\sum_{\ell\neq i} b_\ell\Big] \cdot f_i(x)\, dx, \end{aligned} \tag{12}$$

where

$$\begin{aligned} b_\ell &= \mathsf{Pr}\big(x \le \eta_\ell \le x+\delta,\ \eta_\ell \ge \max_{j\neq i, j\neq \ell} \eta_j\big) \\ &= \int_x^{x+\delta} \Bigg[\prod_{j\neq i,\ j\neq \ell} \mathsf{Pr}(\eta_j \le y)\Bigg] \cdot f_\ell(y)\, dy. \end{aligned} \tag{13}$$

We use Eq. 11 to calculate q_i for each alternative action whenever a new case is added to the case base or the statistics of an existing case are updated.

4.2.3 CBR Procedure

In this section, we describe each step of the CBR procedure as it applies to our problem.

Retrieval The retrieval step involves identifying the set of cases that are most similar to the queried case. We use the Minkowski distance metric, defined as

$$D(\mathbf{x_i}, \mathbf{x_j}) = \Big(\sum_{k=1}^{n} |x_{ik} - x_{jk}|^p\Big)^{1/p} \tag{14}$$

for state vectors of variables scaled to the interval $[0, 1]$. We consider $p = 1$ (Manhattan distance) and $p = 2$ (Euclidean distance) in our experiments. A lesser distance between two states indicates a higher degree of similarity.

Reuse Using a probabilistic classification approach, we attempt to reuse experience in the case base by predicting the likelihood that each available action is optimal in a particular state by training regression models using training examples of the form given by Eq. 6.

For each action and its associated regressor, we predict the action label confidence for the queried state. If the predicted confidence for action a_i is greater than or equal to the specified retrieval threshold ζ, then this action is eligible for retrieval in the current state. If there is more than one action whose predicted confidence exceeds the threshold, we choose the action with the highest predicted value.

We use k-nearest neighbor (kNN) regression to predict the confidence for each action in the queried state. Under this method, we identify the k cases in CB with the minimum distance to the queried case according to Eq. 14. The predicted confidence for action a_i is then the average value q_i among the k neighbors. We also consider the distance-weighted average where neighbors closer to the queried point have a greater influence on the predicted value than those further away.

Revision Once a solution for the queried state is identified, we may choose to revise that solution based on the observed outcome of implementing that solution. Necessary revisions are applied to the solution before storing this new problem-solution pair in the case base.

In our setting, however, it is difficult to retroactively determine if good (or poor) system performance was due to the action selected or the random behavior of the system. Newly encountered cases for which we retrieve an existing solution are therefore not stored in the case base. Experience is stored only if it is obtained as a result of MCTS.

Retention After each instance of MCTS, if the initial state is represented in the case base, we update the case with the results from the new search. These include the mean reward, variance, and sample size of each candidate action in that state. Each statistic is updated incrementally, so that the statistics in the case base reflect the results of every search conducted from that state. As the number of samples for each action increases, we become more confident in the estimation of that action's reward.

If the initial state of an MCTS instance is not already in the case base, we create a new case from the initial state vector and the search statistics and add it to CB. Limiting the number of cases in CB is necessary to limit the time needed to query the case base and to avoid exceeding memory constraints. Once the case base exceeds its maximum specified size, we must determine which cases to keep and which ones to discard.

Three factors are identified by [31] to determine the overall “value” of each case stored in the case base:

- the age of the case,
- the density of cases in the neighborhood in which the case resides, and
- the accuracy of predictions generated by the case.

Given that we are interested in studying a system in its steady state over an indefinite time horizon, the age of a case is less relevant than the other criteria. We therefore only consider the case neighborhood score and the regress error score when evaluating each case.

The first term, the case neighborhood score, determines the density of cases surrounding c^i and is given by

$$S_n(c^i) = \varphi(c^i) \cdot \sigma(c^i) \tag{15}$$

where

$$\varphi(c^i) = \frac{1}{k} \sum_{c^j \in NN_k(c^i)} sim(c^i, c^j), \tag{16}$$

$$\sigma(c^i) = \frac{1}{k} \sum_{c^j \in NN_k(c^i)} 1 - \kappa(c^i, c^j), \tag{17}$$

$NN_k(c^i)$ is the set of k nearest neighbors of c^i, and $\kappa(c^i, c^j)$ is the proportion of actions in $A(s_i) \cap A(s_j)$ for which c^i and c^j agree on whether or not confidence in the action should be above or below the retrieval threshold ζ. If $A(s_i) \cap A(s_j) = \varnothing$ then $\kappa(c_i, c_j) = 0$. A greater value of $\varphi(c^i)$ indicates that a case is very similar to its neighbors on average, while a greater $\sigma_v(c^i)$ indicates greater rate of disagreement (and therefore greater variability) among cases in the neighborhood.

The second term, the regress error score, is given by

$$S_e(c^i) = \sum_{c^j \in NN_k(c^i)} sim(c^i, c^j) \cdot (1 - \kappa(c^j, \hat{c}^j)) \tag{18}$$

where $\hat{c}_v^j$ is the prediction of c_v^j using $CB \setminus c^j$. A lower value of $S_e(c^i)$ indicates that c^i is more useful for predicting the state value of its neighbors.

These components are then combined into a single heuristic:

$$S(c^i) = S_n(c^i) + S_e(c^i). \tag{19}$$

A higher score relative to other cases indicates that a case is "worse" with regards to each of these criteria. We remove the case with the greatest score until the number of cases is within the limit.

5 RESULTS

To demonstrate the proposed methodology, we examine the performance of several maintainers, or "agents", by evaluating 1) the average production (reward) rate obtained by the agent and 2) the simulation effort that is expended over time. If the CBR approach is effective, an agent should be able to maintain a relatively constant reward rate while reducing its simulation effort. This would indicate that an agent is effectively learning from its experience as it interacts with the environment. Throughout our experiments, we use the Python discrete event simulation package Simantha to model and simulate the system of interest [**?**]. This package was developed by the authors to support maintenance optimization via simulation for complex manufacturing systems.

We apply the following procedure to evaluate each CBR maintainer:

1. Simulate each system while applying MCTS at each decision point to gather initial experience.
2. Tune CBR retrieval regressor parameters using the initial experience.
3. Continue simulating the system using the proposed MCTS+CBR method.

The structure of the system studied in this example is shown in Figure 2. We set the maintenance capacity in each experiment to 1. This system includes a complex arrangement of machines for which effective maintenance planning is not easily derived by inspection or achieved by analytical methods.

To model machine degradation we use the notion of Bernoulli machines as described by [33]. Under this reliability model, each machine has some probability of failure at the beginning of each

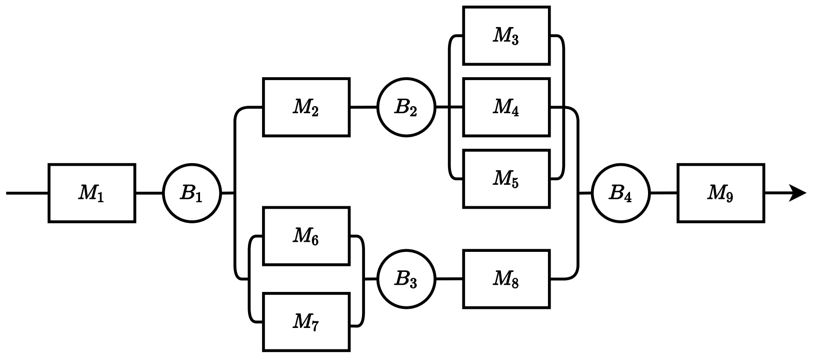

Fig. 2. Nine-machine complex production line.

discrete time step. The time to failure for each machine is therefore geometrically distributed. If a failure occurs, then that machine can no longer function until it receives maintenance that restores it to a healthy state. We can therefore represent the health state of each machine with a binary state variable indicating whether or not the machine is failed. Similarly, the time to repair is geometrically distributed and the repair state of a machine is also a binary variable indicating whether or not the machine is currently under repair.

The proposed method may be used with any discrete distribution of cycle time, time to failure, and time to repair as long as the state variable is chosen such that it is adequate for representing a machine's condition. Choosing a geometric distribution for each of these quantities and the use of binary state variables allows us to reduce the size of the state space. MCTS is therefore able to converge to an optimal solution more quickly which is convenient for our illustrative example.

The production status of each machine must also be captured in the state representation. Assuming geometrically distributed cycle times for each machine again allows us to use binary state variables to indicate whether a machine has a part, and also whether or not that part has finished processing. Considering each of these machine state variables, there are five unique machine states as summarized in Table 1. Our method also allows for a more complex system state space, such as that examined by [23] where state variables are integer-valued rather than binary. If machines do not abide by the Benoulli reliability model, for example, the degradation state of each machine can be represented by a variable indicating the time since the last repair.

Table 2 shows the cycle time distribution for each machine that is used throughout the following example. Each buffer has a maximum capacity of 5 units. The time to repair distribution for machines M_1, M_6, M_7, M_8, and M_9 is $Geom(1/10)$ minutes. For machines M_2, M_3, M_4, M_5,

Table 1. Machine state summary.

Has part	Processing	Failed	Under repair	State Description
0	0	0	0	Starved
1	1	0	0	Processing
1	0	0	0	Blocked
0	0	1	0	Awaiting repair
0	0	1	1	Under repair

Table 2. Machine cycle time distribution in minutes.

Machine	Cycle time
M_1, M_9	$Geom(1/5)$
M_2	$Geom(1/30)$
M_3, M_4, M_5	$Geom(1/90)$
M_6, M_7	$Geom(1/20)$
M_8	$Geom(1/10)$

the time to repair distribution is $Geom(1/20)$ minutes. The time to failure for each machine is distributed $Geom(1/100)$ minutes.

We evaluate FIFO, Birnbaum importance (BI), and the expert heuristic (EH) suggested by [14] as static heuristics for maintenance prioritization in this system. The Birnbaum importance measure for each machine in this arrangement is given in Table 3 where a greater value indicates a higher priority for maintenance. We defer to [15] for details of this calculation and resolve ties using FIFO.

The expert heuristic gives the following list of rules that are used to determine which machine should be prioritized for maintenance:

1. A machine with a longer cycle time should be prioritized over a machine with a shorter cycle time.
2. Machines in a serial section of the line should be prioritized over machines in a parallel section.
3. A machine in a parallel station containing a fewer number of machines should be prioritized over machines in parallel stations with a greater number of machines.

Table 3. Static Birnbaum importance (BI) and expert heuristic (EH) priorities for the nine-machine line where a higher priority value indicates a machine should be repaired earlier.

	Priority	
Machine	BI	EH
M_1, M_9	5	1
M_2	2	4
M_3, M_4, M_5	1	5
M_6, M_7	3	3
M_8	4	2

To apply this heuristic, we first use rule (1) to find the machine in the maintenance queue with the greatest average cycle time. If there is a tie, we then consider rule (2), and finally rule (3). If a tie remains after applying each rule, we defer to FIFO. The priorities of each machine under this method are given in Table 3, where again a greater value indicates a higher priority. Lastly, we also include a random selection policy for reference.

We consider two factors when creating maintainers using the proposed MCTS+CBR method: the retrieval threshold ζ and the maximum size of the case base. We use three levels of ζ in our experiments, 0.5, 0.9, and 0.95. A smaller ζ increases the likelihood that we will be able to retrieve an action from the case base, although we run the risk of retrieving an action that is suboptimal. On the other hand, a greater ζ will force the reasoner to be more selective in the actions it retrieves. We also consider the case with no limit on the maximum case base size as well as a limit of 100 cases. In our experiments, instances with a limited case base size are denoted kNN$_{100}$. Three levels of ζ and two levels of case base capacity results in six total maintainers that we evaluate using our approach. For each case, we set the optimality indifference (δ in Eq. 11) to be 10% of the maximum average observed reward. The confidence label of each action, therefore, represents the probability that an action is within 10% of the best given the observations so far.

Lastly, we also evaluate a maintainer that resolves every maintenance conflict using MCTS and does not retain any experience from the search. For each instance of MCTS, including the searches conducted by the CBR maintainers, we use a search budget of 200 iterations, discount factor $\gamma = 0.95$, and exploration constant $C_p = 10$. These settings performed well throughout our experimentation.

5.1 CBR Maintainer Tuning

We simulate for an initial period of 12 weeks to obtain experience that is used to tune the regression parameters of the CBR maintainers. After 12 weeks, 3,268 unique states were encountered where MCTS was conducted to determine a maintenance action. When tuning the parameters of the action label confidence regression models, we aim to maximize the probabilistic classification accuracy of each model. For a given set of case instances with known action confidence levels, the probabilistic classification accuracy is the proportion of predictions that are correctly above or below the retrieval threshold ζ.

Since we have nine possible maintenance actions corresponding to the nine machines in the system, we tune nine regressors independently so that each predict the likelihood of a particular action is optimal in each state. We perform a grid search of parameters for the kNN regression model and use K-fold validation with $K = 5$ to find the best parameters. Tables 4, 5, and 6 show the resulting best parameters for $\zeta = 0.50$, 0.90, and 0.95, respectively. In each of these tables, the parameter k is the number of neighbors to consider for kNN regression and p is the Minkowski distance metric parameter. $p = 1$ is equivalent to the Manhattan distance while $p = 2$ is Euclidean distance.

5.2 Maintainer Comparison Results

The system is simulated under each maintainer for a period of one week for 48 independent replications. The resulting average throughput for each of the static heuristic maintenance priority rules is shown in Fig. 3. The Birnbaum importance heuristic yields the greatest average throughput with 1.0225 parts per hour.

Fig. 4 compares the throughput of each of the online prioritization maintainers. The average throughput of the best static heuristic maintainer is also included for reference. In each case, online prioritization via MCTS and CBR outperforms the static priority scheduling rules. The average throughput of each maintainer is given in Table 7.

We also consider the rate at which simulation effort is expended by examining the frequency at which MCTS is conducted over time as shown in Fig. 5. The vertical axis gives the average

Table 4. kNN regressor tuning results for $\zeta = 0.50$. Overall accuracy: 76.21%.

	Training	Best model			
	instances	k	p	weights	accuracy
M_1	1119	33	1	distance	76.85%
M_2	1357	25	2	distance	74.73%
M_3	1411	33	2	distance	74.34%
M_4	1358	25	1	distance	74.96%
M_5	1413	29	1	distance	75.31%
M_6	1148	13	1	distance	74.82%
M_7	1103	33	1	distance	72.26%
M_8	637	5	1	uniform	87.28%
M_9	628	9	2	uniform	85.51%

Table 5. kNN regressor tuning results for $\zeta = 0.90$. Overall accuracy: 74.99%.

	Training	Best model			
	instances	k	p	weights	accuracy
M_1	1119	5	2	distance	66.93%
M_2	1357	5	1	distance	78.78%
M_3	1411	9	2	distance	79.73%
M_4	1358	9	2	distance	80.48%
M_5	1413	9	1	uniform	81.31%
M_6	1148	5	2	distance	67.77%
M_7	1103	5	1	distance	69.45%
M_8	637	1	1	uniform	71.27%
M_9	628	1	2	uniform	71.18%

proportion of maintenance actions that are retrieved from the case base at the occurrence of a maintenance conflict. For each CBR maintainer, the rate at which we are able to retrieve actions increases as we gather more experience. When the threshold for retrieval ζ is lower, we retrieve actions more often. Each CBR maintainer provides a relatively constant throughput improvement over the baseline static heuristic policies, indicating that there is no loss in performance when

Table 6. kNN regressor tuning results for $\zeta = 0.95$. Overall accuracy: 80.01%.

	Training	Best model			
	instances	k	p	weights	accuracy
M_1	1119	5	2	distance	66.93%
M_2	1357	5	1	distance	78.78%
M_3	1411	9	2	distance	79.73%
M_4	1358	9	2	distance	80.48%
M_5	1413	9	1	uniform	81.31%
M_6	1148	5	2	distance	67.77%
M_7	1103	5	1	distance	69.45%
M_8	637	1	1	uniform	71.27%
M_9	628	1	2	uniform	71.18%

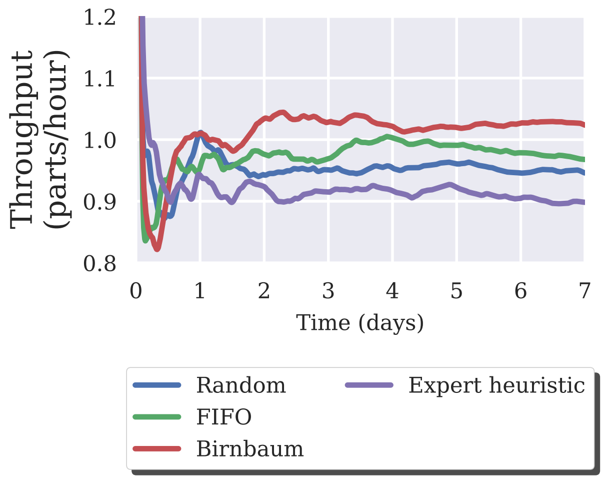

Fig. 3. Complex system throughput comparison for static scheduling rules.

reusing solutions from their experience.

The reduction in simulation effort is also demonstrated in Fig. 6, which shows the average duration of time in seconds needed to make a decision when a maintenance conflict occurs. Since the computational effort needed to retrieve an existing action from the case base is much lower than that needed to conduct MCTS, the maintainers with a higher proportion of retrieved actions are able to make decisions more quickly on average. This is particularly valuable in manufacturing settings where critical maintenance decisions must be made as soon as possible. [**?**] identifies this

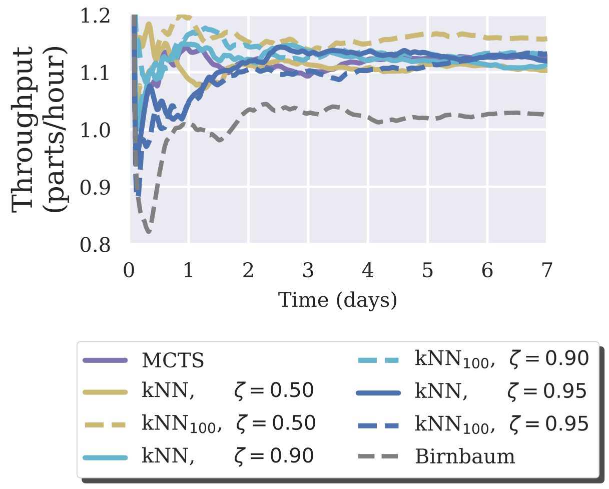

Fig. 4. Complex system throughput comparison for CBR scheduling.

Table 7. Maintainer throughput comparison results.

Maintainer	ζ	Throughput (parts/hour) mean	Throughput (parts/hour) se (%)	Retrieval rate
Random	-	0.9455	1.6611	-
FIFO	-	0.9667	1.9717	-
BI	-	1.0225	1.7977	-
EH	-	0.8977	1.9370	-
MCTS	-	1.1266	1.1145	0.0000
kNN	0.50	1.1031	1.0497	0.1942
kNN_{100}	0.50	1.1584	1.1568	0.2017
kNN	0.90	1.1132	1.3950	0.1063
kNN_{100}	0.90	1.1334	1.3738	0.0587
kNN	0.95	1.1192	1.1574	0.0614
kNN_{100}	0.95	1.1323	1.4562	0.0442

time between machine failure and the start of its repair, or “time to dispatch”, as a key performance indicator (KPI) in maintenance. Reducing this time by making maintenance decisions more quickly can reduce the overall downtime of a machine and increase the productivity of the system.

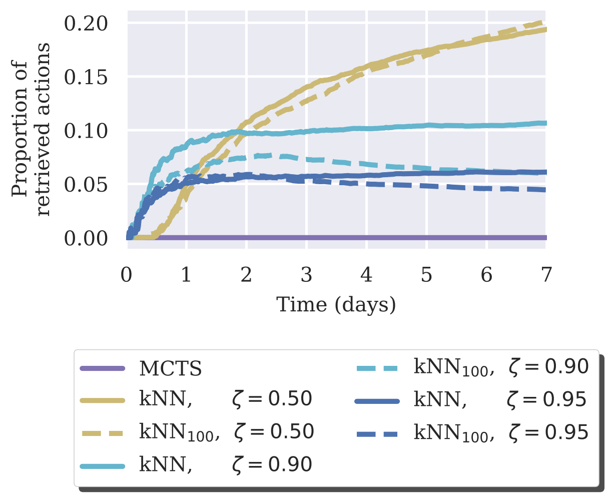

Fig. 5. CBR retrieval rate results.

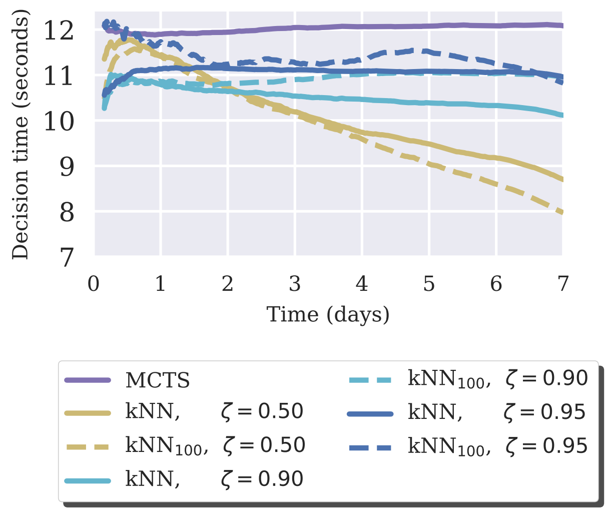

Fig. 6. CBR decision time results.

6 CONCLUSIONS

In this work, we have demonstrated a method of online maintenance prioritization in a dynamic manufacturing setting by using strategic sampling of a generative simulation model via MCTS. Further, we have used CBR to retain and reuse search experience to infer optimal maintenance actions in cases where the current state is similar to those previously encountered. The proposed methods offer improved system-wide performance compared to static heuristic scheduling rules without the need for expert judgement.

A natural extension of this work is to apply the proposed method of maintenance prioritization to more advanced maintenance strategies such as condition-based or predictive maintenance. This would mainly affect the system state representation since we may no longer be able to represent the health state of a machine as a binary variable. Under condition-based maintenance, for example, a machine can be in several possible states of degradation beyond "healthy" or "failed" as we assume in this work. Considering these additional health states will greatly increase the size of the state space and present additional computational challenges that will be addressed in future research.

Future work also includes examining alternative state similarity metrics beyond Minkowski distance. The chosen metric has a significant impact on nearest-neighbor methods, and a learned distance metric may result in better performance of the CBR model. Additionally, an active learning approach to case base management may further improve CBR performance. Such an approach would involve identifying prototypical cases whose best action estimates would provide the most useful information to surrounding states. A search for the best action in prototype states can also be conducted offline, so that the case-based reasoner does not have to wait for a maintenance conflict to occur before gathering additional useful experience.

ACKNOWLEDGEMENTS

Soundar Kumara would like to acknowledge the Clean Energy Smart Manufacturing Innovation Institute (CESMII) for their support of this work. Any opinions, findings, and conclusions or recommendations expressed in this dissertation are the authors' own and do not necessarily reflect the views of CESMII.

REFERENCES

[1] Chang, H. S., Fu, M. C., Hu, J., and Marcus, S. I., 2005. "An adaptive sampling algorithm for solving markov decision processes". *Operations Research,* **53**(1), pp. 126–139.

[2] Keizer, M. C. O., Flapper, S. D. P., and Teunter, R. H., 2017. "Condition-based maintenance policies for systems with multiple dependent components: A review". *European Journal of Operational Research,* **261**(2), pp. 405–420.

[3] Alrabghi, A., and Tiwari, A., 2015. "State of the art in simulation-based optimisation for maintenance systems". *Computers & Industrial Engineering,* **82**, pp. 167–182.

[4] Chong, A. K. W., Mohammed, A. H., Abdullah, M. N., and Rahman, M. S. A., 2019. "Maintenance prioritization–a review on factors and methods". *Journal of Facilities Management*.

[5] Saaty, T. L., 2008. "Decision making with the analytic hierarchy process". *International journal of services sciences,* **1**(1), pp. 83–98.

[6] Sharma, S., Sisodia, A., et al., 2016. "Prioritization of tools in joint production–maintenance environment of auto component manufacturer using ahp–fuzzy–topsis". *Intelligent Industrial Systems,* **2**(1), pp. 73–84.

[7] Khanlari, A., Mohammadi, K., and Sohrabi, B., 2008. "Prioritizing equipments for preventive maintenance (pm) activities using fuzzy rules". *Computers & Industrial Engineering,* **54**(2), pp. 169–184.

[8] Chang, Q., Xiao, G., Biller, S., and Li, L., 2013. "Energy saving opportunity analysis of automotive serial production systems". *IEEE Transactions on Automation Science and Engineering,* **10**(2), pp. 334–342.

[9] Gu, X., Jin, X., and Ni, J., 2015. "Prediction of passive maintenance opportunity windows on bottleneck machines in complex manufacturing systems". *Journal of Manufacturing Science and Engineering,* **137**(3).

[10] Dekker, R., 1995. "Integrating optimisation, priority setting, planning and combining of maintenance activities". *European Journal of Operational Research,* **82**(2), pp. 225–240.

[11] Dekker, R., and Scarf, P. A., 1998. "On the impact of optimisation models in maintenance decision making: the state of the art". *Reliability Engineering & System Safety,* **60**(2), pp. 111–119.

[12] Teng, S.-H. G., and Ho, S.-Y. M., 1996. "Failure mode and effects analysis". *International journal of quality & reliability management*.

[13] Ding, S.-H., Kamaruddin, S., and Azid, I. A., 2014. "Maintenance policy selection model–a case study in the palm oil industry". *Journal of Manufacturing Technology Management*.

[14] Yang, Z., Chang, Q., Djurdjanovic, D., Ni, J., and Lee, J., 2007. "Maintenance priority assignment utilizing on-line production information".

[15] Birnbaum, Z. W., 1968. On the importance of different components in a multicomponent

system. Tech. rep., Washington Univ Seattle Lab of Statistical Research.

[16] Nguyen, K.-A., Do, P., and Grall, A., 2014. “Condition-based maintenance for multi-component systems using importance measure and predictive information”. *International Journal of Systems Science: Operations & Logistics,* **1**(4), pp. 228–245.

[17] Si, S., Liu, M., Jiang, Z., Jin, T., and Cai, Z., 2019. “System reliability allocation and optimization based on generalized birnbaum importance measure”. *IEEE Transactions on Reliability,* **68**(3), pp. 831–843.

[18] Hoffman, M., Song, E., Brundage, M., and Kumara, S., 2018. “Condition-based maintenance policy optimization using genetic algorithms and gaussian markov improvement algorithm”. Vol. 10.

[19] Sutton, R. S., and Barto, A. G., 2018. *Reinforcement learning: An introduction*. MIT press.

[20] Kearns, M., Mansour, Y., and Ng, A. Y., 2002. “A sparse sampling algorithm for near-optimal planning in large markov decision processes”. *Machine learning,* **49**(2), pp. 193–208.

[21] Kocsis, L., and Szepesvári, C., 2006. “Bandit based monte-carlo planning”. In European conference on machine learning, Springer, pp. 282–293.

[22] Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S., 2012. “A survey of monte carlo tree search methods”. *IEEE Transactions on Computational Intelligence and AI in games,* **4**(1), pp. 1–43.

[23] Hoffman, M., Song, E., Brundage, M., and Kumara, S., 2021. “Online improvement of condition-based maintenance policy via monte carlo tree search”. *IEEE Transactions on Automation Science and Engineering* (Accepted).

[24] Kolodner, J. L., 1992. “An introduction to case-based reasoning”. *Artificial intelligence review,* **6**(1), pp. 3–34.

[25] Aamodt, A., and Plaza, E., 1994. “Case-based reasoning: Foundational issues, methodological variations, and system approaches”. *AI Communications,* **7**(1), pp. 39–59.

[26] Bergmann, R., and Wilke, W., 1996. “On the role of abstraction in case-based reasoning”. In European Workshop on Advances in Case-Based Reasoning, Springer, pp. 28–43.

[27] Bengtsson, M., Olsson, E., Funk, P., and Jackson, M., 2004. “Design of condition based maintenance system—a case study using sound analysis and case-based reasoning”.

[28] Tsai, Y., 2009. “Applying a case-based reasoning method for fault diagnosis during maintenance”. *Proceedings of the institution of mechanical engineers, part C: journal of mechanical engineering science,* **223**(10), pp. 2431–2441.

[29] Wan, S., Li, D., Gao, J., and Li, J., 2019. “A knowledge based machine tool maintenance planning system using case-based reasoning techniques”. *Robotics and Computer-Integrated Manufacturing,* **58**, pp. 80–96.

[30] Yu, R., Iung, B., and Panetto, H., 2003. “A multi-agents based e-maintenance system with case-based reasoning decision support”. *Engineering applications of artificial intelligence,* **16**(4), pp. 321–333.

[31] Gabel, T., and Riedmiller, M., 2005. “Cbr for state value function approximation in reinforcement learning”. In International Conference on Case-Based Reasoning, Springer, pp. 206–221.

[32] Varshavskii, P., and Eremeev, A., 2010. “Modeling of case-based reasoning in intelligent decision support systems”. *Scientific and Technical Information Processing,* **37**(5), pp. 336–345.

[33] Li, J., and Meerkov, S. M., 2008. *Production systems engineering*. Springer Science & Business Media.

[34] Nguyen, Q., Valizadegan, H., and Hauskrecht, M., 2011. “Learning classification with auxiliary probabilistic information”. In 2011 IEEE 11th International Conference on Data Mining, IEEE, pp. 477–486.

[35] Aha, D. W., 2013. *Lazy Learning*. Springer Science & Business Media.

[36] Kim, K.-K., Taeho, K., and Song, E., 2021. “Selection of the most probable best under input uncertainty”. In 2021 Winter Simulation Conference (Submitted).

LIST OF FIGURES

LIST OF TABLES