The Pennsylvania State University

The Graduate School

ONLINE MAINTENANCE PRIORITIZATION VIA MONTE CARLO TREE SEARCH AND CASE-BASED REASONING IN COMPLEX MANUFACTURING SYSTEMS

A Dissertation in

Industrial Engineering and Operations Research

by

Michael L. Hoffman

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

August 2021

The dissertation of Michael L. Hoffman was reviewed and approved by the following:

Soundar Kumara
Allen E. Pearce and Allen M. Pearce Professor of Industrial Engineering
Dissertation Co-Adviser, Co-Chair of Committee

Eunhye Song
Harold and Inge Marcus Early Career Assistant Professor
Dissertation Co-Adviser, Co-Chair of Committee

Daniel Finke
Assistant Research Professor, Applied Research Laboratory

Russell Barton
Distinguished Professor of Supply Chain and Information Systems

Michael Brundage
Industrial Engineer, National Institute of Standards and Technology
Special Member

Steven Landry
Department Head of the Harold and Inge Marcus Department of Industrial and Manufacturing Engineering

Abstract

Maintenance serves a critical role in manufacturing systems by ensuring that machines and other assets remain in a productive working condition. The primary objective of maintenance optimization is to determine when to conduct maintenance and which machines should be maintained. Recent advances in industrial maintenance have sought to use online information obtained from sources such as machine sensors and manufacturing execution system software to provide real-time decision support. Such predictive maintenance strategies combine abundant online manufacturing data with techniques in simulation, planning, and artificial intelligence to make effective maintenance decisions and support the overall performance of the system.

In this dissertation we examine several challenges associated with adopting real-time maintenance decision support in complex manufacturing systems. One such challenge is that of modeling complex machine configurations that are often found in modern manufacturing systems. It can be difficult or impossible to model the behavior of these systems analytically without imposing unrealistic simplifying assumptions. One of the goals of this work is therefore to propose a method of maintenance optimization and planning that is generalizable to arbitrarily configured systems. We also introduce a discrete-event simulation package that has been developed as a part of this work and is capable of modeling these systems of interest. Additionally, real-world manufacturing systems are typically subject to constraints on available maintenance resources which limits the number of maintenance jobs that may be conducted simultaneously. In these settings, the maintenance planner must determine how to prioritize competing maintenance activities and allocate these limited resources throughout the system.

This work addresses these challenges by proposing a simulation-based maintenance optimization and planning approach to seek an optimal maintenance policy and prioritize maintenance in complex systems. We formulate condition-based maintenance policy optimization as a discrete optimization via simulation problem and seek a solution using the Gaussian Markov Improvement Algorithm and a genetic algorithm. The result is a degradation threshold for each machine in the system that determines when a machine should request maintenance. To overcome the problem of capacity-constrained maintenance resources, we

first propose a dynamic priority scheduling heuristic that aims to minimize throughput disruption due to downtime for maintenance. We then improve upon this scheduling heuristic by employing a reinforcement learning approach to seek the best maintenance action in each state of the system. We use Monte Carlo tree search to progressively build a search tree within the system state space and evaluate alternative sequences of actions in order to find that which maximizes the expected reward.

We demonstrate that our proposed method of online prioritization results in improved system-level performance when compared to commonly used maintenance prioritization methods. Furthermore, we apply a case-based reasoning framework to retain and reuse relevant experience that improves the decision-making efficiency over time. In addition to improved system productivity, the proposed approach results in reduced time needed to identify optimal maintenance actions which is particularly important when critical maintenance decisions must be made quickly.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to firstly acknowledge the National Institute of Standards and Technology (NIST) for their support of this work through the Graduate Measurement Science and Engineering Fellowship as well as the support provided by the Graduate Fellowships for Science, Technology, Engineering, and Mathematics Diversity. Any opinions, findings, and conclusions or recommendations expressed in this dissertation are my own and do not necessarily reflect the views of the NIST.

I am grateful for the guidance my advisor Dr. Soundar Kumara has given me during my time as an undergraduate student and throughout graduate school. It is to him that I owe my interest in research and attribute my decision to pursue a Ph.D. I would also like to express my deepest gratitude to my advisor Dr. Eunhye Song whose insight helped to steer my research in a productive direction many times. In addition, I would like to sincerely thank Dr. Michael Brundage for his role as a mentor during my time at NIST and for helping me become a better researcher. I also thank my committee members Dr. Dan Finke, especially for his guidance and support early in my Ph.D. studies, and Dr. Russell Barton for his thoughtful questions and feedback on my work.

Most importantly, I thank my parents for their unwavering love and support through all of my pursuits.

Chapter 1

Introduction

Throughout this dissertation we examine the maintenance function of manufacturing systems, namely the optimization and planning of maintenance activities. We will propose a method of scheduling maintenance for complex production systems of arbitrary structure and demonstrate the method through experimentation of various system configurations.

In this chapter, we first provide background and motivation for the study of industrial maintenance and classify various maintenance strategies. We then describe the manufacturing system behavior studied in this work, identify current challenges in maintenance optimization and planning, and lastly, state the objectives and contributions of this dissertation.

1.1 Evolution of Industrial Maintenance

Long seen as a burdensome necessity of manufacturing operations, maintenance has received attention in recent decades as an activity with the potential to reduce operating costs and support overall system performance. Early industrial maintenance strategies were primarily corrective (also called reactive or run-to-failure) policies where a machine or other asset was not repaired until an observable failure occurred. Such corrective maintenance (CM) strategies are costly in most settings due to the severity of damage that can occur when a machine fails, safety risks, and the disruption caused by frequent unplanned downtime [1].

As competition and demand for efficiency began to increase in the mid-twentieth century, the field of reliability engineering saw its rise through the application of statistical control methods to model equipment fatigue and failure [2]. Also around this time, reliability-centered maintenance (RCM) was introduced, which aims to proactively identify maintenance actions that keep equipment in an operational state at minimum cost [3]. By using methods such as these, manufacturers no longer had to wait until the occurrence of a failure

before conducting maintenance on an asset.

These advances led to better maintenance strategies for manufacturers. Preventive maintenance (PM) strategies became more common and sought to improve upon corrective maintenance by identifying points in time prior to the complete failure of a machine to perform a less costly repair that would avoid such failure. A simple preventive maintenance policy schedules repairs on a machine at regular time or processing cycle intervals, while more advanced strategies may use a dynamic interval based on observations of prior failure on a machine. While preventive maintenance improves upon a corrective strategy in many cases, more recent advancements in maintenance have aimed to leverage abundant manufacturing data to support real-time maintenance decision making.

The ubiquity of large-scale sensing and automation in manufacturing has also been extended to maintenance operations. A class of policies known as condition-based maintenance (CBM) uses real-time measurement of a machine's degradation status to decide if maintenance is needed. This is typically accomplished using sensors that monitor signals such as vibration or temperature that can provide insight into the status of a machine. CBM can help to avoid performing a repair on a healthy machine, which is a risk of suboptimal static PM intervals. The goal of CBM is typically to identify a threshold for some sensor signal (or combination of signals from several sensors) at which maintenance should be conducted [4].

Predictive maintenance (PdM) extends CBM by not only measuring the current state of a machine, but also considering other factors such as the state of other machines in the system, the projected deterioration of the machines, operating condition parameters, production and work-in-process information, and the estimated impact of inducing downtime for maintenance [5]. All of this information is used concurrently to determine the ideal time at which to schedule maintenance activities [6]. Accurate prediction of a machine's behavior allows for better maintenance planning, increased system performance, and lower long-term costs. Prognostics and health management (PHM) methods also fall into this class of maintenance policies but tend to place emphasis on identifying root causes of failure and correcting them before the problems arise [7].

The application of Artificial Intelligence (AI) to the domain of industrial maintenance has also received much recent interest and has further enabled effective maintenance practices. [8] refers to the integration of AI methods within a maintenance optimization and planning framework as an "intelligent maintenance decision support system." Such systems may employ one or several AI techniques and have been applied effectively in several areas of maintenance. Some examples include case-based reasoning (CBR) for fault detection and diagnosis [8], evolutionary algorithms for policy optimization [9], and natural language processing (NLP) for parsing and datafication of maintenance logs [10]. We will examine

specific applications relevant to this work in more detail in later chapters. In general, AI has been successful in its application to many of the complex problems encountered in the maintenance domain. We seek to build upon these successes through the work presented here.

1.2 Maintenance Taxonomy

Many classes of maintenance strategies are found in the literature and often similar strategies are described with inconsistent terminology. The terminology used to describe certain aspects of maintenance can vary across industries, organizations, and even maintenance teams. The following will classify various strategies and their relationships and state the terminology as it is used throughout this dissertation. It is not intended to be a comprehensive unifying taxonomy for all maintenance research, but rather is given to aid in the reader's understanding of this work. A more thorough maintenance taxonomy is given by [11].

We first define maintenance as the set of activities whose purpose is to keep degrading assets in their expected working condition. This can include periodic inspections of machine status, machine monitoring via sensors, order and storage of spare parts, repair, replacement, and other related activities [12]. The specific activities that are necessary and appropriate depend on the assets that are being maintained and the environment in which they operate. A maintenance *strategy* is a set of rules that specify which of these activities are carried out as well as the set of decision variables that dictate when and where the activities should be executed. An instance of a maintenance strategy for a particular system is referred to as a *policy* and is defined by a particular set of values corresponding to the decision variables of the strategy. Finding the optimal values of these variables with regards to some objective function is the crux of *maintenance optimization*.

We assume every maintenance action carried out in a system can be classified as either *corrective* or *preventive*. Corrective maintenance involves the "repair or replacement of components as a result of failure" while preventive maintenance is the "repair or replacement of components at predetermined intervals/critera" [13]. If a machine is in a failed state, it can therefore only be restored (either completely or partially) by a corrective maintenance action. Otherwise, a machine is eligible to be restored by a preventive maintenance action.

In general, corrective maintenance actions are more costly and time consuming than preventive actions [1]. This cost and time increase is due to severity of the damage as well as the uncertainty of a corrective maintenance task. For example, changing the oil on an engine every 3,000 miles is a routine, generally low cost maintenance task (preventive), however repairing or replacing the engine due to complete engine failure when not regularly

changing oil (corrective) can be time consuming and expensive in terms of the replacement cost and lost production. As such, preventive maintenance is desirable so that unplanned corrective maintenance is avoided.

1.3 Manufacturing System Dynamics

Throughout this work, we primarily examine mass production discrete manufacturing systems which consider "the production of distinct items" [14]. That is, each machine may only produce an integer number of parts, as opposed to continuous manufacturing which allows for partial production. Mass production or large-volume manufacturing systems are defined by [15] as systems "intended to produce parts and products in multiple copies: cars, computers, refrigerators, and other items of wide use." We therefore treat each part in our target system as identical.

Two main assets make up the system of interest: machines and buffers. Machines retrieve a part from a non-empty upstream buffer, process that part for a specified amount of time, and then place the part in a non-full downstream buffer. If a machine attempts to retrieve a part and none are available, it is considered starved. Conversely, if it attempts to place a part in a buffer and no space is available, it is considered blocked and that part occupies the machine until space becomes available. This convention is sometimes referred to as "block after service" [15]. Buffers are passive containers with a fixed maximum capacity.

Parts arrive in the system via a source, or a buffer with infinite contents. They leave the system when they are placed in a sink, or a buffer with infinite capacity. The change in the contents of the sink over some period of time is the total system production during that period. More details on the implementation of this system behavior are given in Chapter 2.

In this work we use the term "machine" to refer to the the smallest maintainable unit in the system. What constitutes a maintainable unit may vary across applications, but any discrepancy is generally a matter of terminology and modeling.

A system is defined by a set of machines and buffers that are connected by a specified routing. The routing determines from where machines may retrieve parts and where they may place them once processing is finished. The system can be represented by a directed graph where each node is either a machine or a buffer and the routing is defined by the edges of the graph. Generally, the only constraint we assume when creating a system is that no two buffers are adjacent in the directed graph representation. This constraint ensures that there are no "dead ends" in the routing that would cause parts to become trapped in the system.

We refer to these systems of arbitrary configuration as "complex" in terms of system

structure. This is in opposition to systems with with more a strictly constrained arrangement of machines and buffers. For example, serial production lines are often studied and consist of a single path from source to sink. A constrained structure may allow for a convenient analytical model of system behavior, but many real-world systems cannot be modeled under such structural constraints. Chapters 4 and 5 will more thoroughly explore the challenges associated with optimizing maintenance for systems of arbitrary structure.

1.4 Challenges in Maintenance Optimization and Planning

Perhaps the first challenge to consider in implementing maintenance for a manufacturing system is the selection of an appropriate maintenance strategy. Choosing the right strategy depends on many factors including safety considerations, repair costs, failure frequency, failure duration, and condition monitoring capability [16]. For failures that are infrequent and inexpensive to correct, a simple corrective maintenance policy may be optimal. On the other hand, if failures require expensive repairs or jeopardize the safety of personnel, then more advanced strategies are justified. The cost of implementing more advanced monitoring and maintenance decision making techniques should be compared to the potential savings of more effective maintenance. Generally, while more advanced strategies require a higher technology readiness level and significant up front costs, they yield greater cost savings potential in the long run [7]. The maintenance strategy selection problem has been studied thoroughly in recent literature (for examples, see [17–19]) and is not covered explicitly in this work. We will, however, demonstrate that the methods presented here are applicable to a wide variety of maintenance strategies.

Regardless of the maintenance strategy chosen, the strategy must be applied in the form of a maintenance policy for a particular manufacturing system. Typically, the policy is specified by a set of decision variables that determine where and when maintenance should be carried out. Since the system is made up of several machines, the policy must specify the maintenance actions for each. In most cases this is a combinatorial optimization problem where the size of the solution space increases exponentially with the number of machines.

One approach to seeking an optimal maintenance policy is to find the optimal policy for each machine independently and then combine the results into a system-level maintenance policy. This solution may be particularly attractive if machine-level optimization is achievable analytically. For example, in the case of time-based PM (where a policy is defined by the interval between maintenance jobs) or CBM (where some health index threshold determines

when maintenance is scheduled) optimization for a lone machine involves only a single decision variable. With an accurate model of machine behavior these policies are trivially optimized. At the system level, however, combining these independently optimized machine-level policies are not likely to be optimal because the dynamics of interaction between machines are ignored during optimization [20]. This type of dependence among machines is defined by [21] as "performance dependence" and can be difficult to model accurately, particularly for complex systems, thus posing an additional challenge when optimizing maintenance.

Another challenge in maintenance optimization that is frequently overlooked is that of a capacity on maintenance resources. In a real-world system, performing maintenance requires some amount of labor and materials to complete. Limitations on either impose a maximum number of jobs that may be completed simultaneously. [21] refers to this limitation as "resource dependence" and note the additional challenge of deciding where to allocate the limited maintenance resources when there is competition among machines.

A common solution to the challenges described thus far is to make simplifying assumptions regarding the structure or behavior of the manufacturing system. For example, much of the existing work in maintenance optimization imposes strict assumptions on the configuration of machines in a system. In the case of resource dependence, it is often assumed that there is no capacity on maintenance resources, which eliminates the need to consider scheduling conflicts among maintenance jobs. Such assumptions limit the ability to generalize certain maintenance optimization methods to arbitrary manufacturing systems.

1.5 Research Objectives and Contributions

The goals of this work are as follows:

1. Develop a simulation engine for maintenance of manufacturing systems that can be used with optimization via simulation and sample-based planning methods.

Much of the work presented throughout this dissertation relies on simulation experiments for the comparison of various maintenance scheduling strategies. There is a need for a simulation package that allows for the systematic generation of experimental manufacturing systems and extensible behavior of manufacturing objects. We present a software package that has been developed for this work and further describe how it can be used for other simulation-based applications in manufacturing as well.

2. Demonstrate a solution method for the combinatorial maintenance policy optimization problem.

We focus mainly on CBM optimization for much of this work and formulate the policy optimization problem using discrete optimization via simulation (DOvS). For each machine in the system we seek an optimal degradation threshold at which maintenance is scheduled for that machine. To resolve maintenance scheduling conflicts, we initially compare a static scheduling rule to a dynamic priority rule that identifies production-critical machines. In Chapter 4 we propose a method of online prioritization for scheduling maintenance, as addressed by the next objective.

3. Formulate the maintenance capacity scheduling problem as an online prioritization problem for dynamic real-time maintenance planning.

We model the behavior of a manufacturing system as a Markov decision process (MDP) and use Monte Carlo tree search (MCTS), a method of sample-based planning, to make maintenance decisions in real time. This approach allows us to seek the best maintenance action given the evolving state of the system. We again compare this method of online prioritization to more commonly used prioritization heuristics.

4. Improve upon sample-based online prioritization by retaining and reusing experience to aid in future decision making and learn an effective policy over time.

We use a case-based reasoning (CBR) framework to store the experience gathered from MCTS to use when similar system states are encountered in the future. By allocating computational effort more efficiently, we demonstrate that maintenance decisions can be made more quickly and with greater accuracy. This is especially useful in practical settings where little time is available to make urgent maintenance decisions.

1.6 Organization of this Dissertation

In Chapter 2, we describe *Simantha*, the manufacturing system simulation package that has been developed as a part of this work. It is an open source package capable of modeling a variety of manufacturing systems and is designed to be extensible. This chapter will describe its implementation and the scope of its functionality, as well as provide several examples of its use.

Chapter 3 gives a formulation of the CBM optimization problem and compares two optimization algorithms. We also examine the performance of two methods to address the capacity-constrained maintenance scheduling problem.

Formulation of the capacity-constrained maintenance scheduling problem using online maintenance prioritization is given in Chapter 4. We describe the MDP model of the system and the application of MCTS to seek optimal maintenance actions in real time.

Improvements to the solution of the online maintenance scheduling problem are presented in Chapter 5 through the application of CBR. CBR provides a framework for retaining and reusing problem-solving experience to improve decision making. Lastly, conclusions and a discussion of future work are given in Chapter 6.

Chapter 2

Simantha: An Open Source Package for Manufacturing System Simulation

Simantha is an open source discrete event simulation (DES) package written in Python that is designed to model the behavior of discrete manufacturing systems. It provides several core manufacturing object classes whose behavior is designed to be extensible to aid in the modeling of more complex systems. In this chapter, we describe underlying behavior model of each of these objects and how they interact, give several examples of model verification for the package, and provide use cases. Simantha is used for the simulation experiments presented throughout this dissertation.

2.1 Introduction

Simulation is an effective tool for evaluating alternative decisions in many settings, particularly in those like manufacturing where experimentation with the real-world system is prohibitively expensive or impossible. It is also useful in replicating the behavior of complex systems with substantial stochasticity or a high degree of interaction among components. Both of these challenges are commonly found in the manufacturing setting and the ability to model such systems analytically is limited without unrealistic simplifying assumptions [22].

Simantha (simulation for manufacturing) is a DES package developed specifically for modeling the behavior of arbitrarily configured manufacturing systems. It was created initially for the purpose of optimization via simulation, including not only maintenance policy optimization, but other optimization problems found in manufacturing such as line balancing or machine routing. This software package provides tools to model the behavior of core manufacturing assets, namely machines, buffers, and maintainers. These objects serve as the building blocks of the complex manufacturing systems that are of interest in this work. While the focus of Simantha is on high-volume discrete manufacturing systems like those

described in Chapter 1, we will demonstrate its applicability to other types of systems as well.

Section 2.2 discusses related work in the area of manufacturing system simulation and alternative available tools. The design of Simantha is described in Section 2.3 including the behavior of the core manufacturing objects and the event graph formulation that defines the underlying simulation behavior. Section 2.4 describes the procedure for verifying the simulation behavior and several use cases of this package are given in Section 2.5. Lastly, conclusions are stated in Section 2.6. The current version of the Simantha package is available at `https://github.com/m-hoff/simantha`.

2.2 Related Work

DES has become a widely used tool, particularly in domains such as manufacturing where analytical models of complex system are difficult or impossible to derive [22]. There are many options of software for DES, including both commercial and open source packages. A thorough review of available DES software for operations research applications is given by [23]. In this section, we focus on the existing tools that are most relevant to this work and the motivation for developing Simantha.

Many commercial-off-the-shelf (COTS) simulation tools have the advantage of a robust feature set, frequent updates, and readily available support. These come at the financial cost of the licenses needed to use the software. Some commercial packages also have limited support for scripting or use proprietary scripting languages, requiring users to learn the specific language used by the package. Closed source software also makes it difficult modify the underlying behavior of a simulation engine. As we will see in our manufacturing setting, the simulated behavior follows a specific protocol that does not abide by the assumptions of many of these existing tools. Lastly, the abundance of supported features can hinder the runtime of some models, which becomes problematic for optimization via simulation algorithms that must simulate thousands of replications.

Regarding open source alternatives, Python is the language of choice due to its accessibility and growing popularity across many industries, including manufacturing. SimPy [24] is perhaps the most popular DES package for Python. It employs process-based simulation using Python generators to create simulation events. SimPy provides generic simulation functionality and is not specific to a particular domain. One limitation is that the use of Python generators prevents the ability to serialize objects that are built with SimPy. This means that the state of an object or collection of objects cannot be easily copied or stored locally. Additionally, the execution of simultaneous simulation events does not follow the

convention of our model, which we expand upon in Section 2.3.2.

ManPy [25] is another notable open source package that is written in Python and built using SimPy. It is designed specifically for manufacturing applications and provides functionality for a wide range of manufacturing assets. ManPy is also subject to the serialization limitations of SimPy, however. Additionally, ManPy is written in Python 2, which is no longer supported, and has not received a significant update in several years. While it is not ideal for adoption into an ongoing project, the high-level architecture of ManPy serves as inspiration for that of Simantha.

2.3 Design

Simantha is designed to be accessible by individuals with experience primarily in systems modeling and simulation without requiring extensive programming knowledge. An understanding of the Simantha interface and the behavior of the core objects, along with a basic understanding of the Python language, is sufficient for creating and simulating systems with this package. As we will later demonstrate, a more advanced understanding of Python can be leveraged to extend the behavior of the core Simantha assets for specific use cases.

Throughout the description of the Simantha package, we use the convention of "upstream" and "downstream" to refer to the relative arrangement of assets within a specified routing. An object (namely a machine or buffer) can potentially receive a part from any other object that is designated as "upstream" of the receiving object. Similarly, an object can potentially place a part in any object designated as "downstream" of the giving object.

Simantha includes functionality of five main objects that form a manufacturing system:

- Source: Introduces parts to the system according to a specified interarrival time distribution. By default, machines immediately downstream of a source are never starved.
- Machine: Processes parts according to its cycle time distribution and passes finished parts to a downstream object. A machine may also be subject to periodic failure due to gradual degradation, sudden system shocks, or other failure modes. The machine object can also be used to model some other non-instantaneous process, such as transportation or inspection.
- Buffer: Stores parts that are not being processed. Buffers are generally placed between machines to hold intermediate work in process and have a maximum capacity.
- Sink: Collects finished parts that exit the system and tallies the overall production. A sink can also be viewed as a buffer with infinite capacity. The overall production of the

system is generally considered to be the sum of the level of each sink in the system.

- Maintainer: Conducts maintenance on machines according to a user-specified policy. By default, the maintainer will fix failed machines according to a first-in first-out (FIFO) policy, but the user may also specify a preventive policy that repairs machines before failure occurs. A capacity may also be imposed on the maintainer, limiting the number of machines that can be repaired simultaneously.

When using Simantha, a user will instantiate the desired objects with the appropriate parameters and then define the routing among the source, machines, buffers, and sinks. During the simulation phase each machine will repeatedly attempt to retrieve a part from its upstream objects, process that part, and then pass the part to one of its downstream objects. In the case that there are multiple downstream objects eligible to receive a part from a machine, the machine will select one at random by default. The user also has the option to specify a selection priority for such cases.

If a degradation process is specified for one or more machines in the system, this process will operate simultaneously along with the machine's production process. If maintenance begins on a machine or a failure occurs, the production process of that machine is interrupted until maintenance is completed.

The underlying behavior of these processes in defined by an event graph model, described in detail in the following section.

2.3.1 Event Graph Formulation

Event graphs as defined by [26] are a model of discrete event system behavior. With only a few simple components, event graphs are capable of modeling even very complex systems. An event graph is defined by a set of nodes, representing simulation events, and edges, representing the scheduling of future simulation events. A basic event graph is shown in Fig. 2.1.

The event graph in Fig. 2.1 is interpreted as: at the execution of event A, if condition (i) is true, schedule event B after a time delay of t. At the execution of event B, the state of the system changes according to the `{state change}` description. The dashed edge from event node B to A indicates that event A should be canceled once event B is executed (if event A is currently scheduled).

There are two additional node types used for the description of the Simantha implementation shown in Fig. 2.2. The subroutine node indicates some programming logic that leads to the scheduling of one or more event nodes. For example, the maintainer object of a

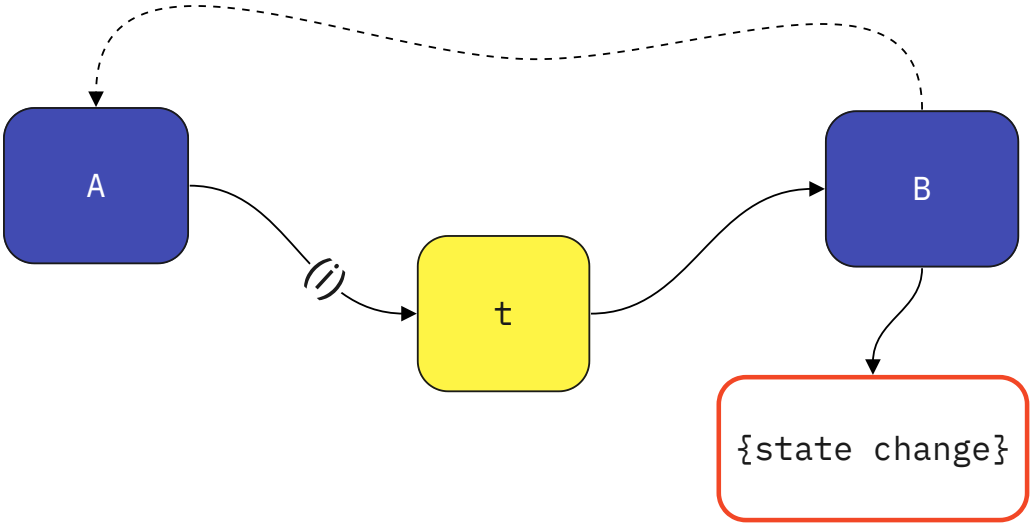

Figure 2.1: Basic event graph components.

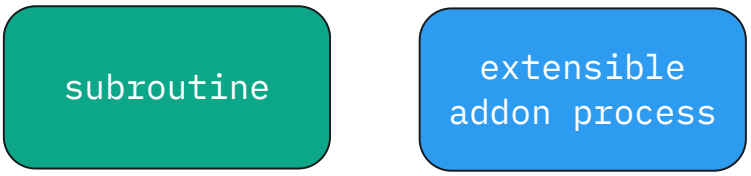

Figure 2.2: Additional event graph nodes.

system must choose which machine to repair if multiple machines are awaiting maintenance. In the case of a FIFO policy, the maintainer evaluates each machine to determine which one requested maintenance the earliest before scheduling a repair event on that machine.

The last node type, add-on process nodes, provide a means of extending the default Simantha event graph. Users can specify additional processes that are triggered by these nodes. For example, the restoration add-on process node can schedule events once a repair on a machine is completed. This might include a custom condition-monitoring process that is restarted once a machine is restored.

Fig. 2.3 shows the complete Simantha event graph. This figure shows one "layer" of a complete system event graph in that all of the simulation event nodes represent events that take place on one machine. Subroutine nodes are used to schedule events on other

machines. For example, when a machine places a part downstream (indicated by the `put part downstream` event node), a subroutine will determine if there is another machine that is eligible to retrieve that part. If so, a request for that part is generated by the downstream machine via the `request part` event node. Conceptually, extending the Simantha event graph is as simple as adding nodes and edges to the existing graph that define the desired behavior.

2.3.2 Simulation Event Execution Order

As simulation events take place at discrete time steps, it is inevitable that the scheduled time of multiple events will coincide. The order in which simultaneous events are executed can affect the simulation output and should be handled in such a way that does not violate the model assumptions. Many existing simulation tools use either a FIFO event order, where events are executed in the order in which they were scheduled, or rely on a user-specified priority [27]. Simantha uses a specific event priority that follows the assumptions of the underlying manufacturing system model with the option to insert user-defined event types into the priority list.

Simantha follows the slotted time discrete event system setting as described by [15] for handling simulation events. The slotted time case indicates that all system state transitions occur at discrete time intervals as opposed to transitions occurring in continuous time. The following excerpt from [15] describes the state changing convention for machines and buffers under the slotted time case:

> *Machine state* ... is determined *at the beginning of each time slot*. This implies that a chance experiment, carried out at the beginning of each time slot, determines whether the machine is up or down during this time slot.
>
> *Buffer state* is determined *at the end of each time slot*. This implies that buffer occupancy may change only at the end of a time slot. For instance, if the state of the buffer was 0 at the end of the previous time slot, then the downstream machine does not produce a part during the subsequent time slot, even if it is up. If the buffer state was $h \neq 0$ and the downstream machine was up and not blocked, then the state of the buffer at the end of the next time slot is h if the upstream machine produces a part, or $h - 1$ if the upstream machine fails to produce.

Simultaneous simulation events are therefore categorized into two classes: events at the end of a time slot and events at the beginning of the next time slot. Consider, for example,

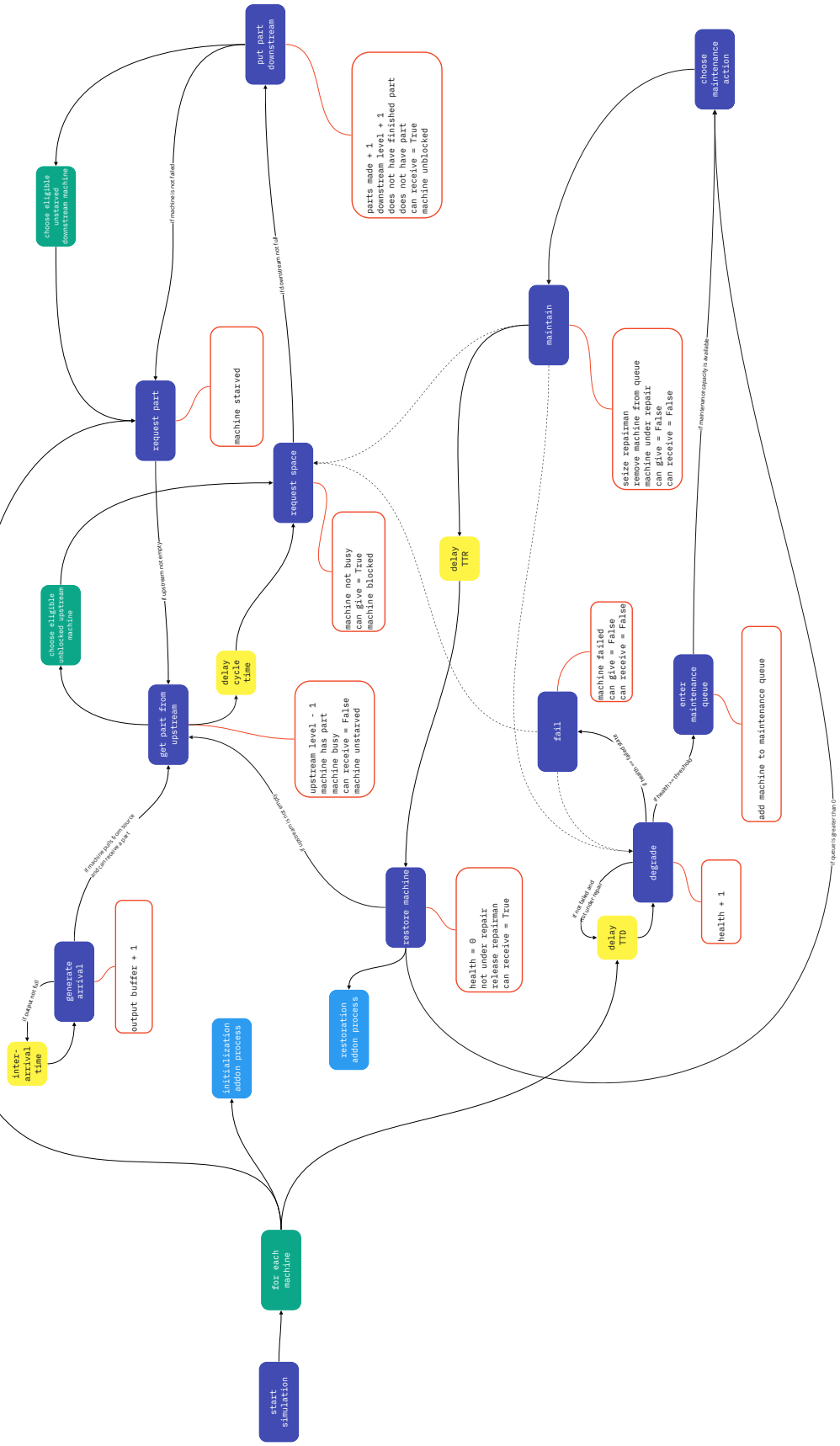

Figure 2.3: Simantha event graph.

a single machine that is scheduled to place a finished part downstream and undergo failure at the same simulation time. According to our state change convention, this machine would first place the part downstream (since buffer state changes occur at the end of a time slot) and then fail (since machine state changes occur at the beginning of a time slot). If the order of these events was reversed, the machine would first fail, causing it to discard its current part and cancel the yet to be executed event of placing that part downstream. Thus a specified order of these events is needed for consistent simulation output.

The following events are listed in order of highest priority (executed earlier) to lowest priority (executed later) as they are implemented in Simantha:

- Events at the end of a time slot:

    1. `generate_arrival`: Source object generates a part if there is a downstream machine eligible to receive it.
    2. `request_space`: Machine with a finished part checks for available space in a downstream buffer or sink.
    3. `put_part`: If space is available, the machine places a part in the downstream buffer. This event also checks if a starved machine was fed by this action.
    4. `restore`: A machine has finished maintenance and is restored to a healthier state. Also schedules a maintainer inspection event.

- Events at the beginning of a time slot:

    5. `degrade`: Machine degrades by one unit. If it is not failed then a time to degrade is sampled and the next degrade event is scheduled.
    6. `enter_queue`: If the previous degradation event caused the machine to reach its maintenance threshold then it is placed in the maintenance queue.
    7. `fail`: If the previous degradation event caused the machine to reach its failed state, then the machine undergoes failure.
    8. `inspect`: Maintainer object inspects the current maintenance queue. If the queue is not empty, the maintainer chooses a machine to repair. Otherwise, the maintainer does nothing.
    9. `maintain`: A machine begins maintenance and a restoration event is scheduled after sampling a time to repair.
    10. `request_part`: An available machine examines upstream buffers and sources for available parts and schedules a retrieval of a part if one is available.

11. `get_part`: A machine retrieves a part from an upstream buffer or source.

- Simulation run time events:

12. `terminate`: The simulation has reached its maximum specified run time and no further events are executed.

If multiple events with the same priority are scheduled simultaneously then their execution order is chosen randomly by default. Alternatively, users can specify a selection priority for objects in the system so that events occurring on an object with a higher priority are executed before those that take place on lower priority objects. User-defined events should be assigned a priority value so that the new event is placed properly in the execution order. A use case including a user-defined event is included in Section 2.5.2.

2.3.3 Simulating a System

Once the system is defined, the user can then simulate the operation of the system for a specified amount of time. Simantha also provides functionality for replicating multiple iterations of a simulation run in parallel, which is especially useful for estimating the expected behavior of a stochastic system.

2.4 Verification Procedure

Several tests were constructed to verify the behavior of the Simantha package. These include unit tests that verify the functionality of individual functions and object methods as well as integration tests that verify the interaction between system components. The random behavior of the simulator is also verified by comparing the simulation result to the theoretical expected behavior. For example, given the distribution of time to failure and time to repair for a single machine, we can calculate the expected long term average availability of that machine. If the simulator reflects this expected behavior, then that is an indication the simulation behaves as intended.

Several such tests are presented in the following sections. The complete suite of tests can be found in the publicly available Github repository.

2.4.1 Deterministic Machine Behavior

The first set of tests deals with the deterministic behavior of machines. Machines in each of these cases have a constant cycle time and are not subject to random degradation and

failure. The purpose of these tests is to verify the production function of machines and the flow of parts through a system.

2.4.1.1 Deterministic Machine Throughput

In the simplest case, we consider the deterministic production of a single machine. For one machine with a cycle time of one minute, the number of parts produced by that machine should be equal to the number of minutes its operation is simulated. Simulating for one day (1,440 minutes) should result in 1,440 parts produced by the machine.

The code to construct this test case is shown below, while the complete code for the remaining test cases is included in included in Appendix 6.

```
from simantha import Source, Machine, Sink, System, utils

# Create manufacturing system objects
source = Source()
M1 = Machine(cycle_time=1)
sink = Sink()

# Specify object routing
source.define_routing(downstream=[M1])
M1.define_routing(upstream=[source], downstream=[sink])
sink.define_routing(upstream=[M1])

# Instantiate system
system = System(objects=[source, M1, sink])

# Simulate system
system.simulate(simulation_time=utils.DAY)
```

Execution of this code produces the output:

```
Simulation finished in 0.05s
Parts produced: 1440
```

which verifies that the machine produces the number of parts expected.

2.4.1.2 Deterministic Serial Machine Throughput

Another key function of this package is the ability to arrange a sequence of machines in a system. In a serial configuration, there is a single path from source to sink that all parts must traverse. In this test, we arrange three machines in series each with a cycle time of one

minute. If we start with an empty system, the last machine will be starved for the first two minutes of operation while it waits for the first part to be processed by the other machines. Beyond that point, we expect the last machine to produce parts without interruption for the remaining simulation time.

Simulating this three machine line for one day produces the following output:

```
Simulation finished in 0.20s
Parts produced: 1438
```

thus verifying the behavior of the serial arrangement.

2.4.1.3 Deterministic Parallel Machine Throughput

When machines are arranged in parallel, a part must visit just one of these machines before proceeding to the next station. Parallel machines can also be viewed as a single machine with the capacity to process more than one part simultaneously.

For this test, we arrange three machines in parallel, each with a cycle time of one minute. Each machine should therefore produce one part per minute, resulting in an overall production volume of three times the simulation time.

The output of this test is

```
Simulation finished in 0.17s
Parts produced: 4320
```

which is as expected since $3 \times 1{,}440 = 4{,}320$.

2.4.2 Stochastic Machine Behavior

In the following tests we verify the stochastic behavior of machines, namely the process of random degradation and failure. These tests are conducted by comparing the long-term average simulation behavior to the theoretical expectation.

2.4.2.1 Degrading Machine Availability

The availability of a machine is the proportion of time that it is not in a failed state. Given probability distributions for time to failure and time to repair, the long-term average availability is

$$A = \frac{m_f}{m_f + m_r} \tag{2.1}$$

where m_f is the mean time to failure and m_r is the mean time to repair [28]. We can compare this result to the behavior of a simulated machine with known time to failure and repair distributions.

For the following tests, we use the geometric distribution for time to failure and time to repair, specified by $Geom(p)$ where p is the success probability parameter with mean $1/p$.

In this example, we consider a single machine with a constant cycle time of one minute, time to failure distributed $Geom(p_f)$ with $p_f = 1/90$, and time to repair distributed $Geom(p_r)$ with $p_r = 1/10$. The resulting expected availability is

$$A = \frac{1/p_f}{1/p_f + 1/p_r} = \frac{90}{90 + 10} = 0.90. \tag{2.2}$$

To conduct this test, we replicate the simulation of a single machine under these settings 100 times for a period of one month. The resulting output is

```
Finished 100 replications in 23.94s
Average availability: 90.0368%
```

We can further verify this output by conducting a one sample t-test to determine if the observed mean availability is significantly different from its expected value. The p-value of this test is 0.5371, indicating we fail to reject the null hypothesis that the observed machine availability is equal to 90%.

2.4.2.2 Degrading Machine Throughput

Periodic downtime of a machine affects its productivity by reducing the amount of time that is spent processing parts. The effective mean cycle time of a machine is given by [28]

$$t_e = \frac{t_0}{A} \tag{2.3}$$

where t_0 is the mean cycle time of the machine while it is in a fully operational state.

Using the same single-machine system described in the previous section, we find that the expected effective cycle time of the machine is

$$t_e = \frac{1}{A} = \frac{1}{0.90} = 1.11 \tag{2.4}$$

which corresponds to a production rate of 0.90 parts per minute. Over the one month period considered previously, we would expect to see an average of $0.90 \times 30 \times 24 \times 60 = 38{,}880$ units produced in this time. Over the 100 replications of the previous test, the average production

count is 38,849.29 resulting in an average throughput of 0.8993 parts per minute. We again conduct a one sample t-test to compare the observed average throughput to its expected value which results in a p-value of 0.2107. We therefore fail to reject the null hypothesis that the observed throughput is equal to 0.90 parts per minute.

2.4.2.3 Degrading Bottleneck Machine Throughput

In the last test case included here, we examine the interaction of multiple machines subject to stochastic degradation. In a system of multiple connected machines, the long run throughput for each machine is expected to converge to the effective throughput of the bottleneck machine [28]. That is, the system throughput is given by

$$TH = \text{bottleneck utilization} \times \text{bottleneck production rate.} \tag{2.5}$$

The bottleneck machine is defined as the that with the highest long-term utilization when accounting for downtime due to failures, shift changes, setup, and other sources of production disruption. Since we primarily consider failures and maintenance as the sole sources of downtime, then the bottleneck in a system of machines subject to the same degradation conditions is the machine with the greatest average cycle time.

We arrange three machines in series with constant cycle times of 1 minute, 3 minutes, and 2 minutes, respectively. The bottleneck machine in this configuration has a production rate of 1/3 parts per minute. A buffer with capacity 10 is placed between each machine. The machines are again subjected to geometrically distributed times to failure and repair with $p_f = 1/90$ and $p_r = 1/10$ resulting in an expected availability of 90%. Following Eq. 2.5, the expected system-level throughput is therefore given by

$$TH = 0.90 \times 1/3 = 0.3 \text{ parts per minute.} \tag{2.6}$$

Over a period of one month, the approximate expected production volume is $0.3 \times 43{,}200 = 12{,}960$ units. This value is approximate because it does not account for work in process (WIP) that is lost due to failure. The actual expected system production volume is slightly less, since some parts are periodically discarded if a machine fails while it is processing a part.

Fig. 2.4 shows the result of simulating this system for a period of 30 days. As expected, the throughput of each machine converges to the bottleneck rate (the production rate of M_2) once the system reaches its steady state. At the end of this period, the system has produced 12,617 units at an average rate of 0.2921 parts per minute.

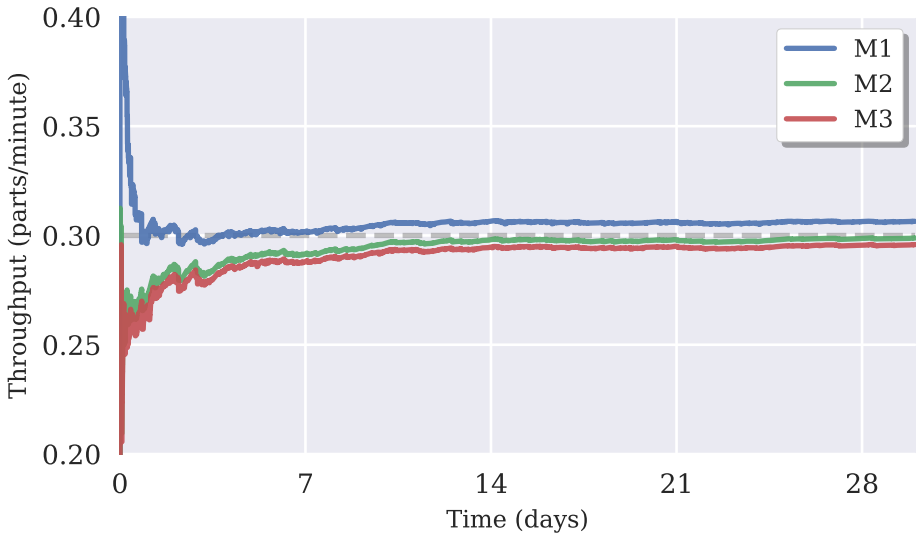

Figure 2.4: Bottleneck throughput test result.

2.4.3 Simulation Event Trace

In addition to the previously described tests, the simulation behavior can be verified by inspecting the trace of simulation events. In Simantha, each simulation event is recorded including the time and location of the event as well as the previous event that led to its scheduling.

Table 2.1 shows a subset of the event trace from the three-machine line described in the previous section. The location of each event corresponds to an object in the system and the action is a method of that object. The source includes the location and action that caused each event to be scheduled. Each source and location + action pair is represented by an arc of the event graph shown in Fig. 2.3.

2.5 Use Cases

In this section we present several use cases for the Simantha package. We will demonstrate the extensibility of the simulation object behavior and give an example of maintenance policy optimization via simulation. The associated code and additional examples are available in the Github repository.

	time	**location**	**action**	**source**	**status**
30	6	M3	request_space	M3.get_part at 4	
31	6	M1	request_space	M1.get_part at 5	
32	6	M3	put_part	M3.request_space at 6	
33	6	M1	put_part	M1.request_space at 6	
34	6	M3	degrade	M3.initialize	
35	6	M3	fail	M3.degrade at 6	
36	6	repairman	inspect	M3.fail at 6	
37	6	M3	maintain	repairman.inspect at 6	
38	6	M1	request_part	M1.put_part at 6	
39	6	M3	request_part	M3.put_part at 6	canceled
40	6	M1	get_part	M1.request_part at 6	
41	7	M1	request_space	M1.get_part at 6	
42	7	M2	request_space	M2.get_part at 4	
43	7	M1	put_part	M1.request_space at 7	
44	7	M2	put_part	M2.request_space at 7	
45	7	M1	request_part	M1.put_part at 7	
46	7	M2	request_part	M2.put_part at 7	
47	7	M2	get_part	M2.request_part at 7	
48	7	M1	get_part	M1.request_part at 7	
49	8	M1	request_space	M1.get_part at 7	
50	8	M1	put_part	M1.request_space at 8	

Table 2.1: Simantha simulation event trace.

2.5.1 Maintenance Policy Optimization

Maintenance optimization involves seeking an optimal set of parameters that define the maintenance policy for a system. The optimal policy or set of policies will maximize one or more performance measures of the system. Such performance measures typically include throughput of the system, availability of machines, average cost rate, and other performance indicators. Optimization via simulation is particularly useful in cases where system performance under a particular policy cannot be evaluated analytically due to a high degree of stochasticity in the system or interaction among machines.

In this example we demonstrate the condition-based maintenance optimization for a single machine with the objective of maximizing its production volume. The policy is defined by a single decision variable: the machine health index at which to conduct maintenance. The machine health h begins at 0 and increases by one unit at each time step with probability 0.1 until reaching a maximum of 10. The machine can be repaired at any time, which restores its health to 0. The time to repair is a function of the current health state h and is normally distributed according to

$$TTR \sim N(\mu, \sigma) \tag{2.7}$$

with mean $\mu = 2^{(h/8)^3}$ and standard deviation $\sigma = h$. The expected value of this distribution is the mean time to repair and increases as the machine progresses to higher states of degradation. Scheduling maintenance at lower health states results in lower repair times on average, but also reduces the uptime between repairs that the machine is available for production. Conversely, a higher threshold for maintenance will increase the average time between repairs but the average repair times will be higher.

To find the best policy, we simulate the system under various maintenance thresholds over a period of one week for 30 replications and choose that with the highest average production volume. Since this is a simple system with a single decision variable, we can enumerate the specified solution space to evaluate each possibility.

Fig. 2.5 shows the simulation result from the evaluation of maintenance thresholds on the interval $[1, 10]$. The maximum production of 9,931.93 units per week is achieved with a threshold of 6 for this machine.

Finding an effective maintenance policy quickly becomes more difficult for larger systems. In the case of a condition-based maintenance strategy, the solution space grows exponentially with each additional machine and it may no longer be feasible to enumerate every possible policy. More complex maintenance optimization problems will be examined later in this work.

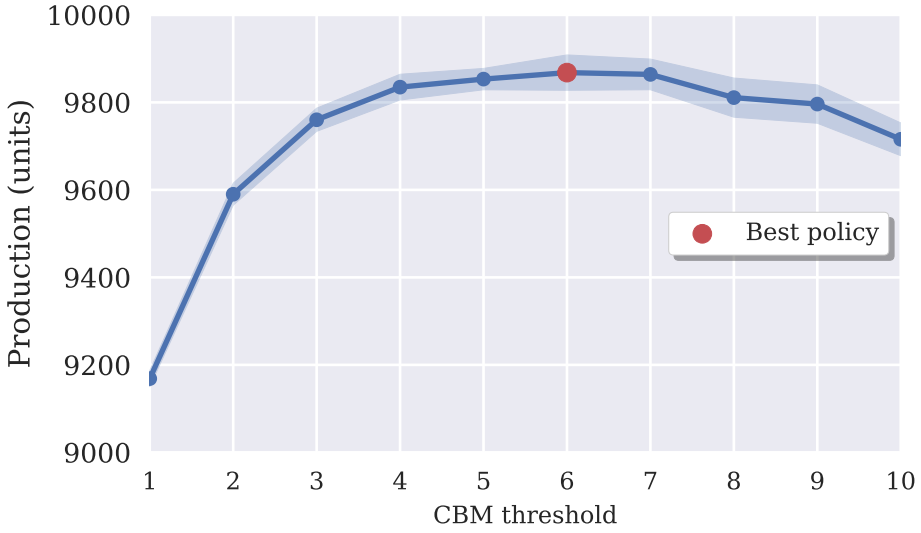

Figure 2.5: One-dimensional maintenance optimization.

2.5.2 Sensor-based Condition Monitoring

To demonstrate the extensibility of machine behavior, we implement a condition-monitored machine with imperfect sensor readings. In some settings, the exact health state of a machine is unknown but sensor readings are used to infer its degradation status. These sensor readings can be continuous or periodic, and often include some amount of random noise.

In terms of the event graph formulation, we add an event node to represent the generation of a health signal. When a machine is initialized or repaired, this health signal event is scheduled immediately. Then, until the machine is chosen for maintenance, it will generate a health signal regularly at a specified interval. A subset of the modified event graph with the added health signal generation event node is shown in Fig. 2.6. This new event should occur just after any degradation events and just before any maintenance queue arrival events. This allows a maintainer to observe the sensor signal that results from the current degradation state and determine if the machine should be placed in the maintenance queue as a result of that signal. Based on the event priority list in Section 2.3.2, this new health signal generation event should be assigned a numeric priority of 5.5 so that is is executed in the proper order.

This health signal can be made observable to the maintainer and used during the maintenance decision making process. For example, the machine may be scheduled for maintenance if some number of consecutive health signals exceed a specified threshold. This behavior may be more realistic in some settings where the exact health index is not known.

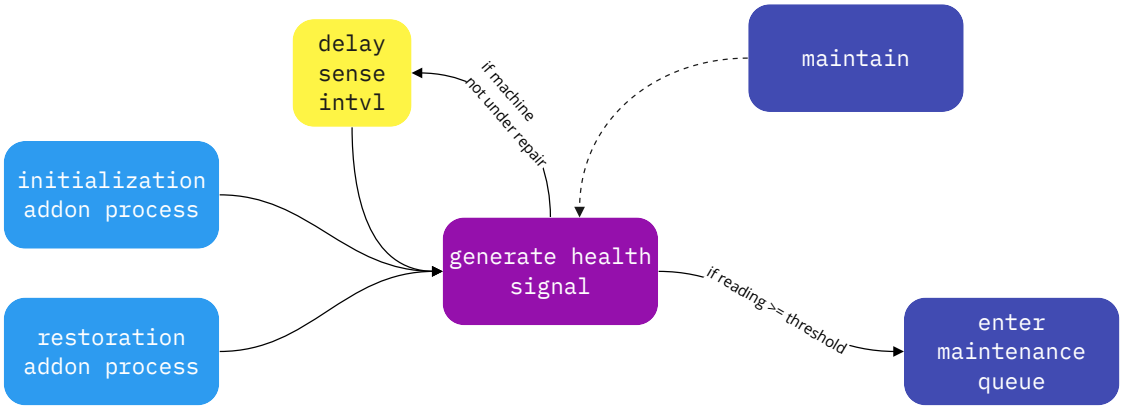

Figure 2.6: Modified condition monitoring event graph.

For this implementation, the health signal at the current time t_0 is given by

$$S(t_0) = H(t_0) + \varepsilon \tag{2.8}$$

where $H(t_0)$ is the current true health of the machine and ε is a Gaussian noise term with mean 0 and standard deviation 0.5.

We simulate a single machine under these settings for six hours and observe a health signal every two minutes. When the true health state reaches 10, the machine undergoes maintenance for 10 minutes before being restored to a health state of 0.

Fig. 2.7 shows the health index and sensor readings of a single machine over time. While the observable sensor readings follow the trend of the true health state, it is clear how maintenance decision making is more difficult in this setting.

2.6 Conclusions

We have introduced the Simantha package for discrete event simulation of manufacturing systems. The primary goal of the package is to provide a set of extensible core manufacturing objects and basic simulation functionality. We have also provided several examples of use cases that demonstrate the extensibility of the basic objects.

Future development of Simantha will include support for additional manufacturing system objects, such as conveyors or inspection stations, more detailed reporting of simulated experiments, improvement of simulation efficiency, and other features depending on their demand. Various developmental versions of Simantha are used for the simulation experiments presented throughout the remaining chapters.

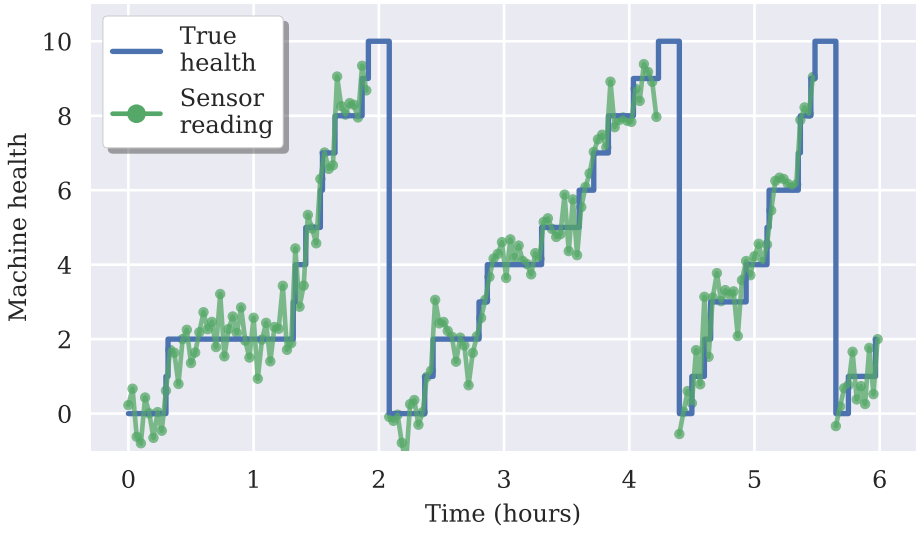

Figure 2.7: Condition monitored machine health over time.

Chapter 3

Condition-based Maintenance Policy Optimization Using Genetic Algorithms and Gaussian Markov Improvement Algorithm

3.1 Introduction

We focus on the development of a condition-based maintenance (CBM) policy for continuously-monitored deteriorating machines in this chapter. For our system of interest, we consider non-uniform machines, a capacity for maintenance resources (which imposes a maximum number of concurrent maintenance jobs), and non-instant repair times. Much of the previous related work makes simplifying assumptions and does not consider all of these factors in combination. A capacity for maintenance resources and non-instant repairs results in frequent occurrences of conflicting maintenance jobs. When a conflict occurs, a single maintenance resource must choose among multiple competing maintenance jobs to complete immediately. This makes the optimization of a maintenance policy more difficult because, in addition to deciding when to repair each machine, we must decide how to reconcile maintenance scheduling conflicts.

The rest of the chapter is organized as follows. Section 3.2 describes some of the previous work related to maintenance policy optimization and capacity-constrained maintenance scheduling. Section 3.3 presents the system description including the machine degradation model that is used throughout this work, the maintenance queue formulation, and the maintenance cost measurement model. The algorithms used to optimize the maintenance policy are described in Section 3.4. In Section 3.5 the results from an experimental example are shown and, lastly, we summarize the chapter in Section 3.6.

3.2 Related Work

Problem settings for CBM policy optimization can be classified in several ways. Such classifications are based on the maintenance policy formulation and decision variables, machine degradation model and mode of condition monitoring, system configuration, maintenance resource configuration, and optimization objectives [11]. This section provides examples of previous research in each classification scheme as it relates to our work.

3.2.1 Degradation Model and Condition Monitoring

Two main categories of decision variables are considered when defining a CBM policy. One is the interval between inspections of components in the system. When continuous monitoring is not available, the health of components can only be measured by performing an inspection that typically incurs some fixed cost. Upon inspection, the decision must be made as to whether or not maintenance should be performed. Such models are thoroughly described by [29]. The alternative, and the approach that is used here, is a continuously-monitored system where the health of components is known at each discrete time step. In this case, the decision is at what health level should maintenance be scheduled.

Continuously-monitored signals, such as vibration data, acoustic data, or temperature, are now commonplace in many manufacturing settings as sensing and data communication become increasingly cost-effective [30]. Further, multiple disparate signals can be aggregated into a monotonic composite health index that reflects the state of machine degradation for the purpose of maintenance decision support [31]. A discrete-time Markov model is often used to represent the discrete health index of deteriorating machines. Given an initial health index state, a machine transitions to another health index at the next discrete time step according to the specified degradation transition matrix. One variation of this model is that with the addition of random shocks, where a machine may transition into a complete failure state at any time, as examined by [32]. Some work has considered dependence among multiple machines, which can have a significant impact on the optimal maintenance policy. For instance, [33] develops a CBM policy for a system of components with stochastic dependence in which the degradation rate of a machine depends on that of others in the system.

3.2.2 Multi-component Policy Optimization

A large portion of the CBM optimization literature has considered only single-component systems where the policy is optimized for one machine in isolation. For examples of such work under a discrete-state degradation model, see [32, 34–36]. In multi-component manu-

facturing systems, however, the routing configuration of machines has a substantial impact on the formulation of the CBM policy optimization problem. Because of the performance dependence among machines, an independently optimized policy for a single machine may not be optimal when that machine is examined in the context of a larger manufacturing system [20, 21]. Downtime events due to maintenance or failure of a machine can propagate through a system causing other machines to become starved or blocked which complicates the modeling of such systems [37]. Nevertheless, the problem has been addressed by some recent literature.

[21] reviews recent research in CBM optimization for multi-component systems among which [38–40] consider a problem most similar to our setting with continuously-monitored serial system arrangements. However, each assumes that maintenance activities are instantaneous and that there is no capacity on maintenance resources. We aim to alleviate these simplifying assumptions in our work. In this chapter we examine only a serial production line, but subsequent chapters will consider more complex arrangements of machines.

3.2.3 Optimization and Scheduling Under Maintenance Capacity

Constraining maintenance capacity also increases the complexity of a maintenance optimization problem because there will be cases where multiple machines are competing for the same maintenance resource. These situations impose the additional challenge of resolving such conflicts. One class of maintenance strategy that deals explicitly with maintenance capacity is selective maintenance. Often under a selective maintenance strategy, each job consumes some amount of constrained resources (such as time or labor) during a "maintenance break" for the entire system. Maintenance breaks occur between two consecutive production missions and the objective when scheduling maintenance is usually to minimize maintenance time or cost or to maximize asset reliability during the next mission. This class of policies was first introduced by [41] and a thorough review of recent work in selective maintenance is given by [42]. Although selective maintenance accounts for maintenance capacity, having predetermined maintenance breaks and fixed mission duration is not always applicable. In a realistic manufacturing setting, unplanned failures can occur at any point in time and need to be addressed immediately, whereas selective maintenance does not allow production to be interrupted until the next maintenance break.

[21] further 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. For example, [43] assumes n machines are in a system and there are at least $n-1$ maintenance resources available, implying that

there is no instance where a maintenance resource must decide among multiple machines to maintain. Meanwhile, [44, 45] optimize the maintenance policy of a k-out-of-N system (a system of N identical components of which at least k must be functional for the system to be operational) with maintenance capacity. Since all machines are identical, the choice of which ones to repair has no impact on system performance. Such simplifying assumptions are often made for mathematical convenience and are not representative of many real-world manufacturing systems [22].

In some cases, a maintenance policy is already established for a system and does not need to be optimized. In these instances there is still the challenge of scheduling maintenance when resources are limited. Maintenance prioritization subject to limited capacity is examined by [46] in a variety of industries, including manufacturing. Four prioritization methods are identified 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 pairwise comparisons of weighted criteria and sub-criteria [47]. 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. [48, 49] 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 [50, 51], 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 [52]. Several case studies using priority criterion for maintenance are presented by [53] which considers priority criterion for both long-term strategic planning and short-term operational decision support. 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 [54]. 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 [55]. 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 interaction among machines and consequence of downtime are not easily inferred.

Each of these methods of maintenance priority management relies on expert input to derive and evaluate priority criteria. Obtaining the necessary expert opinion may not be feasible in some cases due to cost or time constraints. To overcome this, traditional queueing service disciplines have also been applied to maintenance and typically do not require prior expert knowledge. Several common scheduling rules are compared by [56] 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 [57]. 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 [58,59]. While these scheduling rules are simple to implement and may perform well in some scenarios, they do not consider the ever-evolving state of the system. In this chapter we compare our proposed maintenance priority rule to a FIFO scheduling discipline, although we consider additional scheduling heuristics in later chapters.

3.2.4 Optimization Objective Function

While the minimization of cost is a typical objective in the design of maintenance policies, other competing objectives are also considered. In addition to minimizing cost, [43] aims to maximize the availability of the system. [60] attempts also to maximize the throughput of the system by scheduling maintenance so that downtime does not hinder production. [12] identifies overall equipment effectiveness (OEE), machine availability, and finished part quality as the most commonly used maintenance performance measures in manufacturing, although they are not explicitly used as optimization objectives. Many of these metrics and objectives can be combined into a single cost objective. For example, production that is lost due to downtime for maintenance can be assigned a cost per unit that worsens the objective function value. Including such measures in the cost objective function allows us to consider a single objective.

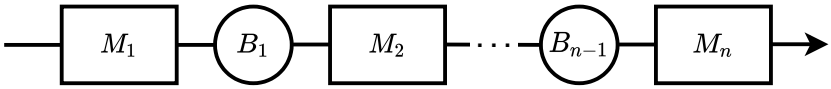

Figure 3.1: Serial manufacturing system example.

3.3 System Description

In this section, we define notation that is used throughout the remainder of this dissertation and the underlying assumptions of the system of interest. A manufacturing system consists of n machines where M_i is the label of the i^{th} machine. For a serial line, we assume that machines M_i and M_{i+1} are connected by a single intermediate buffer B_i, as shown in Fig. 3.1. For a general arbitrary configuration of machines, we make no such assumption but continue to use machine labels of the form M_i for convenience. We will examine complex machine arrangements in more detail in later chapters, but an example of such a system is shown in Fig. 5.3.

Machine M_i has cycle time t_i which may be a constant or a discrete random variable with a specified probability distribution. We assume that each buffer has maximum capacity $b_{\max}$. We further assume that each machine will produce at its maximum rate while it is functional, so long as it is not starved or blocked and that the first machine in the system is never starved and the last machine is never blocked.

The maintenance capacity dictates the maximum number of maintenance jobs that may be executed simultaneously and is denoted $c_{\max}$. A time t, the available maintenance capacity is $c_{\text{idle}}(t)$ while the occupied capacity is $c_{\text{busy}}(t)$. It follows that $c_{\text{idle}}(t) + c_{\text{busy}}(t) = c_{\max}$ for all t.

3.3.1 Degradation Model

As described in Section 3.2.1, many environments employing CBM assume that component deterioration can be modeled as a discrete Markov process. Under this model, machines begin in a perfect health state and move to states of increased degradation over time according to the specified degradation state transition matrix. The advantages of this model are that we only need to know the current state of a machine to estimate its future degradation and the discrete states provide a well-defined and intuitive domain for the CBM decision variables [61]. This model also allows for variations such as sudden failures in the form of system shocks [32] or stochastic dependence of degradation among machines [33]. Examples of previous work that employ the continuously-monitored discrete health state Markov model for maintenance optimization problems include [43, 62–65].

Using this model, we denote the health index level of machine M_i at time t as $H_i(t)$.

We assume that M_i is in perfect working condition when its health index $H_i(t) = 0$ and that it degrades at each time step with some probability. As the machine degrades, its health index increases until it is repaired or experiences a complete failure. The degradation rate of a machine can depend on many factors including age of the machine, stress on the machine, utilization, or the degradation state of other components in the system [66]. When a random failure occurs the machine stops functioning completely until it is repaired. When maintenance is completed on a machine, whether preventive or in response to a complete failure, its health index is restored to zero and degradation resumes.

In this chapter, we define the degradation process of machine M_i by a degradation rate p_i, representing the probability of degrading by a single unit at each discrete time step, and a failure state $h_{\max}$ which indicates the degradation level at which a machine fails. The resulting $(h_{\max}+1) \times (h_{\max}+1)$ transition matrix $\mathbf{P}_i$ of the degradation process of machine M_i is:

$$\mathbf{P}_i = \begin{bmatrix} 1-p_i & p_i & & & \\ & 1-p_i & p_i & & \\ & & \ddots & & \\ & & & 1-p_i & p_i \\ & & & & 1 \end{bmatrix} \tag{3.1}$$

Note that $\mathbf{P}_i$ is upper bidiagonal in this case. In general, we only assume $\mathbf{P}_i$ is upper triangular; that is, we assume M_i cannot transition to a state of lower degradation without the intervention of a maintenance action. An upper triangular degradation transition matrix allows for the machine degradation level to increase by more than one unit in a single time step and the possibility sudden failures of the machine. We will examine these cases in later chapters.

An example of machine degradation over time is shown in Fig. 3.2. In this example, the machine reaches its threshold for maintenance at time $t = 6$ and is restored to perfect health at $t = 8$. The machine reaches its maintenance threshold again at $t = 13$, but is not repaired immediately, perhaps because all maintenance resources are occupied elsewhere in the system. The machine reaches failure at $t = 15$, at which point it can only be restored by corrective repair. Finally, corrective maintenance on the machine is completed at $t = 19$.

Throughout this work, we specify the machine degradation transition probabilities, though several methods exist for estimating a Markov degradation transition matrix from observed data. For examples see [67, 68].

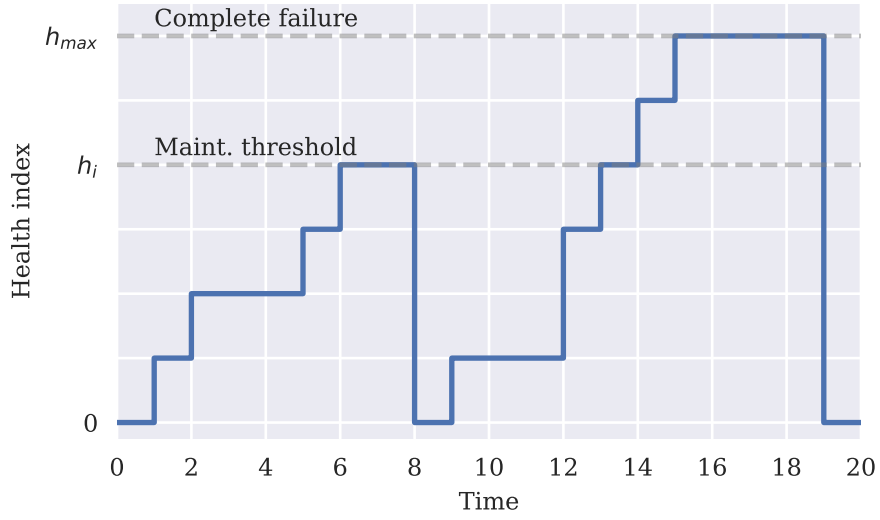

Figure 3.2: Markovian machine degradation example.

3.3.2 Maintenance Queue

As machines degrade over time, a maintenance request is generated whenever a machine's degradation level exceeds its threshold for maintenance. The set of current maintenance requests forms a virtual "maintenance queue" containing 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

$$1 \leq c_{\text{idle}}(t) < |L(t)|. \tag{3.2}$$

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. While waiting in the queue for maintenance, machines continue to degrade until their health index reaches $h_{\max}$.

In this chapter, we consider two queueing disciplines: first-in, first-out (FIFO) and priority queues. Under the FIFO rule, maintenance jobs are serviced in the order that they arrive in the queue by available maintenance resources. Any ties are resolved by random selection. While this policy is simple to implement, it ignores the fact that high-risk machines with a greater potential to disrupt system throughput (e.g., the bottleneck machine) will be ignored

if they are not in the front of the queue.

An alternative approach is to assign each maintenance job a priority measure and always service the job in the queue with the highest priority. To minimize lost production due to machine down time, maintenance jobs are assigned a priority that is related to the size of each machine's *maintenance opportunity window*. This concept is explained further in the following section. If two or more machines have equal priority, we defer to the FIFO discipline to resolve the tie.

3.3.2.1 Maintenance Opportunity Window

The maintenance opportunity window is the length of time a machine can stop production without hindering the overall system throughput. Throughput loss is avoided through the use of buffers in the system and by making sure the bottleneck machine does not become blocked or starved. In the following we use β to denote the index of the bottleneck machine which is the machine with the greatest cycle time among those in a serial line. For simplicity we assume there is a single unique bottleneck machine.

If a machine M_i in a serial line is upstream from the bottleneck machine M_β ($i < \beta$), the opportunity window for M_i is the time it takes for all buffers between M_i and M_β to become empty. At this point, the bottleneck machine is starved and throughput is hindered. If M_i is downstream from M_β ($i > \beta$), the opportunity window for M_i is the time for all buffers between M_β and M_i to become full. The bottleneck machine will then be blocked. This concept is described thoroughly by [50] and is summarized by Eq. 3.3:

$$W_i(t) = \begin{cases} t_\beta \sum_{j=m+1}^{\beta} b_j(t), & i < \beta \\ 0, & i = \beta \\ t_\beta \sum_{j=\beta+1}^{n} (b_{\max} - b_j(t)), & i > \beta, \end{cases} \tag{3.3}$$

where $b_i(t)$ is the level of buffer B_i at time t and $W_i(t)$ is the duration of the opportunity window for machine M_i at time t. This equation assumes that M_i is the only machine that is failed over the duration of the opportunity window; however, work has shown how this equation also provides an approximation of the opportunity window of each machine with simultaneous failures [69].

Machines with the smallest maintenance opportunity window will be assigned the highest priority. By minimizing the downtime of high-risk machines, we can reduce the throughput impact of performing maintenance. A comparison of the performance of the FIFO and priority queue disciplines is described in Section 3.5.

3.3.3 Cost Objective Function

In general, minimizing the cost of the maintenance policy as given by Eq. 3.4 will be the primary objective. The cost of a policy over some time horizon consists of three components: planned maintenance activities, unplanned maintenance activities, and lost production due to downtime in the system. The cost of a planned preventive maintenance job is incurred when a machine is repaired before reaching the complete failure state. The total cost of preventive maintenance is the product of the number of preventive jobs that occur over a specified time horizon (x_P) and the cost of each preventive maintenance activity (C_P). Similarly, the total cost of unplanned corrective maintenance is the number of repairs completed on a machine in a failed state (x_U) multiplied by the cost of the activity (C_U). The number of maintenance events over the observed time horizon T is represented by a random variable.

Cost is also measured in terms of lost production due to machine downtime. Lost production is defined as the difference between the production requirement in units over the time horizon and the actual number of units produced by the system. The production requirement can be a fixed number of units, or a fraction of the "ideal production" that is obtained from a perfect system with no downtime events. We assign a cost of C_{LP} to each unit of lost production.

The maintenance policy cost function is therefore

$$C_T = C_P x_P + C_U x_U + C_{LP}\left(\frac{T}{t_\beta} - u\right) \tag{3.4}$$

where T/t_β is the ideal production obtained with no downtime events according to Little's law [28] and u is the actual system production in number of units over T.

Generally, planned maintenance is less costly than unplanned maintenance because the machine avoids a complete failure and downtime that results in system throughput disruption [70]. Unplanned maintenance occurs when a machine's health index reaches the total failure state, $h_{\max}$, and it is forced to stop production until repaired. Machine failures can occur when a machine that is waiting for planned maintenance continues to degrade to the point of total failure while waiting for maintenance resources to become available. Since we consider the duration of maintenance activities in the model, maintenance on a machine will disrupt the machine's production and possibly the overall throughput of the system.

This cost function can also be used to maximize the throughput of the system by setting C_P and C_U to 0. Since C_{LP}, T, and t_β are fixed, such an objective is equivalent to maximizing the number of parts u produced by the system over the time horizon T.

3.4 Methodology

The goal of this work is to find an optimal CBM policy for a serial manufacturing system under capacity-constrained maintenance resources. As described in Section 3.2, a CBM policy is defined by the health index thresholds at which CBM is scheduled for each machine. Since there are n machines in the system, a solution, or policy, $\mathbf{x}$ can be value encoded by a set of n thresholds $\mathbf{x} = \{h_1, h_2, \cdots, h_n\}$. The minimum value of a threshold is 1, which would indicate maintenance is scheduled on a machine as soon its health index deteriorates by one unit. The maximum value is $h_{\max}$, the health index that indicates a machine has experienced a complete failure. A threshold at its maximum value is equivalent to a corrective maintenance policy.

For the problem presented here, the primary objective is to find the maintenance policy that minimizes expected cost per unit time. Due to the stochasticity and complex interactions that occur in the system under consideration, it is difficult to analytically determine the cost of the policy as a function of a set of CBM thresholds for each machine. Analytically determining cost often requires simplifying assumptions that reduce the accuracy of the cost measurement [22]. For this reason simulation will be used to evaluate the solutions by estimating the expected cost of a policy. We will compare the effectiveness of a genetic algorithm and the Gaussian Markov Improvement Algorithm in finding a solution to this problem.

3.4.1 Genetic Algorithm

A genetic algorithm (GA) is a metaheuristic method of problem solving that attempt to replicate evolutionary behavior observed in nature. A population of individuals (or maintenance policies) "evolves" over time by selecting the best candidates, as determined by a fitness function, to produce the succeeding generation. Starting with an initial set of random individuals, a new population is generated by reproduction and added to the total population. The best individuals from this group are then chosen to produce the next generation, and the process repeats until some termination criteria is met. The problem of optimizing a CBM policy is well-suited for GAs as this approach is robust and effective for large, complex manufacturing systems [8].

We apply GA as it is described by [71]. For the CBM policy optimization problem, we use a candidate policy $\mathbf{x} = \{h_1, h_2, \ldots, h_n\}$ as a value-encoded individual. The fitness of each individual is determined by the production volume that is obtained by the system over a fixed time horizon when simulated under the policy it represents. Individuals are selected for reproduction with likelihood proportional to their estimated fitness, so that better solutions

are more likely to be chosen to aid in the creation of the next generation. Once two individuals are selected for reproduction, an offspring is produced using uniform single-point crossover. That is, for two parent policies $\mathbf{x}_1 = \{h_1^1, h_2^1, \ldots, h_n^1\}$ and $\mathbf{x}_2 = \{h_1^2, h_2^2, \ldots, h_n^2\}$, we choose a position uniformly between 0 and n (inclusive) to serve as the crossover point. Elements to the left of this point in $\mathbf{x}_1$ are combined with elements to the right of this point in $\mathbf{x}_2$ to form a complete policy and create a new individual. For example, if the crossover point is m, the offspring of $\mathbf{x}_1$ and $\mathbf{x}_2$ would be $\{h_1^1, \ldots h_m^1, h_{m+1}^2, \ldots h_n^2\}$.

To maintain diversity in the population, there is a small chance of mutation in each element of a newly created individual. If mutation occurs, the element changed to a random value on its domain. We also employ an elitist strategy wherein the best subset of individuals from the current generation are carried over into the new generation. This approach improves the performance of the GA by retaining good solutions once they are identified [72]. For our implementation we use a population size of 30 and a mutation probability of 0.01.

3.4.2 Gaussian Markov Improvement Algorithm

Discrete optimization via simulation (DOvS) refers to finding an optimal solution of a problem with discrete decision variables whose objective function does not have a closed-form expression, but can be evaluated by stochastic simulation. Because of its flexibility, DOvS is a popular method for solving a complex stochastic problem. For the problems with small-to-medium feasible solution spaces where one can afford to simulate all feasible solutions, ranking and selection (R&S) has been successfully applied [73]; however, when the feasible solution space is large, it is practically impossible and inefficient to simulate all solutions. For the latter category of DOvS problems, several adaptive random search (ARS) algorithms have been developed. In general, an ARS algorithm initially simulates a small number of solutions and iteratively selects the next solution to simulate based on the simulation history. Since these initially sampled solutions are often used to estimate the necessary parameters of the algorithm, they are referred to as initial design points. A good ARS algorithm uses statistical inference based on simulated solutions to choose the next solution to simulate balancing exploration and exploitation.

The Gaussian Markov Improvement Algorithm (GMIA) finds the globally optimal solution of a DOvS problem with probability 1 when the simulation budget increases without a bound [74]. GMIA is an ARS that draws statistical inference on the performance of feasible solutions by fitting a metamodel of the objective function at all feasible solutions based on the simulated solutions. The particular metamodel GMIA employs is a Gaussian Markov random field (GMRF), which models the unknown objective function values at the solu-

Table 3.1: Experiment machine parameters.

Machine	1	2	3
t_i	3	4	5
q_i	0.02	0.06	0.01

tion as Gaussian random variables with positive spatial correlations among solutions. From the simulation results of the initial design points, parameters of the GMRF model are estimated. Then, at each iteration, the distribution of the GMRF is updated conditional on the cumulative simulation results up to that iteration.

GMIA imposes the correlation such that nearby solutions have stronger positive correlation in the objective function values. This works for DOvS as solutions that are close in the feasible solution space often have similar objective function values. Therefore, even if a solution is not simulated yet, we can infer the objective function value at the solution based on simulated solutions and guide the search towards a more promising region of the feasible solution space. We defer the implementation details of GMIA in this paper; see [74].

3.4.3 Simulation

We use simulation to evaluate the quality of a maintenance policy solution for both GA and GMIA. The system is simulated in its steady state for some period of time and then the cost of the defined maintenance policy is calculated using Eq. 3.4 as described in Section 3.3.3. The system is considered to be in its steady state when the production rate of each machine is relatively constant over time. Once the steady state is achieved, the system is observed for the specified time horizon, T.

3.5 Results

A three-machine serial production line is used to demonstrate the methodology presented in the previous section. The system will be evaluated under both FIFO and priority queue disciplines. The machines in the system are described in Table 3.1 and Table 3.2 describes additional parameters of the system.

GA and GMIA can be compared by evaluating the performance of each for a specified simulation budget. The simulation budget will be a maximum number of fitness function evaluations (NFE) that will occur. NFE for GMIA is given by

$$(2 \cdot \text{max iterations} + k) \cdot r, \tag{3.5}$$

where k is the initial number of design points and r is the number of simulation replications of each sampled solution. The NFE for GA depends on the population size, maximum number of generations, and the number of replications and is given by

$$(2n + 1) \cdot \text{population size} \cdot \text{replications}, \tag{3.6}$$

where n is the number of generations.

For the GMIA instance shown here, the maximum number of iterations is 200, the number of initial design points is $k = 10$, and the number of simulation replications for each sampled solution is $r = 10$. This results in a total of 4,100 fitness function evaluations at the termination of the GMIA. The parameters for the GA are a population size of 20, a maximum of 10 generations, and 10 simulation replications per solution. 4,200 fitness function evaluations are used for GA.

For this system, the solution space is small enough that all solutions can be exhausted in order to find the global optimum. At 10,000 replications, the largest standard error observed was 3.23. The algorithms are also compared to see if they converge to this solution.

3.5.1 FIFO Maintenance Queue

Using a FIFO scheduling discipline, the overall best policy is $\mathbf{x} = \{8, 7, 8\}$ which was found to have a cost of 382.07 when simulated for 10,000 replications. Under this policy, machines 1, 2, and 3 are scheduled for repair when their health index reaches 8, 7 and 8, respectively. Fig. 3.3 shows the convergence of each algorithm to the optimal solution for the system under a FIFO maintenance queueing discipline. The objective function value at each stage is the average of ten simulation replications. The cost shown on the vertical axis is the true cost of the best solution, as found by exhausting the solution space. When evaluating a solution, the algorithms are not likely to obtain an estimate that is equal to the true expected cost of the policy due to the high variance in the simulation. This results in the selection of "worse"

Table 3.2: Experiment system parameters.

Parameter	Value
Buffer capacity ($b_{\max}$)	2
Planned maintenance time to repair	Uniform$(5, 15)$
Planned maintenance cost (C_P)	50
Unplanned maintenance time to repair	Uniform(10, 20)
Unplanned maintenance cost (C_U)	300
Unit lost production cost (C_{LP})	10

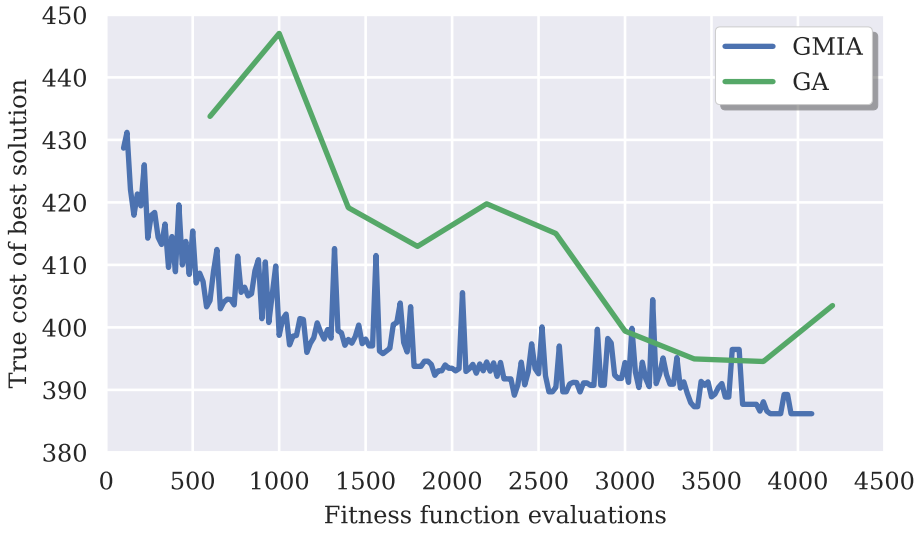

Figure 3.3: FIFO discipline policy optimization result.

solutions at some steps of each algorithm. Both algorithms are able to improve the solution over time, but on an average GMIA finds a policy with a lower cost.

In many cases, the true cost of a policy is different than that determined by the GA. As shown in Fig. 3.4, the cost of the best solution at each generation as predicted by the GA (referred to as the observed cost) is much lower than the true cost of that solution. In fact, the observed cost of the best solution at termination is lower than the true minimum cost (indicated by the horizontal red line). This is due to the small number of simulation replications that are made when the GA evaluates a solution. The high degree of replication variability makes it difficult to accurately measure the fitness of a solution with only a few replications.

3.5.2 Priority Maintenance Queue

Similar results can be examined for the system under a priority queue discipline for maintenance jobs. The true minimum cost is obtained for the policy $\mathbf{x} = \{8, 7, 8\}$ which has an average objective function value of 383.21 after 10,000 simulation replications, so the cost is not improved by a priority queue. Both algorithms are again compared using a prescribed maximum number of fitness function evaluations. In Fig. 3.5 the convergence of each algorithm is shown. Again it appears that on average the GMIA obtains a better solution for a given NFE.

Just as in the FIFO maintenance queue case, the GA tends to underestimate the cost of the best solution, as shown in Fig. 3.6. This is again a result of the small number of

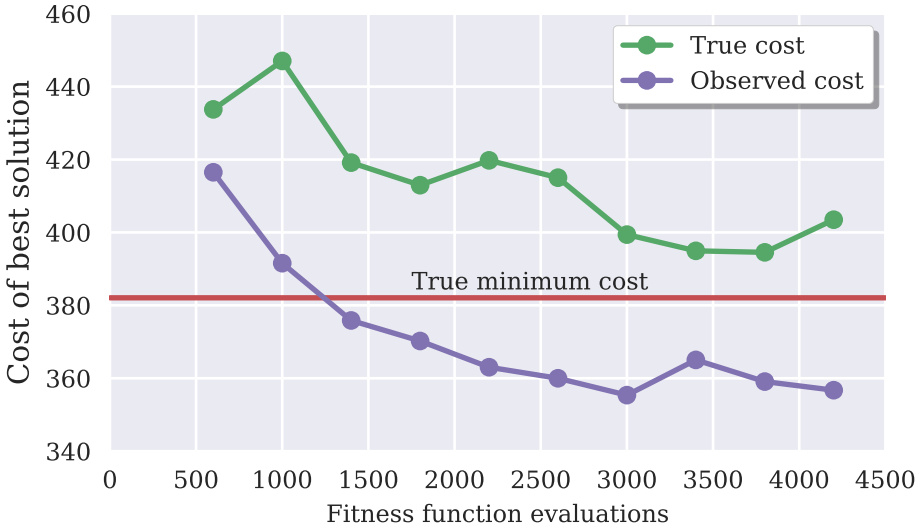

Figure 3.4: FIFO discipline GA objective function estimation result.

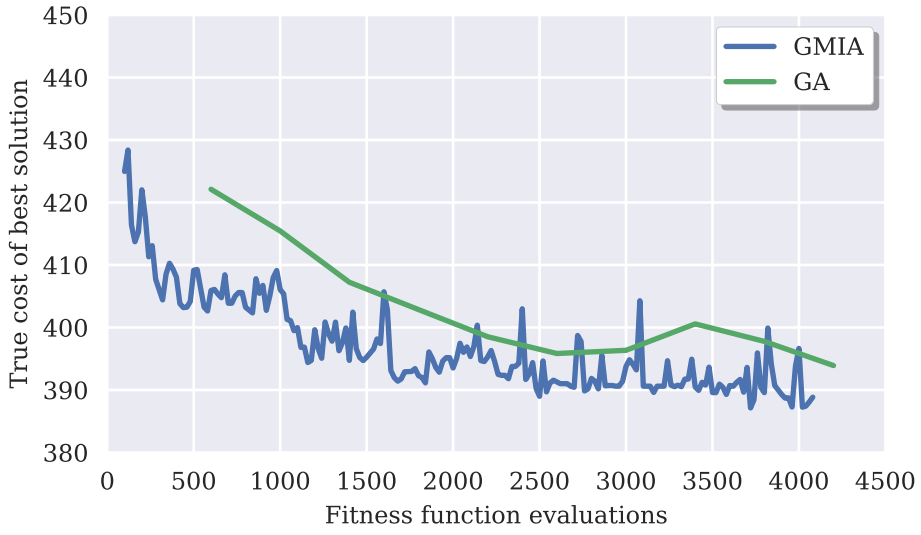

Figure 3.5: Priority discipline policy optimization result.

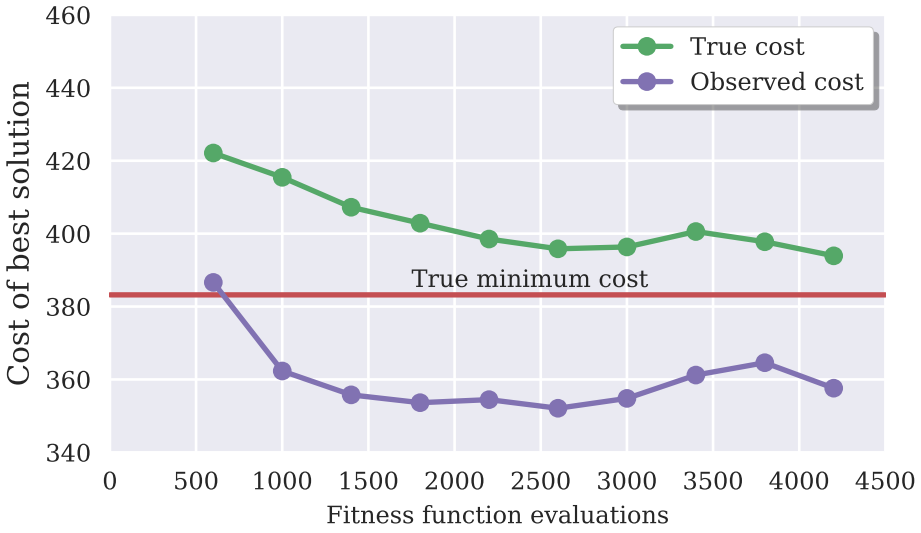

Figure 3.6: Priority discipline GA objective function estimation result.

replications that are used to evaluate the candidate solutions. There is a trade off between the accuracy of solution evaluations and the number of unique solutions evaluated. Conversely, another disadvantage of the GA is that favorable solutions may be overlooked due to the variability in their evaluation. Just as the fitness of some solutions is overestimated, it is likely that fitness is frequently underestimated as well. This could result in better solutions not being selected for reproduction, resulting in a non-optimal population of solutions.

3.6 Summary

For the CBM policy optimization problem of a serial manufacturing system, both GA and GMIA have shown to be effective search techniques. When constrained to a fixed simulation budget, GMIA is able to find a better solution at termination on average. This is an important consideration as the simulation of complex systems can be computationally expensive and time-consuming.

The average maintenance cost of the example system presented here did not benefit from a priority maintenance job queueing discipline when compared to a FIFO discipline. This could be due to fact that failures of other machines are ignored when finding the opportunity window of a machine. Improving upon the opportunity window priority measure is among the next steps of this work. It may also be the case that there are few instances of conflicting maintenance jobs, and so there is little need to decide the order in which jobs should be performed. A priority queue would likely be more effective for a system with a greater

number of machines or machines with higher rates of degradation. In both cases, a greater number of maintenance jobs would occur over a given time horizon, thus increasing the rate of scheduling conflicts. We explore these possibilities further in the following chapter by examining alternative system configuration arrangements.

Chapter 4

Online Improvement of Condition-based Maintenance Policy via Monte Carlo Tree Search

4.1 Introduction

In this chapter, we develop a maintenance framework that uses real-time system state information to schedule maintenance in support of system throughput by avoiding lost production through unnecessary idling of production-critical machines. The framework provides a policy that aims to strategically schedule downtime for maintenance so that throughput disruption is avoided as much as possible. The proposed method can be applied to multi-component systems of arbitrary configuration as well as to those with constrained maintenance capacity.

We again focus on a condition-based maintenance (CBM) strategy, which relies on prescribing maintenance actions as a function of a machine's current degradation state. Although this problem can be formulated as a Markov decision process (MDP) as reviewed in Section 4.2, it becomes prohibitively expensive to solve using traditional methods as simplifying modeling assumptions are relaxed and the problem scale increases. Instead, another thread of research is finding a set of static health index thresholds at which to schedule maintenance for each machine. In optimizing these thresholds the goal is to maximize the long-term average performance of the system; however, this problem is also challenging due to a combinatorially large solution space of health indices and lack of an analytical expression of the performance measure. The goal of this chapter is to present a method to find an optimal CBM policy for a general multi-component manufacturing system that is subject to maintenance capacity constraints and to seek optimal maintenance actions in real time in this setting.

Our approach is to first find a static CBM policy that maximizes the throughput using a

genetic algorithm (GA). We use a static queueing priority discipline whenever the number of machines due for maintenance exceeds the available maintenance capacity during optimization. The output of the GA is a health index threshold for each machine that determines when a machine will request maintenance.

Once a static CBM policy is found, it is adopted until a maintenance scheduling conflict occurs. We then solve an MDP problem online by seeking the optimal action in the current state of the system whenever the number of machines requesting maintenance exceeds the available maintenance capacity. Here, by "online" we mean that the static policy is updated incorporating the real-time information of the system. The state space for the MDP is reduced significantly compared to the original problem since we only consider taking a maintenance action for a machine that has exceeded its health index threshold as specified by the static CBM policy. Thus, for each machine, health states below the threshold can be merged together in the online scheduling problem. For the solution method, we adopt Monte Carlo tree search (MCTS) to efficiently navigate the state-action space under a given computation budget. The result of this search is the "best action" in the form of which machine to prioritize for maintenance given the current state of the system. This procedure is repeated each time a maintenance resource must choose between two or more pending maintenance jobs.

As defined by [75], online scheduling problems are distinguished by incomplete information due to either a lack of knowledge of future jobs, unknown duration of scheduled jobs, or unknown times between machine failure. Our online maintenance prioritization problem has each of these characteristics and so we refer to our method of maintenance conflict resolution as an "online improvement" of the static policy. This is in contrast to use of the term "online" elsewhere in maintenance optimization literature to refer to online parameter estimation for non-stationary system processes. For example, [76, 77] update the parameters of the degradation process for a single-component system as more degradation observations are gathered over time.

The rest of the chapter is organized as follows. Section 4.2 reviews existing literature relevant to CBM optimization and maintenance scheduling. The problem statement is given in Section 4.3 followed by the methodology in Section 4.4. Section 4.5 shows experiment results and lastly, a summary is given in Section 4.6.

4.2 Related Work

In Section 3.2 we examined previous research related to CBM optimization that has included a discrete Markov machine degradation process and a serial machine configuration as we

have considered thus far. We also reviewed work that has considered capacity-constrained maintenance resources and identified several methods for prioritizing maintenance under this constraint. In this section, we extend the literature review by considering previous work that has studied the CBM optimization and maintenance prioritization problems in structurally complex systems.

Besides a serial arrangement, there are several machine configuration types that result in increased structural complexity. These include parallel, series-parallel, k-out-of-N arrangements, and various combinations of these. A thorough description of each arrangement in the context of reliability is given by [78]. Often the policy optimization method is derived under an assumption of the system configuration making it difficult to transfer the method to other systems that do not abide by the structural assumptions. Even more rarely is a maintenance strategy generalizeable to arbitrary system configurations.

Many complex real-world systems cannot be characterized by one of these configuration types, yet little previous work has allowed for arbitrary system structure. Among that which has, [58, 79, 80] optimize CBM for general system configurations, but rely on the assumption that maintenance activities are instantaneous and that there is no limit to the number of components that can be repaired simultaneously. Maintenance duration is imposed in [81] for CBM of a complex system arrangement, yet maintenance capacity is unlimited. A capacity on maintenance resources is a common real-world constraint that this work also aims to address.

Regarding capacity-constrained maintenance prioritization, [56] seeks the best priority arrangement (which is equivalent to a static sequence of jobs) with limited maintenance resources for pending jobs in a complex system of a given state. Although this approach addresses the problem of scheduling maintenance on heterogeneous machines under a fixed capacity, it uses a predetermined maintenance schedule that does not adapt to the evolving state of the system and thus is not appropriate for online prioritization. Even if a schedule performs well compared to common scheduling heuristics such as first-in, first-out (FIFO), shortest processing time first, etc., it may not necessarily be the best policy in every scenario the system encounters. In Chapter 3 we sought to improve upon FIFO by using a priority heuristic based on the concept of the opportunity window for maintenance jobs that minimizes the disruption to system production. However, the priority measure considers each job in isolation and does not accurately evaluate the impact a sequence of jobs has on system performance [82].

The literature reviewed thus far has studied the optimization of CBM policies under a variety of configurations. However, simplifying assumptions have been made in previous work that limit the effectiveness of these maintenance strategies in a generalized dynamic

manufacturing setting with maintenance capacity. There is therefore a need for a method of dynamic maintenance scheduling that makes the best scheduling decisions under the current conditions of a complex system. Our proposed approach improves upon this work by first seeking an optimal CBM policy in the form of a set of static maintenance thresholds that serve to reduce the size of the problem state space. We then update this policy online when needed by formulating and solving an MDP using MCTS.

Monte Carlo planning algorithms aim to improve upon static rules for the maintenance 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 system is modeled as an MDP, which is formalized for our system of interest in Section 4.4.2.

[83] proposes a method of seeking optimal actions for large or infinite MDPs using an online search in the current state of the system. The proposed 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. While this method provides useful theoretical guarantees on the optimality of the result, it is possible that significant simulation effort is expended on suboptimal action trajectories. This limitation is overcome by [84] with the upper confidence bound for trees (UCT) algorithm. 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 [85].

4.3 System Description

In this chapter we again consider discrete manufacturing systems with a finite maintenance capacity. In contrast to the problem presented in the previous chapter, here we allow for an arbitrary configuration of machines instead of restricting the system to a serial arrangement. This means that a machine may have more than one upstream buffer from which it retrieves parts as well as multiple downstream buffers into which it places finished parts. Similarly, multiple machines may share either an upstream or downstream buffer. An example of a complex configuration is shown in Fig. 4.3.

For complex systems, we introduce the concept of a station as a set of identical machines arranged in parallel relative to each other. These machines are identical in terms of their cycle

time and their degradation transition matrix. We denote a station as $S_j = \{M_1^j, \ldots, M_{n_j}^j\}$, $j = 1, \ldots, m$. Since machines in a station are identical, we apply the same static CBM threshold to each machine in a station. This allows us to reduce the dimension of the solution space during the static policy optimization procedure.

4.3.1 Machine Degradation

We use the machine degradation model described in Section 3.3.1. To summarize, we assume the machine health index is continuously observable and model its behavior over time as a discrete-time Markov chain where higher states represent increased degradation and wear. Machines undergo time-based degradation, meaning that degradation of a working machine continues even if it is idle due to blockage or starvation. An example of this degradation process is shown in Fig. 3.2.

We again assume that the degradation transition matrix for each machine is upper triangular. This follows the monotonicity assumption of the health index measure by ensuring that a machine cannot transition to a healthier state without the intervention of a maintenance action.

4.3.2 Maintenance Queue

Since we consider a capacity on maintenance resources, we use the maintenance queue as described in Section 3.3.2 to handle competing maintenance jobs. The maintenance queue contains machines that are failed and require corrective action as well as functional machines that are awaiting preventive repairs. Under CBM, a machine enters the maintenance queue when its health reaches the CBM threshold prescribed by the policy. When a maintenance resource becomes available, it chooses a machine to prioritize for repair from among those in the queue.

4.4 Methodology

We formulate the CBM optimization problem in two parts: static CBM threshold optimization and its online improvement whenever there are competing maintenance jobs. The two components of this formulation are summarized in Fig. 4.1. The static CBM policy reduces the state space by excluding relatively healthy machines from consideration for maintenance. This state space reduction allows MCTS to avoid allocating its search budget to evaluating the possible action of repairing a healthy machine, which is not likely to be optimal if there are other machines in more advanced degradation states.

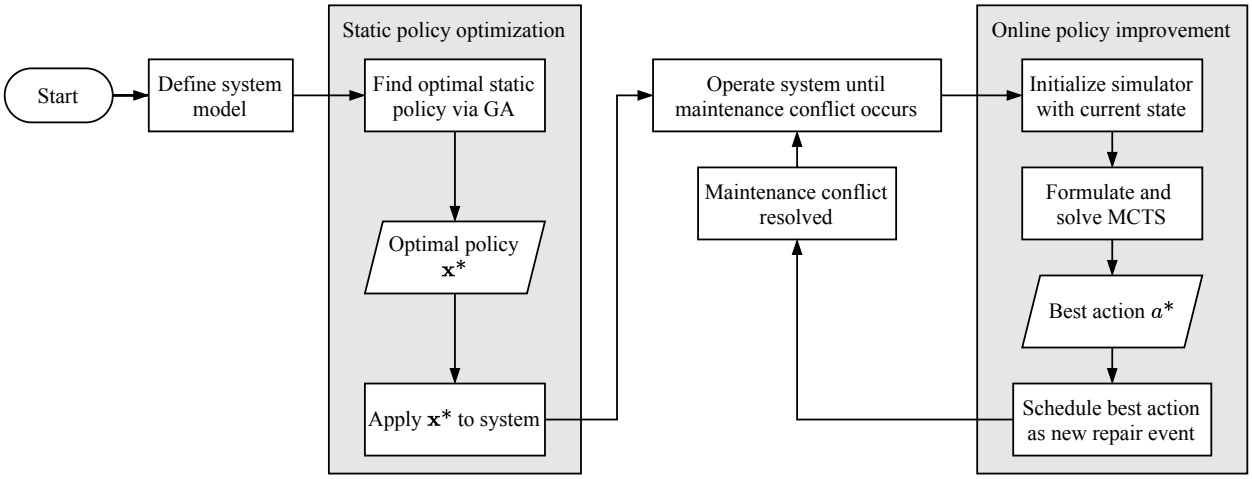

Figure 4.1: Static policy optimization and online improvement procedure.

4.4.1 Static Policy Optimization

We use a GA as described in Section 3.4.1 to optimize the static CBM policy of the system. The result of this optimization is a set of maintenance threshold for each machine in the system, denoted $\mathbf{x} = \{h_1, h_2, \ldots, h_n\}$, which indicate the health index level at which each machine should request maintenance.

If machines in the system are grouped into stations, we can instead seek to optimize a station-level policy, represented by $\mathbf{x}_S = \{h_{S_1}, h_{S_2}, \ldots, h_{S_m}\}$. Using this formulation, the CBM threshold h_{S_j} is applied to all machines at station S_j. The dimensionality of the solution space is reduced if $m < n$.

The objective function during optimization is to maximize the throughput of the system in number of parts produced per unit time. This is equivalent to the minimizing the cost objective function given in Eq. 3.4 with $C_P = 0$, $C_U = 0$, and $C_{LP} = 1$.

4.4.2 Online Improvement of Static Policy

To improve upon the static scheduling policy used to resolve maintenance scheduling conflicts, we use Monte Carlo tree search (MCTS) to seek the best machine to prioritize for maintenance when a conflict occurs in online fashion. We formulate MCTS online each time a maintenance resource becomes available in the system and there is more than one machine in the maintenance queue (i.e., in states where Eq. 3.2 is true). MCTS has seen success in its application to artificial intelligence in games by using a single-player stochastic game formulation, or an MDP, where there is one decision maker and the state transition is random. We adopt this formulation for our problem.

4.4.2.1 Markov Decision Process Formulation

An MDP is 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

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

- 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

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

$$Q(s_0, \pi(s_0)) = \sum_{s' \in S} T(s_0, \pi(s_0), s') \cdot (R(s, \pi(s_0), s') + \gamma Q(s', \pi(s'))) \tag{4.1}$$

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

$$\pi^* = \max_{\pi} \; Q(s, \pi(s)) \; \forall \; s \in S. \tag{4.2}$$

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 maximize the expected reward.

The online prioritization problem is formulated with s_0 representing the current state of the system when a maintenance conflict occurs. 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 the cumulative number of parts produced in state s' and s. By choosing $\gamma < 1$, we place preference on obtaining rewards earlier instead of deferring rewards.

4.4.2.2 System State Representation

State representation is another important consideration when formulating an MDP. For the maintenance scheduling problem, the state of the system is defined by the health index for each machine, remaining processing time for machines with a part in progress, and elapsed repair time for machines under repair as well as the current level of each buffer. In practice, this method requires that this information is readily available in real time. As discussed previously, we assume that the health of each machine is known continuously through state-of-the-art condition monitoring techniques. In most modern manufacturing environments, information regarding the work in progress at each machine and the current buffer levels are typically available through the use of manufacturing execution system (MES) software or similar technologies [86]. For machines currently under repair, we need to estimate the distribution of the remaining repair time in order to simulate the future behavior from the current point in time. Since the time to repair distribution is known, we can find the appropriate conditional probability distribution of the remaining repair time given the time that has already elapsed. This initial current state is denoted s_0.

The state of a system at time t can be represented as the set of state variables for each component in the system. For a machine M_i, the health state variable is -1 if it is under repair, 0 if $H_i(t) < h_i$ and $H_i(t) \in [h_i, h_{\max}]$ otherwise. We use a health state of 0 to indicate a general "healthy" state for machines whose health does not exceed their threshold for maintenance. There are therefore $h_{\max} - h_i + 3$ possible encoded health states for M_i.

The remaining process time is encoded as -1 if M_i is failed, under repair, or starved (it is in an operational state and does not have a part), 0 if it is blocked (it is in an operational state and holding a completed part and cannot place it downstream), or $t_i - u_i$ if it is holding a part that has been processed for duration $u_i \in [0, t_i - 1]$. The remaining process time can take on one of $t_i + 2$ values for each machine.

The elapsed repair time of a machine is encoded -1 if it is not under repair or r_i if it began repair at time $t - r_i$. Therefore, for a general time to repair distribution $\mathcal{G}$, the remaining repair time on M_i (q) is distributed $\mathcal{G}(q|q \geq r_i)$. The repair state can be simplified in the case of geometrically distributed repair times; the repair state can be represented as

a binary variable with 1 indicating a machine is currently under repair and 0 indicating otherwise.

Lastly, the level of each buffer is integer-valued between 0 and its maximum capacity, b_j. Combining each of these state variables, we obtain a numerical state vector which allows for convenient representation and comparison of states. In particular, the total size of the state space for a system consisting of n machines subject to geometric repair times and m buffers can be bounded above by

$$|S| \leq \left[\prod_{i=1}^{n}(h_{\max} - h_i + 3) \cdot (t_i + 2) \cdot 2\right] \cdot \left[\prod_{j=1}^{m}(b_j + 1)\right]. \tag{4.3}$$

Notice that the upper bound increases exponentially with each additional machine or buffer. However, the upper bound tends to be loose since some included states are not valid. For instance, states where the number of machines under repair exceeds the capacity for maintenance will never be encountered.

From each state s, the available set of actions $A(s)$ are to repair a machine currently in the maintenance queue or, if the queue is empty, do nothing and wait until a machine enters the queue. The maximum number of possible actions in a particular state is therefore equal to the number of machines currently in the maintenance queue, which is again upper-bounded by n. When executing an action, the transition to the new state is sampled using simulation since the transition function T is unknown.

Given this MDP formulation, each node of the search tree represents a unique state-action sequence that starts from the current state, s_0. Choosing action a_0 from this state results in traversing an edge of the tree from s_0 to s_1. Thus, a node of depth d represents a state-action sequence $s_0, a_0, s_1, a_1, \ldots, a_{d-1}, s_d$.

4.4.2.3 Monte Carlo Tree Search

The general MCTS algorithm, as described by [87], is initialized by creating a root node v_0 representing the current state of the physical system, s_0. The algorithm then repeats four basic steps until a specified search budget is expended: selection, expansion, simulation, and backpropagation. 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. This steps of this procedure are outlined in the following sections and the algorithm is summarized in Algorithm 1 where $N_c(v_i)$ is the set of child nodes of v_i and $N_p(v_i)$ is the parent node of v_i.

4.4.2.3.1 Selection Since the tree contains only one node at the first iteration, the selection step is trivial at this stage. For the node selection policy beyond the first iteration, we must choose an action from each node and sample the future state that results from taking that action in the current state. We use the action selection strategy described by [88] and the upper confidence bound for trees (UCT) algorithm proposed by [84] to evaluate candidate actions. At each node v_i in the tree, we choose the action a from among the candidate actions that maximizes the quantity

$$UCT(v_i, a, C_p) = \hat{Q}_i(s(v_i), a) + 2C_p\sqrt{\frac{\ln \bar{n}}{N_{a,i}^{s(v_i)}}} \tag{4.4}$$

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. 4.1, 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. Larger values of C_p will favor the search of less visited actions, while smaller values will cause the search to spend more time evaluating promising actions. For $Q(s, a) \in [0, 1]$, $C_p = 1/\sqrt{2}$ has been shown to be ideal for rewards in this range [84]. If rewards are not bounded to this range, an effective C_p can be found through experimentation.

Once an action is selected, the resulting future state is sampled by simulating the execution of that action. The node selection process is repeated recursively until we reach either an unexpanded node (as defined under the *expansion* step) or a terminal node (as defined under the *simulation* step).

4.4.2.3.2 Expansion A node is considered "expanded" if every available action from that node has been sampled at least once. Otherwise, the node is unexpanded. If an unexpanded node is encountered during the node selection procedure, we randomly select a valid action that has not yet been chosen and execute that action to sample a future state. A new node representing this future state is then added to the current search tree. We simulate from the new node to obtain an estimate of its reward.

4.4.2.3.3 Simulation The rollout policy used in the simulation step determines how we obtain an outcome from an intermediate non-terminal node. The default policy chooses random children nodes (i.e., selecting random available actions) until finding a leaf node of sufficient depth. The reward at each node is the production volume that is observed over the elapsed time horizon divided by the ideal production over that horizon. We use a discount factor of $\gamma = 0.9624$, which results in 99% of the cumulative reward at the root node being obtained within two hours of the initial time.

4.4.2.3.4 Backpropagation After the simulation result is obtained, the statistics are updated for nodes along the path from the simulated leaf node to the root. These statistics include the number of visits to each node and their cumulative reward. The number of visits is incremented only for nodes that were chosen by the node selection policy. Nodes visited during the rollout of the simulation step are not considered visited.

4.4.2.3.5 Termination Once the search budget is expended, we have an estimation of the expected discounted reward for each action available in the current state s_0. Typically in MCTS the action with the highest average observed reward is chosen as the best. However, with a fixed search budget we may not be able to conclude that a single action is statistically significantly better than the others. We therefore use statistical testing to find the best candidate actions and apply knowledge of the system to resolve ties among such actions.

To find the best set of actions, we first use one-way analysis of variance to test the hypothesis that each sample of rewards shares an equal mean. If this hypothesis is not rejected, we cannot conclude that the mean rewards for each action are different, and so we include all available actions in the best set. Otherwise, we use Tukey's honestly significant difference (HSD) test, a multiple comparison test for determining if the mean of several samples are significantly different. This test overcomes the inflated type I error that may occur when conducting pairwise statistical tests independently.

Tukey's HSD test provides a pairwise comparison of the mean reward for each action. From this result, we choose the actions for which there is no statistically significantly better action as the best set. If this set contains more than one potential action, we select the action according to the static heuristic policy in place. For example, under FIFO we would choose the action of repairing the machine in this set with the earliest request for maintenance. This method of breaking ties among the best actions ensures the scheduling procedure will not perform worse than the static rule in the cases where MCTS cannot clearly identify a single best action.

4.4.3 Alternative Scheduling Disciplines

In addition to FIFO and MCTS, we compare several other commonly used methods of maintenance conflict resolution: shortest processing time first (SPTF), longest processing time first (LPTF), and Birnbaum importance.

Under a SPTF discipline, the machine in the maintenance queue with the shortest expected repair time is chosen first. Since we consider two types of maintenance jobs, preventive and corrective, SPTF results in preferring preventive jobs over corrective ones. If there are multiple machines awaiting preventive maintenance, we break ties according to FIFO.

Algorithm 1: General Stochastic Monte Carlo Tree Search

function `Mcts`(Initial state s_0)**:**
  Initialize tree with root node v_0
  while search budget remains **do**
    $v_\ell \leftarrow$ TreePolicy(v_0)
    $\Delta \leftarrow$ DefaultPolicy($s(v_\ell)$)
    Backpropagate(v_ℓ, Δ)
  end
return BestAction(v_0, 0), search tree $\mathcal{T}$

---

function `TreePolicy`(Node v_i)**:**
  while v_i is not a leaf node **do**
    $a_i \leftarrow$ BestAction(v_i, $1/\sqrt{2}$)
    $s' \leftarrow \mathcal{G}(s(v_i), a_i)$
    if $\nexists\ v_j \in N_c(v_i) : s' = s(v_j)$ **then**
      Create leaf node v_j
    end
    $v_i \leftarrow v_j$
  end
return v_i

---

function `BestAction`(Node v_i, exploration constant C_p)**:**
  $\hat{a}^* \leftarrow \max_{a \in A(s(v_i))} UCT(v_i, a, C_p)$
return $\hat{a}^*$

---

function `DefaultPolicy`(State s)**:**
  while s is not terminal **do**
    Choose a randomly from $A(s)$
    $s \leftarrow \mathcal{G}(s, a)$
  end
  $\Delta \leftarrow R(s)$
return Δ

---

function `Backpropagate`(Node v_i, simulation result Δ)**:**
  while $v_i \neq v_0$ **do**
    Increment v_i cumulative reward by Δ
    Increment v_i visits by 1
    $v_i \leftarrow N_p(v_i)$
  end
return

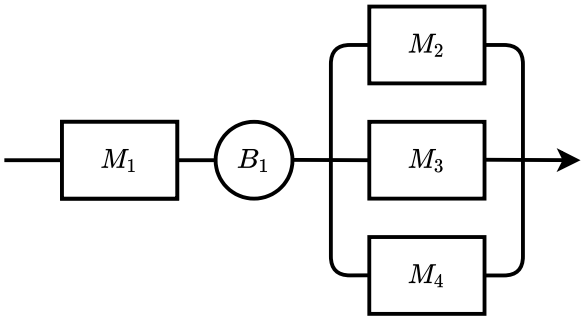

Figure 4.2: A four-machine production line.

Similar to SPTF, under LPTF we choose maintenance jobs with longer expected repair times. This strategy prefers completing corrective jobs earlier. Ties in this case are also broken with FIFO.

Birnbaum importance aims to quantify the structural importance of each component in a system [57]. It has been used effectively for maintenance prioritization in complex manufacturing systems by identifying machines with a greater potential to disrupt system throughput and selecting this critical machines for prioritized maintenance [58,59]. Consider, for example, the four-machine system depicted in Fig. 4.2. If machine M_1 is failed while M_2, M_3, and M_4 are functioning, the system-wide throughput is blocked since each part produced must pass through M_1. However, if M_2 is failed while the rest of the machines are working, the system throughput is not disrupted since parts can traverse through M_3 or M_4 as alternatives to M_2 due to the parallel arrangement of these machines.

When calculating Birnbaum importance, the operational state of each machine M_i is represented by a binary variable y_i, where

$$y_i = \begin{cases} 1 & \text{if } M_i \text{ is operational} \\ 0 & \text{if } M_i \text{ is failed} \end{cases} \tag{4.5}$$

and the state vector of the system is $\mathbf{y} = (y_1, y_2, \ldots, y_n)$. The system state is described by the binary function $\phi(\mathbf{y})$ where

$$\phi(\mathbf{y}) = \begin{cases} 1 & \text{if the system is operational} \\ 0 & \text{otherwise.} \end{cases} \tag{4.6}$$

For the purposes of this calculation, we consider the system to be operational if there is at least one path of working machines that allows parts to completely traverse through the

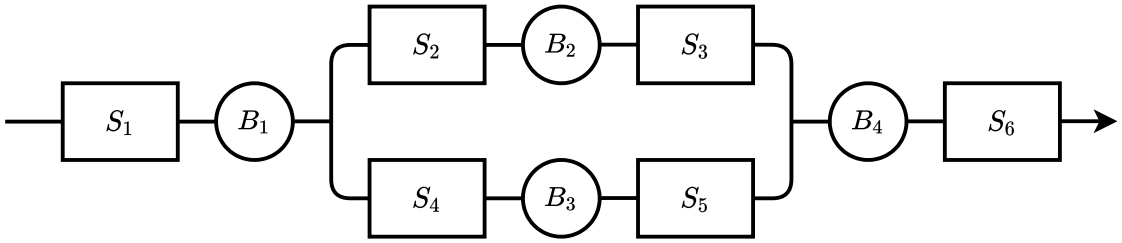

Figure 4.3: A six-station complex production line.

system. The Birnbaum importance of machine M_i is then given by

$$IB_i = \frac{\sum_{(\cdot_i,\mathbf{y})} \big(\phi(1_i,\mathbf{y}) - \phi(0_i,\mathbf{y})\big)}{2^{n-1}} \tag{4.7}$$

where $(\cdot_i,\mathbf{y})$ is the set of all 2^{n-1} state vectors with $y_i \in \{0,1\}$, $\phi(1_i,\mathbf{y})$ is the system state with $y_i = 1$, and $\phi(0_i,\mathbf{y})$ is the system state with $y_i = 0$. Therefore, the term $\phi(1_i,\mathbf{y}) - \phi(0_i,\mathbf{y})$ is equal to 1 if a failure on M_i would change the system state from operational to non-operational. In these cases, M_i is considered to be critical to the functionality of the system. Machines that are deemed critical in a higher proportion of system states are assigned a higher measure of importance and should be prioritized for maintenance. If multiple machines due for maintenance are tied for the highest importance then one is selected randomly.

Each of these scheduling rules serves as a baseline rule for obtaining a static policy to which our proposed online scheduling method is compared.

4.5 Results

In this section we compare the performance of the online scheduling improvement to static CBM policies for the complex production line shown in Fig. 4.3 under the two machine configurations outlined in Table 4.1. For each arrangement, we consider machines in a parallel station (such as machines M_3, M_4, and M_5 in this example) to be identical with a common cycle time and degradation rate.

We first optimize the static CBM policy using GA under the aforementioned static scheduling rules and then improve each policy using MCTS. We use the same maintenance threshold for each machine at a station when applying the CBM policy.

For each example shown, machines at stations S_1 to S_6 have constant cycle times of 10, 60, 180, 40, 20, and 10 minutes, respectively. Each buffer has a maximum capacity of 10 parts. Also in all scenarios, the duration of maintenance follows the geometric distribution, which

Table 4.1: Stations for system configurations A and B.

	Configuration	
	A	B
S_1	$\{M_1\}$	
S_2	$\{M_2\}$	
S_3	$\{M_3, M_4, M_5\}$	
S_4	$\{M_6, M_7\}$	$\{M_6, \ldots, M_{13}\}$
S_5	$\{M_8\}$	$\{M_{14}\}$
S_6	$\{M_9\}$	$\{M_{15}\}$

is the distribution of X number of independent Bernoulli trials with a specified probability of success that are needed until one success is observed. For success probability p, the probability mass function is $P(X = k) = (1-p)^{k-1}p$ and the expected value of X is $1/p$. We choose $p = 1/20$ for all preventive maintenance jobs and consider several success probabilities for corrective maintenance in the following experiments.

We use the matrix $\mathbf{P}_i$ for the degradation transition matrix of machine M_i of the form shown in Eq. 4.8. In this matrix, p_i is the probability of degrading by one unit at each time step or the degradation rate of M_i. We also allow for the possibility of sudden failures from any health state as indicated by the probability f_j. For each of the following results we set $f_j = 0.005 \cdot (j+1)$ for all machines. This results in an increasing probability of sudden failure at higher states of degradation. Bear in mind that our method allows much more complex upper triangular degradation matrices than Eq. 4.8.

$$\mathbf{P}_i = \begin{bmatrix} 1-(p_i+f_0) & p_i & & & f_0 \\ & 1-(p_i+f_1) & p_i & & f_1 \\ & & \ddots & & \vdots \\ & & & 1-(p_i+f_{h_{\max}-1}) & p_i+f_{h_{\max}-1} \\ & & & & 1 \end{bmatrix} \tag{4.8}$$

4.5.1 Maintenance Policy Optimization

We demonstrate optimization of the CBM policy for the system in Fig. 4.3 where each machine is subject to a degradation rate of $p_i = 0.03$, the maintenance capacity is 3, and corrective maintenance duration is geometrically distributed with success probability 1/60 (configuration 10 in Table 4.4 below). We use a warm up period of one week to achieve steady-state performance and a time horizon of one week for evaluation. Since the dynamics of the system are stationary, the duration of the evaluation horizon will not affect optimization

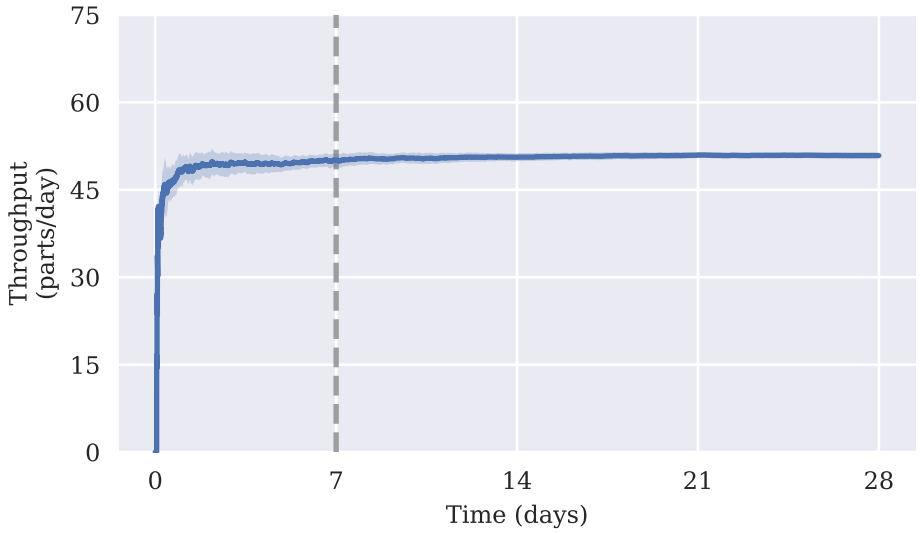

Figure 4.4: Average system throughput and chosen warm up period.

as long as the system is observed in its steady-state. To justify this warm up period, we consider the average throughput (in parts per day) over time of this system as shown in Fig. 4.4. System throughput is measured as the rate at which finished parts leave the system which is equivalent to the observed rate of production of station S_6 in this arrangement of machines. At time $t = 0$, the system is empty and all machines begin in a perfectly healthy state. As the buffers become populated, the throughput of the system eventually converges to an approximately constant level. Beyond this point the throughput is stable and so we can conclude the system is in its steady state.

Fig. 4.5 shows the convergence of the solution under Birnbaum priority scheduling averaged over 30 runs of the GA. A 95% confidence interval on the mean objective function value is indicated by the shaded region. Across these runs, the best station-level policy found was $\mathbf{x}_S = \{7, 8, 10, 10, 7, 10\}$ (recall that $h_{\max} = 10$) with an average objective function value of 419.01 units produced over the one week evaluation period.

Fig. 4.6 shows the improvement in production throughput under the optimal policy when using MCTS maintenance scheduling. MCTS was applied each time a maintenance scheduling conflict occurred. The improved policy provides an average throughput increase of 3.37 parts per day, or 23.60 parts per week. This is a 6.19% improvement for this configuration.

4.5.2 Scheduling Problem Instance

We demonstrate the online policy improvement by generating an instance of maintenance scheduling conflict in the system described in Section 4.5.1 by simulating the system until

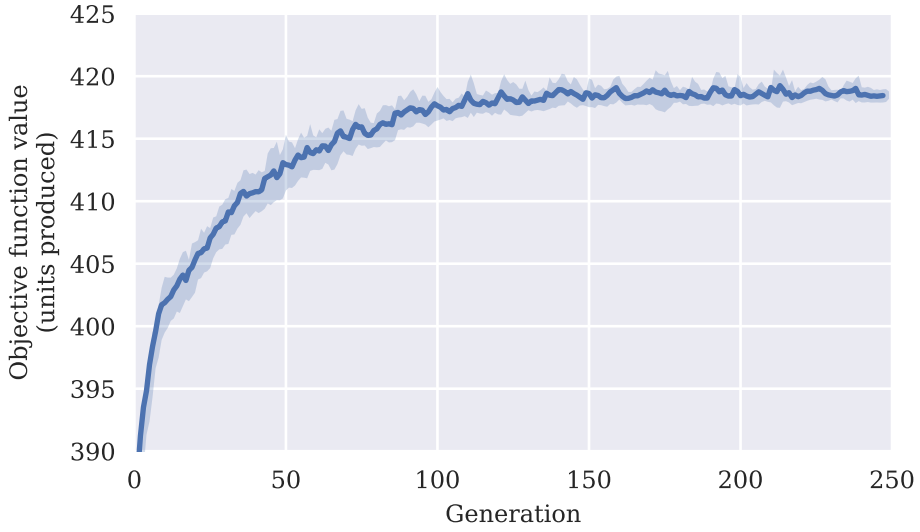

Figure 4.5: GA policy optimization convergence over 250 generations.

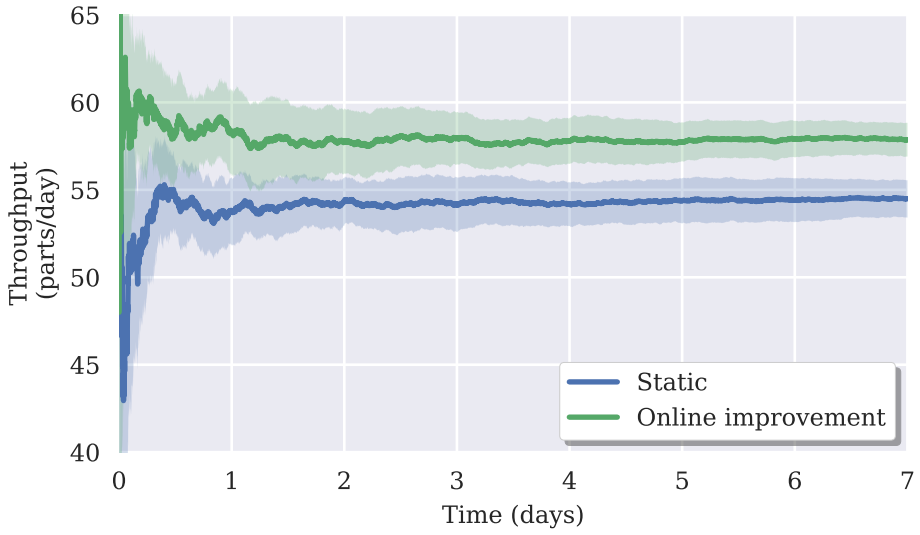

Figure 4.6: Throughput comparison of Birnbaum and MCTS scheduling.

four machines are waiting in the maintenance queue. We formulate the online scheduling problem at the point in time when the maintenance resource is released and must now decide which machine to repair next.

At simulation time $t = 366$ minutes of the replication, a maintenance resource completes a repair on machine M_{13} and observes machines M_1, M_4, M_{11}, and M_{14} with respective health states 6, 10, 10, and 4 in the maintenance queue. Machines M_8 and M_{12} are currently under repair while the remaining machines are in use. We must allocate the available capacity to conducting a repair on a machine in the queue.

According to the default Birnbaum priority scheduling policy as outlined in Table 4.3, machine M_1 should be repaired as it has the highest Birnbaum importance in this system. Instead, we apply the online improvement procedure described in Section 4.4.2 to attempt to find a better maintenance action.

In addition to the health states of each machine, we first gather the necessary additional information about the current system state including the remaining cycle time of each machine and the current level of each buffer. In the current state we also indicate that machines M_8 and M_{12} are undergoing corrective repair. This information forms the initial state from which we conduct MCTS to seek the optimal action.

With the initial state specified in the simulator, we run the MCTS for 1,000 iterations. The resulting best action is to repair machine M_{14}, which is awaiting preventive maintenance. Maintaining M_{14} is identified as the best action because it is estimated to maximize the reward function stated in Eq. 4.1, indicating that this action provides the greatest expected discounted production volume in the current state. The action is carried out in the real-world system and it continues to operate until another maintenance scheduling conflict occurs.

4.5.3 System Configuration Experiments

In this section we examine the performance of the online maintenance policy improvement on a variety of systems under different parameters. The system configuration factors and corresponding levels are given in Table 4.2. A full-factorial design of these four factors results in 24 system configurations. For each configuration, we find the optimal static CBM policy for each of the baseline scheduling heuristics described in Section 4.4.3 using the GA as described in Section 4.4.

We then simulate each system under its optimal policy using each baseline heuristic for maintenance conflict resolution and again using our online policy improvement via MCTS for comparison. We use a one week warm up period and a one week evaluation horizon to measure the production observed under each method. The results are summarized in

Table 4.2: Experimental system configuration settings.

Factor	Level		
	1	2	3
1. Degradation rate, p_i	0.03	0.07	-
2. Maintenance capacity	1	2	3
3. Corrective time to repair	$Geom(1/60)$	$Geom(1/80)$	-
4. Machine configuration	A	B	-

Table 4.3: Station-level Birnbaum importance for nine-machine line configurations A and B.

		Birnbaum importance					
		S_1	S_2	S_3	S_4	S_5	S_6
Config.	A	0.324	0.137	0.020	0.035	0.105	0.324
	B	0.359	0.110	0.016	0.001	0.140	0.359

Table 4.4.

In Table 4.4, for each static scheduling heuristic we give the baseline production volume and the production obtained under MCTS scheduling as well as the relative improvement MCTS provides. An asterisk in the p column signifies a p-value less than 0.05 for the one sided two-sample t-test for equal mean production volume, indicating that MCTS provides a significant improvement over the corresponding static heuristic. In 90 of the 96 cases studied, MCTS either improves or preserves the performance of the static policy.

The throughput improvement obtained from MCTS scheduling strongly depends on the system configuration parameters and the resulting behavior of the system. For example, if maintenance conflicts occur very rarely in a system (such as in the case of a high maintenance capacity or machines with low degradation rate) then there will be few opportunities to apply MCTS to make scheduling decisions. This will result in approximately equal performance to any static scheduling heuristic. On the other hand, if machines degrade more quickly and maintenance resources are scarce, we expect to see more maintenance conflicts and instances of online scheduling via MCTS. Among the system configurations studied in this work, MCTS scheduling provided a 10% or greater increase in throughput only in cases where maintenance resource utilization was at least 90%.

Under certain system configurations, it may also be possible to derive an optimal or near-optimal heuristic for maintenance scheduling by inspection of the system. Consider, for example, a system with n machines arranged in parallel where the cycle time of machine M_i is i minutes. A simple and effective heuristic would be to repair the machine with the shortest cycle time first, since it offers the greatest contribution to the overall system throughput.

Table 4.4: System configuration experiment results.

Config.	Factor Levels				FIFO				SPTF				LPTF				Birnbaum			
	1	2	3	4	Baseline Prod.	MCTS Prod.	Impr.	p	Baseline Prod.	MCTS Prod.	Impr.	p	Baseline Prod.	MCTS Prod.	Impr.	p	Baseline Prod.	MCTS Prod.	Impr.	p
1	1	1	1	1	386.93	422.50	9.19%	*	429.03	460.10	7.24%	*	371.27	446.63	20.30%	*	417.67	443.37	6.15%	*
2	1	1	1	2	255.03	494.97	94.08%	*	311.30	496.23	59.41%	*	248.10	484.10	95.12%	*	95.10	488.50	413.67%	*
3	1	1	2	1	274.07	334.27	21.97%	*	339.53	328.97	-3.11%		266.67	324.90	21.84%	*	298.77	337.67	13.02%	*
4	1	1	2	2	183.93	360.10	95.78%	*	220.53	364.80	65.42%	*	181.20	356.90	96.96%	*	70.03	363.77	419.42%	*
5	1	2	1	1	690.87	702.03	1.62%	*	715.97	694.70	-2.97%		700.97	710.57	1.37%	*	693.70	699.93	0.90%	
6	1	2	1	2	601.67	744.90	23.81%	*	697.97	756.47	8.38%	*	572.40	748.77	30.81%	*	657.97	746.17	13.40%	*
7	1	2	2	1	647.03	677.27	4.67%	*	682.53	676.67	-0.86%		634.03	656.93	3.61%	*	672.73	679.13	0.95%	
8	1	2	2	2	440.50	635.13	44.18%	*	605.90	632.43	4.38%	*	427.00	631.10	47.80%	*	513.13	629.20	22.62%	*
9	1	3	1	1	749.60	752.97	0.45%	*	759.50	758.37	-0.15%		752.10	754.80	0.36%		754.03	756.00	0.26%	
10	1	3	1	2	721.07	784.87	8.85%	*	735.23	795.13	8.15%	*	700.17	790.60	12.92%	*	757.07	807.20	6.62%	*
11	1	3	2	1	728.77	742.73	1.92%	*	744.13	738.57	-0.75%		741.00	742.63	0.22%		730.70	749.33	2.55%	*
12	1	3	2	2	645.70	735.10	13.85%	*	697.30	743.00	6.55%	*	586.77	730.03	24.42%	*	691.00	733.37	6.13%	*
13	2	1	1	1	162.10	166.50	2.71%		173.87	174.63	0.44%		160.70	183.70	14.31%	*	104.90	170.70	62.73%	*
14	2	1	1	2	114.33	202.50	77.11%	*	111.10	206.93	86.26%	*	120.40	208.03	72.79%	*	5.47	209.40	3730.49%	*
15	2	1	2	1	121.17	121.17	0.00%		129.10	122.33	-5.24%		121.67	119.50	-1.78%		58.47	123.23	110.78%	*
16	2	1	2	2	82.53	154.23	86.87%	*	87.67	146.73	67.38%	*	84.63	151.60	79.13%	*	5.93	136.23	2196.07%	*
17	2	2	1	1	313.90	344.80	9.84%	*	378.43	359.57	-4.99%		311.20	309.53	-0.54%		349.40	358.13	2.50%	
18	2	2	1	2	262.57	378.00	43.96%	*	322.63	390.90	21.16%	*	256.77	386.60	50.56%	*	104.53	393.03	275.99%	*
19	2	2	2	1	243.63	230.77	-5.28%		298.57	263.13	-11.87%		232.90	238.13	2.25%		267.63	254.33	-4.97%	
20	2	2	2	2	188.47	296.47	57.30%	*	225.07	294.73	30.95%	*	184.20	291.83	58.43%	*	42.80	297.37	594.78%	*
21	2	3	1	1	482.73	492.93	2.11%	*	485.57	494.50	1.84%	*	482.03	494.13	2.51%	*	485.57	496.47	2.24%	*
22	2	3	1	2	409.53	496.83	21.32%	*	446.47	503.97	12.88%	*	391.60	493.03	25.90%	*	370.37	496.20	33.98%	*
23	2	3	2	1	448.43	470.10	4.83%	*	469.60	464.47	-1.09%		442.73	468.40	5.80%	*	465.50	464.70	-0.17%	
24	2	3	2	2	279.93	394.57	40.95%	*	405.23	407.87	0.65%		285.07	401.40	40.81%	*	206.03	396.67	92.53%	*

In this scenario it is unlikely that MCTS scheduling would result in a significantly higher productivity over this static heuristic.

Based on these results, we expect our proposed method of online static policy improvement to offer the greatest throughput increase for systems with a high rate of maintenance resource utilization as well as a sufficiently complex configuration for which an effective static heuristic cannot easily be derived.

4.6 Summary

We have demonstrated the performance of GA in static CBM policy optimization and of MCTS for improving the policy online. MCTS is effective especially when the maintenance resource utilization is high. It overcomes the limitations of simple scheduling heuristics by looking ahead to the future to evaluate the outcome of sequential maintenance decisions. Although MCTS has performed well in the examples shown in this work, for very large problems it may require significant run time to converge to a good solution. This can be burdensome in instances where there is very little time to make maintenance decisions, such as the sudden occurrence of a critical failure.

Future work includes examining the effectiveness of MCTS over heuristic scheduling for arbitrary manufacturing systems. For example, if a system consists of very reliable machines that require infrequent maintenance, then it may be rare for scheduling conflicts to occur. A simple priority policy such as FIFO would be effective if the average maintenance utilization

level is very low as seen from our experiments. Under these conditions, the effort required to implement an MCTS scheduling policy may not be worthwhile. We identified several such cases for a system of particular arrangement under various parameter settings, but doing so for generalized system structure is also of interest.

Further, the application of MCTS to the maintenance scheduling problem can be improved by learning from the results of each search to influence future decisions. If the MCTS problem is formulated and solved for a particular system state, that information can likely be used again if a scheduling conflict arises in the future while the system is in a similar state. We explore this approach of reusing search experience to aid in decision making in the following chapter.

Chapter 5

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

5.1 Introduction

In this chapter we continue to study the problem of online maintenance prioritization in a manufacturing setting with capacity-constrained maintenance resources. In addition to using Monte Carlo tree search (MCTS) to seek optimal maintenance actions, we retain the experience from each search to learn an effective policy and improve the decision making process over time. As more experience 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.

In Chapter 4 we have applied MCTS to the maintenance prioritization problem for a system subject to condition-based maintenance (CBM). 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 chapter, we aim to improve upon this method by retaining and reusing experience that is gathered during each search through the application of a Case-based Reasoning (CBR) framework.

The rest of this chapter is organized as follows. Section 5.2 contains a review of relevant existing work in both maintenance prioritization and simulation-based planning that builds upon the literature reviewed thus far. Section 5.3 gives a more detailed description of the problem setting and the assumed behavior of the target system. The methodology is described in Section 5.4 and experimental results of the proposed method are presented in Section 5.5. Lastly, a summary is given in Section 5.6.

5.2 Related Work

In the following we expand upon the literature that has been reviewed thus far. We defer to Section 3.2 for a description of existing methods of capacity-constrained maintenance optimization and prioritization and to Section 4.2 for a discussion of capacity-constrained maintenance in complex manufacturing systems.

CBR is a framework for using past experience to understand and solve newly encountered problems and is founded on the premise that similar problems share similar solutions [89–91]. We adopt this framework to aid in providing maitnenance decision support for complex manufacturing systems. CBR reflects the everyday human reasoning process that is often used when we attempt to solve problems. For example, from [90]:

> 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 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, such as those discussed in Section 3.2.

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 [92–94]. Planning under limited maintenance resources is not considered, however. [95] 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. To our knowledge, CBR has not yet been applied to the problem of online maintenance prioritization.

CBR has also been effectively applied in some reinforcement learning (RL) settings which share a similar formulation with our maintenance planning problem. Many RL problems are also formulated as a Markov decision process (MDP) with the goal of choosing the best action in each state of the system. For example, [96] uses CBR to estimate the expected reward of actions in an episodic RL task. Actions are chosen at each step by identifying that which provides the greatest predicted reward. [96] addresses 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.

[97] also examines the real-time decision support problem (in the context of real-time control of energy systems) and proposes a general CBR framework for such settings which is referred to as a "real-time intelligent decision support system" (RT-IDSS). An RT-IDSS is differentiated from traditional non-real-time decision support systems primarily by the importance of the time and computational complexity considerations of such a system. Given that we are interested in using CBR to provide real-time maintenance decision support, we adopt the general RT-IDSS framework in our application.

5.3 System Description

We again study a system consisting of machines and buffers with the same behavior that has been described in previous chapters and allow for arbitrary complex configurations of these objects. However, in the following we consider only binary state machines (a machine is either operational 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 presented in Section 3.3.2. 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.

To model machine degradation we use the notion of Bernoulli machines as described by [15]. Under this reliability model, each machine has some probability of failure at the beginning of each 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 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 5.1.

Table 5.1: Bernoulli machine state summary.

State variable				
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

5.4 Methodology

As demonstrated in Section 4.4.2.1, 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 MCTS, to strategically sample the state space using the generative function $\mathcal{G}$ and evaluate alternative actions. The application of MCTS to the maintenance prioritization problem has been introduced and demonstrated in Chapter 4.

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 effectiveness while reducing the computational effort needed over time. The proposed approach uses MCTS along with CBR to retain and reuse the experience that is gained over time. This approach is illustrated in Fig. 5.1 (which has been adapted from [98]), 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 identify the best action, we query the gathered experience 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

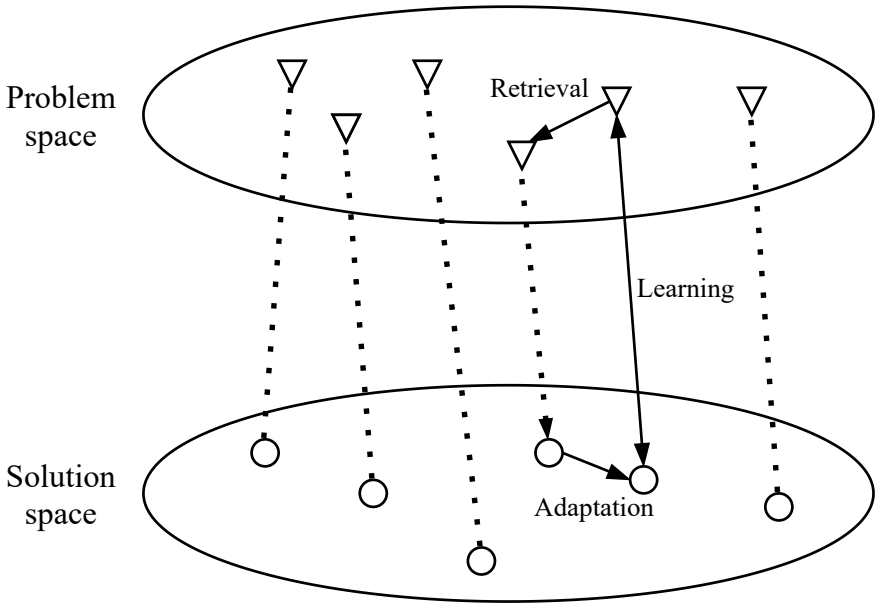

Figure 5.1: An illustration of experience storage, retrieval, and adaptation.

action.

1. Determine if the case base contains sufficient experience related to the current state.
2. If sufficient experience exists, retrieve the solution from the existing relevant cases and go to step 4.
3. Formulate MCTS to seek a solution.
   (a) Update the case base with experience gained during this search.
4. Return the estimated optimal action as the best action as for the current state and go to step 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. We use MCTS as it is described in Section 4.4.2.3, while the following sections describe the additional steps of this approach in more detail.

5.4.1 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

$$c^i = (s_i, a_i) \tag{5.1}$$

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

$$c^i = \big(s_i, \langle a_1, q_1\rangle, \ldots \langle a_m, q_m\rangle\big) \tag{5.2}$$

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 [99] 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 avoid 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 [100]. 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.

5.4.2 Measuring Action Label Confidence

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

$$a_i^* := \arg\max_{1 \leq i \leq m} \mathrm{E}[X_i] \tag{5.3}$$

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 [101], we assume

$$X_i \sim N(\eta_i, \lambda_i^2),\ \eta_i \sim N(\mu_i, \sigma_i^2) \tag{5.4}$$

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 Eq. 5.4 using

$$\mu_i = \bar{X}_i,\ \sigma_i^2 = S_i^2/n_i. \tag{5.5}$$

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

$$\begin{aligned} q_i &= \Pr(\eta_i \geq \max_{j \neq i} \eta_j) \\ &= \int \Pr(x \geq \max_{j \neq i} \eta_j) \cdot f_i(x)\ dx \\ &= \int \Big[\prod_{j \neq i} \Pr(\eta_j \leq x) \Big] \cdot f_i(x)\ dx \end{aligned} \tag{5.6}$$

where $f_i(\cdot)$ is the probability density function of the distribution $N(\mu_i, \sigma_i^2)$ as defined in

Eq. 5.4. 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 $\delta > 0$ of the best alternative. This probability is given by

$$\begin{aligned} q_i &= \Pr\big(|\eta_i - \max_{1\le j\le m} \eta_j| \le \delta\big) \\ &= \Pr\big(\eta_i + \delta \ge \max_{1\le j\le m} \eta_j\big) \\ &= \Pr\big(i \text{ is best}\big) + \Pr\big(\eta_i + \delta \ge \max_{1\le j\le m} \eta_j,\ i \text{ is not best}\big) \end{aligned} \tag{5.7}$$

The term $\Pr(i \text{ is best})$ is q_i as defined by Eq. 5.6. The second term of Eq. 5.7 is

$$\begin{aligned} \Pr\big(\eta_i \le \max_{j\ne i} \eta_j \le \eta_i + \delta\big) &= \int \Pr\big(x \le \max_{j\ne i} \eta_j \le x + \delta\big) \cdot f_i(x)\ dx \\ &= \int \Big[\sum_{\ell\ne i} b_\ell\Big] \cdot f_i(x)\ dx \end{aligned} \tag{5.8}$$

where

$$\begin{aligned} b_\ell &= \Pr\big(x \le \eta_\ell \le x + \delta,\ \eta_\ell \ge \max_{j\ne i, j\ne \ell} \eta_j\big) \\ &= \int_x^{x+\delta} \left[\prod_{j\ne i,\ j\ne \ell} \Pr(\eta_j \le y)\right] \cdot f_\ell(y)\ dy. \end{aligned} \tag{5.9}$$

We use this method 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.

5.4.3 Case-based Reasoning Procedure

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 which are summarized in Fig. 5.2 [89].

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.

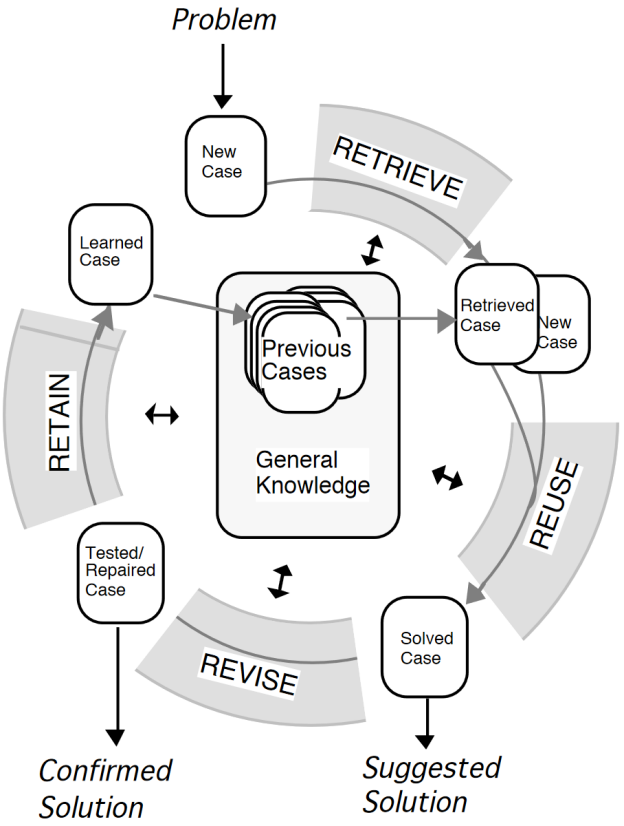

Figure 5.2: An overview of the Case-based Reasoning procedure.

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 Section 4.4.2.2 for problem descriptions in the case base under our manufacturing setting. 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 describes each step of the CBR algorithm in more detail and the procedure is by Algorithm 2.

5.4.3.1 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

$$D(\mathbf{x}_i, \mathbf{x}_j) = \Big(\sum_{k=1}^{n} |x_{ik} - x_{jk}|^p \Big)^{1/p} \tag{5.10}$$

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. Alternatively, we can measure the similarity between two vectors using

$$sim(\mathbf{x}_i, \mathbf{x}_j) = 1 - \frac{D(\mathbf{x}_i, \mathbf{x}_j)}{n^{1/p}} \tag{5.11}$$

which returns a value between 0 and 1. Thus, a lesser distance between two states indicates a higher degree of similarity.

5.4.3.2 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. 5.2.

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. 5.10. The predicted confidence for action a_i is then the average value of q_i among the k neighbors. In our experiments, we also consider the distance-weighted average where neighbors closer to the queried point have a greater influence on the predicted value than those that are farther away.

5.4.3.3 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.

5.4.3.4 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 [96] to determine the overall "value" of each case stored in the case base in the context of CBR for real-time decision support problems:

- 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 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

$$S_n(c^i) = \varphi(c^i) \cdot \sigma(c^i) \tag{5.12}$$

where

$$\varphi(c^i) = \frac{1}{k} \sum_{c^j \in NN_k(c^i)} sim(c^i, c^j), \tag{5.13}$$

$$\sigma(c^i) = \frac{1}{k} \sum_{c^j \in NN_k(c^i)} 1 - \kappa(c^i, c^j), \tag{5.14}$$

$NN_k(c^i)$ 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

$$S_e(c^i) = \sum_{c^j \in NN_k(c^i)} sim(c^i, c^j) \cdot (1 - \kappa(c^j, \hat{c}^j)) \tag{5.15}$$

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:

$$S(c^i) = S_n(c^i) + S_e(c^i). \tag{5.16}$$

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.

5.5 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.

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 5.3. We set the maintenance capacity in each experiment to 1. This system includes a complex arrangement

Algorithm 2: Case-based Reasoning with k-NN Action Prediction

function `RetrieveReuse`(Query case c^i, case base CB, neighbors k, confidence threshold ζ)**:**

- $a \leftarrow \texttt{Reuse}(\texttt{s}, \texttt{CB})$
- $a = \arg\max_{A(s)} \frac{1}{k} \sum_{c^j \in NN_k(c^i)} \left(\prod_{a' \neq a} \Pr(Q(s,a) \geq Q(s,a')) \right)$
- **if** $\prod_{a' \neq a} \Pr(Q(s,a) \geq Q(s,a')) \geq \zeta$ **then**
  - $\hat{a}^* \leftarrow a$
- **else**
  - $\hat{a}^*, \mathcal{T} \leftarrow \texttt{Mcts}(s(c^i))$
  - $CB \leftarrow \texttt{Retain}(c^i, \mathcal{T}, CB)$
- **end**

return Estimated best action $\hat{a}^*$

---

function `Retain`(Case c^i, search tree $\mathcal{T}$, case base CB)**:**

- **if** $c^i \in CB$ **then**
  - Increment node visits and rewards with results from $\mathcal{T}$
- **else**
  - $CB \leftarrow CB \bigcup c^i$
- **end**
- **if** $|CB| > maxCbSize$ **then**
  - Score each case according to $S(c^j)$
  - Remove ϱ cases with highest score
- **end**

return CB

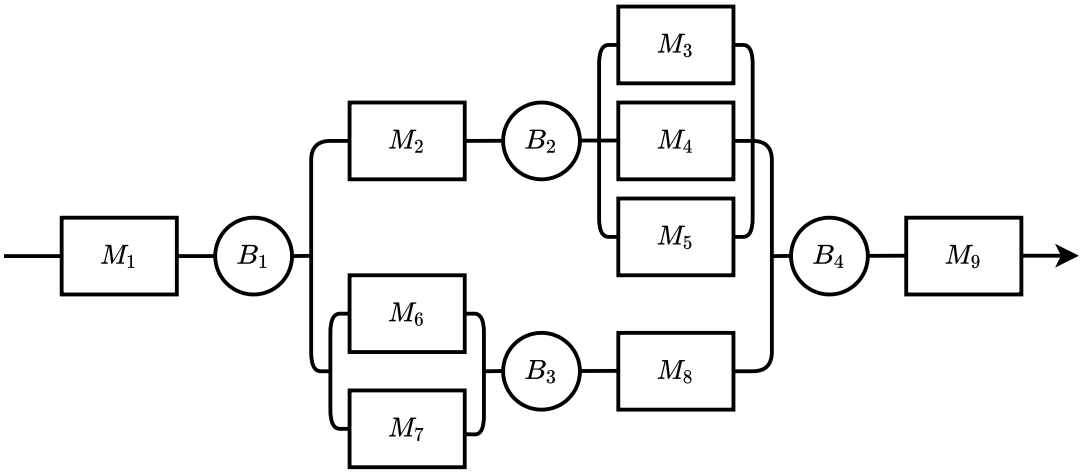

Figure 5.3: Nine-machine complex production line.

Table 5.2: 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	$Geom(1/10)$

of machines for which effective maintenance planning is not easily derived by inspection or achieved by analytical methods.

Table 5.2 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, 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 [56] as static heuristics for maintenance prioritization in this system. The Birnbaum importance measure for each machine in this arrangement is given in Table 5.3 where a greater value indicates a higher priority for maintenance. A description of the calculation of Birnbaum importance is given in Section 4.4.3. We resolve any ties in priority 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.

Table 5.3: Static priorities for the nine-machine complex system.

	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

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.

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 5.3, 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 CBR method: the retrieval threshold ζ and the maximum size of the case base. We use three levels of ζ in our experiments, 0.50, 0.90, and 0.95. A smaller ζ increases the likelihood that we will be able retrieve an action from the case base, although there is greater 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 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. 5.7) 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.

Table 5.4: kNN regressor tuning results for $\zeta = 0.50$. Overall accuracy: 76.21%.

	Training instances	Best model			
		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%

5.5.1 Case-based Reasoning 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 that each predict the likelihood 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 5.4, 5.5, and 5.6 show the resulting best parameters for $\zeta = 0.5$, 0.9, and 0.95, respectively.

5.5.2 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. 5.4. The Birnbaum importance heuristic yields the greatest average throughput with 1.0225 parts per hour.

Fig. 5.5 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

Table 5.5: kNN regressor tuning results for $\zeta = 0.90$. Overall accuracy: 74.99%.

	Training instances	Best model			
		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%

Table 5.6: kNN regressor tuning results for $\zeta = 0.95$. Overall accuracy: 80.01%.

	Training instances	Best model			
		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%

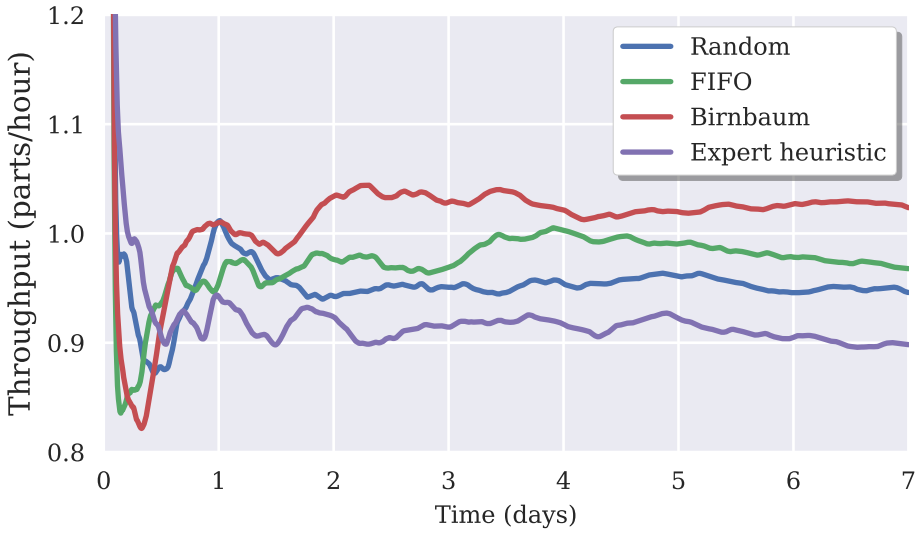

Figure 5.4: Complex system throughput comparison for static scheduling rules.

rules. The average throughput of each maintainer is given in Table 5.7.

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. 5.6. The vertical axis gives the average 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 reusing solutions from their experience.

The reduction in simulation effort is also demonstrated in Fig. 5.7, 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.

5.6 Summary

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

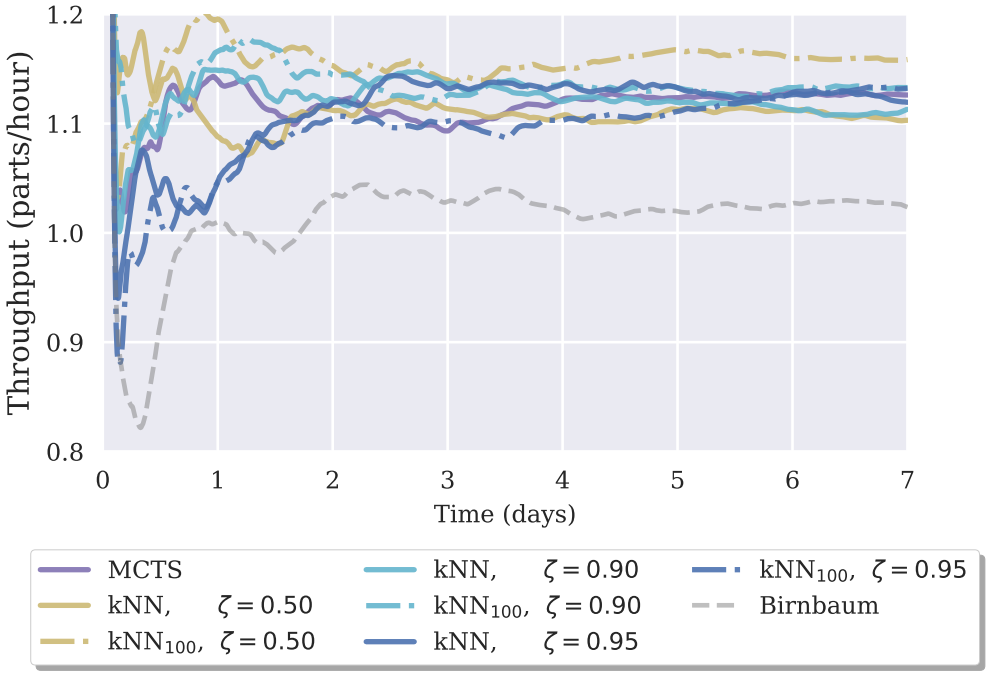

Figure 5.5: Complex system throughput comparison for CBR scheduling.

Table 5.7: Maintainer throughput comparison results.

Maintainer	ζ	Throughput (parts/hour)		Retrieval rate
		mean	se (%)	
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

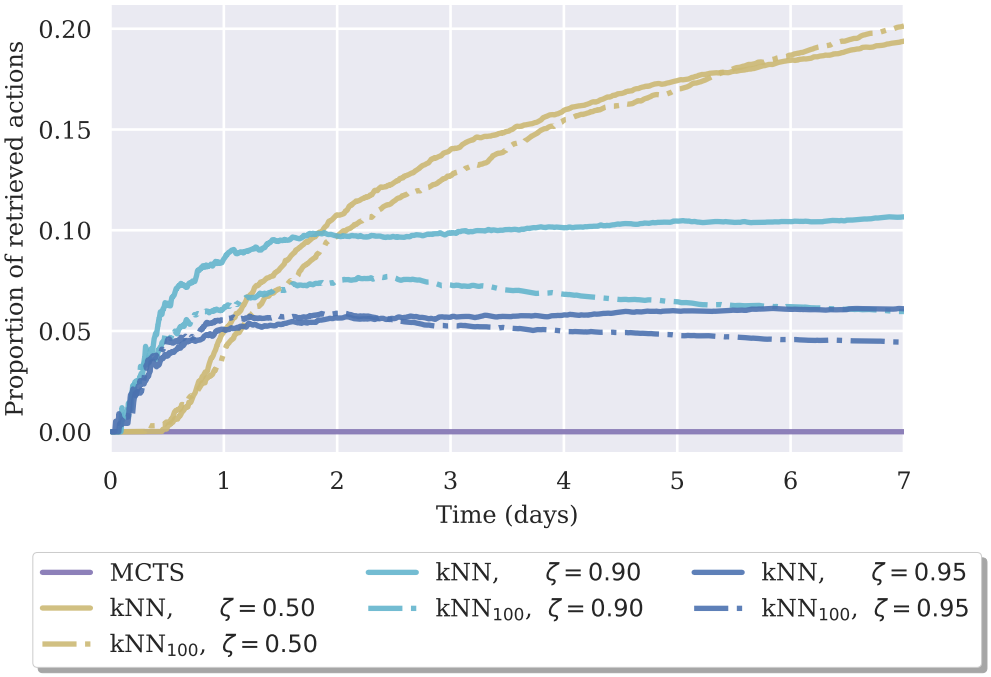

Figure 5.6: CBR retrieval rate results.

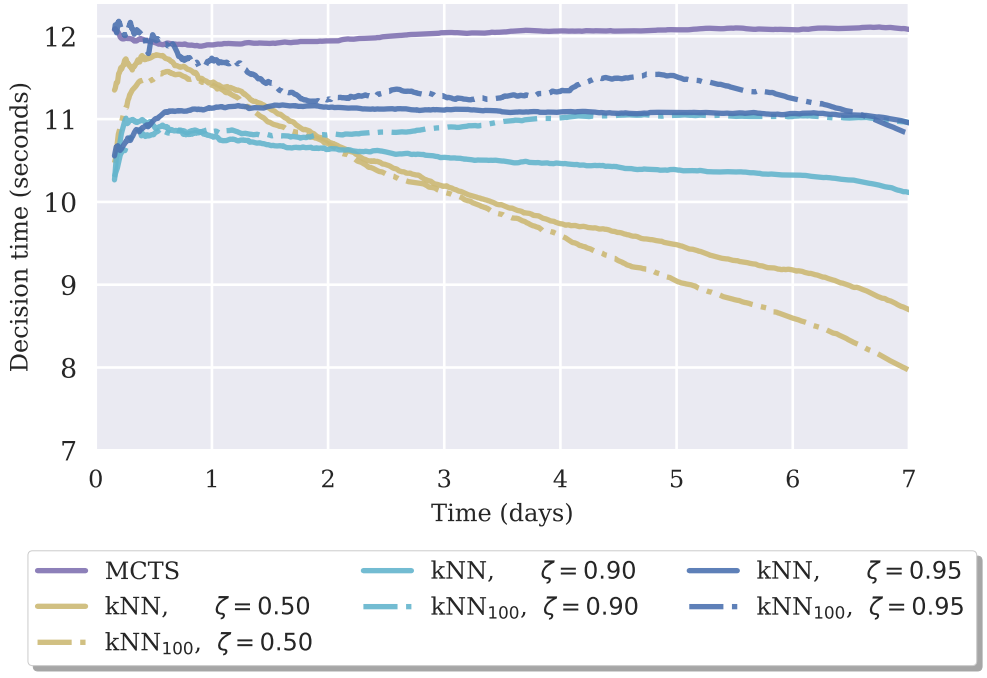

Figure 5.7: CBR decision time results.

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.

Future work 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.

Chapter 6

Conclusions and Future Work

In this dissertation, we have examined several research problems related to maintenance policy optimization and capacity-constrained maintenance resource scheduling in a manufacturing setting.

First, in Chapter 2 we introduced Simantha, an open source simulation package designed to model the operation and maintenance of complex manufacturing systems. The package provides functionality of several basic manufacturing system components including machines, buffers, sources, sinks, and maintainers. Users can instantiate and arrange these objects to replicate the behavior of a variety of system configurations. Further, the package supports serialization of objects and parallel simulation, both of which are important criteria for efficient optimization via simulation and simulation-based planning methods. The object behavior provided by Simantha is also designed to be extensible so that users have the capability to extend the basic object behavior when modeling more complex manufacturing processes.

In Chapter 3, we studied a CBM policy optimization problem for a serial production line and addressed the challenge of scheduling maintenance under a limited capacity. Both GA and GMIA were effective in finding a good policy for a particular serial line, although GMIA was shown to find a better policy under a fixed simulation budget. We also proposed a dynamic priority service discipline to resolve maintenance scheduling conflicts. The priority rule sought to schedule maintenance so that the system-level throughput disruption was minimized. When compared to a static FIFO policy, however, there was no significant improvement in the objective function. Although we consider a discrete CBM policy optimization problem for a serial production line, in practice there are many classes of maintenance strategies that can be applied to a variety of system configurations. Regardless of the maintenance strategy or system configuration, the maintenance capacity scheduling problem arises in systems where maintenance resources are limited. The maintenance scheduling

methods proposed in subsequent chapters are therefore intended to be generalizable to arbitrary maintenance strategies and system configurations.

To address the maintenance scheduling problem in complex manufacturing system arrangements, Chapter 4 proposes the application of MCTS to seek maintenance actions under an MDP formulation of the system behavior. We again consider a system consisting of machines that are subject to a CBM strategy. A two-stage approach is used, wherein the maintenance policy is first optimized using GA under a static scheduling rule and maintenance decisions are then improved in an online manner by applying MCTS when conflicts occur. This approach demonstrated an improvement in system throughput for a variety of complex system configurations when compared to several static heuristic scheduling rules. The greatest throughput improvement was observed in systems with a higher average maintenance resource utilization rate.

The methods presented in Chapter 5 aim to improve MCTS maintenance scheduling by retaining and reusing search experience through the use of CBR. In applying a CBR framework to the maintenance scheduling problem, upon encountering a system state for which a maintenance decision is needed, we can effectively query our knowledge of previously encountered similar states to determine if we can retrieve an action from this existing experience. This allows us to reduce the computational effort that is needed to conduct the online search, resulting in reduced time needed to make maintenance decisions. This result is especially valuable in maintenance settings where critical maintenance decisions must be made quickly. We again demonstrate the improvement in system throughput for a complex arrangement that our method provides over several common methods of heuristic-based maintenance prioritization.

This dissertation has demonstrated an effective strategy for scheduling capacity constrained maintenance resources in complex manufacturing systems. The work presented here is in line with the pursuit of AI applications in industrial maintenance for real time decision support. There are several items of future work that will build upon this dissertation. For instance, employing an active learning approach to the case base management procedure may improve the decision-making effectiveness of CBR. This would involve identifying prototypical cases to include in the case base that would be most informative when predicting the optimal action in other states. Such an approach could be conducted in an offline manner when the manufacturing system is operating normally and a maintenance decision is not immediately needed. The case-based reasoner would then be able to gather valuable experience offline by interacting with the simulator instead of remaining idle while waiting for a maintenance conflict to occur. Another avenue of future work involves capturing alternative forms of information to store in the case base beyond that which is gathered by MCTS.

This may include tribal knowledge that is held by maintenance technicians regarding which machines or maintenance jobs should be prioritized. Such knowledge is rarely captured and formalized, but would be valuable to a decision support system that seeks to learn from prior decision-making experience. In some cases, substantial historical maintenance data is also available, but is rarely used to support maintenance planning and scheduling. This information may be useful for initializing the case base for CBR instead of starting with no prior knowledge.

Appendix: Simantha Test Code

The following sections include the code used to verify the behavior of the Simantha package as described in Chapter 2.4. The current version of Simantha is available at `https://github.com/m-hoff/simantha`.

Deterministic Serial Throughput

```
from simantha import Source, Machine, Buffer, Sink, System, utils

source = Source()
M1 = Machine(cycle_time=1)
B1 = Buffer(capacity=5)
M2 = Machine(cycle_time=1)
B2 = Buffer(capacity=5)
M3 = Machine(cycle_time=1)
sink = Sink()

source.define_routing(downstream=[M1])
M1.define_routing(upstream=[source], downstream=[B1])
B1.define_routing(upstream=[M1], downstream=[M2])
M2.define_routing(upstream=[B1], downstream=[B2])
B2.define_routing(upstream=[M2], downstream=[M3])
M3.define_routing(upstream=[B2], downstream=[sink])
sink.define_routing(upstream=[M1])

system = System(objects=[source, M1, B1, M2, B2, M3, sink])

system.simulate(simulation_time=utils.DAY)
```

Output:

```
Simulation finished in 0.20s
Parts produced: 1438
```

Deterministic Parallel Throughput

```python
from simantha import Source, Machine, Buffer, Sink, System, utils

source = Source()
M1 = Machine(cycle_time=1)
M2 = Machine(cycle_time=1)
M3 = Machine(cycle_time=1)
sink = Sink()

source.define_routing(downstream=[M1, M2, M3])
for Mi in [M1, M2, M3]:
    Mi.define_routing(upstream=[source], downstream=[sink])
sink.define_routing(upstream=[M1, M2, M3])

system = System(objects=[source, M1, M2, M3, sink])

system.simulate(simulation_time=utils.DAY)
```

Output:

```
Simulation finished in 0.17s
Parts produced: 4320
```

Stochastic Degrading Machine Availability

```python
from simantha import Source, Machine, Sink, System, utils

p_f = 1/90
p_r = 1/10

# Markovian degradation transition matrix
Q = [
    [1-p_f, p_f],
    [0,     1  ]
]

source = Source()
M1 = Machine(
    name='M1',
    cycle_time=1,
    degradation_matrix=Q,
    cm_distribution={'geometric': p_r}
```

```
)
sink = Sink()

source.define_routing(downstream=[M1])
M1.define_routing(upstream=[source], downstream=[sink])
sink.define_routing(upstream=[M1])

system = System(objects=[source, M1, sink])

n = 100
results = system.iterate_simulation(
    replications=n,
    simulation_time=utils.MONTH,
    jobs=10
)
availability = sum([r[2][0] for r in results]) / n
print(f'Average availability: {availability:.4%}')

_, p_value = scipy.stats.ttest_1samp([r[2][0] for r in results], 0.90)
print(f'\nOne sample t-test p-value for availability: {p_value:.4f}')

_, p_value = scipy.stats.ttest_1samp([r[0]/utils.MONTH for r in results],
    0.90)
print(f'One sample t-test p-value for throughput: {p_value:.4f}')
```

Output:

```
Finished 100 replications in 23.94s
Average availability: 90.0368%

One sample t-test p-value for availability: 0.5371
One sample t-test p-value for throughput: 0.2107
```

Degrading Bottleneck Machine Throughput

```
from simantha import Source, Machine, Buffer, Sink, System, utils

p_f = 1/90
p_r = 1/10

Q = [
    [1-p_f, p_f],
    [0,     1  ]
```

```
]

source = Source()
M1 = Machine(
    name='M1',
    cycle_time=1,
    degradation_matrix=Q,
    cm_distribution={'geometric': p_r}
)
B1 = Buffer(name='B1', capacity=30)
M2 = Machine(
    name='M2',
    cycle_time=3,
    degradation_matrix=Q,
    cm_distribution={'geometric': p_r}
)
B2 = Buffer(name='B2', capacity=30)
M3 = Machine(
    name='M3',
    cycle_time=2,
    degradation_matrix=Q,
    cm_distribution={'geometric': p_r}
)
sink = Sink()

source.define_routing(downstream=[M1])
M1.define_routing(upstream=[source], downstream=[B1])
B1.define_routing(upstream=[M1], downstream=[M2])
M2.define_routing(upstream=[B1], downstream=[B2])
B2.define_routing(upstream=[M2], downstream=[M3])
M3.define_routing(upstream=[B2], downstream=[sink])
sink.define_routing(upstream=[M3])

system = System(objects=[source, M1, B1, M2, B2, M3, sink])

system.simulate(simulation_time=utils.MONTH)
```

Output:

```
Simulation finished in 2.67s
Parts produced: 12681
```

Bibliography

[1] T. Wireman, "Maintenance organizations," in *Benchmarking Best Practices in Maintenance Management*, ch. 3, pp. 55–84, Industrial Press Inc., 2004.

[2] J. H. Saleh and K. Marais, "Highlights from the early (and pre-) history of reliability engineering," *Reliability Engineering & System Safety*, vol. 91, no. 2, pp. 249–256, 2006.

[3] F. S. Nowlan and H. F. Heap, "Reliability-centered maintenance," tech. rep., United Air Lines Inc, San Francisco, CA, 1978.

[4] J.-H. Shin and H.-B. Jun, "On condition based maintenance policy," *Journal of Computational Design and Engineering*, vol. 2, no. 2, pp. 119–127, 2015.

[5] G. W. Vogl, B. A. Weiss, and M. Helu, "A review of diagnostic and prognostic capabilities and best practices for manufacturing," *Journal of Intelligent Manufacturing*, vol. 30, no. 1, pp. 79–95, 2019.

[6] R. K. Mobley, *An Introduction to Predictive Maintenance*. Elsevier, 2002.

[7] X. Jin, D. Siegel, B. A. Weiss, E. Gamel, W. Wang, J. Lee, and J. Ni, "The present status and future growth of maintenance in US manufacturing: results from a pilot survey," *Manufacturing Review*, vol. 3, 2016.

[8] K. A. H. Kobbacy, "Artificial intelligence in maintenance," in *Complex System Maintenance Handbook*, ch. 9, pp. 209–231, Springer London, 2008.

[9] S.-H. Ding and S. Kamaruddin, "Maintenance policy optimization—literature review and directions," *The International Journal of Advanced Manufacturing Technology*, vol. 76, no. 5, pp. 1263–1283, 2015.

[10] T. Sexton, M. P. Brundage, M. Hoffman, and K. C. Morris, "Hybrid datafication of maintenance logs from AI-assisted human tags," in *2017 IEEE International Conference on Big Data (Big Data)*, pp. 1769–1777, IEEE, 2017.

[11] K. Khazraei and J. Deuse, "A strategic standpoint on maintenance taxonomy," *Journal of Facilities Management*, vol. 9, no. 2, pp. 96–113, 2011.

[12] U. M. Al-Turki, T. Ayar, B. S. Yilbas, and A. Z. Sahin, *Integrated Maintenance Planning in Manufacturing Systems*. Springer, 2014.

[13] "Railway applications – Heating, ventilation and air conditioning systems for rolling stock – Part 1: Terms and definitions," standard, International Organization for Standardization, Aug. 2017.

[14] Wikipedia contributors, "Discrete manufacturing — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=Discrete_manufacturing&oldid=960518724. [Online; accessed 15-June-2021].

[15] J. Li and S. M. Meerkov, *Production Systems Engineering*. Springer Science & Business Media, 2008.

[16] M. Bevilacqua and M. Braglia, "The analytic hierarchy process applied to maintenance strategy selection," *Reliability Engineering & System Safety*, vol. 70, no. 1, pp. 71–83, 2000.

[17] M. Shafiee, "Maintenance strategy selection problem: An MCDM overview," *Journal of Quality in Maintenance Engineering*, vol. 21, no. 4, pp. 378–402, 2015.

[18] R. Velmurugan and T. Dhingra, "Maintenance strategy selection and its impact in maintenance function," *International Journal of Operations & Production Management*, vol. 35, no. 12, pp. 1622–1661, 2015.

[19] C. Vishnu and V. Regikumar, "Reliability based maintenance strategy selection in process plants: A case study," *Procedia Technology*, vol. 25, pp. 1080–1087, 2016.

[20] D. I. Cho and M. Parlar, "A survey of maintenance models for multi-unit systems," *European Journal of Operational Research*, vol. 51, no. 1, pp. 1–23, 1991.

[21] M. C. Olde Keizer, S. D. P. Flapper, and R. H. Teunter, "Condition-based maintenance policies for systems with multiple dependent components: A review," *European Journal of Operational Research*, vol. 261, no. 2, pp. 405–420, 2017.

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

[23] G. Dagkakis and C. Heavey, "A review of open source discrete event simulation software for operations research," *Journal of Simulation*, vol. 10, no. 3, pp. 193–206, 2016.

[24] "SimPy." https://simpy.readthedocs.io/en/latest/, 2021. [Online; accessed 15-June-2021].

[25] "ManPy - discrete event simulation in Python." https://www.manpy-simulation.org/, 2021. [Online; accessed 15-June-2021].

[26] A. H. Buss, "Modeling with event graphs," in *Proceedings of the 1996 Winter Simulation Conference*, pp. 153–160, IEEE, 1996.

[27] T. J. Schriber, D. T. Brunner, and J. S. Smith, "Inside discrete-event simulation software: How it works and why it matters," in *Proceedings of the 2017 Winter Simulation Conference*, pp. 735–749, IEEE, 2017.

[28] W. J. Hopp and M. L. Spearman, *Factory Physics*. Waveland Press, 2011.

[29] M. J. Kallen and J. M. van Noortwijk, "Optimal periodic inspection of a deterioration process with sequential condition states," *International Journal of Pressure Vessels and Piping*, vol. 83, no. 4, pp. 249–255, 2006.

[30] A. K. Jardine, D. Lin, and D. Banjevic, "A review on machinery diagnostics and prognostics implementing condition-based maintenance," *Mechanical Systems and Signal Processing*, vol. 20, no. 7, pp. 1483–1510, 2006.

[31] K. Liu, A. Chehade, and C. Song, "Optimize the signal quality of the composite health index via data fusion for degradation modeling and prognostic analysis," *IEEE Transactions on Automation Science and Engineering*, vol. 14, no. 3, pp. 1504–1514, 2015.

[32] L. Yang, X. Ma, and Y. Zhao, "A condition-based maintenance model for a three-state system subject to degradation and environmental shocks," *Computers & Industrial Engineering*, vol. 105, pp. 210–222, 2017.

[33] N. Rasmekomen and A. K. Parlikad, "Condition-based maintenance of multi-component systems with degradation state-rate interactions," *Reliability Engineering & System Safety*, vol. 148, pp. 1–10, 2016.

[34] M. Kurt and J. P. Kharoufeh, "Monotone optimal replacement policies for a Markovian deteriorating system in a controllable environment," *Operations Research Letters*, vol. 38, no. 4, pp. 273–279, 2010.

[35] M. L. Neves, L. P. Santiago, and C. A. Maia, "A condition-based maintenance policy and input parameters estimation for deteriorating systems under periodic inspection," *Computers & Industrial Engineering*, vol. 61, no. 3, pp. 503–511, 2011.

[36] V. Makis, "Optimal condition-based maintenance policy for a partially observable system with two sampling intervals," *The International Journal of Advanced Manufacturing Technology*, vol. 78, no. 5-8, pp. 795–805, 2015.

[37] X. Gu, S. Lee, X. Liang, M. Garcellano, M. Diederichs, and J. Ni, "Hidden maintenance opportunities in discrete and complex production lines," *Expert Systems with Applications*, vol. 40, no. 11, pp. 4353–4361, 2013.

[38] J. Koochaki, J. A. Bokhorst, H. Wortmann, and W. Klingenberg, "Condition based maintenance in the context of opportunistic maintenance," *International Journal of Production Research*, vol. 50, no. 23, pp. 6918–6929, 2012.

[39] M. Shafiee, M. Finkelstein, and C. Bérenguer, "An opportunistic condition-based maintenance policy for offshore wind turbine blades subjected to degradation and environmental shocks," *Reliability Engineering & System Safety*, vol. 142, pp. 463–471, 2015.

[40] M. Shafiee and M. Finkelstein, "A proactive group maintenance policy for continuously monitored deteriorating systems: Application to offshore wind turbines," *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, vol. 229, no. 5, pp. 373–384, 2015.

[41] W. Rice, C. Cassady, and J. Nachlas, "Optimal maintenance plans under limited maintenance time," in *Proceedings of the Seventh Industrial Engineering Research Conference*, pp. 1–3, 1998.

[42] W. Cao, X. Jia, Q. Hu, J. Zhao, and Y. Wu, "A literature review on selective maintenance for multi-unit systems," *Quality and Reliability Engineering International*, vol. 34, no. 5, pp. 824–845, 2018.

[43] M. Marseguerra, E. Zio, and L. Podofillini, "Condition-based maintenance optimization by means of genetic algorithms and Monte Carlo simulation," *Reliability Engineering and System Safety*, vol. 77, no. 2, pp. 151–165, 2002.

[44] K. S. de Smidt-Destombes, M. C. van der Heijden, and A. van Harten, "On the availability of a k-out-of-N system given limited spares and repair capacity under a condition based maintenance strategy," *Reliability Engineering & System Safety*, vol. 83, no. 3, pp. 287–300, 2004.

[45] R. Moghaddass, M. J. Zuo, and M. Pandey, "Optimal design and maintenance of a repairable multi-state system with standby components," *Journal of Statistical Planning and Inference*, vol. 142, no. 8, pp. 2409–2420, 2012.

[46] A. K. W. Chong, A. H. Mohammed, M. N. Abdullah, and M. S. A. Rahman, "Maintenance prioritization–a review on factors and methods," *Journal of Facilities Management*, vol. 17, no. 1, pp. 18–39, 2019.

[47] T. L. Saaty, "Decision making with the analytic hierarchy process," *International Journal of Services Sciences*, vol. 1, no. 1, pp. 83–98, 2008.

[48] S. Sharma, A. Sisodia, *et al.*, "Prioritization of tools in joint production–maintenance environment of auto component manufacturer using AHP–Fuzzy–TOPSIS," *Intelligent Industrial Systems*, vol. 2, no. 1, pp. 73–84, 2016.

[49] A. Khanlari, K. Mohammadi, and B. Sohrabi, "Prioritizing equipments for preventive maintenance (PM) activities using fuzzy rules," *Computers & Industrial Engineering*, vol. 54, no. 2, pp. 169–184, 2008.

[50] Q. Chang, G. Xiao, S. Biller, and L. Li, "Energy saving opportunity analysis of automotive serial production systems," *IEEE Transactions on Automation Science and Engineering*, vol. 10, no. 2, pp. 334–342, 2013.

[51] X. Gu, X. , and J. Ni, "Prediction of passive maintenance opportunity windows on bottleneck machines in complex manufacturing systems," *Journal of Manufacturing Science and Engineering*, vol. 137, no. 3, p. 031017, 2015.

[52] R. Dekker, "Integrating optimisation, priority setting, planning and combining of maintenance activities," *European Journal of Operational Research*, vol. 82, no. 2, pp. 225–240, 1995.

[53] R. Dekker and P. A. Scarf, "On the impact of optimisation models in maintenance decision making: the state of the art," *Reliability Engineering & System Safety*, vol. 60, no. 2, pp. 111–119, 1998.

[54] S.-H. G. Teng and S.-Y. M. Ho, "Failure mode and effects analysis," *International Journal of Quality & Reliability Management*, vol. 13, no. 5, pp. 8–26, 1996.

[55] S.-H. Ding, S. Kamaruddin, and I. A. Azid, "Maintenance policy selection model–a case study in the palm oil industry," *Journal of Manufacturing Technology Management*, vol. 25, no. 3, pp. 415–435, 2014.

[56] Z. Yang, Q. Chang, D. Djurdjanovic, J. Ni, and J. Lee, "Maintenance priority assignment utilizing on-line production information," *Journal of Manufacturing Science and Engineering*, vol. 129, no. 2, pp. 435–446, 2007.

[57] Z. W. Birnbaum, "On the importance of different components in a multicomponent system," tech. rep., Washington University Seattle Lab of Statistical Research, 1968.

[58] K.-A. Nguyen, P. Do, and A. Grall, "Condition-based maintenance for multi-component systems using importance measure and predictive information," *International Journal of Systems Science: Operations & Logistics*, vol. 1, no. 4, pp. 228–245, 2014.

[59] S. Si, M. Liu, Z. Jiang, T. Jin, and Z. Cai, "System reliability allocation and optimization based on generalized Birnbaum importance measure," *IEEE Transactions on Reliability*, vol. 68, no. 3, pp. 831–843, 2019.

[60] Y. Lei, J. Liu, J. Ni, and J. Lee, "Production line simulation using STPN for maintenance scheduling," *Journal of Intelligent Manufacturing*, vol. 21, no. 2, pp. 213–221, 2010.

[61] X. S. Si, W. Wang, C. H. Hu, and D. H. Zhou, "Remaining useful life estimation – a review on the statistical data driven approaches," *European Journal of Operational Research*, vol. 213, no. 1, pp. 1–14, 2011.

[62] Y. Zhou, T. R. Lin, Y. Sun, and L. Ma, "Maintenance optimisation of a parallel-series system with stochastic and economic dependence under limited maintenance capacity," *Reliability Engineering & System Safety*, vol. 155, pp. 137–146, 2016.

[63] C. D. Dao and M. J. Zuo, "Optimal selective maintenance for multi-state systems in variable loading conditions," *Reliability Engineering & System Safety*, vol. 166, pp. 171–180, 2017.

[64] C. D. Dao and M. J. Zuo, "Selective maintenance of multi-state systems with structural dependence," *Reliability Engineering & System Safety*, vol. 159, pp. 184–195, 2017.

[65] Y. Zhou, Y. Guo, T. R. Lin, and L. Ma, "Maintenance optimisation of a series production system with intermediate buffers using a multi-agent FMDP," *Reliability Engineering & System Safety*, vol. 180, pp. 39–48, 2018.

[66] R. P. Nicolai and R. Dekker, "Optimal maintenance of multi-component systems: A review," in *Complex System Maintenance Handbook*, ch. 11, pp. 263–286, Springer London, 2008.

[67] H.-S. Baik, H. S. Jeong, and D. M. Abraham, "Estimating transition probabilities in Markov chain-based deterioration models for management of wastewater systems," *Journal of Water Resources Planning and Management*, vol. 132, no. 1, pp. 15–24, 2006.

[68] Y. Tsuda, K. Kaito, K. Aoki, and K. Kobayashi, "Estimating Markovian transition probabilities for bridge deterioration forecasting," *Structural Engineering/Earthquake Engineering*, vol. 23, no. 2, pp. 241–256, 2006.

[69] M. P. Brundage, Q. Chang, Y. Li, J. Arinez, and G. Xiao, "Implementing a real-time, energy-efficient control methodology to maximize manufacturing profits," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 46, no. 6, pp. 855–866, 2016.

[70] T. Chitra, "Life based maintenance policy for minimum cost," in *Annual Reliability and Maintainability Symposium*, pp. 470–474, IEEE, 2003.

[71] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd ed., 2010.

[72] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[73] S.-H. Kim and B. L. Nelson, "Selecting the best system," *Handbooks in Operations Research and Management Science*, vol. 13, pp. 501–534, 2006.

[74] P. Salemi, E. Song, B. L. Nelson, and J. Staum, "Gaussian Markov random fields for discrete optimization via simulation: Framework and algorithms," *Operations Research*, vol. 67, no. 1, pp. 250–266, 2019.

[75] S. Albers, "Online scheduling," in *Introduction to Scheduling* (Y. Robert and F. Vivien, eds.), ch. 3, CRC Press, 2009.

[76] A. H. Elwany, N. Z. Gebraeel, and L. M. Maillart, "Structured replacement policies for components with complex degradation processes and dedicated sensors," *Operations Research*, vol. 59, no. 3, pp. 684–695, 2011.

[77] N. Chen, Z.-S. Ye, Y. Xiang, and L. Zhang, "Condition-based maintenance using the inverse Gaussian degradation model," *European Journal of Operational Research*, vol. 243, no. 1, pp. 190–199, 2015.

[78] C. E. Ebeling, "Reliability of systems," in *An Introduction to Reliability and Maintainability Engineering*, ch. 5, Tata McGraw-Hill Education, 2004.

[79] A. Gupta and C. Lawsirirat, "Strategically optimum maintenance of monitoring-enabled multi-component systems using continuous-time jump deterioration models," *Journal of Quality in Maintenance Engineering*, vol. 12, no. 3, pp. 306–329, 2006.

[80] K.-A. Nguyen, P. Do, and A. Grall, "Multi-level predictive maintenance for multi-component systems," *Reliability Engineering & System Safety*, vol. 144, pp. 83–94, 2015.

[81] Y. Liu and H.-Z. Huang, "Optimal replacement policy for multi-state system under imperfect maintenance," *IEEE Transactions on Reliability*, vol. 59, no. 3, pp. 483–495, 2010.

[82] M. Hoffman, E. Song, M. P. Brundage, and S. Kumara, "Condition-based maintenance policy optimization using genetic algorithms and Gaussian Markov improvement algorithm," in *Annual Conference of the PHM Society*, 2018.

[83] M. Kearns, Y. Mansour, and A. Y. Ng, "A sparse sampling algorithm for near-optimal planning in large Markov decision processes," *Machine Learning*, vol. 49, no. 2, pp. 193–208, 2002.

[84] L. Kocsis, C. Szepesvári, and J. Willemson, "Improved Monte-Carlo search," 2006.

[85] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of Monte Carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.

[86] B. Saenz de Ugarte, A. Artiba, and R. Pellerin, "Manufacturing execution system–a literature review," *Production Planning and Control*, vol. 20, no. 6, pp. 525–539, 2009.

[87] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, "Monte-Carlo tree search: A new framework for game AI," in *Proceedings of the Annual Artificial Intelligence and Interactive Digital Entertainment Conference*, 2008.

[88] H. S. Chang, M. C. Fu, J. Hu, and S. I. Marcus, "An adaptive sampling algorithm for solving Markov decision processes," *Operations Research*, vol. 53, no. 1, pp. 126–139, 2005.

[89] A. Aamodt and E. Plaza, "Case-based reasoning: Foundational issues, methodological variations, and system approaches," *AI Communications*, vol. 7, no. 1, pp. 39–59, 1994.

[90] J. L. Kolodner, "An introduction to case-based reasoning," *Artificial Intelligence Review*, vol. 6, no. 1, pp. 3–34, 1992.

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

[92] M. Bengtsson, E. Olsson, P. Funk, and M. Jackson, "Technical design of condition based maintenance system—a case study using sound analysis and case-based reasoning," in *8th International Conference of Maintenance and Reliability*, 2004.

[93] Y. Tsai, "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*, vol. 223, no. 10, pp. 2431–2441, 2009.

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

[95] R. Yu, B. Iung, and H. Panetto, "A multi-agents based E-maintenance system with case-based reasoning decision support," *Engineering Applications of Artificial Intelligence*, vol. 16, no. 4, pp. 321–333, 2003.

[96] T. Gabel and M. Riedmiller, "CBR for state value function approximation in reinforcement learning," in *Proceedings of the 6th International Conference on Case-Based Reasoning Research and Development*, pp. 206–221, Springer, 2005.

[97] P. Varshavskii and A. Eremeev, "Modeling of case-based reasoning in intelligent decision support systems," *Scientific and Technical Information Processing*, vol. 37, no. 5, pp. 336–345, 2010.

[98] Y. Avramenko, L. Nyström, and A. Kraslawski, "Selection of internals for reactive distillation column—case-based reasoning approach," *Computers & Chemical Engineering*, vol. 28, no. 1, pp. 37–44, 2004.

[99] Q. Nguyen, H. Valizadegan, and M. Hauskrecht, "Learning classification with auxiliary probabilistic information," in *2011 IEEE 11th International Conference on Data Mining*, pp. 477–486, IEEE, 2011.

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

[101] K.-K. Kim, K. Taeho, and E. Song, "Selection of the most probable best under input uncertainty," in *Proceedings of the 2021 Winter Simulation Conference* (Submitted), 2021.

Vita

Michael Hoffman

Education

The Pennsylvania State University, University Park, PA

Ph.D., Industrial Engineering and Operations Research	*August 2021*
M.S., Industrial Engineering	*December 2018*
B.S., Industrial Engineering	*December 2015*

Research Positions

National Institute of Standards and Technology

Graduate Measurement Science and Engineering Fellow *May 2017 - July 2021*

Laboratory for Intelligent Systems and Analytics

Member *January 2016 - August 2021*

Applied Research Laboratory, Penn State University

Walker Graduate Assistant	*January 2017 - May 2017*
Graduate Assistant	*January 2016 - December 2016*

Selected Publications

M. Hoffman, E. Song, M. Brundage, and S. Kumara, "Online Improvement of Condition-based Maintenance Policy via Monte Carlo Tree Search," in IEEE Transactions on Automation Science and Engineering, 2021 (Accepted)

M. Hoffman, E. Song, M. Brundage, and S. Kumara, "Condition-based Maintenance Policy Optimization Using Genetic Algorithms and Gaussian Markov Improvement Algorithm," in Prognostics and Health Management Society Conference, 2018.

M. Brundage, K. C. Morris, T. Sexton, S. Moccozet, and **M. Hoffman**, "Developing Maintenance Key Performance Indicators from Maintenance Work Order Data," in International Manufacturing Science and Engineering Conference, 2018.

T. Sexton, M. Brundage, **M. Hoffman**, and K. C. Morris, "Hybrid Datafication of Maintenance Logs from AI-assisted Human Tags," in 2017 IEEE International Conference on Big Data, 2017.

Awards

Graduate Fellowship for STEM Diversity *May 2017 - July 2021*