The Pennsylvania State University
The Graduate School

MANUFACTURING SYSTEMS MODELING AND ANALYSIS

A Dissertation in
Industrial Engineering
by
Juxihong Julaiti

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

December 2021

The dissertation of Juxihong Julaiti was reviewed and approved by the following:

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

GuoDong Pang
Professor of Industrial and Manufacturing Engineering
Dissertation Co-Adviser, Committee Co-Chair

Eunhye Song
Assistant Professor of Industrial and Manufacturing Engineering

Vasant Honavar
Professor and Chair of Information Sciences and Technology

BiCheng Chen
CEO/CTO, CPNet
Special Member

Steve Landry
Professor and Head of the Department of Industrial and Manufacturing Engineering

Abstract

In the United States, an increasing amount of industries are becoming high-mix and low-volume (HMLV) facilities to provide various products and to stay competitive. The heterogeneity of products introduces frequent reconfiguration to the production line and therefore increases the chance of unplanned downtime. Because of the significant cost of unplanned downtime, any effort to reduce it is a welcome endeavor to U.S. manufacturers. This thesis contains three chapters, the first two address HMLV reconfiguration problem. The third chapter deals with adaptive controls in unreliable single server queues, which can represent a typical HMLV environment.

In the first chapter, a parallel machine scheduling problem with stochastic machine breakdowns is studied to minimize the weighted tardiness. We propose a reinforcement learning-based framework with a novel sampling method. The results indicate that the new sampling approach expedites the learning process and the resulting policy significantly outperforms static dispatching rules.

In a HMLV facility, similar jobs are often scheduled together to decrease additional setup times. However, in dry machining processes, utilizing a tool for a prolonged period of time overheats the tool and increases chances of tool damage and scrapped parts. Therefore, the optimal schedule should avoid tool overheating. In the second chapter, we propose a mixed integer programming model to minimize the makespan in a job-shop scheduling environment with overheating constraints. Numerical studies are conducted to validate the model.

In queuing systems, the unplanned machine downtime is commonly modeled using queues with interruptions and it has received considerable attention since the late 1950s. In the third chapter, we study adaptive service rate control problems for single server queues with server breakdowns. We focus on a particular dependent structure where the breakdown rate of the server is a linear function of the service rate. Since the relation between these two rates might be unknown in practice, we develop online algorithms to obtain the optimal policy. Numerical studies are conducted to analyze the optimal policy and validate proposed algorithms.

Table of Contents

List of Figures

List of Tables

Acknowledgments

Throughout the time I had at Penn State, I have received a tremendous amount of support and assistance from my advisors, professors, family members, friends, colleagues from TE connectivity. Without having you on this journey, I will not be where I am today. For this, I am forever grateful.

First and foremost, I would like to say thank you to my advisor Professor Soundar Kumara for accepting me into his group. During my time in the lab, Dr. Kumara has given me intellectual freedom in my work, supported my attendance at various conferences, engaged me in new ideas, and demanded a high quality of work in my endeavors. And because of this, I have had a rewarding graduate school experience. I would also like to say thank you to my co-advisor Professor Guodong Pang for his patience and guidance on this fruitful journey. Because of Dr. Pang, I have had opportunities to explore and learn challenging topics by self-studying, which is one of the most valuable skills I have gained. By offering me opportunities to be a lecturer to teach undergrads operations research, Dr. Pang gave me one of the best memories I had at Penn State. Not only have I learned teaching is the best way of learning, but I also met many excellent students. Additionally, I would like to thank my committee members Professor Vasant Honavar, Professor Eunhye Song, and Dr. Bicheng Chen for their interest and guidance in my work.

The work I have done in TE connectivity under the guidance of Dr. Bicheng Chen greatly benefited the work in this thesis. Even after my internship, Dr. Bicheng Chen has guided me along the way and inspired me to read many books, papers, and to meditate. I could not express my appreciation enough to Dr. Bicheng Chen for what he has done for me.

I would also thank my labmates and friends Dr. Seifu Chonde, Dr. Deepak Agarwal, Dr. YiShan Song, Professor Dika Handayani, Professor Cheng Bang Chen, Qian Yu Hu, Rico Polim, Mihir Mehta, Ankur Verma, Yi Zheng for their numerous assistance, suggestions, and company on this journey.

Last but not least, I would like to thank my wife and great friend Guli for her

love, selfless support and sacrifices. Of course, without the love and support of my father Juret Balaji, my mother Rihangul Kanjiahun, and my brother Jiesur Juret, the journey would have been impossible.

Chapter1
Real-time Scheduling of Heterogeneous Jobs on Non-stationary Unreliable Parallel Machines to Minimize Weighted Tardiness

1.1 Introduction

The basic definition of unplanned machine downtime is when a manufacturing machine fails to operate productively due to unforeseen hardware or software component failures. It may occur because of different factors, such as tool failures, machine malfunctions, and operator errors. High-mix and low-volume manufacturing facilities (HMLV-fab) process jobs with relatively small order sizes and heterogeneous specifications [1]. Due to the high variety of products, machines are often reconfigured to process a new job, such as tool changes, tool path and cutting parameters adjustments. The resulting frequent interactions between operators and machines often results in more unplanned downtime [2, 3].

To stay competitive, minimizing tardiness of orders is critical for manufacturers. However, variations in the availability of machines in HMLV-fab introduces challenges for achieving such a goal. When an unplanned downtime occurs, the finishing time of orders is often delayed, and it even halts business operations temporarily. As a result, a massive disruption is caused in the production, resulting in the decrease in customer satisfaction.

Though the downtime cost varies across different manufacturing industries, the cost can be very high as it reduces the production capacity of the plant. In 2014, Aberdeen [4] estimated the cost of unscheduled downtime in all businesses to be around $163,000 per hour. The number skyrocketed to $260,000 in 2016, illustrating the burden that organizations continue to face during these dormant times. Additionally, a survey of 101 executives in the automotive industry [5] indicates that the U.S. automotive industry's interruption cost stands at $22,000 per minute, an equivalent of $1.3 million per hour.

Since the interactions can be reduced in the context of HMLV-fab if similar jobs are scheduled together, there is potential to obtain a schedule such that the unplanned downtime is minimized. Given that an increasing number of industries are becoming HMLV-fab in the U.S. [6], any effort in reducing unplanned downtime is a welcome endeavor to U.S. manufacturers. Therefore, it is vital to consider the heterogeneity of jobs and its impact on machines' availability during the scheduling process.

Machines are rarely identical for any given plant, therefore, we assume machines are nonidentical. Moreover, since the stochasticity of machines can manifest in many stages of the production, we focus on the following critical ones:

1. When an unplanned downtime occurs to a machine, we state that the machine is in a down-state, and the amount of time it stays in a down-state is a random variable as it may be impacted by outside factors such as the availability of

maintenance workers.

2. When a part/job is assigned to an available machine, two random variables are used to model the process. The first one is the amount of time the machine takes to finish the job, also known as the process time (in the context of queuing theory, this random variable is often known as the service requirement), and the second one is the time until the machine breaks, which is often known as the on-time or availability of the machine. When the process time is less than or equal to the on-time, then the job can be finished without the machine breaks down. However, when the process time is greater than the on-time, the machine is going into a down-state without finished the job. Therefore, these two variables form a competing process. Moreover, when the job encounters an unplanned downtime during its production, it will be sent back to the queue, and when it is assigned to a machine after it has been sent back to the queue, it will be started from the scratch.

More importantly, in this chapter, the key characteristics of the HMLV-fab are modeled into the distributions of the process time and on-time. When two similar jobs are assigned to a machine back-to-back, by the definition of similarity, when the first job is finished, the second one requires less number of operations to reconfigure the machine. We therefore assume that there will be a reduced process time for the second job, and we also assume that the machine has a smaller probability to go into a down-state when the second job is being processed. Therefore, the distribution of the process time is parameterized with both the class of the job and the class of the previous job of the machine it is assigned to. Similarly, the distribution of the on-time is parameterized by classes of jobs (to capture the impact of heterogeneity of jobs on the on-time), the machine ID (to capture the non-identicalness of machines), and the time since the last maintenance (to capture

the non-stationarity of machines). The details of the distribution is described in the third section.

The tardiness of a job measures how late the job is finished compared to its due date. If the job is finished before its due date, the tardiness is defined as zero. The total tardiness of a given set of jobs reflects how late jobs are, and it gives the decision-maker the overall performance of the schedule. The average tardiness, on the other hand, gives an average measure. However, since the tardiness is zero if the job is early, averaging the measure is biased. Therefore, we use the total weighted tardiness in this work, where the weight of a job is associated with the importance of the job.

Therefore, in this work, a nonidentical parallel machine scheduling problem with stochastic machine breakdowns is studied to minimize the total weighted tardiness. And our contributions in this chapter are:

1. Both the impact of heterogeneity of jobs on machines on-time and the degeneration of machine's on-time are considered in the model. To the best of our knowledge this is the first study to model both factors in the scheduling process. To solve the scheduling problem, we propose a novel Markov Decision Process (MDP) model such that the state is independent of the number of jobs and machines. Moreover, the action space can be constructed with any scheduling dispatching rules. To schedule jobs in real-time, a simulation environment needs to be constructed such that it represents the interested manufacturing environment. After training a RL agent within the MDP, the obtained policy can be used to schedule jobs in real-time.

2. Though the proposed sampling method is tested with the deep deterministic policy gradient (DDPG) method, it can be used with any actor-critic-based RL algorithms. During the training process, the actor and critic are trained

using different batches of experiences. The sampling method is tested using the pendulum simulator from OpenAI gym [7], and the result indicates that a faster learning can be achieved via the proposed sampling method.

1.2 Literature Review

Research in stochastic scheduling can be classified into 1) proactive, 2) reactive and 3) hybrid approaches [8].

In proactive approaches, an initial schedule is generated by optimizing performance measures of interest such as makespan, total weighted tardiness, average flow time, etc. Subsequently, the schedule is improved based on robustness measurements, such as the average difference between the completion times of jobs and realized ones. Al-Hinai and EiMekkawy [9] propose a two-stage Hybrid Genetic Algorithm (HGA) to generate a robust and stable schedule. At the first stage, makespan is minimized as scheduling without uncertainty. At the second stage, the model takes the result from the first stage as the initial population, and searches for better schedules in terms of robustness and stability using Genetic Algorithm. Xiong et al. model the flexible job-shop scheduling problem (FJJSP) with random machine breakdowns into a multi-objective optimization problem [10]. In their work, makespan and robustness are considered simultaneously, where the robustness of the schedule is measured by the expected difference between deterministic and actual makespan. An evolutionary algorithm is developed to solve the model.

The reactive approaches generate an initial schedule based on the initial availability of machines without taking any uncertainty into account, and a new deterministic schedule is generated whenever a machine breakdown is observed or a machine becomes available. Given the reactive nature, the solution time is one of the key performance measures of reactive approaches. When domain knowledge is available,

such as the expected down time for a given the type of breakdown, jobs can be assigned to a down machine before it is brought back online [11]. Nevertheless, this type of approaches give an optimistic estimate of the objective value when the internal measurement like makespan is considered but may perform badly when the objective function considers the external measurement like tardiness of jobs.

Hybrid approaches utilize both proactive and reactive approaches. The initial schedule is generated in a proactive manner, and when an uncertainty is realized, a new schedule is generated in a proactive manner. A hybrid method that considers machine breakdowns is proposed by Smith [12]. In his work, a right shift rescheduling rule is analyzed. When a breakdown impacts tasks, they are moved forward in time on the schedule. This method reacts instantaneously to the event. However, the future uncertainties are only anticipated when the first schedule is generated, and it might degenerate the quality of the schedule in terms of robustness as more realizations of breakdowns happen. Subramaniam develops a modified Affected Operation Rescheduling (mAOR) [13], which considers multiple disruptions such as absenteeism of workers, process time variations, the arrival of unexpected jobs, and machine breakdowns. Heuristics are applied to repair the schedule, and the work shows that this method outperforms right shift rescheduling.

Reinforcement learning (RL) is used to learn a policy under a stochastic environment to maximize discounted cumulative rewards. In the context of scheduling with unreliable machines, it provides a scheduling policy that can be used to schedule jobs. The resulting policy not only considers uncertainties, it can also be reactive by incorporating the availability of machines into state space design. Therefore, we classify it as a hybrid approach in the context of stochastic scheduling.

Several studies used RL to solve scheduling problems decades ago. Zhang and Thomas [14] are the pioneers who used RL to solve job-shop scheduling problem (JSSP) and applied to NASA space shuttle scheduling. The state space is designed

like the current schedule, and the action space contains reassign-pool and move. The agent performs reassign-pool only when all resource constraints are satisfied, and the action reassigns another pool to the resource requirements of a job. When there is at least one constraint violation, the agent moves the job to the next earlier or the next later time until there is no violation. The reward signal is designed as the resource dilation factor, which is the ratio of total pool utilization to total resource utilization. The multi-step temporal difference method ($TD(\lambda)$) is used in the RL framework. The result indicates that their RL method outperforms the state of the art methods at that time in the domain of space shuttle payload processing jobs. Nonetheless, the design of the state space restricts the robustness of the method, since any different numbers of jobs and resources require new learning processes. Hence, it has a relatively large solution time, and it is not applicable to the scope of JSSP with machine breakdowns.

Aydin and Oztemel [15] applied RL to select dispatching rule to schedule jobs in real-time. The state space is constructed by the queue size of each machine and the corresponding mean slack time of the queue. The action space contains three dispatching rules: 1) Shortest Processing Time (SPT), 2) Cost over time (COVERT), and 3) Critical Ratio (CR). With similar designs, Wang and Usher [16], Park et al. [17] and Tseng et al. [18], use RL to solve dynamic scheduling for a single machine or multiple machines. Though the state vectors are robust in both works, studies show that the design of action space using dispatching rules do not have good quality to meet optimization objectives [19].

In a scheduling problem setting, the traditional stochastic optimization method takes future uncertainty into account by utilizing the probability distribution (chance-constrained methods) or the historical data (two-stage methods). However, it requires a large computational time when the problem size is large; hence, it cannot generate a new policy in a timely manner. This limits the usage of the

traditional optimization in the reactive and hybrid approaches, and it often used as proactive approaches.

When a random event is realized, hybrid approaches exploit the realization and generate a new schedule that considers future uncertainty. The reactive nature of the approach requires the method to have a short computational time. Moreover, the hybrid approaches are able to utilize the resources better as it has more accurate data in terms of the availability of resources when a new schedule is generated. Since urgent, important jobs can be rescheduled to available assets when uncertainty is realized, hybrid approaches are especially practical when the due date of jobs is one of the optimization criteria. This improvement has the potential to increase the on-time rate and reduce the lead time compared to the traditional optimization methods.

A reinforcement learning (RL) approach naturally falls into the hybrid approaches category for the following reasons:

1. The resulting scheduling policy generated by an RL model considers future uncertainty as the policy is generated to maximize the future return, which is characterized by the total discounted rewards or average rewards.

2. The resulting scheduling policy can be used to generate a new schedule decision with a very short computational time.

As it is not practical to let the RL agent learn the policy by interacting with the real-world, a simulation model is a practical option to train the agent. And probability distributions estimated from the real-world data can be utilized to build the simulation model. The RL agent can be trained in a model-based manner or model-free manner. There are various methods for both types, such as the value iteration, policy iteration, SARSA, Q-learning, stochastic policy gradients, Actor-Critic, deterministic policy gradients. Moreover, the utilization

of nonlinear approximators such as deep neural networks empowered RL methods to solve problems with infinite/continuous state and action spaces and learn a well-performing policy.

Therefore, an RL approach is preferred due to:

1. The resulted scheduling policy takes future uncertainty into account events even when the modeled uncertainty is extremely complex.

2. The resulted scheduling policy parameterized by nonlinear approximators such as deep neural networks gives the policy a good generalization power to react to the changes (the realizations of uncertainty) of state space. This allows better utilization of resources, especially when due data is one of the optimization criteria.

3. Even the computational time is large when training the RL algorithm, the obtained scheduling policy can react to the uncertainty instantaneously; hence, the schedule can adapt to the environment change accordingly.

1.3 Methodology

1.3.1 Reinforcement Learning

Reinforcement learning (RL), supervised, and unsupervised learning are three types of machine learning technique. In the standard RL framework, the learner or decision maker is called as the agent, and the agent could observe and interact with the environment via actions. In the context of control, the agent, environment, and action are controller, plant, and control signal. When the agent takes action, it will receive a numerical reward signal. The goal of RL is finding a policy that maximizes the long-term total reward signal. Similar to natural learning processes,

such as a newborn gazelle learns to walk, a master chess player makes a move from the intuitive judgments [20], the agent discover the desired policy from its experiences and by exploring the environment via trial-and-error. In most of the challenging cases, actions would affect the next state of environment and reward, and through that, all subsequent environments and rewards, therefore, the rewards are delayed. In fact, the trial-and-error search and delayed rewards are two crucial characteristics of RL.

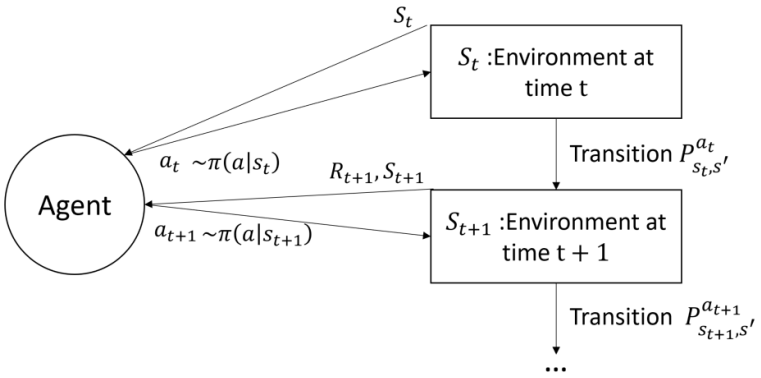


Figure 1.1: Reinforcement Learning Diagram

As Figure 1.1 shows, at any time point t, the agent would observe environment via state vector $S_t \in \mathcal{S}$, where $\mathcal{S}$ is the set of possible states. Based on S_t, the agent would draw an action a_t from current policy $\pi(a|S_t) = \mathbb{P}[A = a|S = S_t]$ for $\forall a$. After the agent executes the action a_t, the environment would transit to S_{t+1} with probability $\mathcal{P}^a_{S_t,S_{t+1}}$ for all reachable states $S_{t+1} \in \mathcal{S}$ from S_t, and agent would receive a reward signal $R_{t+1} \in$ R and observe the new environment S_{t+1}, where R is the reward set. This process would be continued until the environment reaches the terminate state or the policy is converged. After each action, the agent would receive a reward signal R_t, and the maximization of expected cumulative rewards should describe all goals. In the JSSP context, the agent would receive a large negative reward if the action makes a job overdue. Also, the agent would receive a

negative reward proportional to the total time to finish the set of jobs at the end. By maximizing the expected cumulative rewards, the number of overdue jobs and total makespan are minimized. Formally, the expected cumulative reward G_t can be defined using discount factor $\gamma \in [0, 1]$:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} ... = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1} \tag{1.1}$$

where T is the time point when the agent reaches the terminate state. The discount factor γ can prevent problematic calculation when the learning task has an infinite time horizon. Moreover, when γ is set to closer to 1, the future rewards have a more significant effect on G_t, and it would make agent farsighted. Similarly, when γ is set to closer to 0, the immediate reward plays a more significant role, and the agent would act towards getting more immediate rewards.

1.3.2 Value Functions

Value functions and Bellman equations are the core of RL. There are two types of value functions:

1. State-value function for policy π: $v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$
2. Action-value function for policy π: $q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$

Both value functions indicate the long-term rewards under a policy π, given the current state or the state-action pair.

Using Bellman equation for $v_\pi(s)$ shows that the state-value function can be decomposed into two parts: 1) immediate reward R_{t+1} and 2) discounted value of successor state $\gamma v_\pi(S_{t+1})$:

$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$

$= \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} ...|S_t = s]$

$= \mathbb{E}_\pi[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3}...)|S_t = s]$

$= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s]$

$= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s]$

Similarly, the Bellman equation for $q_\pi(s, a)$ shows that action-value function can also be decomposed into immediate reward R_{t+1} and discounted value of successor state-action pair $\gamma q_\pi(S_{t+1}, A_{t+1})$:

$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$

$= \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3}...|S_t = s, A_t = a]$

$= \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a]$

Therefore, given the dynamics of a finite MDP, by applying recursion with memorization method in dynamic programming, Bellman equations can be used to calculate state-value for all the states and action-value for all the state-action pairs. With the value functions, the optimal path can be found easily by merely acting greedily towards the terminal state.

Using the transition probability of the given MDP, the state value function of any state $s \in \mathcal{S}$ under a policy π can be expressed as

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} P(s'|s, a)[r + \gamma v_\pi(s')] \tag{1.2}$$

where r is a random reward given by the environment when the agent takes action a in the state s. The expression is shown in Figure 1.2, the backup diagram of state value function.

Similarly, the state-action value function can be rewritten as

$$q_\pi(s, a) = \sum_{s' \in \mathcal{S}} P(s'|s, a)[r + \gamma \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s', a')] \tag{1.3}$$

and the vitalization of state-action value function is shown in 1.3.

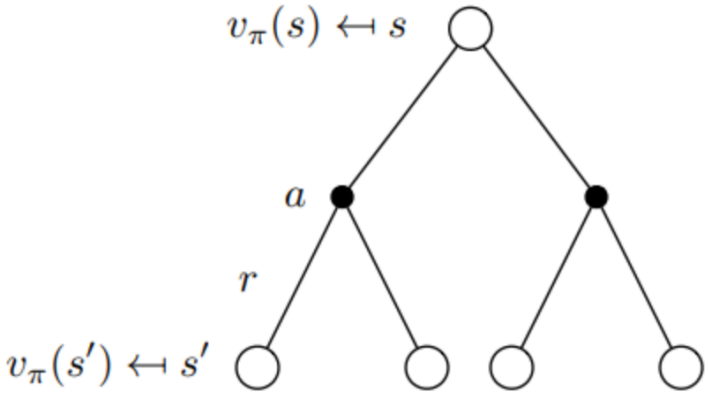


Figure 1.2: Backup Diagram of State Value Function

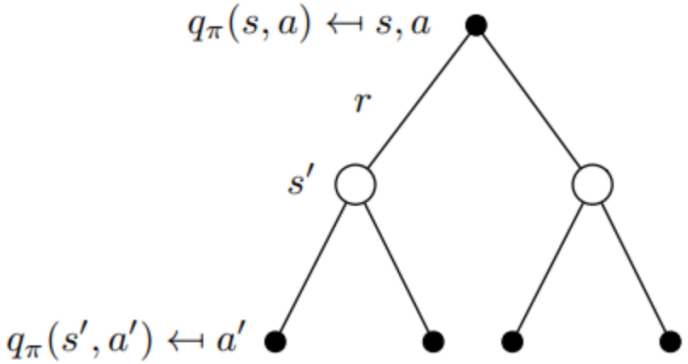


Figure 1.3: Backup Diagram of State-Action Value Function

1.3.3 Bellman Optimality Equation

For any given state, or state-action pair, the value functions are determined by the policy. A policy π is better than π' if the following holds

$$v_\pi(s) \geq v_{\pi'}(s) \forall s \in \mathcal{S} \tag{1.4}$$

and the optimal value functions of a given state $s \in \mathcal{S}$ is therefore defined as

$$v_{\pi^*}(s) = \max_{\pi} v(s) \tag{1.5}$$

the optimal state-action value function for a given state-action pair is defined as

$$q_{\pi^*}(s, a) = \max_{\pi} q(s, a) \tag{1.6}$$

The objective of a RL task is to find the optimal policy π^*, and the Bellman optimality equations state that

$$v_{\pi^*}(s) = \max_{a} \sum_{s' \in \mathcal{S}} P(s'|s, a)[r + \gamma v_{\pi^*}(s')] \tag{1.7}$$

$$q_{\pi^*}(s, a) = \sum_{s' \in \mathcal{S}} P(s'|s, a)[r + \gamma \max_{a'} q_{\pi^*}(s', a')] \tag{1.8}$$

and it is evident that

$$v_{\pi^*}(s) = \max_{a} q(s, a) \tag{1.9}$$

hence, we have

$$q_{\pi^*}(s, a) = \sum_{s' \in \mathcal{S}} P(s'|s, a)[r + \gamma v_{\pi^*}(s')] \tag{1.10}$$

There are three types of methods in RL to find the optimal policy in a given MDP. The first one is value-based method, in which value functions are calculated or estimated, and the optimal policy can be found by acting greedily based on the state-action value function. The second type is policy-based method, instead of calculate or estimate the value functions, the policy is parameterized and improved. The third one is actor-critic method, it is a combination of value-based method and policy-based method.

Furthermore, depending on if the system dynamics of the MDP are utilized or

available, algorithms in each type of method can be categorized as model-free or model-based algorithm. In a model-based method, the Bellman equations will be solved to obtain the value functions, and in a medel-free method, the value functions are estimated using the experiences collected from the interactions between the agent and the MDP. When the size of the MDP is small and the system dynamics are available, a model-based method is able to provide the optimal policy efficiently. However, as the size of the MDP growth or when dynamics of the MDP are unavailable, the value functions can be estimated by utilizing either the Monte Carlo Methods or Temporal-Difference Learning.

1.3.4 Value-based Method

Commonly seen value-based method includes the policy iteration [21, 22], value iteration [23, 24], Sarsa [25], and Q-learning [26, 27]. Many advanced algorithms are based on these algorithms, such as the Expected Sarsa, n-step Sarsa, double Q-learning, and deep Q-learning with prioritized experience replay.

Policy Iteration

The policy iteration algorithm contains two major steps, the first step is called policy evaluation, and it calculates the state value functions under a given policy for all states; the second step is to improve the policy greedily with respect to the value function of the original policy, and the step is based on the policy improvement thorem [21, 22] and the step is often known as policy improvement. The details of the algorithm is shown in Algorithm 1 and the process of the algorithm is shown in Figure 1.4.

As the value function of all states are evaluated and improved, when the state space is continuous, the algorithm cannot be implemented directly without

Algorithm 1: Policy Iteration

Result: $\pi \approx \pi^*$

Input: θ a small positive number determining the accuracy of the estimation
$\gamma \in (0, 1]$ discount factor

1. Initialization:

$V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation:

while *True* **do**
  $\Delta \leftarrow 0$
  for $s \in \mathcal{S}$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_{r,s'} p(s', r|s, \pi(s))[r + \gamma V(s')]$
    $\Delta \leftarrow max(\Delta, |v - V(s)|)$
  end
  if $\Delta \leq \theta$ **then**
    break
  end
end

3. Policy Improvement

$converge \leftarrow true$
for $s \in \mathcal{S}$ **do**
  $a_0 \leftarrow \pi(s)$
  $\pi(s) \leftarrow \text{argmax}_a \sum_{r,s'} p(s', r|s, a)[r + \gamma V(s')]$
  if $\pi(s) \neq a_0$ **then**
    $converge \leftarrow false$
  end
end
if *converge* **then**
  Return π
end
else
  Go to step 2
end

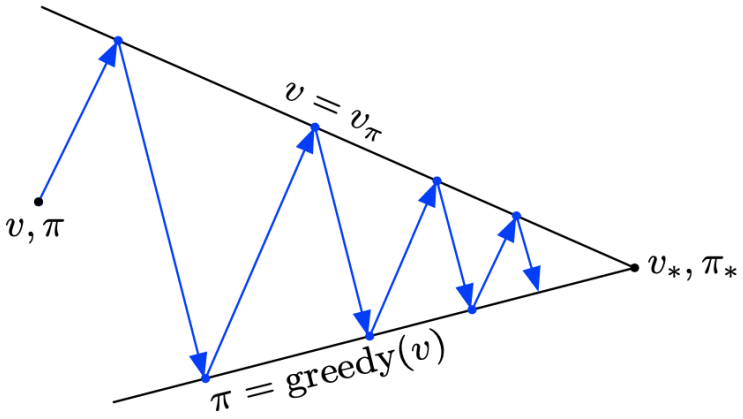


Figure 1.4: Policy Iteration

discretization of the state space or approximating the state value functions. Furthermore, as both of policy evaluation and improvement steps use the transition probability of the MDP, policy iteration algorithm is a model-based method.

Value Iteration

In the policy iteration, since both policy evaluation and improvement steps are swapping the state space, the method is computationally expensive. The value iteration algorithm, which is shown in Algorithm 2, can be viewed as a special case of the policy iteration where the policy evaluation step is done using the Bellman optimality equation, namely,

$$V(s) \leftarrow \max_a \sum_{r,s'} p(s', r|s, a)[r + \gamma V(s')]$$

Value iteration combines a single sweep of policy evaluation and policy improvement in every step. However, when the action space is continuous, the maximization step might become computationally expensive. As the system dynamics are used in the algorithm, value iteration is also a model-based method.

Algorithm 2: Value Iteration

Result: $\pi \approx \pi^*$
Input: θ a small positive number determining the accuracy of the estimation
$\gamma \in (0, 1]$ discount factor
1. Initialization:
$V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}$ arbitrarily for all $s \in \mathcal{S}$
while *True* **do**
  $\Delta \leftarrow 0$
  for $s \in \mathcal{S}$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{r,s'} p(s', r|s, a)[r + \gamma V(s')]$
    $\Delta \leftarrow max(\Delta, |v - V(s)|)$
  end
  if $\Delta \leq \theta$ **then**
    break
  end
end
Return π

Monte Carlo Exploring Starts

When the dynamics of the MDP are not available, value iteration or policy iteration can no longer be used. Mente Carlo (MC) methods [28,29], however, can be used to find optimal policy [20] as the they require only sample sequences of states, actions, and rewards from actual or simulated interaction with the MDP. Since the core of MC methods is averaging sample returns, the methods can be applied when the MDP is a episodic task, namely, all episodes eventually terminate no matter what actions are selected.

Learning from the actual interaction with the MDP might be expensive as it might cause undesired outcomes, MC methods are often used to learn from simulated experiences. Though the simulation model is required, the model does not need the complete probability distributions of all possible transitions. Instead, the MC methods only require the model to generate sample transitions. In face, it is feasible to obtain experience sampled according to the desired probability distributions, and it is infeasible to express the distributions in explicit form [20]. Hence, MC methods are model-free.

There are different MC methods, such as first-visit MC control, off-policy MC controls, we only introduce the basic one called Monte Carlo Exploring Starts, and the details are shown in Algorithm 3.

A sufficient exploration is an issue in MC methods, as a actual better action may never be picked based on the current policy, hence it may never be learned. The concern is resolved when the number of episodes is large enough as the initial action is selected randomly [20].

Algorithm 3: Monte Carlo Exploring Starts

Result: $\pi \approx \pi^*$
Input: N number of episodes
1. Initialization:
$\pi(s) \in \mathcal{A}$ arbitrarily for all $s \in \mathcal{S}$
$Q(s,a) \in \mathbb{R}$ arbitrarily for all $s \in \mathcal{S}, a \in \mathcal{A}$
$Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}, a \in \mathcal{A}$
for $i = 1, ..., N$ **do**
  Select initial state $S_0 \in \mathcal{S}$ and initial action $a_0 \in \mathcal{A}$ randomly
  Generate a trajectory from S_0, a_0 following π: $S_0, a_0, R_1, ..., S_{T-1}, a_{T-1}, R_T$
  $G \leftarrow 0$
  for $t = T-1, ..., 0$ **do**
    $G \leftarrow \gamma G + R_{t+1}$
    Append G to $Returns(s_t, a_t)$
    $Q(s_t, a_t) \leftarrow average(Returns(s_t, a_t))$
    $\pi(s_t) \leftarrow arg\max_a Q(s_t, a)$
  end
end
Return π

Sarsa

The temporal-difference (TD) learning is the central and novel idea to reinforcement learning as it is a combination of the MC methods and value iteration [20]. Like MC methods, TD methods are model-free. However, TD methods do not require the MDP to be episodic, hence, TD methods can be applied in more general settings include an online learning setting. Though TD learning is introduced under Sarsa, it is also the basic of Q-learning.

The core of TD learning is from the Bellman equations,

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \tag{1.11}$$

by minimizing the TD error δ_t, which is defined as

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \tag{1.12}$$

the estimation of value function under the policy can be updated,

$$V(S_t) \leftarrow V(S_t) + \alpha\delta_t \tag{1.13}$$

where $\alpha \in (0, 1]$ is the learning rate. The update is intuitive, when the current estimated value function of S_t is optimistic then δ_t would be a negative number, hence, after the update step, the value function of S_t will be less optimistic. When the estimation is pessimistic, then a positive δ_t will result in a more optimistic estimation of S_t. This TD method is call one-step TD (also known as TD(0)) as the estimation of the value function is updated based on one step of sample, and there is also n-step TD.

TD(0) bootstraps by updating the estimates based on other estimates, and it has been proven that when α is sufficiently small, for any fixed policy π, TD(0) converges to V_π in the mean [20].

Sarsa is developed based on the TD(0), instead of working with the state value functions, Sarsa uses state-action value functions, and the state-action value function of a state-action pair in a trajectory is updated as following,

$$Q(S_t, a_t) \leftarrow Q(S_t, a_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, a_{t+1}) - Q(S_t, a_t)]$$

and the details are shown in Algorithm 4

Algorithm 4: Sarsa

Result: $Q \approx q^*$
Input: N number of episodes
α learning rate
1. Initialization:
$Q(s,a) \in \mathbb{R}$ arbitrarily for all $s \in \mathcal{S}, a \in \mathcal{A}$
$Q(s,a) = 0$ if s is terminal for all $a \in \mathcal{A}$
for $i = 1, ..., N$ **do**
  Select initial state $S \in \mathcal{S}$ randomly
  Select a using $Q(S, \bullet)$ function (e.g., $\epsilon-$greedy)
  while S *is not terminal* **do**
    Take action a, observe R, S'
    Select a' using $Q(S', \bullet)$ function (e.g., $\epsilon-$greedy)
    $Q(S,a) \leftarrow Q(S,a) + \alpha[R + \gamma Q(S', a') - Q(S,a)]$
    $S \leftarrow S'$
    $a \leftarrow a'$
  end
end
Return Q

Q-learning

Q-learning is first introduced in Watkins's Ph.D. thesis [26], and the convergence proof was made later with Dayan [27]. The core of Q-learning is also minimizing the TD error δ_t, however, instead of using Bellman equations like TD methods, Bellman optimality equations 1.8 are used. The Q function is updated as follow,

$$Q(S,a) \leftarrow Q(S,a) + \alpha[R + \gamma \max_{a'} Q(S', a') - Q(S,a)]$$

The action-value function is learned independent of the policy being followed, hence, the algorithm is off-policy. Furthermore, this simplifies the analysis of the algorithm and as long as all the state-action pairs are updated, Q-learning is guaranteed to find the optimal policy [20]. The details of Q-learning is shown in Algorithm 5.

When the state space is small and discrete, Q-learning can be easily implemented by creating and updating a lookup table that contains value functions of all state-

Algorithm 5: Q-learning

Result: $Q \approx q^*$
Input: N number of episodes
α learning rate
1. Initialization:
$Q(s,a) \in \mathbb{R}$ arbitrarily for all $s \in \mathcal{S}, a \in \mathcal{A}$
$Q(s,a) = 0$ if s is terminal for all $a \in \mathcal{A}$
for $i = 1, ..., N$ **do**
  Select initial state $S \in \mathcal{S}$ randomly
  while S *is not terminal* **do**
    Select a using $Q(S, \bullet)$ function (e.g., $\epsilon-$greedy)
    Take action a, observe R, S'
    $Q(S,a) \leftarrow Q(S,a) + \alpha[R + \gamma \max_{a'} Q(S', a') - Q(S,a)]$
    $S \leftarrow S'$
  end
end
Return Q

action pairs. But when the state space is large or continuous, one needs to approximate the state-action value function.

The function approximation has made RL more powerful but also potentially more difficult to manage and understand [20]. The fundamental of function approximation in RL is to parameterize the value functions, and the simplest form of approximation is to use a linear function [30], and commonly used ones are supervised machine learning models such as neural networks [31], and radial basis functions [32]. Long-short term memory (LSTM) [33] neural networks are also used in partially observable MDPs.

1.3.5 Policy-based Method

Instead of consulting a value function to find optimal policies like algorithms in value-based method do, a policy-based method selects actions without consulting a value function. The key ideas in the policy-based method are policy gradient theorem [34] and parameterization of a policy.

The policy π is parameterized by Θ and optimized using gradient ascent. The

objective function can be rewritten as following,

$$J(\pi_\Theta) = \max_\Theta \int_{\tau=s_{t_0},a_{t_0},R_{t_0},...} \mathbb{P}_{\pi_\Theta}(\tau)G(\tau)d\tau \tag{1.14}$$

Additionally, the probability of any trajectory $\tau = s_{t_0}, a_{t_0}, R_{t_0}, s_{t_1}, a_{t_1}, ..., R_{t_H}$ under a policy π_Θ can be calculated as

$$\mathbb{P}_{\pi_\Theta}(\tau) = d(s_0) \prod_{q=1}^{H} \pi_\Theta(a_{t_q}|s_{t_q})\mathbb{P}(s_{t_{q+1}}, R_{t_q}|s_{t_q}, a_{t_q}) \tag{1.15}$$

where $d(s_0)$ is the probability of the initial state being $s_0 \in \mathcal{S}$.

Since $G(\tau)$ is a scalar for any given τ, the derivative with respect to Θ can be therefore written as

$$\nabla_\Theta J(\pi_\Theta) = \int_\tau G(\tau)\nabla_\Theta \mathbb{P}_\Theta(\tau)d\tau = \int_\tau G(\tau) \sum_{s,a\in\tau} \nabla_\Theta log\pi_\Theta(a|s)d\tau \tag{1.16}$$

By regrouping the terms, the derivative can be expressed as

$$\nabla_\Theta J(\pi_\Theta) = \mathbb{E}[G(\tau) \sum_{s,a\in\tau} \nabla_\Theta log\pi_\Theta(a|s)] = \frac{1}{N}\sum_{q}^{N}\Big[\Big(\sum_{R\in\tau_q} R\Big) \sum_{s,a\in\tau_q} \nabla_\Theta log\pi_\Theta(a|s)\Big] \tag{1.17}$$

where N is number of trajectories used to estimate the gradient.

The equation 1.17 is the result of the policy gradient theorem [34], and it shows that the derivative of parameterized policy π_Θ can be obtained using simulated trajectories under the policy π_Θ and it has good theoretical convergence properties [20].

REINFORCE

When $N = 1$, 1.17 results in the following parameter update,

$$\Theta_{t+1} = \Theta_t + \alpha(\sum_{R \in \tau} R) \sum_{s,a \in \tau} \nabla_\Theta log \pi_\Theta(a|s) \tag{1.18}$$

where α is the learning rate. The update from 1.18 is known as REINFORCE update [35], and it is the key step of the REINFORCE algorithm. The details of REINFORCE is shown in Algorithm 6.

Algorithm 6: REINFORCE

Result: $\pi \approx \pi^*$
Input: A differentiable policy parameterization π_Θ
N number of episodes
α learning rate
Randomly initialize Θ
for $i = 1, ..., N$ **do**
  Generate a trajectory following π_Θ: $S_0, a_0, R_1, ..., S_{T-1}, a_{T-1}, R_T$
  for $t = 0, 1, 2, ..., T-1$ **do**
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    $\Theta \leftarrow \Theta + \alpha \gamma^t G \nabla_\Theta ln \pi_\Theta(a_t|S_t)$
  end
end
Return π

Though REINFORCE has good theoretical convergence properties, it has high variances and it produces a slow learning [20] and it is restricted to episodic MDPs.

REINFORCE with Baseline

Define an arbitrary baseline $b(s)$, which can be any function of the state or even a random variable that does not depend on action a, then, we have

$$\sum_a b(s) \nabla_\Theta \pi_\Theta(a|s) = b(s) \nabla_\Theta \sum_a \pi_\Theta(a|s) = 0 \tag{1.19}$$

it is evident that the update rule in Algorithm 6 can be rewritten as

$$\Theta \leftarrow \Theta + \alpha\gamma^t(G - b(S_t))\nabla_\Theta ln\pi_\Theta(a_t|S_t)$$

and it results in a variance of REINFORCE called REINFORCE with Baseline. Since the update rule is not changed, the convergence properties are unchanged. However, the baseline is able to reduce the variance and speed up the learning [20]. The detailed algorithm in shown in 7, in which the approximated state-value function is used as the baseline.

Algorithm 7: REINFORCE with Baseline

Result: $\pi \approx \pi^*$
Input: A differentiable policy parameterization π_Θ
N number of episodes
$\alpha^\Theta, \alpha^\omega$ learning rates
Randomly initialize Θ and ω
for $i = 1, ..., N$ **do**
  Generate a trajectory following π_Θ: $S_0, a_0, R_1, ..., S_{T-1}, a_{T-1}, R_T$
  for $t = 0, 1, 2, ..., T-1$ **do**
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    $\delta \leftarrow G - \hat{V}_\omega(S_t)$
    $\omega \leftarrow \omega + \alpha^\omega \delta \nabla_\omega \hat{V}_\omega(S_t)$
    $\Theta \leftarrow \Theta + \alpha^\Theta \gamma^t \delta \nabla_\Theta ln\pi_\Theta(a_t|S_t)$
  end
end
Return π

Both algorithms under policy-based method use MC to update the parameters of the policy, hence, it can only work with episodic MDPs. However, policy-based method provides a way to work with MDPs in which actions are continuous.

In a continuous action space setting, the parameterized policy is often defined using the normal probability density function,

$$\pi_\Theta(a|s) = \frac{1}{\sqrt{(2\pi)^2|\Sigma_{s,\Theta}|}} e^{-\frac{1}{2}(a-\mu_{s,\Theta})^T \Sigma_{s,\Theta}^{-1}(a-\mu_{s,\Theta})} \tag{1.20}$$

where $\mu_{s,\Theta} \in \mathbb{R}^{|\mathcal{A}|}, \Sigma_{s,\Theta} \in \mathbb{R}^{|\mathcal{A}|\times|\mathcal{A}|}$ are the mean vector and covariance matrix that depend on the state and parameter Θ. For simplicity, $\Sigma_{s,\Theta}$ can be assumed to be a diagonal matrix, thus, instead of asking the policy to provide actions, we ask for $[\mu_{s,\Theta}, diag(\Sigma_{s,\Theta})]$, the actions are then drawn from $\mathcal{N}(\mu_{s,\Theta}, \Sigma_{s,\Theta})$.

1.3.6 Actor-Critic Method

Since actor-critic method combines both value-based and policy-base methods, algorithms under this category can handle MDPs that have no terminal states and actions are continuous. Though REINFORCE with baseline utilized a state-value function, it is not considered as a actor-critic method as its state-value function is used only as a baseline instead of bootstrapping [20]. The most basic Actor-Critic (AC) algorithm uses a critic $\hat{v}_{\Theta'}(s)$ and the details are shown in Algorithm 8.

Algorithm 8: Actor-Critic

Result: $\pi_\Theta \approx \pi^*$
Input: a differentiable policy parameterization π_Θ
a differentiable state-value function parameterization $\hat{v}_{\Theta'}$
α_A, α_C learning rates
N number of episodes
Initialize
policy parameter Θ
state-value parameters Θ'
episode count $epc = 0$
while *True* **do**
  Initialize first state of episode S
  while S *is not terminal* **do**
    $a \sim \pi_\Theta(\bullet|S)$
    Take action a and observe S', R
    $\delta \leftarrow R + \hat{v}_{\Theta'}(S') - \hat{v}_{\Theta'}(S)$
    $\Theta \leftarrow \Theta + \alpha_A \delta \nabla ln \pi_\Theta(a|S)$
    $\Theta' \leftarrow \Theta' + \alpha_C \delta \nabla \hat{v}_{\Theta'}(S)$
    $S \leftarrow S'$
  end
  $epc \leftarrow epc + 1$
  if *epc=Iter* **then**
    break
  end
end

Deep Deterministic Policy Gradient

Denote a deterministic policy $\mu_\Theta(s) : \mathcal{S} \rightarrow \mathcal{A}$ with parameter vector Θ, the objective function in 1.14 can be rewritten with respect to Q function,

$$J(\mu_\Theta) = \max_\Theta \int_\mathcal{S} \rho^{\mu_\Theta}(s) Q^{\mu_\Theta}(s, \mu_\Theta(s)) ds \tag{1.21}$$

where $\rho^{\mu_\Theta}(s)$ is the stationary probability of visiting state s under policy μ_Θ. And the deterministic policy gradient (DPG) thorem [36] provides the gradient of the parameter vectors,

$$\begin{aligned} \nabla_\Theta J(\mu_\Theta) &= \int_\mathcal{S} \rho^{\mu_\Theta}(s) \nabla_\Theta \mu_\Theta(s) \nabla_a Q^{\mu_\Theta}(s, a)|_{a=\mu_\Theta(s)} ds \\ &= \mathbb{E}_{s\sim\rho^{\mu_\Theta}}[\nabla_\Theta \mu_\Theta(s) \nabla_a Q^{\mu_\Theta}(s, a)|_{a=\mu_\Theta(s)}] \end{aligned} \tag{1.22}$$

and this provides a way to update the parameters of the deterministic policy and the parameter of the deterministic policy can be updated by,

$$\Theta_{t+1} = \Theta_t + \alpha \nabla_\Theta \mu_\Theta(s_t) \nabla_a Q^{\mu_\Theta}(s_t, a_t) \tag{1.23}$$

The deep deterministic policy gradient (DDPG) algorithm is model-free, it combines both DPG and deep Q-network [31] (DQN) method. It utilizes DPG to update the actor (the deterministic policy). Both experience replay and the target network from DQN are used in DDPG in order to stabilize the learning process of the critic. The target Q-network is periodically frozen for certain period of time before the update in DQN; however, parameters of target networks are updated via a soft update procedure in DDPG. To ensure the exploration, a Gaussian noise $\mathcal{N}$ is added to the action drawn from the deterministic policy. Algorithm 9 shows the details of details of DDPG.

Algorithm 9: DDPG

Result: $\mu_\Theta \approx \mu^*$

Input: a differentiable policy parameterization μ_Θ, a target policy $\mu_{\Theta_{targ}} := \mu_\Theta$

Input: a differentiable state-action value function parameterization $Q_{\Theta'}$, a target Q-function $Q_{\Theta'_{targ}} := Q_{\Theta'}$

Parameters: α_A, α_C, $Iter$, ρ, $freq$, γ, a_{Low},a_{High}

Initialize policy parameter Θ, state-value parameters Θ', episode count $epc = 0$, step count $step = 0$, an empty experience buffer $\mathcal{D}$

while *True* **do**

  Initialize first state of episode S

  $d = 0$

  while S *is not terminal* **do**

    $A = min[max[\mu_\Theta(S) + \epsilon, a_{Low}], a_{High}]$, where $\epsilon \sim \mathcal{N}$

    Take action A and observe S', R

    $d = 1$ if S' is terminal

    Store (S, A, R, S', d) in the replay buffer $\mathcal{D}$

    $S \leftarrow S'$

    $step \leftarrow step + 1$

    if $step\%freq = 0$ **then**

      Sample a batch of transitions, $B = (s, a, r, s', d)$ from $\mathcal{D}$

      Compute targets

$$y(r, s', d) = r + \gamma(1 - d)Q_{\Theta'_{targ}}(s', \mu_{\Theta_{targ}}(s'))$$

      Update Q-function by one step of gradient descent using

$$\Theta' \leftarrow \Theta' - \alpha_C \nabla_{\Theta'} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} (Q_{\Theta'}(s, a) - y(r, s', d))^2$$

      Update policy by one step of gradient ascent using

$$\Theta \leftarrow \Theta + \alpha_A \nabla_\Theta \frac{1}{|B|} \sum_{s \in B} Q_{\Theta'}(s, \mu_\Theta(s))$$

      Update the target networks with

$$\Theta_{targ} \leftarrow \rho\Theta_{targ} + (1 - \rho)\Theta$$

$$\Theta'_{targ} \leftarrow \rho\Theta'_{targ} + (1 - \rho)\Theta'$$

  $epc \leftarrow epc + 1$

  if *epc=Iter* **then**

    break

1.3.7 DDPG with Separate Sampling

In Algorithm 9, the critic (the Q-function) is updated by minimizing the one-step temporal difference errors (TD error),

$$\Theta' \leftarrow \Theta' - \alpha_C \nabla_{\Theta'} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} (Q_{\Theta'}(s,a) - y(r,s',d))^2 \tag{1.24}$$

and the actor (the deterministic policy) is updated by moving parameters towards actions that increase the Q-value.

$$\Theta \leftarrow \Theta + \alpha_A \nabla_{\Theta} \frac{1}{|B|} \sum_{s \in B} Q_{\Theta'}(s, \mu_{\Theta}(s)) \tag{1.25}$$

When estimation of $Q_{\Theta'}$ at the samples in B (1.25) is inaccurate, the gradient will therefore be off, and it results in an inaccurate actor. For instance, say the estimated state-action value of a state s and an action a, namely $Q_{\Theta'}(s,a)$ has a large error compared to the true state-action value $Q(s,a)$, and when the actor adjusts its parameter based on the gradient at $Q_{\Theta'}(s,a)$, then, the actor will have a large error after the update. This results in a fluctuation learning for the actor, and therefore an overall slower convergence. By simply feeding the actor samples where the critic has a low error, the learning of actor will be expedited. However, if experiences are sampled in a way such that the critic has a lower TD error, then it might overfit the critic and lead to a bad generalization.

Inspired by the prioritized experience replay (PER) [37], a sampling method is proposed to generate two batches of transitions based on the TD error of critic. In the first step, experiences B are sampled randomly with a sample size of B_{total}. The absolute TD error is then calculated for experiences in B,

$$E = |Q_{\Theta'_{targ}}(s,a) - (r + \gamma(1-d)Q_{\Theta'_{targ}}(s', \mu_{\Theta_{targ}}(s')))| \in \mathcal{R}^{|B|}$$

where $(s, a, r, s', d) \in B$. To utilize the TD error in the sampling process, the error is scaled into value between 0 and 1, and the normalized error is defined as

$$\tilde{E} = \frac{E - min(E) + 10^{-8}}{max(E) - min(E) + 10^{-8}} \in [0, 1]^{|B|}$$

where 10^{-8} is used to prevent computational error when $max(E) = min(E)$.

Then, two batches of transition are sampled. In the first batch,B_C, transitions with a large TD error has a higher probability to be sampled, and it will be used to train the critic, and the sampling probability is defined as

$$P_C(i) = \frac{\tilde{E}_i^{\rho_C}}{\sum \tilde{E}_i^{\rho_C}}$$

where i is the index of samples in B and ρ_C is a parameter to tune the priority of the sampling process similar to the prioritized experience replay (PER) [37]. When $\rho_C = 1$, the sampling probability is purely based on the normalized error,namely the sample with largest TD error has the highest probability to be sampled, and when $\rho_C = 0$, every sample in B has the same probability to be sampled.

In the second batch, B_A, transitions with a smaller TD error has a higher probability to be sampled, and it will be used to train the actor where the sampling probability for each sample in B is defined as

$$P_A(i) = \frac{(1 - \tilde{E}_i)^{\rho_A}}{\sum (1 - \tilde{E}_i)^{\rho_A}}$$

Algorithms 10 and 11 show the details of DDPG with Separate Sampling (SSDDPG), the workflow is also shown in 1.5.

Algorithm 10: SSDDPG

Result: $\mu_\Theta \approx \mu^*$

Input: a differentiable policy parameterization μ_Θ, a target policy $\mu_{\Theta_{targ}} := \mu_\Theta$

Input: a differentiable state-action value function parameterization $Q_{\Theta'}$, a target Q-function $Q_{\Theta'_{targ}} := Q_{\Theta'}$

Parameters: α_A, α_C, $Iter$, ρ, $freq$, γ, a_{Low},a_{High}, $B_{total}, B_{train}, \rho_C, \rho_A$

Initialize policy parameter Θ, state-value parameters Θ', episode count $epc = 0$, step count $step = 0$, an empty experience buffer $\mathcal{D}$

while *True* **do**

  Initialize first state of episode S

  $d = 0$

  while S *is not terminal* **do**

    $A = min[max[\mu_\Theta(S) + \epsilon, a_{Low}], a_{High}]$, where $\epsilon \sim \mathcal{N}$

    Take action A and observe S', R

    $d = 1$ if S' is terminal

    Store (S, A, R, S', d) in the replay buffer $\mathcal{D}$

    $S \leftarrow S'$

    $step \leftarrow step + 1$

    if $step\%freq = 0$ **then**

      $B_C, B_A = SeparateSampling(B_{total}, B_{train}, D)$

      Compute targets for transitions in B_C

$$y(r, s', d) = r + \gamma(1 - d)Q_{\Theta'_{targ}}(s', \mu_{\Theta_{targ}}(s')), (r, s', d) \sim B_C$$

      Update Q-function by one step of gradient descent using

$$\Theta' \leftarrow \Theta' - \alpha_C \nabla_{\Theta'} \frac{1}{B_{train}} \sum_{(s,a,r,s',d) \in B_C} (Q_{\Theta'}(s, a) - y(r, s', d))^2$$

      Update policy by one step of gradient ascent using

$$\Theta \leftarrow \Theta + \alpha_A \nabla_\Theta \frac{1}{B_{train}} \sum_{s \in B_A} Q_{\Theta'}(s, \mu_\Theta(s))$$

      Update the target networks with

$$\Theta_{targ} \leftarrow \rho\Theta_{targ} + (1 - \rho)\Theta$$

$$\Theta'_{targ} \leftarrow \rho\Theta'_{targ} + (1 - \rho)\Theta'$$

    end

  end

  $epc \leftarrow epc + 1$

  if *epc=Iter* **then**

    break

  end

end

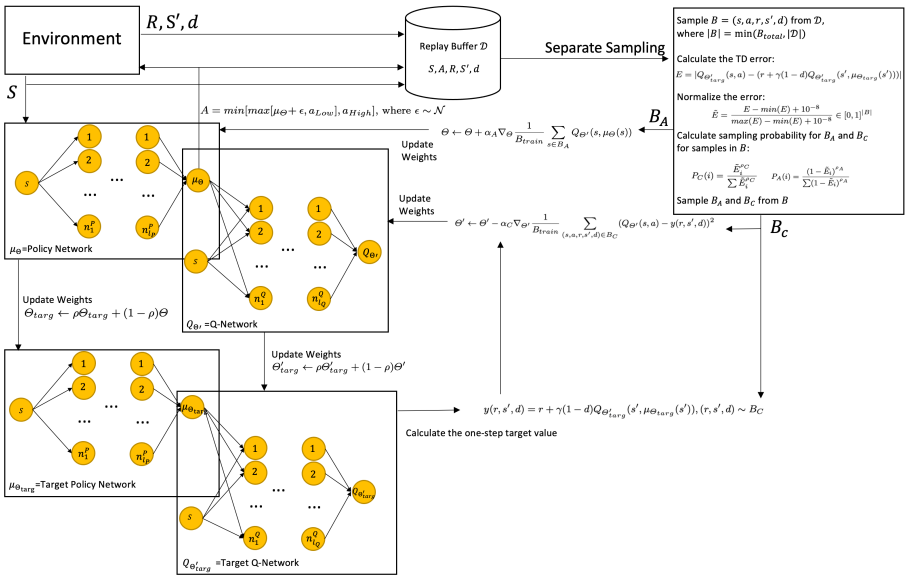


Figure 1.5: Overall Framework of the Proposed Method

1.4 Model

1.4.1 Problem Setting

The assumptions and notations for the problem are as follow,

1. There are N independent jobs at $t = 0$, $N \in \mathbb{Z}^+, N < \infty, t \in \mathbb{R}^+$, and jobs are indexed by j.

2. There are M independent machines, $M \in \mathbb{Z}^+, M < \infty$, and machines are indexed by i. All machine are available at $t = 0$ and each machine can process a single job at any given time.

3. Jobs can be assigned to any machine i.

4. Each job j has a class $k_j \in \{0, 1, 2, ..., K\}, K \in \mathbb{Z}^+$ from a predefined distribution, and its due date $D_j \in \mathbb{R}^+$ and priority $\omega_j \in [0, 1]$ are random variable from general distributions that are parameterized by k_j

Algorithm 11: SeparateSampling

Result: B_C, B_A

Input: $B_{total}, B_{train}, \rho_C, \rho_A, \mathcal{D}$

Randomly sample $B = (s, a, r, s', d)$ transition from $\mathcal{D}$, where $|B| = min(B_{total}, |\mathcal{D}|)$

Compute absolute TD error for transitions in B

$$E = |Q_{\Theta'_{targ}}(s, a) - (r + \gamma(1 - d)Q_{\Theta'_{targ}}(s', \mu_{\Theta_{targ}}(s')))| \in \mathcal{R}^{|B|}$$

Normalize the absolute TD error by

$$\tilde{E} = \frac{E - min(E) + 10^{-8}}{max(E) - min(E) + 10^{-8}} \in [0, 1]^{|B|}$$

Assign sampling probability to each transition in B indexed by i,

$$P_C(i) = \frac{\tilde{E}_i^{\rho_C}}{\sum \tilde{E}_i^{\rho_C}}$$

Sample B_C from B based on the probability P_C, where $|B_C| = min(B_{train}, |B|)$

Assign sampling probability to each transition in B indexed by i,

$$P_A(i) = \frac{(1 - \tilde{E}_i)^{\rho_A}}{\sum (1 - \tilde{E}_i)^{\rho_A}}$$

Sample B_A from B based on the probability P_A, where $|B_A| = min(B_{train}, |B|)$

5. The on-time t_i^{on} of machine i is a random variable from a general distribution $G(i, K_1(i), K_0(i), T_i)$, where $K_1(i)$ and $K_0(i)$ are the classes of the current and previous job of machine i, T_i is the previous maintenance time of machine i. If the current job is the first job for the machine, $K_0(i) = K_1(i)$.

6. If job j is assigned to machine i, job j requires $p_{K_0(i),j} \in \mathbb{R}^+$ time units to be setup and completed, and $p_{K_0(i),j}$ is a random variable from general distribution parameterized by k_j and $K_0(i)$.

7. If a breakdown occurs while it is processing a job, the job will be put back into the queue. And when the job is reassigned to any available machine, the job will be processed from the beginning.

8. As machine's off-time t_i^{off} may be impacted by outside factors, we assume off-time of all machines is from the same general distribution.

9. The status of machine i is indicated by I_i, for $i = 1, 2, 3, ..., M$, and $I_i = -1/0/1$ represent i is down/idle/busy.

10. The transportation time from queue to machine is assumed to be negligible.

11. All other resources are assumed to be available during the manufacturing process.

Jobs are heterogeneous as the due date, process time, and priority of each job vary based on the class of the job. And since the on-time distributions change over time and are non-identical across machines, machines are heterogeneous and non-stationary.

Let $Q(t)$ be the set of jobs in queue at time t and define

$$\mathcal{T} = \{t : |Q(t)| > 0\} \tag{1.26}$$

When $j \in Q(t)$ is assigned to an available machine i at $t \in \mathcal{T}$. If $p_{K_0(i),j} \leq t_i^{on}$, j will be finished after $p_{K_0(i),j}$. Otherwise, j will be sent back to the queue after t_i^{on} as the machine is going to a down-state.

Therefore, given the current time is t, the next event will be happening at

$$t' = min\{\mathcal{M}|I_i| + t,\ \mathcal{M}|I_i - 1| + C_i^{on},\ \mathcal{M}|I_i + 1| + C_i^{off} \text{ for } \forall i\} \tag{1.27}$$

where $\mathcal{M}$ is a large number; C_i^{on} is the time when the job on i is finished if the machine does not have a breakdown during the producing it, otherwise, it is the time the machine i is down; C_i^{off} is the time when the maintenance is finished if the machine is down. Without loss of generality, both C_i^{on} and C_i^{off} are set to 0 by default for $i = 1, 2, 3, ..., M$. When $Q(t) = 0$, the next event will be at

$$t' = min\{\mathcal{M}|I_i - 1| + C_i^{on},\ \mathcal{M}|I_i + 1| + C_i^{off} \text{ for } \forall i\} \tag{1.28}$$

When $Q(t) = 0$ and all machines are idle, then the session is completed.

As every decision making is for assigning a job $j \in Q(t)$ to an available machine i at $t \in \mathcal{T}$, the urgency of a job $j \in Q(t)$ is defined as

$$U_j^{(i)} = \omega_j |t + p_{K_0(i),j} - D_j|^+ \tag{1.29}$$

The urgency of jobs in the queue for the machine i therefore can be defined as

$$U^{(i)}(t) = \{U_j^{(i)} \text{ for } j \in Q(t)\} \in \mathbb{R}^{|Q(t)|} \tag{1.30}$$

Similarly, the processing time of jobs in the queue is defined as

$$p^{(i)}(t) = \{p_{K_0(i),j} \text{ for } j \in Q(t)\} \in \mathbb{R}^{|Q(t)|} \tag{1.31}$$

Let $\beta_i^{on}(t)$ be the mean of on-time distributions $G(i, K_1(i), K_0(i), T_i)$ of machine i at time t, it can be written as

$$\beta_i^{on}(t) = \begin{bmatrix} \beta_{i,0,0}^{on}(t - T_i) & \beta_{i,0,1}^{on}(t - T_i) & \dots & \beta_{i,0,k}^{on}(t - T_i) \\ \beta_{i,1,0}^{on}(t - T_i) & \beta_{i,1,1}^{on}(t - T_i) & \dots & \beta_{i,1,k}^{on}(t - T_i) \\ \vdots & \vdots & \ddots & \vdots \\ \beta_{i,k,0}^{on}(t - T_i) & \beta_{i,k,1}^{on}(t - T_i) & \dots & \beta_{i,k,k}^{on}(t - T_i) \end{bmatrix} \tag{1.32}$$

The element $\beta_{i,K_1(i),K_0(i)}^{on}(t - T_i)$ represents the mean on-time of machine i at t when the current job class is $K_1(i)$ and the previous job class is $K_0(i)$.

When a job is assigned to a machine to greedily maximize the mean on-time of the machine, it might decrease the maximum mean on-time for other machines. Therefore, the total mean on-time that can be obtained for an assignment should be considered during the scheduling. Let $\beta_{i,j}^{on^*}(t)$ be the maximum average on-time mean at time t if j is assigned to i where $j \in Q(t)$, and it can be obtained from

$$
\begin{aligned}
\beta_{i,j}^{on^*}(t) = \max_{x} \quad & \tfrac{1}{M} \sum_{i'=1}^{M} \sum_{j' \in Q(t)} \beta_{i,K_1(i),K_0(i)}^{on}(t - T_i) x_{j',i'} \\
\text{subject to} \quad & x_{j,i} = 1 \\
& \textstyle\sum_{i'=1}^{M} x_{j',i'} \leq 1 && \forall j' \in Q(t) \\
& \textstyle\sum_{j' \in Q(t)} x_{j',i'} \leq 1 && \forall i' \\
& x_{j',i'} \in \{0, 1\} && \forall i', \forall j' \in Q(t)
\end{aligned}
$$

where $x_{j',i'}$ is the decision variable, and $x_{j',i'} = 1$ if job j' is assigned to i', otherwise, $x_{j',i'} = 0$.

We denote $\beta_i^{on^*}(t) \in \mathbb{R}^{|Q(t)|}$ as maximum average on-time mean of jobs in queue at time t.

A function $\mathcal{F} : \cup_{1 \leq d \leq N} \mathbb{R}^d \to \mathbb{R}^7$ is defined to return basic statistics of the input, including mean, standard deviation, minimum, 25%, 50%, 75% and maximum, $n \in \mathbb{N}$. In the case of $d = 1$, the standard deviation is defined as 0.

For any available machine i at $t \in \mathcal{T}$, the processing time, urgency, and the maximum mean on-time of jobs in the queue can therefore be characterized by $\mathcal{F}(p^{(i)}(t))$, $\mathcal{F}(U^{(i)}(t))$, and $\mathcal{F}(\beta_i^{on^*}(t))$ respectively.

1.4.2 Markov Decision Process

The scheduling problem is modeled as a Markov Decision Process (MDP), namely, a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$, where

- $\mathcal{S} \in \mathbb{R}^{7 \times 3}$ is the state space, and $S_t = [\mathcal{F}(p^{(i)}(t)), \mathcal{F}(U^{(i)}(t)), \mathcal{F}(\beta_i^{on^*}(t))] \in \mathcal{S}$. Note that the state is observed whenever an event happens, see equation (2) and (3). When there are multiple available machines at a given time t, the agent randomly selects a machine to assign jobs.

- $\mathcal{A} \in \mathbb{R}^3$ is the action space, and $A_t = [\theta_u, \theta_p, \theta_\beta]$. A job j will be assigned to

machine i if i is available at time t:

$$j = argmax_{j' \in Q(t)} \theta_u U_{j'}^{(i)} + \theta_p \frac{1}{p_{K_0(i),j'}} + \theta_\beta \beta_{i,j'}^{on^*}(t) \tag{1.33}$$

- $\mathcal{P}$ is a state transition probability matrix,

$$\mathcal{P}_{ss'}^{a} := \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a] \tag{1.34}$$

  As the system dynamics in the problem setting is complex, the probability matrix will not be explored explicitly.

- $R \in \mathbb{R}^-$ is a reward signal. Machine i will receive R_t when it finishes a job j at time t, where

$$R_t = -\omega_j |t - D_j|^+ \tag{1.35}$$

  The agent receives the maximum reward 0 unless it misses the deadline.

- $\gamma \in [0, 1]$ is a discount factor, and γ is set to 0.9999 to ensure the decision making process considers future impacts.

A policy π is a distribution over actions given states,

$$\pi(a|s) = \mathcal{P}[A_t = a | S_t = s] \tag{1.36}$$

For any given trajectory $\tau = s_{t_0}, a_{t_0}, R_{t_0}, s_{t_1}, a_{t_1}, ..., R_{t_H}$, the total cumulative reward of the trajectory is calculated as

$$G(\tau) = \sum_{q=0}^{H} R_{t_q} \tag{1.37}$$

And for a given policy π, the expected reward, $\mathbb{E}[G]$ is defined as

$$\mathbb{E}[G] = \int_{\tau=s_{t_0},a_{t_0},R_{t_0},\ldots} \mathbb{P}_\pi(\tau)G(\tau)d\tau \tag{1.38}$$

Therefore, the objective function of the defined MDP is

$$J = \max_\pi \int_{\tau=s_{t_0},a_{t_0},R_{t_0},\ldots} \mathbb{P}_\pi(\tau)G(\tau)d\tau \tag{1.39}$$

1.5 Numerical Study

1.5.1 Comparing DDPG and SSDDPG

To test the performance of proposed sampling method, a DDPG agent and a SSDDPG agent are trained to learn Pendulum using OpenAI gym. Note that since the AC algorithm (Algorithm 1) has the discrete action space, we skip the AC algorithm in this performance test. Other than the SSDDPG agent uses the proposed prioritized sampling method, two agents have the same hyperparameters. The actor and critic in both agents are parameterized by neural networks. Detailed configurations are shown in Table 1.1. The resulting performances over 50 runs

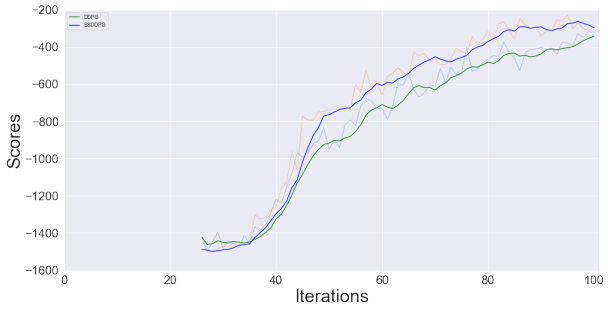


Figure 1.6: Performances of DDPG and SSDDPG on the Pendulum task

are shown in Figure 1.5. In the first 25 iterations in each run, both agents will act randomly to collect experiences without training. Once the pure exploration

Table 1.1: Agent Configurations for Pendulum

Parameters	Value
α_A	0.001
α_C	0.001
ρ	0.995
γ	0.999
$freq$	1
$Iter$	100
a_{Low}	-2
a_{High}	2
B_{total}	256
B_{train}	128
ρ_A	1
ρ_C	1
Neural network structure (actor)	[32,16]
Activation function (actor)	ReLU,ReLU,Linear
Neural network structure (critic)	[32,16]
Activation function (critic)	ReLU,ReLU,Linear

period is over, agents will start to learn based on the transitions they have collected. The two lines show the average rewards of DDPG agent and SSDDPG agent over 50 runs in the last 75 iterations. As we can see after 10 iterations after agents start to learn, the performance of SSDDPG agent dominates the performance of DDPG agent. The result indicates that proposed sampling method does expedite the learning process of DDPG. Though they both will reach a same performance at the end, the proposed sampling method offers a faster convergence.

1.5.2 Scheduling using SSDDPG

To test the performance of SSDDPG agent in the scheduling task, numerical experiments are conducted. As the DDPG method is a model-free method, the agent is not given any model dynamics and the agent will learn a policy simply based collected observations and the reward signal. The performance of SSDDPG is compared against three heuristic methods, namely, earliest weighted due date (EWDD) rule, shortest processing time (SPT) rule and maximum mean on-time

(MMOT) rule.

For EWDD rule, a job j is selected at time t if a machine is available, where

$$j = argmin\{\frac{D_j}{\omega_j}, \forall j \in Q(t)\} \tag{1.40}$$

For SPT rule, a job j is selected at time t if a machine i is available, where

$$j = argmin\{p_{K_0(i),j}, \forall j \in Q(t)\} \tag{1.41}$$

For MMOT rule, a job j is selected at time t if a machine i is available, where

$$j = argmax\{\beta_i^{on^*}(t), \forall j \in Q(t)\} \tag{1.42}$$

System dynamics of the MDP is described in Table 1.2 and the configurations of RL agent is shown in Table 1.3.

Table 1.2: System Dynamics

Random Variable	Distribution parameters
Job type	Uniformly distributed integer between 1 and K
Job process time	Exponential distribution
Job due date	Uniformly distributed between 1 and 24 hours
Job priority	Uniformly distributed between 0 and 1
Machine on-time	Exponential distribution
Machine off-time	Exponential distribution with a rate of 0.5

Since distribution of the process time of a job is parameterized by the its type as well as the type of its previous job, a matrix of rates are used to define the rate of the exponential distribution. Since our assumption is that if similar jobs are assigned to a machine back-to-back, the second job will have shorter process time, a symmetric similarity matrix $SM \in \mathbb{R}^{K \times K}$ is generated where each element is uniformly distributed between 0.25 and 1. Each job has a base rate BR uniformly

Table 1.3: Agent Configurations for Scheduling

Parameters	Value
α_A	0.00003
α_C	0.00003
ρ	0.9
γ	0.99999
$freq$	1
$Iter$	200
a_{Low}	[-10, -10, -10]
a_{High}	[10, 10, 10]
B_{total}	256
B_{train}	128
ρ_A	1
ρ_C	1
Neural network structure (actor)	[128, 64, 32]
Activation function (actor)	ReLU, ReLU, ReLU, Linear
Neural network structure (critic)	[128, 64, 32]
Activation function (critic)	ReLU, ReLU, ReLU, Linear

distributed between $\frac{1}{12}$ and 1. If a job j_2 is assigned after j_1 to a the same machine, the rate of j_2 is $BR_{j_2} \times SM[k_{j_1}, k_{j_2}]$. If j_1 and j_2 has a similarity of 1, then the rate equals to its base rate, otherwise, the rate decreases and therefore result in a longer process time. In practice, the rate matrix for the process time can be generated based on the data.

The distribution of the machine on-time is parameterized by the machine ID, types of the current job and privous job, as well as the time since the last breakdown, and we modeled the rate based on the function shown in (1.32). The initial on-time rate (right after a breakdown/when the system starts) for each machine is uniformly distributed between $\frac{1}{2}$ to $\frac{1}{24}$. Similar to the rate matrix of the process time,the impact of types of jobs on the on-time rate is modeled by multiplying the rate to one over the similarity matrix SM. Then, the rate increases with a factor of $1.001^{(t-T)}$ where t is the current time, and T is the previous breakdown time of the machine.

Settings and results of the experiments are shown in Table 1.3. The Results column contains the average performance over 500 simulations under each method

and setting. The t-test is used to compare the outcomes of 500 simulations of each heuristic against proposed method. From the Results column, it can be observed that proposed method outperforms the heuristics in every setting, and P-value column indicates the the proposed method is significantly better than the heuristics as all p-values are less than 0.01.

Table 1.4: Results of Experiments

N	M	K	Methods	Results	P-value
30	3	10	EWDD	1553.91	3.96×10^{-6}
			SPT	1625.45	5.35×10^{-9}
			MMOT	1528.11	1.04×10^{-5}
			SSDDPG	**1315.34***	-
50	5	15	EWDD	2287.34	1.61×10^{-3}
			SPT	2290.60	3.27×10^{-4}
			MMOT	2282.01	1.49×10^{-3}
			SSDDPG	**2077.87***	-
100	10	35	EWDD	4227.11	3.36×10^{-3}
			SPT	4324.44	9.61×10^{-4}
			MMOT	4226.10	6.12×10^{-3}
			SSDDPG	**4033.19***	-

In short, Table 1.4 indicates a clear domination relationship that would distinguishes SSDDPG and other heuristics. Some potential reasons that might explain SSDDPG outperforms other heuristics are:

- The action space of the approach is composed of weights for heuristics, namely, θ_u,θ_p,θ_β. It gives the RL agent the flexibility to select among heuristics.
- Since the weights for heuristics are continuous variables, it allows the agent to create new heuristics by interpolations.
- The weights are selected dynamically based on the current state.

In a practitioner space, skilled subject matter experts are likely to have prior knowledge on useful heuristics but not likely to know the relative significance

between them. The definition of action space used in this approach is useful in these cases because it can be used to discover the relative and dynamic significance between the rules.

Additionally, since the state space does not have dependency on the number of machines and jobs, it makes the approach robust and the data collected across different plants can be used to train the agent.

One of the assumptions in the paper is that types of jobs next to each other have an impact on the mean on-time of machines. Specifically, since similar jobs require less configurations to the machine, when the jobs next to each other have similar job types, the average on-time of the machine is relatively high when the new job is processed. Given this assumption, we would expect that an intelligent agent considers this factor during the scheduling process and it should tend to pick the next job for the available machine such that the type-pair of the new job and the previous job has a relatively high mean on-time for the machine. The Fig. 1.7, obtained from the first experiment setting, shows that the type-pair that has a higher mean on-time are selected more frequently. Since the mean on-time matrix (1.32) is generated randomly, and different trajectories may have a different maximum value, the mean on-time of each trajectory is normalized between 0 to 1 in the Fig. 1.7. The result indicates that the agent does pick new job frequently to ensure the on-time of the machine is relatively high such that the machine is less likely to have a breakdown.

From equation (1.33), we can see that the action of the agent is a function of θ_u,θ_p, and θ_β. When θ_u is larger than the remaining two, a job that has a large urgency is likely to be selected. Naturally, an intelligent agent should tend to have an relatively large θ_u when the urgency of waiting jobs is high. The Fig. 1.8 shows the urgency of waiting jobs and normalized θ_u, where the normalized θ_u is calculated by

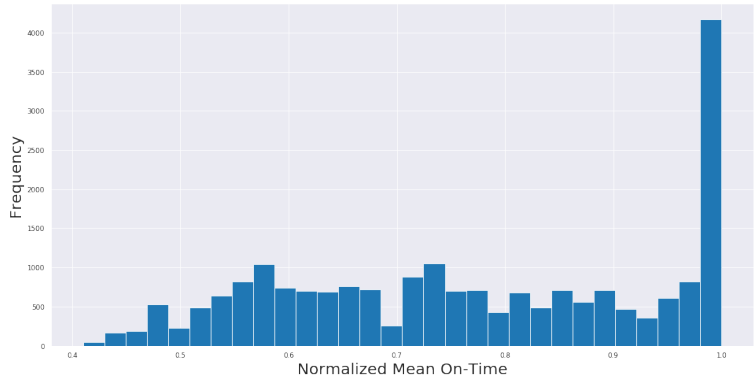


Figure 1.7: The Frequency of Mean On-time Selection of the Resulting RL Agent

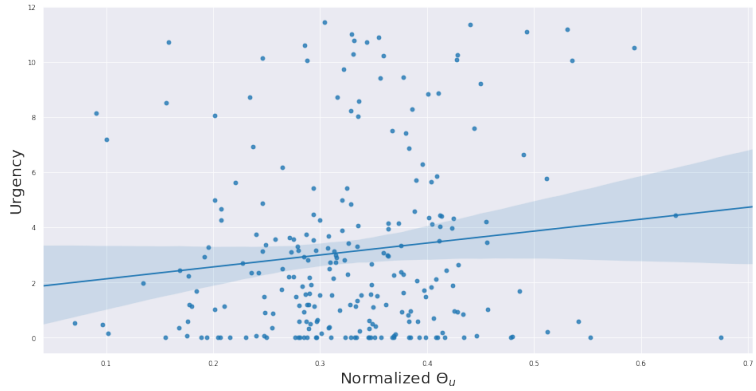


Figure 1.8: Urgency of Waiting Jobs and the Value of Normalized θ_u

$$\text{Normalized } \theta_u = \frac{\theta_u - min([\theta_u, \theta_p, \theta_\beta])}{\theta_u + \theta_p + \theta_\beta - 3min([\theta_u, \theta_p, \theta_\beta])}$$

and the fitted regression line has a significant coefficient with a p-value less than 0.001. The result indicates that the resulting agent tends to prioritize the urgency when the due date of waiting jobs is approaching.

1.6 Conclusions

In this chapter, a parallel machine scheduling problem with heterogeneous jobs and non-stationary unreliable machines is modeled as an MDP, and a SSDDPG agent

is used to learn to schedule to minimize the weighted tardiness in the MDP. The obtained policy and other heuristics are then used in the simulated environment to compare their performance, and the result indicates that the proposed method is able to produce a policy that significantly outperforms the heuristics. Since the SSDDPG is able to learn in such complex MDP, it indicates that given proper state space and action space formulations, reinforcement learning is able to tackle complicated scheduling problems in the real world. To schedule jobs in real-time, a simulation environment needs to be constructed such that it represents the interested manufacturing environment. After training a RL agent within the MDP, the obtained policy can be used to schedule jobs in real-time.

A separated sampling method is proposed in this chapter, and the computational results indicate that it boosts up the learning speed of DDPG agent and therefore results in a faster convergence. In the context of scheduling, this gives a shorter iteration cycle which can be valuable in today's competitive manufacturing environment.

Since a model free RL approach is used in the work, we do not have assumptions on the system dynamics of the underlying MDP. However, the system dynamics might have an impact on the performance of the algorithm, therefore, a sensitivity analysis can be done to test the performance of the proposed method across various system dynamics. Additionally, other RL methods can be used to compare the performances.

Chapter2
A Mixed Integer Programming Approach for Job Shop Scheduling with Sequence-dependent Setup Time and Tool Overheating Constraints

2.1 Introduction

High-mix and low-volume manufacturing facilities (HMLV-fab) process jobs with relatively small order sizes and heterogeneous specifications [1]. Due to the high variety of products, machines are often reconfigured to process a new job, such as tool changes, tool path and cutting parameters adjustments. In addition, it is preferred to schedule jobs with similar specifications together. Failing to do so might cause additional setup times and potentially increase the number of unplanned machine breakdowns [2]. Figure 2.1 shows two schedules of HMLV jobs.

In Schedule 1 (S1), the same type of jobs are scheduled together, and in Schedule 2 (S2), the same jobs are not scheduled together. The resulting total setup time of S2 is larger than S1, and therefore S1 is preferred in general.

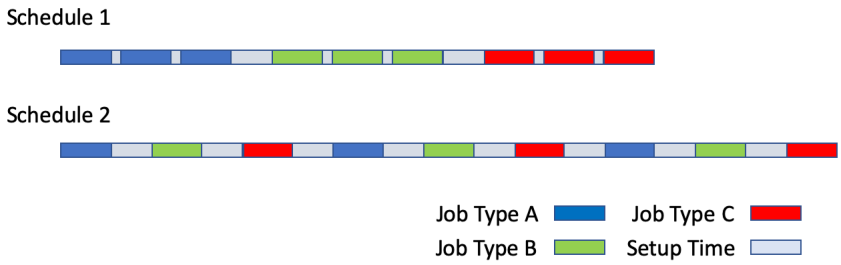
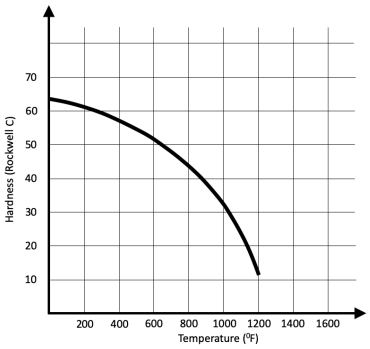


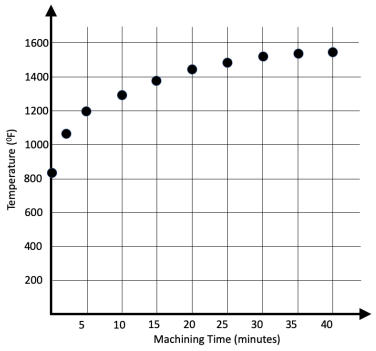
Figure 2.1: Schedules of Mixed Jobs

However, in a dry machining environment where the usage of liquid coolants is restricted, using the same tool for a prolonged period of time is likely to cause the tool to be overheated. When the temperature of the tool increases, its hardness decreases, which might cause a tool breakage leading to scrapped parts. For example, high-speed steel (HSS) is one of the commonly used machining tools [38], and Figure 2.2 a) shows that the hardness of HSS decreases as the temperature increases [39]. When the hardness of the HSS decreases to a value that is close to the hardness of the work piece, a tool failure might arise. Figure 2.2 b) shows the tool tip temperature of HSS in a continuous dry machining process [40], and it can be observed that the hardness of HSS can decrease to 10 HRC (Rockwell C) after 5 minutes. Therefore, it is crucial to avoid tool overheating by scheduling jobs that require different tools together in a dry machining environment.

Though waiting for the tool to cool down is an alternative solution, the resulting scheduling can be inefficient in a dry machining environment. A typical cooling rate for air blowing ranges from $3K/s$ (slowly cooled) to $10K/s$ (fast cooled) [41], and based on the Continuous Cooling Transformation (CCT) Diagram [42] of AISI M42 HSS, it takes around $200s$ to $600s$ to cool down the tool from 1600^oF to



(a) Hardness and Temperature of HSS



(b) Machining Time and Temperature of HSS in A Dry Machining Process

Figure 2.2: Temperatures, hardness and Machining Time of HSS

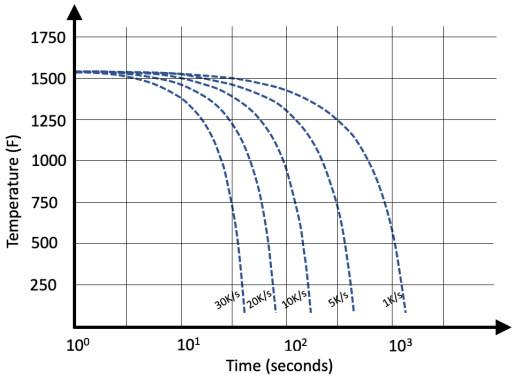


Figure 2.3: CCT Diagram of AISI M42 HSS

800^oF. When the number of tools being limited in the tool magazine, it is favored to switch to a different job and allow the overheated tool to cool down.

While there are researches on tool constraints in scheduling, the focus has been on tool allocations and replacements. To the best of our knowledge, there is no research consider overheating constraints in scheduling. Therefore, in this chapter, a mixed integer programming (MIP) model is proposed for job shop scheduling (JSSP) with job sequence-dependent setup time (SDST) and overheating constraints (OC). The rest of the paper is organized as follows. In the following section, related studies are reviewed. In section 3, the proposed model is described in detail. Results

of numerical studies are presented and examined in section 4. In the last section, conclusions and future work are discussed.

2.2 Related Works

The JSSP have been studied since the 1950s and the minimization of the makespan in JSSP is a well-known NP-hard problem [43]. One of the standard assumptions of classic JSSP is that the setup times are included in processing times; however, it is not a valid assumption when the setup time of a job depends on the previous one. The more similar two successive jobs are, the less setup time the later one will require. Such setup times are called SDST.

The JSSP with SDST to minimize the makespan is addressed by many researchers due to the applicability of the problem [44, 45] to different contexts. As it is a combinatorial optimization problem, most of the works are on developing meta-heuristics, and hybrid methods to tackle the problem. Commonly used meta-heuristics are the Genetic algorithm [46–49], Tabu search [50, 51], and Simulated annealing [49, 52]. The formulation of MIP models is still an active area, and a new model is proposed recently [53] , and the model has a smaller number of constraints and variables compared to other formulations [54, 55].

Constraints and objectives that are related to tools are often considered in the context of flexible manufacturing systems. The schedule of operations and allocation of tools to tool magazine with limited capacity is studied by Stecke [56]. A nonlinear MIP model is formulated, and linearizing methods are suggested to use the method to solve practical problems. In the later work [57], Stecke considers the minimization of the number of tool changes. The research finds that the number of tool changes due to product variety is small compared to changes due to tool wear. However, there are more ways to extend tool life, and tools with a longer lifespan

are built, and products are getting highly mixed. However, such possibilities may not be valid in many cases.

For the last three decades, various methodologies are developed to solve scheduling that minimizes the number of tool changes [58]. Nonlinear integer programming models are developed by Bard [59] and Van Hop [60] . Bard solves the model using a dual-based relaxation heuristic method. Van Hop develops a Genetic algorithm to solve the model. Mixed integer nonlinear programming models are proposed by Lee [61] and Özpeynirci [62]. Heuristic methods are used in the work of Khan [63], Fathi [64], and Gükgör [65]. Solving large problems might require meta-heuristics, but theoretical models are needed to define the problem.

Both JSSP with SDST and scheduling with tool-related objectives/constraints are well studied. Nonetheless, the minimization of setup times and the number of tool changes in a dry machining environment may result in overheated tools. It decreases the tool life, and it also increases the chance of producing scrapped parts. Therefore, this work aims to fill the gap in the literature by adding upper bounds on usage times of tools.

2.2.1 Performance Measures

Commonly used performance measures in JSSP are:

1. Maximum completion time, or makespan: $C_{max} = max(C_1, ..., C_i, ..., C_n)$, where C_i is the completion time of job i. C_{max} represents the overall time to complete all n jobs, which is the completion time of the last job minus the starting time of the first job.

2. Maximum flow time: $F_{max} = max(F_1, ..., F_i, ..., F_n)$, where $F_i = C_i - S_i$, and S_i is starting time of job i. Minimizing F_{max} is able to minimize the worst case of time each job spends in the system.

3. Expected flow time $\bar{\mathrm{F}}= \frac{1}{n}\sum_i^n F_i$

4. Maximum lateness: $L_{max} = max(C_1 - D_1, ..., C_i - D_i, ..., C_n - D_n)$, where D_n is the due date of the job i.

5. Total tardiness $T = \sum_i^n [C_i - D_i]^+$

6. Total weighted tardiness $T = \sum_i^n \omega_i [C_i - D_i]^+$, where ω_i is the weight or priority of job i.

7. Expected tardiness $\bar{\mathrm{T}}= \frac{1}{n}\sum_i^n [C_i - D_i]^+$

8. Number of tardy jobs $n_T = \sum_i^n \mathbb{1}([C_i - D_i]^+)$, where $\mathbb{1}([C_i - D_i]^+) = 1$ if $[C_i - D_i]^+ > 0$, and $\mathbb{1}([C_i - D_i]^+) = 0$ otherwise.

2.3 Methodology

In this section, a MIP model is formulated for JSSP with SDST and OC based on the classic disjunctive model by Manne [66], and notations are listed in the Table 2.1.

2.3.1 Notations

A JSSP consists of a finite set of jobs J and a finite set of machines M. Each job has a finite set of operations $O_j, j \in J$, and each operation requires a machine $M_{i,j}$ and it takes $P_{i,j}$ units of time to process, $i = 1, 2, 3, ..., n_j, j \in J$. There are precedence constraints between operations and the operation order of each job is fixed, $O_{i,j}$ can only be processed after $O_{i',j}, \forall j \in J$ if $i' < i$. Additionally, we assume that

- All jobs, machines and other resources are available from the beginning.

- An operation can be processed on one machine at a time.
- A machine can process one job at a time.
- Each operation of a job has to be processed once.
- All operations of a job has be completed to finish the job.
- Jobs are independent.
- Preemption is not allowed.
- Each machine is equipped with one of each type of tools.
- $P_{i,j} \geq D_{p,k}$, $\forall O_{i,j} \in OT_{p,k}, p = 1, 2, 3, ..., p_k, k \in M$.
- $P_{i,j} \leq C_{p,k}$, $\forall O_{i,j} \in OT_{p,k}, p = 1, 2, 3, ..., p_k, k \in M$.

Though two successive jobs on a machine have a smaller setup time if they require the same tool, a smaller setup time does not necessarily imply the successive jobs share the same tool. Therefore, SDST and types of tool are defined separately in the model.

2.3.2 Model

The mixed integer programming model is formulated as following,

- Decision variables
  1. $x_{i,j} \in \mathbb{R}^+$: the starting time of the operation i of job j, $\forall O_{i,j} \in O_j$, $j \in J$
  2. $y_{i_1,j_1,i_2,j_2} \in [0, 1]$: $y_{i_1,j_1,i_2,j_2} = 1$ if operation O_{i_2,j_2} is scheduled right after O_{i_1,j_1} on the machine k, where $k = M_{i_1,j_1} = M_{i_2,j_2}$, and $y_{i_1,j_1,i_2,j_2} = 0$ otherwise, $\forall O_{i_1,j_1} O_{i_2,j_2} \in OM_k, O_{i_1,j_1} \neq O_{i_2,j_2}, k \in M$

n	the number of jobs
m	the number of machines
J	the set of jobs, $J = \{1, 2, 3, ..., n\}$, indexed by j
M	the set of machines, M=$\{1,2,3,...,m\}$, indexed by k
n_j	the number of operations of job j
O_j	the set of operations of job j, $O_j = \{O_{1,j}, O_{2,j}, O_{3,j}, ..., O_{n_j,j}\}$
M_j	the set of machines required by operations of job j, $M_j = \{M_{1,j}, M_{2,j}, M_{3,j}, ..., M_{n_j,j}\}$
P_j	the set of processing time of operations of job j, $P_j = \{P_{1,j}, P_{2,j}, P_{3,j}, ..., P_{n_j,j}\}$
T_j	the set of tool required by operations of job j, $T_j = \{T_{1,j}, T_{2,j}, T_{3,j}, ..., T_{n_j,j}\}$
OM_k	the set of operations of machine k, $OM_k = \{O_{i,j} \in O_j, \forall j \in J \mid M_{i,j} = k\}, k \in M$
p_k	the number of tools in the tool magazine of machine k
$OT_{p,k}$	the set of operations require tool p on machine k, $OT_{p,k} = \{O_{i,j} \in O_j, \forall j \in J \mid M_{i,j} = k, T_{i,j} = p\}$, $k = 1, 2, ..., p_k, k \in M$
S_{i_1,j_1,i_2,j_2}	the setup time of O_{i_2,j_2} if it is scheduled after O_{i_1,j_1} on the same machine k, where $k = M_{i_1,j_1} = M_{i_2,j_2}$
$C_{p,k}$	the capacity of tool p of machine k
$D_{p,k}$	the length of cool down period required by tool p of machine k after it is overheated
$\mathcal{M}$	a big number

Table 2.1: Notations

3. $g_{i_1,j_1,i_2,j_2} \in [0,1]$: $g_{i_1,j_1,i_2,j_2} = 1$ if $y_{i_1,j_1,i_2,j_2} = 1$ and the gap between ending of O_{i_1,j_1} and beginning time of O_{i_2,j_2} is greater than the length of required cool down time for tool p, and $g_{i_1,j_1,i_2,j_2} = 0$ otherwise, where $p = T_{i_1,j_1} = T_{i_2,j_2}, \forall O_{i_1,j_1}, O_{i_2,j_2} \in OM_k, O_{i_1,j_1} \neq O_{i_2,j_2}, \forall k \in M$

4. $d_{i,j,f} \in [0,1]$: $d_{i,j,f} = 1$ if $O_{i,j}$ is the first operation on machine $M_{i,j}$, and $d_{i,j,f} = 0$ otherwise, $\forall O_{i,j} \in O_j$, $j \in J$

5. $d_{i,j,l} \in [0,1]$: $d_{i,j,l} = 1$ if $O_{i,j}$ is the last operation on machine $M_{i,j}$, and $d_{i,j,l} = 0$ otherwise, $\forall O_{i,j} \in O_j$, $j \in J$

- Constraints

1. Precedence constraints

$$x_{i_1+1,j_1} \geq x_{i_1,j_1} + P_{i_1,j_1} + \sum_{\substack{O_{i_2,j_2} \in OM_{M_{i_1,j_1}} \\ O_{i_1,j_1} \neq O_{i_2,j_2}}} S_{i_2,j_2,i_1,j_1} y_{i_2,j_2,i_1,j_1}, \forall i_1 < n_{j_1}, j_1 \in J \tag{2.1}$$

2. No more than one operation is scheduled on the same machine at the same time

$$\begin{aligned} x_{i_2,j_2} \geq x_{i_1,j_1} + P_{i_1,j_1} + S_{i_1,j_1,i_2,j_2} - (1 - y_{i_1,j_1,i_2,j_2})\mathcal{M}, \\ \forall O_{i_1,j_1} O_{i_2,j_2} \in OM_k, O_{i_1,j_1} \neq O_{i_2,j_2}, k \in M \end{aligned} \tag{2.2}$$

3. Unless an operation is the first/last one, it has to be scheduled after/before another operation on a machine

$$\begin{aligned} \sum_{\substack{O_{i_2,j_2} \in OM_{M_{i_1,j_1}} \\ O_{i_1,j_1} \neq O_{i_2,j_2}}} y_{i_2,j_2,i_1,j_1} = 1 - d_{i_1,j_1,f}, \qquad \forall O_{i_1,j_1} \in O_{j_1}, j_1 \in J \\ \sum_{\substack{O_{i_2,j_2} \in OM_{M_{i_1,j_1}} \\ O_{i_1,j_1} \neq O_{i_2,j_2}}} y_{i_1,j_1,i_2,j_2} = 1 - d_{i_1,j_1,l}, \qquad \forall O_{i_1,j_1} \in O_{j_1}, j_1 \in J \end{aligned} \tag{2.3}$$

4. For each machine, only one operation is the first/last

$$\begin{aligned} \sum_{O_{i,j} \in J_k} d_{i,j,f} = 1, \quad \forall k \in M \\ \sum_{O_{i,j} \in J_k} d_{i,j,l} = 1, \quad \forall k \in M \end{aligned} \tag{2.4}$$

5. Check if the cool down requirement is satisfied for two successive jobs that require the same tool

$$\begin{aligned} x_{i_2,j_2} - x_{i_1,j_1} - p_{i_1,j_1} - D_{T_{i_1,j_1},M_{i_1,j_1}} \geq -\mathcal{M}(1 - g_{i_1,j_1,i_2,j_2}), \\ x_{i_2,j_2} - x_{i_1,j_1} - p_{i_1,j_1} - D_{T_{i_1,j_1},M_{i_1,j_1}} \leq \mathcal{M} g_{i_1,j_1,i_2,j_2}, \\ \forall O_{i_1,j_1}, O_{i_2,j_2} \in OM_k, O_{i_1,j_1} \neq O_{i_2,j_2}, T_{i_1,j_1} = T_{i_2,j_2}, \forall k \in M \end{aligned} \tag{2.5}$$

6. Overheating constraints

$$\begin{aligned} C_{p,k} \geq P_{i_1,j_1} + P_{i_2,j_2} - (1 - y_{i_1,j_1,i_2,j_2} + g_{i_1,j_1,i_2,j_2})\mathcal{M} \\ \forall O_{i_1,j_1} O_{i_2,j_2} \in OT_{p,k}, O_{i_1,j_1} \neq O_{i_2,j_2}, p = 1, 2, ..., p_k, k \in M \end{aligned} \tag{2.6}$$

$$\begin{aligned} C_{p,k} \geq P_{i_1,j_1} + P_{i_2,j_2} + P_{i_3,j_3} - [2 - y_{i_1,j_1,i_2,j_2} - y_{i_2,j_2,i_3,j_3} \\ + g_{i_1,j_1,i_2,j_2} + g_{i_2,j_2,i_3,j_3}]\mathcal{M}, \forall O_{i_1,j_1} O_{i_2,j_2} O_{i_3,j_3} \in OT_{p,k}, \\ O_{i_1,j_1} \neq O_{i_2,j_2} \neq O_{i_3,j_3}, p = 1, 2, ..., p_k, k \in M \\ ... \end{aligned} \tag{2.7}$$

$$\begin{aligned} C_{p,k} \geq P_{i_1,j_1} + ... + P_{i_z,j_z} - [z - y_{i_1,j_1,i_2,j_2} - ... - y_{i_{(z-1)},j_{(z-1)},i_z,j_z} \\ + g_{i_1,j_1,i_2,j_2} + ... + g_{i_{(z-1)},j_{(z-1)},i_z,j_z}]\mathcal{M}, \forall O_{i_1,j_1}, ..., O_{i_z,j_z} \in OT_{p,k}, \\ O_{i_1,j_1} \neq O_{i_2,j_2} \neq ... \neq O_{i_z,j_z}, p = 1, 2, ..., p_k, k \in M \end{aligned} \tag{2.8}$$

7. Non-negative and binary constraints

$$
\begin{aligned}
x_{i,j} &\geq 0, \forall O_{i,j} \in O_j, j \in J \\
y_{i_1,j_1,i_2,j_2} &\in [0,1], \forall O_{i_1,j_1}, O_{i_2,j_2} \in OM_k, O_{i_1,j_1} \neq O_{i_2,j_2}, k \in M \\
g_{i_1,j_1,i_2,j_2} &\in [0,1], \forall O_{i_1,j_1}, O_{i_2,j_2} \in OM_k, O_{i_1,j_1} \neq O_{i_2,j_2}, k \in M \\
d_{i,j,f} &\in [0,1], O_{i,j} \in J_k, k \in M \\
d_{i,j,l} &\in [0,1], O_{i,j} \in J_k, k \in M
\end{aligned} \tag{2.9}
$$

- Objective function

$$
\min_x \max_x (x_{i,j} + P_{i,j} + \sum_{\substack{O_{i',j'} \in OM_{M_{i,j}} \\ O_{i,j} \neq O_{i',j'}}} S_{i',j',i,j} y_{i',j',i,j}) \tag{2.10}
$$

2.4 Numerical Study

To validate the proposed model, three experiments are conducted. The model is solved using the version 12.8.0.0 of CPLEX optimization solver with Python API. Settings of experiments are as follows:

I A well-known example from literature [54] is used as inputs (see Section 2.4.1). By setting $C_{p,k} = +\infty, p = 1, 2, 3, ..., p_k, \forall k \in M$, the overheating constraints are removed and it simplifies the model to the basic JSSP with SDST. This case provides the best schedule that minimizes the makespan for the example.

II The inputs from case I is used to run the experiment. By setting $p_k = 1, \forall k \in M$, each machine has only one tool and all operations on the same machine share the same tool. This condition will idle the machine when its tool is overheated. The resulting makespan is increased as expected.

III With additional inputs related to tools, the example from case I is used to

test the model. Though there are overheating constraints and multiple tools, the obtained optimal schedule provides the same makespan as case I. This is because the optimal schedule uses additional setup times to cool down the tool, and the additional setup times did not increase the makespan in this case.

2.4.1 Case I: Basic JSSP with SDST

Descriptions of input jobs [54] are shown in Table 2.2. There are 4 jobs, each job has 4 operations, and each operation requires one machine. The required machine is different across all the operations of each job and there are 4 machines in total. The setup times are described in Table 2.4.

The optimal makespan of 24 is obtained, which is the same as the optimal result provided in the literature [54]. The optimal schedule is shown in Figure 2.2 and jobs are colored by Job ID.

Table 2.2: Descriptions of Input Jobs

Job ID	Operation ID	Required Machine	Processing Time
1	1	4	2
1	2	3	3
1	3	2	2
1	4	1	3
2	1	4	3
2	2	1	2
2	3	2	7
2	4	3	2
3	1	3	4
3	2	2	3
3	3	4	6
3	4	1	4
4	1	1	10
4	2	2	3
4	3	3	4
4	4	4	5

Table 2.3: Setup Times

		Machine 1			
	Job ID	1	2	3	4
Job ID	Opt ID	4	2	4	1
1	4	0	1	2	0
2	2	1	0	1	0
3	4	1	0	0	1
4	1	1	0	2	0

		Machine 2			
	Job ID	1	2	3	4
Job ID	Opt ID	3	3	2	2
1	3	0	0	1	1
2	3	0	0	1	0
3	2	0	2	0	0
4	2	0	2	1	0

		Machine 3			
	Job ID	1	2	3	4
Job ID	Opt ID	2	4	1	3
1	2	0	0	2	1
2	4	1	0	1	1
3	1	0	2	0	1
4	3	0	2	0	0

		Machine 4			
	Job ID	1	2	3	4
Job ID	Opt ID	1	1	3	4
1	1	0	3	1	3
2	1	2	0	2	2
3	3	1	4	0	3
4	4	1	1	2	0

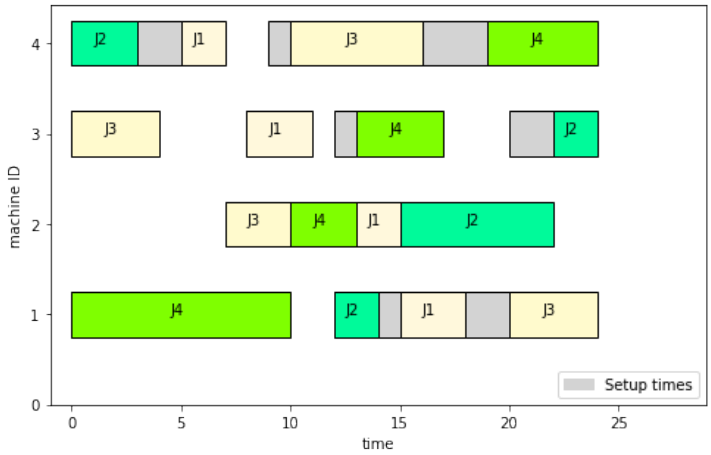


Figure 2.4: Gantt of the optimal schedule for the problem in Case I

2.4.2 Case II: Single Tool JSSP with SDST & OC

The input jobs and setup times of case II are the same as ones of case I. Additionally, all operations on the same machine share the same tool, namely, $p_k = 1, \forall k \in M$, and $C_{1,k} = 10, D_{1,k} = 2, \forall k \in M$. The obtained optimal result is 26, and the schedule is shown in Figure 2.5 (jobs are colored by Job ID). Since some of the

setup times are larger than the required cool down time, machines are not always idled to allow for the tool to cool down. But, as the Gantt chart shows, there are periods in which machines are forced to idle to wait for the tool to cool down. For example, the gap between job 1 (J1) and job 3 (J4) on machine 3, and the gap between job 1 (J1) and job 2 (J2) on machine 2.

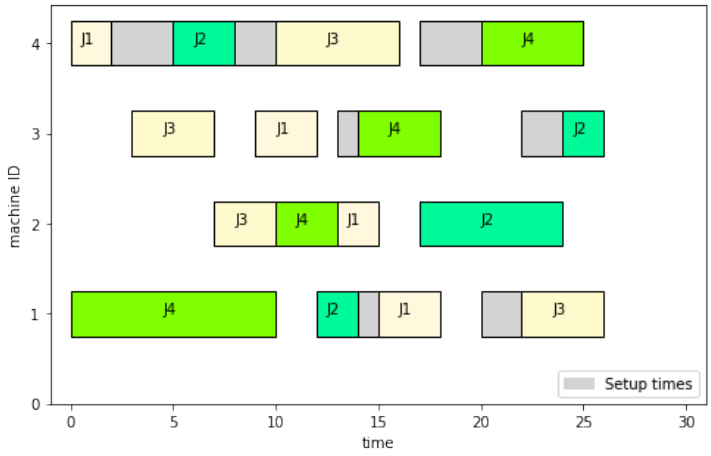


Figure 2.5: Gantt of the optimal schedule for the problem in Case II

2.4.3 Case III: JSSP with SDST & OC

In case III, inputs and setup times are the same as ones in case I, $p_k = 2$ and $C_{1,k} = C_{2,k} = 10, D_{1,k} = D_{2,k} = 2, \forall k \in M$. The tool requirement of each operation is shown in Table 2.4. The resulting optimal schedule has a makespan of 24, which is the same as the result of case I. However, the optimal schedule is different. It does not overheat the tool and it allows the overheated ones to cool down. The schedule is shown in Figure 2.6. Note that jobs are colored by the required tool. By comparing the Gantt chart of case I and case III, we can see the schedule utilizes the additional setup time to cool down the tool. For instance, on machine 4, the sequence (J1,J2,J3,J4) of case III requires 8 units of setup times, and the sequence (J2,J1,J3,J4) of case I requires 6 units of time. The additional setup time is utilized

to cool down the tool.

Table 2.4: Description of Input Jobs

Machine ID	Tool 1	Tool 2
1	$OT_{1,1} = \{O_{4,3}, O_{1,4}\}$	$OT_{2,1} = \{O_{4,1}, O_{2,2}\}$
2	$OT_{1,2} = \{O_{2,3}, O_{3,1}\}$	$OT_{2,2} = \{O_{2,4}, O_{3,2}\}$
3	$OT_{1,3} = \{O_{4,2}, O_{1,3}\}$	$OT_{2,3} = \{O_{2,1}, O_{3,4}\}$
4	$OT_{1,4} = \{O_{1,1}\}$	$OT_{2,4} = \{O_{1,2}, O_{4,4}, O_{3,3}\}$

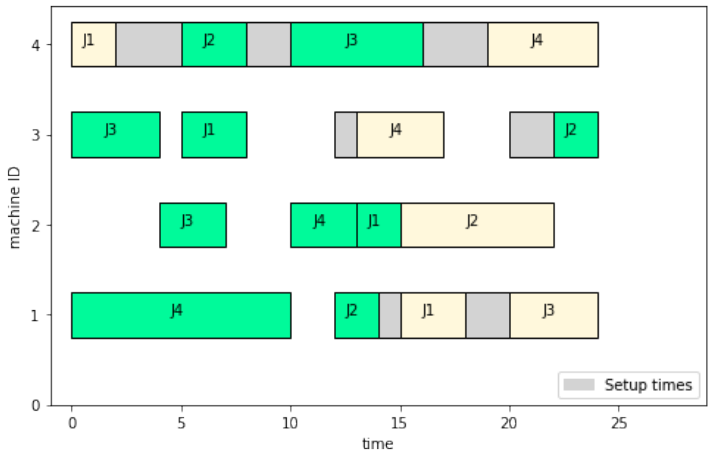


Figure 2.6: Gantt of the optimal schedule for the problem in Case III

2.5 Conclusions

During a continuous high speed machining, cooling is necessary. Nevertheless, a liquid coolant may introduce a thermal stress on the tool by intensifying temperature gradients of the surrounding region of the cut [67]. A significant thermal stress can bring a thermal shock to the tool and accelerate a crack formation and propagation. Because tools are costly, avoiding tool damages using dry machining is preferred in certain scenarios. During a dry machining process, however, scheduling jobs that require the same tool together becomes alarming as it carries the risk of overheating tools. As no study considers such an issue, a MIP model is proposed to minimize

makespan and avert overheating tools. Three case studies are conducted to validate the model.

The computational time of solving the MIP model increases exponentially as the problem size grows, a meta-heuristics should be used to solve large problems. Regarding the further work, we plan to optimize the process rate which interacts with the tool capacity. For instance, a lower rotation speed will heat up the tool temperature slower than a faster rotation speed, therefore it will increase the tool capacity. Furthermore, when the data of tool capacity is available, chance constraints can be added to the model to obtain a robust schedule.

Chapter3

Service Rate Control in a Finite Capacity Single-server Queue with an Unreliable Server and Unknown Breakdown Rate

3.1 Introduction

The unplanned downtime commonly observed in manufacturing systems is due to reasons such as tool failures, machine malfunctions, and operator errors. The downtime cost varies across different manufacturing industries. In 2014, Aberdeen [4] estimated the cost of unplanned downtime across all businesses to be $163,000 an hour, and this number skyrocketed to $260,000 in 2016. A survey of 101 executives in the automotive industry [5] indicates that the downtime cost in the U.S. automotive industry is $22,000 per minute, which is equivalent to $1.3 millions per hour. Hence, any effort in reducing unplanned downtime can create considerable savings for the business. For example, progressive stamping processes

are widely used in the automotive industry and are economically justified only at higher service rates [68]. However, an increase in the press speed significantly improves the punch velocity and it creates additional vibrations and introduces thermal growth, which results in a higher probability of unplanned downtime [69]. In queuing systems, the unplanned machine downtime is commonly modeled using queues with interruptions and it has received considerable attention since the late 1950s [70–74]. In this chapter, we focus on a particular dependent structure where the breakdown rate of the server is a linear function of the service rate. Since the relationship between these two rates might be unknown in practice, we develop online algorithms to obtain the optimal policy in the long-run.

We apply a self-tuning approach to the control problem when system has unknown parameters. The self-tuning scheme is introduced in [75]. Mandl [75] provides several models for controlled Markov processes with unknown parameters. The self-tuning approach is identified as a procedure that the controlled policy is continuously modified based on the estimation of unknown parameters to approach the optimal policy for the problem with true parameters. In this chapter, we estimate the unknown parameters in the relation between the breakdown and service rates based on the historical data at each jump time of the Markov process, and then the control implemented is characterized by an optimality equation with the current parameter estimate. The optimality equation used is the same as the equation used in the aforementioned optimal control problem when the true parameters are known. Since the linear relation between the service and breakdown rates leads to a nonlinear relation between the mean service times and mean 'up' times of the system, the quasi-maximum likelihood estimates are used for the estimation of unknown parameters. This method has been studied in [76, 77] and references therein.

The next section presents a detailed review of related studies. The model of

the service rate control problem is described in the third section. The is a finite capacity single-server queue with an unreliable server. In the fourth section, the necessary and sufficient conditions of stationarity are shown for the problem. The optimality of the obtained policy is proven in Section five. In Section six, two learning lgorithms are proposed. The first one is to obtain the control policy when the down rate parameters are unknown, and the second one is to learn a control policy with mild system dynamics assumptions. Finally, numerical studies are presented to validate the algorithms in Section seven, and the study is concluded in Section eight.

3.2 Related Work

In the context of queuing systems, unplanned machine shutdowns are commonly modeled using queues with interruptions. In general, queuing models with servers that are not available continuously can be classified into the following four types [78]:

1. Queuing models with vacations: In this type of model, the server takes a vacation when there is no job in the system. The server resumes services only when the length of the queue reaches a predefined threshold.

2. Queuing models with service interruptions: In this type of model, the server can either be in the on-state or the off-state. When the server is in the on-state, the system works as the classic queuing system (without interruptions); however, the server can become unavailable because of a random event and therefore goes into the off-state. These events can occur either during a service, or when the server is idle. After the event is completed, the server returns to the on-state. When a busy server enters the off-state, the job that was being processed is commonly modeled to return to the queue [78].

3. Queuing models with customer interrupted services: In a system with customer interrupted services, customers may leave the system during a service due to a random event.

4. Queuing models with catastrophic events: In a this type of model, a random external event may occur that empties the system. After the repair is complete, the server resumes the service when there is a new arrival.

Because the occurrence of unplanned downtime prevents the machine from working, and maintenance takes time to bring the machine back online, in this work, we formulate the problem using queuing models with service interruptions.

3.2.1 Queuing Models with Service Interruptions

Queuing models with service interruptions have been studied since the late 1950s by pioneers White and Christie [70], Gaver [71], Avi-Itzhak and Naor [72]. Queuing systems with a single-server and Poisson arrival process are represented as M/M/1, M/E_k/1, M/PH/1 and M/G/1 if the service times have an exponential distribution, Erlang distribution, phase type distribution and general distribution, respectively.

Analysis of single-server Queues

White and Christie [70] have shown that a single-server queue with service breakdown is equivalent to a single-server queue with preemptive priority arrivals. Gaver [71] investigates a single-server compound Poisson queue with server breakdowns and priorities. Five single-server Poisson queues with server breakdowns with various settings are studied by Avi-Itzhak and Naor [72]. Variations in models are at conditions of interruptions, and repairs can occur. The performance measures, such as the expected queue length and waiting time are derived in all three works above.

Kella and Whitt [79], as well as Chen and Whitt [80] analyze heavy-traffic stochastic process limits for single-server systems with service interruptions. Their work investigates the limits for queues in which a sequence of stochastic processes converges to another stochastic process. Because the converging processes are obtained by appropriately scaling the time and space of the initial process, the limits provide a macroscopic view of uncertainty.

Gray et al. [81] consider a general single-server queuing model. Server failures can occur for many types of reasons. Therefore, the breakdown requires a finite random number of stages to be repaired. The authors assume that the breakdown occurs when the server is busy, and it has no impact on the arrival process. The necessary and sufficient condition for the stationary queue length distribution to exist is obtained, and the authors derive the expected queue length distribution using matrix-geometric methods.

Analysis of single-server Queues with batch arrivals

Time-dependent M/M/1 queuing models with fixed batch sizes and server interruptions are studied in Madan [82, 83]. The performances in the steady state are derived. By characterizing the repairs into two phases, a queue model with batch arrival and departure is analyzed by Madan in [84].

Altiok [85] investigates a M/G/1 queue with batch arrivals and service interruptions, wherein the distribution of service times is a mixture of generalized Erlangs. Service failures are generated by a Poisson process, and the corresponding repair times follow a mixture distribution of generalized Erlangs. The interruptions occur during service, and all customers are removed from the system once a breakdown occurs, but new customers can arrive when the server is under repair.

Tadj and Choudhary [86] analyze an M/G/1 queue with a fixed size batch arrival and service interruptions. Server failures occur when the system is busy and

the repair times follow an arbitrary distribution. In addition, the breakdowns have no effect on the arrival process. The stability condition and steady-state system size are derived.

A single-server queue with set-up time and server breakdown is studied by Chang and Wang [87]. The server is turned off when the system is empty, and failure may happen when a newly arrived customer reactivates the off server. When the server is successfully turned on, it requires a set-up time. When the server fails to be reactivated, it is repaired immediately, and the repair time is exponentially distributed. This study considers a model with imperfect repairs and a model with perfect repairs. In the first model, the server might fail to reactivate after a repair, and in the second model, the server will always be successfully turned back on after a repair. An explicit expression for the stationary distribution of the queue length is obtained.

Analysis of single-server Queues with bulk service

Jayaraman [88] studies a single-server queuing system with a Poisson process and general bulk service. The lengths of the operating and repair periods follow an exponential and phase-type distribution, respectively. When a server failure occurs, customers arrive at a lower rate compared to when the server is operational. The system is not cleared when a failure occurs, and the job that was being severed returns to the queue, and its service starts over when the repair period ends. The stability condition and expected length of the queue in the steady state are obtained using a matrix-geometric algorithmic approach.

3.2.2 Optimal Control in single-server Queues

The Markov decision process [22] is a basic method in modern dynamic control theory [89], and it has been applied in many practical areas such as inventory control, supply chain management, transportation networks and communication networks to name a few. Moreover, MPDs in queues play a critical role in the dynamic control of stochastic systems such as manufacturing systems [90], production lines [91], and energy-efficient management [92]. A detailed survey is conducted by Li et al. [89].

In the literature on optimal control in single-server queues, some commonly studied control targets are

1. Admission rate
2. Rate of the arrival process
3. Type of the arrival process
4. Service rate
5. Type of the service process
6. Threshold of queue length to resume services in systems with server vacations
7. Priority of queues

In this work, we focus on service rate control in single-server queues.

One of the earliest work on service rate control in a single-server queue is to adjust the service rate based on the queue length of the system [93], in which the service rate is chosen from a finite set. Another pioneering work controls the service rate in a closed set in which the holding cost is a convex function [94]. As in many later works [24, 95, 96], the objective function involves two cost components. The first component is the service rate cost, which is a non-decreasing function of the

service rate. The second component is the holding cost, which is a non-decreasing function of the queue length.

In the majority of the work, the arrival is a Poisson process and the service time follows an exponential distribution. However, a single-server queue with a non-stationary arrival is studied in which the arrival is Markov-modulated [96].

The objective of this body of literature is to develop control policies to balance the cost of effort and holding cost. Moreover, they all show that the optimal service rate is non-decreasing as a function of the queue length. There is also a stream of work which considers switching cost for changing the service rate [97–99] resulting in hysteric policies.

Another body of work jointly controls the service rate and admission, in which a new customer can be rejected [100–102]. Most of the work still considers the two costs associated with service rate and queue length but the uniformization is not applicable because the transition rates are generally unbounded [102].

Although the optimal control with an unreliable server has been studied [103–105], the control target is the vacation threshold, namely, the queue length in which the server resumes services.

3.2.3 Learning and Control in Queuing Systems

Maximizing System Payoff in Multi-Server Queues

A study [106] integrates learning and control in a multi-server queue to maximize the total system payoff when the type of new customer and its payoff information is unknown. In the model, each arrival has multiple tasks, and the control target is the probability of assigning a client's tasks to servers. The paper proposes an algorithm to iteratively estimate the payoff using a truncated Upper-Confidence-Bound (UCB) and solve the objective function based on the estimated payoff vectors. In the

estimation part, the the client's payoff vector is estimated based on the payoff feedback.

Another related work [107] characterizes the structure of the optimal policy in the limit in which each server performs many jobs. In their model, the arrivals have unknown payoff vectors, and the objective is to maximize the total system payoff. An algorithm is proposed to solve the problem by balancing the exploration and exploitation.

In both of the above problems, the unknown is in the arrival process, and the policy obtained based on the estimated parameters does not have an impact on the future observations. In our model, however, the breakdown of the server is affected by its service rate; hence, the control policy has an impact on the future observations.

Server Allocation and Routing in Queuing Networks

A recent study [108] propose a model-based reinforcement learning algorithm to determine the network control policy from observed data from systems without the information of underlying system dynamics. Because the state space is unbounded, their policy is determined by the RL algorithm when the state is below a threshold, and when the state is above the threshold, a simple baseline algorithm is applied. The gap between the resulting policy and the optimal policy is shown to go to zero when the threshold goes to infinity.

In the above study, the model-based RL algorithm utilizes the underlying system mechanics to obtain the optimal control. In this chapter, we apply a model-free RL method, which learns control policies via interacting with the system, the numerical results indicate that a near-optimal policy can be obtained in our problem setting.

3.2.4 Contributions

In this chapter, we study finite capacity single-server queues when the server is subject to breakdown, and the rate of breakdown is unknown. We first show the necessary and sufficient conditions of the stationarity, and then we show the optimality of the obtained policy. By applying an inference and learning scheme, we propose algorithms that estimate the parameters and then solve the control problem. We also apply a model-free reinforcement learning method to determine the control policy without imposing any assumptions on the system dynamics. Numerical studies are conducted to validate these algorithms. Our main contributions are summarized as follows.

1. We show the necessary and sufficient conditions of stationarity for the finite capacity single-server queues with an unreliable server when the arrival process is a Poisson process. To the best of our knowledge, there are no studies on the service rate control in unreliable single-server queues.

2. We propose inference and learning algorithms to solve the service rate control problems when the breakdown parameters are unknown. This algorithm can be helpful in practice because the machine behavior can be shifting as it ages, and our algorithm can adapt to the change and provide an optimal control policy.

3. We apply a model-free RL algorithm to show that a near-optimal policy can be obtained from a pure data-driven method. This can be useful in the current Industrial 4.0 era as data can be collected and stored cost-efficiently.

3.3 Model

In this section, an unreliable single-server queue with a Poisson arrival and exponential service time with a finite capacity is modeled. The arrival process has a rate $\lambda > 0$, and the system has a capacity of N, that is, a job will be rejected if it arrives when the system has N jobs. The number of jobs in the system is denoted as $x \in [0, N]$.

The unreliable server has an up state and a down state denoted by $k \in \{0, 1\}$. When the server is in the up state, $k = 1$, and $x > 0$, the server processes jobs with a service rate $\mu \in [\underline{\mu}, \bar{\mu}]$, where $\underline{\mu}$ and $\bar{\mu}$ are the minimum and maximum service rates, respectively. Additionally, $\underline{\mu} \geq \lambda$ and $\bar{\mu} < \infty$ are assumed. Since turning the server up has a relatively large energy cost in practice, the server is assumed to operate at the minimum rate $\underline{\mu}$ if x=0.

When the server is in the down state, $k = 0$, it cannot process any jobs as $\mu = 0$, but new jobs can join the system when $x < N$. A unit time maintenance cost C_m will occur when the server goes to the down state, and the time required for the server to come return to the up state follows an exponential distribution with a positive parameter β_u. Moreover, we assume that the time that it takes the server to the down state from the up state follows an exponential distribution with a parameter $\beta_d(\mu)$, and

$$\beta_d(\mu) = a_u \mu + b_u, \tag{3.1}$$

where a_u and b_u are positive constants. This indicates that if the server operates at a high rate, then it has a high rate of going to the down state. A linear relationship between the breakdown rate and service rate can be observed in many manufacturing settings, such as stamping, molding and machining. For instance, Figure 3.1 shows the service rate and breakdown rate of molding machines in a U.S.

factory. From the plot we can see that as the service rate increases, the overall on-time decreases, which implies that the breakdown rate increases. The fitted linear regression line is plotted as the blue line, and both of the coefficient and intercept are statistically significant with p-values less than 0.001.

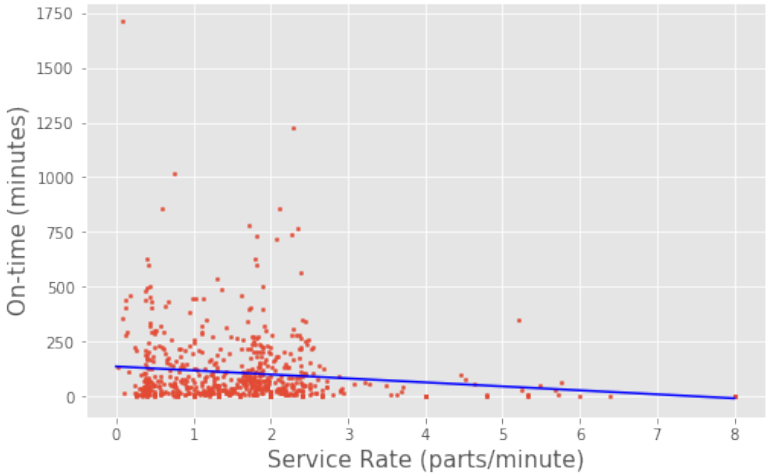


Figure 3.1: Service Rate and Machine On-Time

Therefore, the state space is defined as $\mathcal{S} \equiv [0, N] \times \{0, 1\}$ and the action space is defined as $\mathcal{A} \equiv \{0\} \cup [\underline{\mu}, \bar{\mu}]$.

A control policy $\pi : \mathcal{S} \to \mathcal{A}$ is admissible if it satisfies $\pi(x, 0) \equiv 0$, $\pi(x, 1) \in [\underline{\mu}, \bar{\mu}]$ for $\forall x \in [0, N]$ and $\pi(0, 1) \equiv \underline{\mu}$. We denote Π as the set of admissible control policies. Hence, under any $\pi \in \Pi$, we have

$$\mu = \pi(x, k), \forall (x, k) \in \mathcal{S}. \tag{3.2}$$

Let $X(t)$ and $K(t)$ denote the number of jobs in the system and server up-down state k at time $t \geq 0$, then $\{(X(t), K(t)) : t \geq 0\}$ forms a continuous-time Markov chain (CTMC) under an admissible policy $\pi \in \Pi$. The transition diagram is shown in Figure 3.1.

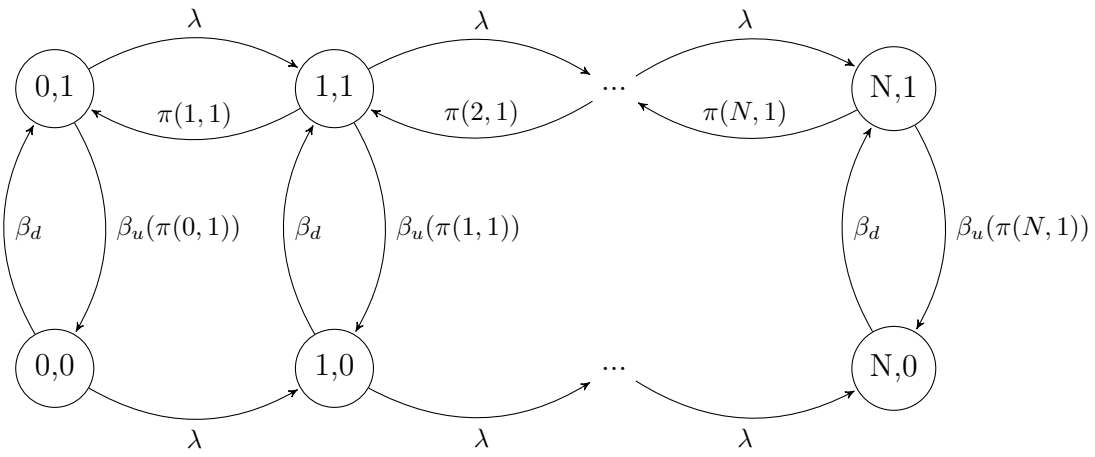


Figure 3.2: The State Transition Diagram

The steady-state distribution for a state is defined as

$$u(x,k) = \lim_{t\to\infty} Pr\{(X(t) = x, K(t) = k)\}. \tag{3.3}$$

The Markov chain is ergodic if the steady-state distribution exists. We say that a policy π is stable if the steady-state distribution under π exists, and we denote the set of stable admissible policies as Π_s. We denote u^π as a steady-state distribution under $\pi \in \Pi_s$

The existence of a stable policy for the Markov chain $\{(X(t), K(t)) : t \geq 0\}$ is shown in Section 3.3.

In this problem, we consider a convex unit time running cost $R(\mu)$ and a convex unit time holding cost $H(x)$. When the server is down, a constant unit cost $C_m > 0$ will be considered. In addition, if a new job arrives while the system is at its full capacity, a rejection cost $p > 0$ is considered.

In this chapter, we consider the finite capacity ergodic control problem. Hence, the cost function for the model with a Poisson arrival is defined as

$$J^\pi := \sum_{x=0}^{N} \Big[u^\pi(x,1)(H(x)+R(\pi(x,k)))+u^\pi(x,0)(H(x)+C_m)\Big] + \lambda p(u^\pi(N,1)+u^\pi(N,0)). \tag{3.4}$$

The optimal cost is defined by

$$J^* := \inf_{\pi \in \Pi_s} J^\pi. \tag{3.5}$$

3.4 Stable Admissible Policy

The sufficient conditions that ensure the existence of stationary distribution of the CTMC $(X(t), K(t))$ are (see, e.g. Theorem 4.1 in [109]):

1. All states communicate

$$Pr\{(X(t+s) = n, K(t+s) = k) | (X(s) = n', K(s) = k')\} > 0 \tag{3.6}$$

   $\forall n, n' \in [0, N]$, $\forall k, k' \in \{0, 1\}$ and $t, s \geq 0$;

2. The Markov chain is positive recurrent, namely, each state has a finite mean return time

Since $\pi(x, 1) \in [\underline{\mu}, \bar{\mu}] > 0 \ \forall x \in [0, N]$ under any admissible control policy, the CTMC $(X(t), K(t))$ is irreducible; hence, the condition 1 is satisfied.

Proposition 3.4.1. *The CTMC $(X(t), K(t))$ is positive recurrent under admissible policy π if and only if*

$$\frac{\lambda}{\pi(x,1)} \frac{\beta_u(\pi(x,1)) + \beta_d}{\beta_d} < 1, x = 0, 1, 2, 3, ..., N.$$

Proof. The generator matrix Q of the CTMC under an admissible policy $\pi \in \Pi$ is

$$Q = \begin{bmatrix} -(\beta_d+\lambda) & \beta_d & \lambda & 0 & ... & 0 & 0 \\ \beta_u(\mu_0) & -(\beta_u(\mu_0)+\lambda) & 0 & \lambda & ... & 0 & 0 \\ 0 & 0 & -(\beta_d+\lambda) & \beta_d & ... & 0 & 0 \\ 0 & \mu_1 & \beta_u(\mu_1) & -(\beta_u(\mu_1)+\mu_1+\lambda) & ... & 0 & 0 \\ & & \ddots & & \ddots & & \ddots \\ 0 & 0 & 0 & 0 & ... & -\beta_d & \beta_d \\ 0 & 0 & 0 & 0 & ... & \beta_u(\mu_N) & -(\beta_u(\mu_N)+\mu_N) \end{bmatrix},$$

where $\mu_i = \pi(i, 1)$, for $i = 0, 1, 2, ..., N$.

Using the uniformization technique, a given CTMC can be represented as a Discrete-time Markov Chain (DTMC). The probability of the DTMC corresponding to the CTMC is defined as

$$P = I + \frac{Q}{q}, \tag{3.7}$$

where $q \geq \max_i(|Q_{i,i})|)$. Since under any admissible policy π, $\max \mu_i = \bar{\mu}$, q is defined as

$$q = \beta_u(\bar{\mu}) + \beta_d + \lambda + \bar{\mu}, \tag{3.8}$$

and we have the transition probability matrix for the CTMC as

$$P = \begin{bmatrix} \frac{\beta_u(\bar{\mu})+\bar{\mu}}{q} & \frac{\beta_d}{q} & \frac{\lambda}{q} & 0 & ... \\ \frac{\beta_u(\mu_0)}{q} & \frac{q-\beta_u(\mu_0)-\lambda}{q} & 0 & \frac{\lambda}{q} & ... \\ 0 & 0 & \frac{\beta_u(\bar{\mu})+\bar{\mu}}{q} & \frac{\beta_d}{q} & ... \\ 0 & \frac{\mu_1}{q} & \frac{\beta_u(\mu_1)}{q} & \frac{q-\beta_u(\mu_1)-\mu_1-\lambda}{q} & ... \\ & & \ddots & & \ddots \end{bmatrix}.$$

Since P has a tridiagonal block structure, it can be written as

$$P = \begin{bmatrix} L_0 & F & 0 & 0 & 0 & ... & 0 & 0 \\ B_1 & L_1 & F & 0 & 0 & ... & 0 & 0 \\ 0 & B_2 & L_2 & F & 0 & ... & 0 & 0 \\ 0 & 0 & B_3 & L_3 & F & ... & 0 & 0 \\ & & \ddots & & & \ddots & & \\ 0 & 0 & 0 & 0 & 0 & ... & L_{N-1} & F \\ 0 & 0 & 0 & 0 & 0 & ... & B_N & L_N \end{bmatrix},$$

where

$$F = \begin{bmatrix} \frac{\lambda}{q} & 0 \\ 0 & \frac{\lambda}{q} \end{bmatrix}, \qquad L_0 = \begin{bmatrix} \frac{\beta_u(\bar{\mu})+\bar{\mu}}{q} & \frac{\beta_d}{q} \\ \frac{\beta_u(\mu_0)}{q} & \frac{q-\beta_u(\mu_0)-\lambda}{q} \end{bmatrix},$$

$$B_i = \begin{bmatrix} 0 & 0 \\ 0 & \frac{\mu_i}{q} \end{bmatrix}, \quad \text{and} \quad L_i = \begin{bmatrix} \frac{\beta_u(\bar{\mu})+\bar{\mu}}{q} & \frac{\beta_d}{q} \\ \frac{\beta_u(\mu_i)}{q} & \frac{q-\beta_u(\mu_i)-\mu_i-\lambda}{q} \end{bmatrix},$$

for $i = 1, 2, 3, ..., N-1$. And

$$B_N = \begin{bmatrix} 0 & 0 \\ 0 & \frac{\mu_N}{q} \end{bmatrix}, \quad \text{and} \quad L_N = \begin{bmatrix} \frac{\lambda+\beta_u(\bar{\mu})+\bar{\mu}}{q} & \frac{\beta_d}{q} \\ \frac{\beta_u(\mu_N)}{q} & \frac{q-\beta_u(\mu_N)-\mu_N}{q} \end{bmatrix},$$

The DTMC therefore can be viewed as a Quasi-Birth-and-Death Process (QBD).

Define

$$L = F + B_1 + L_1 = \begin{bmatrix} \frac{q-\beta_d}{q} & \frac{\beta_d}{q} \\ \frac{\beta_u(\mu_1)}{q} & \frac{q-\beta_u(\mu_1)}{q} \end{bmatrix} \tag{3.9}$$

and $\theta \in \mathbb{R}^2$.

By solving

$$\theta^T L = \theta^T, e^T\theta = 1, \tag{3.10}$$

we obtain

$$\theta = \begin{bmatrix} \frac{\beta_u(\mu_1)}{\beta_u(\mu_1)+\beta_d} \\ \frac{\beta_d}{\beta_u(\mu_1)+\beta_d} \end{bmatrix}. \tag{3.11}$$

Theorem 3.2.1 [110] indicates that the QBD is ergodic if and only if $\theta^T F e < \theta^T B_i e$ for θ satisfies (3.10). Using(3.11), we obtain a condition for the rates such that the MDP has a stationary distribution,

$$\frac{\lambda}{\mu_i}\frac{\beta_u(\mu_i)+\beta_d}{\beta_d} < 1, i = 0, 1, 2, 3, ..., N.$$

This shows that the embedded DTMC is ergodic, which guarantees the original CTMC is positive recurrent. Therefore, for any π, we have

$$\frac{\lambda}{\pi(x,1)}\frac{\beta_u(\pi(x,1))+\beta_d}{\beta_d} < 1, x = 0, 1, 2, 3, ..., N, \tag{3.12}$$

and the stationary distribution exists, we denote the set of such policy as Π_{sm} $\square$

Under any policy $\pi \in \Pi_{sm}$. We have shown that the stationary distribution exists, and the balance equations take the form

$$\begin{aligned}
(\lambda + \beta_u(\mu_0))u^\pi(0,1) &= \beta_d u^\pi(0,0) + \mu_1 u^\pi(1,1) \\
(\lambda + \beta_d)u^\pi(0,0) &= \beta_u(\mu_0)u^\pi(0,1) \\
(\lambda + \beta_u(\mu_i) + \mu_i)u^\pi(i,1) &= \lambda u^\pi(i-1,1) + \beta_d u^\pi(i,0) + \mu_{i+1}u^\pi(i+1,1), \quad \forall i \in [1, N-1] \\
(\lambda + \beta_d)u^\pi(i,0) &= \lambda u^\pi(i-1,0) + \beta_u(\mu_i)u^\pi(i,1), \quad \forall i \in [1, N-1] \\
(\beta_u(\mu_N) + \mu_N)u^\pi(N,1) &= \lambda u^\pi(N-1,1) + \beta_d u^\pi(N,0) \\
\beta_d u^\pi(N,0) &= \lambda u^\pi(N-1,0) + \beta_u(\mu_N)u^\pi(N,1)
\end{aligned}$$

It is evident that

$$\mu_i u^\pi(i,1) = \lambda(u^\pi(i-1,0) + u^\pi(i-1,1)) \quad \forall i \in [1,N]. \tag{3.13}$$

By substituting (3.13) into the balance equations, we obtain

$$\lambda u^\pi(i-1,0) = (\lambda+\beta_d)u^\pi(i,0) - \beta_u(\mu_i)u^\pi(i,1) \qquad \forall i \in [1,N], \tag{3.14}$$

$$\lambda u^\pi(i-1,1) = -(\lambda+\beta_d)u^\pi(i,0) + (\beta_u(\mu_i)+\mu_i)u^\pi(i,1) \quad \forall i \in [1,N]. \tag{3.15}$$

3.5 Optimality

For an admissible stable policy $\pi \in \Pi_{sm}$, the ergodic cost is given by (3.3) and the optimal ergodic cost is defined in (3.3).

The differential value function of each state under an admissible stable policy $\pi \in \Pi_{sm}$ is given by

$$v(0,0) = \frac{1}{q}[C_m - J^* + \beta_d v(0,1) + \lambda v(1,0) + (q-\beta_d-\lambda)v(0,0)] \tag{3.16}$$

$$v(0,1) = \frac{1}{q}[R(\underline{\mu}) - J^* + \beta_u(\underline{\mu})v(0,0) + \lambda v(1,1) + (q-\beta_u(\underline{\mu})-\lambda)v(0,1)] \tag{3.17}$$

$$v(x,0) = \frac{1}{q}[H(x) + C_m - J^* + \beta_d v(x,1) + \lambda v(x+1,0) + (q-\beta_d-\lambda)v(x,0)] \tag{3.18}$$

$$\begin{aligned} v(x,1) = \min_\mu \frac{1}{q}[&H(x) + R(\mu) - J^* + \beta_u(\mu)v(x,0) + \lambda v(x+1,1) + \mu v(x-1,1) \\ &+ (q-\beta_u(\mu)-\mu-\lambda)v(v,1)] \end{aligned} \tag{3.19}$$

$$v(N,0) = \frac{1}{q}[H(N) + C_m - J^* + \lambda p + \beta_d v(N,1) + (q-\beta_d)v(N,0)] \tag{3.20}$$

$$\begin{aligned} v(N,1) = \min_\mu \frac{1}{q}[&R(\mu) + H(N) - J^* + \lambda p + \beta_u(\mu)v(N,0) + \mu v(N-1,1) \\ &+ (q-\beta_u(\mu)-\mu)v(N,1)] \end{aligned} \tag{3.21}$$

As in George and Harrison [95], to simplify the optimality equation, we can

define the following functions,

$$W(x) = v(x,1) - v(x-1,1) \qquad \forall x \in [1,N] \tag{3.22}$$

$$Y(x) = v(x,0) - v(x,1) \qquad \forall x \in [0,N] \tag{3.23}$$

$$\phi(w,y) = \max_{\mu}\{\mu w - R(\mu) - \beta_u(\mu)y\} \tag{3.24}$$

The value functions can then be expressed as

$$\lambda W(1) = -\beta_u(\underline{\mu})Y(0) - H(0) - R(\underline{\mu}) + J \tag{3.25}$$

$$\lambda W(1) + \lambda Y(1) = (\beta_d + \lambda)Y(0) - H(0) - C_m + J \tag{3.26}$$

$$\lambda W(x+1) = \phi(W(x), Y(x)) - H(x) + J \tag{3.27}$$

$$\lambda W(x+1) + \lambda Y(x+1) = (\beta_d + \lambda)Y(x) - C_m - H(x) + J \tag{3.28}$$

$$\phi(W(N), Y(N)) = H(N) - J + \lambda p \tag{3.29}$$

$$\beta_d Y(N) = H(N) + C_m - J + \lambda p \tag{3.30}$$

where $x = 1, 2, 3, ..., N-1$, and J is estimated minimum average cost.

We now provide a theorem similar to Proposition 1 in [111], which allows us to rigorously prove the optimality of a policy derived from a solution of (3.25) to (3.30).

Theorem 3.5.1. *Let $J < \infty$ and $(W(1), W(2), .., W(N), Y(0), Y(1), ..., Y(N))$ be a solution to (3.25) to (3.30). If $Y(x) \geq 0$ $\forall x \in [0,N]$, and $\pi^*(x,1) = \arg\max_{\mu}\{\mu W(x) - R(\mu) - \beta_u(\mu)Y(x)\} \in [\underline{\mu}, \bar{\mu}]$ $\forall x \in [1,N]$, then π^* is optimal and $J^{\pi^*} = J = J^*$.*

Proof. Let π be a feasible rate control policy of (3.3), and let $\mu_x = \pi(x,1)$ for $x = 1, 2, 3, ..., N$.

By definition of $\phi(.)$, one writes

$$\phi(W(x), Y(x)) \geq \mu_x W(x) - R(\mu_x) - \beta_u(\mu_x)Y(x) \quad \forall x \in [1, N-1].$$

Then, by (3.27), it follows that

$$\lambda W(x+1) + H(x) - J \geq \mu_x W(x) - R(\mu_x) - \beta_u(\mu_x)Y(x) \quad \forall x \in [1, N-1].$$

Multiplying both sides of the equation above by $u^\pi(x, 1)$, we obtain

$$(H(x)+R(\mu_x)-J)u^\pi(x,1) \geq (\mu_x W(x)-\lambda W(x+1)-\beta_u(\mu_x)Y(x))u^\pi(x,1). \quad (3.31)$$

Multiplying both sides of (3.28) by $u^\pi(x, 0)$, we have

$$(C_m+H(x)-J)u^\pi(x,0) = [(\beta_d+\lambda)Y(x)-\lambda(W(x+1)+Y(x+1))]u^\pi(x,0). \quad (3.32)$$

By multiplying $Y(x)$ to both sides of (3.14), it follows

$$\lambda u^\pi(x-1,0)Y(x) = [(\lambda+\beta_d)u^\pi(x,0) - \beta_u(\mu_x)u^\pi(x,1)]Y(x), \quad (3.33)$$

and adding $\lambda\mu_x u^\pi(x, 1)W(x)$ to both sides of (3.33), we obtain

$$\begin{aligned} &\lambda u^\pi(x-1,0)Y(x) + \lambda(u^\pi(x-1,1) + u^\pi(x-1,0))W(x) \\ &= [(\lambda+\beta_d)u^\pi(x,0) - \beta_u(\mu_x)u^\pi(x,1)]Y(x) + \lambda\mu_x u^\pi(x,1)W(x). \end{aligned} \quad (3.34)$$

By adding (3.31) and (3.32) and applying (3.34), it is evident that for $x \in [1, N-1]$ we have

$$H(x)(u^\pi(x,1) + u^\pi(x,0)) + R(\mu_x)u^\pi(x,1) + C_m u^\pi(x,0) - J(u^\pi(x,1) + u^\pi(x,0))$$

$$\geq \lambda W(x)(u^\pi(x-1,1)+u^\pi(x-1,0))+\lambda Y(x)u^\pi(x-1,0) \tag{3.35}$$

$$-\lambda[W(x+1)(u^\pi(x,1)+u^\pi(x,0))+Y(x+1)u^\pi(x,0)].$$

Similarly, we can show that

$$H(N)(u^\pi(N,1)+u^\pi(N,0))+R(\mu_N)u^\pi(N,1)+C_m u^\pi(N,0)$$

$$-J(u^\pi(N,1)+u^\pi(N,0))+\lambda p(u^\pi(N,1)+u^\pi(N,0)) \tag{3.36}$$

$$\geq \lambda W(N)(u^\pi(N-1,1)+u^\pi(N-1,0))+\lambda Y(N)u^\pi(N-1,0).$$

Summing over $x=1,2,3,...,N-1$ of (3.35) and adding to (3.36) gives the following:

$$\sum_{x=1}^{N}[(H(x)+R(\mu_x))u^\pi(x,1)+(H(x)+C_m)u^\pi(x,0)]-J\sum_{x=1}^{N}(u(x,1)+x(x,0))$$

$$\geq [(\beta_d+\lambda)Y(0)-C_m+J]u^\pi(0,0)+(J-\beta_u(\mu_0)Y(0)-R(\mu_0)+J)u^\pi(0,1). \tag{3.37}$$

Then, by (3.3) and the balance equations, it follows that,

$$J^\pi - J \geq (\beta_d+\lambda)u(0,0)-\beta_u(\mu_0)u(0,1)=0. \tag{3.38}$$

Therefore,

$$J^\pi \geq J. \tag{3.39}$$

As $\pi \in \Pi_{sm}$, all inequalities in the preceding can be replaced with equalities. Therefore, it follows that

$$J^{\pi^*}=J=J^*. \tag{3.40}$$

Hence, the optimal policy can be obtained by solving the system (3.25) to (3.30). □

3.6 Numerical Study: Optimal Controls

In this section, we present the numerical results and optimal service rate controls for the queueing systems discussed in Sections 3.3.1 and 3.3.2. To ensure that the numerical study is representative of real-world scenarios, we provide the results for the optimal controls under different system dynamics and cost settings. We assume $\beta_u(\mu) \leq \beta_d$ for $\underline{\mu} \leq \mu \leq \bar{\mu}$ where $\beta_u(\mu) = a_u\mu + b_u$ and β_d, a_u, b_u are positive constants. This implies that the machine spends more time in the up-state compared to the down-state on the average, which is a commonly observed real-world phenomenon.

We first show the optimal controls when the system has linear costs, and then we show the optimal policies when the system is under the quadratic cost setting.

In the linear costs setting, the holding cost function $H(x)$ satisfies

$$H(x) = C_h x$$

where x is the number of jobs in the system and C_h is a positive constant. We set the effort cost function $R(\mu)$ to

$$R(\mu) = C_r \mu$$

where C_r is a positive constant, and $\mu \in [\underline{\mu}, \bar{\mu}]$ with $\underline{\mu}, \bar{\mu} > 0$.

In the quadratic costs setting, we set the holding cost function $H(x)$ to

$$H(x) = C_h x^2$$

where x is the number of jobs in the system and C_h is a positive constant. In

addition, the effort cost function $R(\mu)$ satisfies

$$R(\mu) = C_r \mu^2$$

where C_r is a positive constant, and $\mu \in [\underline{\mu}, \bar{\mu}]$ with $\underline{\mu}, \bar{\mu} > 0$.

In both settings, the maintenance cost is C_m per unit time, and when a new job arrives when the system is at its full capacity, a rejection cost p will occur.

Table 3.1: Parameter Combinations for the System Dynamics

Scenario	λ	β_d	a_u	b_u
1	5	1	$\frac{1}{50}$	$\frac{1}{50}$
2	10	1	$\frac{1}{50}$	$\frac{1}{50}$
3	5	5	$\frac{1}{50}$	$\frac{1}{50}$
4	10	5	$\frac{1}{50}$	$\frac{1}{50}$
5	5	1	$\frac{1}{25}$	$\frac{1}{50}$
6	10	1	$\frac{1}{25}$	$\frac{1}{50}$
7	5	5	$\frac{1}{25}$	$\frac{1}{50}$
8	10	5	$\frac{1}{25}$	$\frac{1}{50}$

Table 3.2: Cost Parameter Settings

Settings	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
C_r	1	100	1	1	1	100	100	100	1	1	1	100	100	100	1
C_h	1	1	100	1	1	100	1	1	100	100	1	100	100	1	100
C_m	1	1	1	100	1	1	100	1	100	1	100	100	1	100	100
p	1	1	1	1	100	1	1	100	1	100	100	1	100	100	100

In this section, equations (3.25) to (3.30) are solved to obtain the optimal control. The parameters for the system dynamics are listed in the Table 3.1. In all the scenarios, we set $\underline{\mu} = 1$, $\bar{\mu} = 20$, and $N = 100$. The parameters of the cost functions are given in Table 3.2. The cost parameters take values in the set $\{1, 100\}$ and are permuted to show the impact of costs on the optimal policy.

There are 120 parameter combinations for the numerical study of the optimal controls under both linear and quadratic cost settings.

Overall, scenarios 1, 3, 5, and 7 represent a less busy system compared to the

scenarios 2, 4, 6, and 10. Moreover, scenarios 1, 2, 5, and 6 represent a more responsive system in terms of maintenance compared to scenarios 3, 4, 7, and 8. The first four scenarios represent a system with a more reliable machine compared to the last four scenarios.

In the first and second scenarios, the fraction of time for up states approximately ranges from 70.4% to 96.2%. On average, 5 and 10 jobs arrive per unit of time in these two scenarios, respectively. In the third and fourth scenarios, the fraction of time for the up states approximately ranges from 92.3% to 99.2%. On average, 5 jobs arrive per unit time in the third scenario and 10 jobs arrive per unit time on average in the scenario 5 . Because β_d is set to 5 in scenarios 4 and 5, the machine is able to recover much faster from a down state compared to the first two scenarios, which explains why the fraction of time in the up states is relatively higher in the third and fourth scenarios.

In the fifth and seventh scenarios, the average interarrival time is one fifth of unit time. The fraction of time for up states approximately ranges from 54.9% to 94.3% in the fifth scenario, and it ranges from 85.9% to 98.8% in the seventh scenario. In scenarios 6 and 8, 10 jobs arrive per unit of time on average. In the scenario 6, when the system always operates on the maximum/minimum service rate, the fraction of time when the system is in an up state is 54.9%/94.3%. In the scenario 8, when the system always operates at the maximum/minimum service rates, the fraction of time when the system is in an up state is 85.9%/98.8%. Because of the value of a_u is set to a larger number ($\frac{1}{25}$) compared to the first four scenarios ($\frac{1}{50}$), the impact of β_d on the fraction of up-state time is larger in the last four scenarios.

3.6.0.1 Linear Costs

For every parameter combination, the optimal controls are obtained by solving equations (3.25) to (3.30) where the cost functions are set to linear functions. The

optimal controls are shown in Figures 3.3 - 3.10, where the x-axis represents the number of jobs in the system and the y-axis corresponds to the service rate under the optimal policy.

The rejection rate is the ratio between the number of rejected jobs and the total number of arrivals. The server utilization and rejection rate from implementing the optimal policy across all the scenarios are listed in the Table 3.3 and 3.4, which are obtained via simulating 500,000 events.

Table 3.3: Server Utilization (%) across Scenarios 1 - 8 (Linear Costs)

Setting	C_r	C_h	C_m	p	S1	S2	S3	S4	S5	S6	S7	S8
1	1	1	1	1	87.54	76.49	97.32	95.12	76.29	59.74	95.14	91.75
2	100	1	1	1	96.22	96.58	99.05	99.16	94.42	94.80	98.76	98.93
3	1	100	1	1	87.23	78.79	97.28	95.19	76.33	60.12	94.98	91.67
4	1	1	100	1	87.85	77.53	97.28	95.61	76.46	58.49	95.04	91.44
5	1	1	1	100	86.65	77.74	97.21	95.31	75.91	60.71	94.95	91.13
6	100	100	1	1	86.97	77.89	97.38	95.63	76.15	58.30	95.08	91.54
7	100	1	100	1	96.32	96.08	99.17	99.12	94.64	94.40	98.80	98.91
8	100	1	1	100	86.96	77.51	97.33	95.52	76.81	60.03	95.09	91.42
9	1	100	100	1	86.98	77.79	97.18	95.51	76.91	59.20	94.99	91.44
10	1	100	1	100	87.60	78.06	97.40	95.26	76.29	59.73	95.03	91.33
11	1	1	100	100	86.82	77.73	97.39	95.26	76.04	59.06	95.13	90.98
12	100	100	100	1	86.90	78.44	97.38	95.53	76.74	57.68	95.23	91.62
13	100	100	1	100	85.72	78.14	97.26	95.55	75.71	58.80	94.86	91.23
14	100	1	100	100	86.79	76.79	97.26	95.65	75.40	59.69	94.99	90.97
15	1	100	100	100	86.86	78.07	97.30	95.57	77.21	59.53	95.07	91.49

Table 3.4: Rejection Rates (%) across Scenarios 1 - 8 (Linear Costs)

Setting	C_r	C_h	C_m	p	S1	S2	S3	S4	S5	S6	S7	S8
1	1	1	1	1	0.00	0.26	0.00	0.00	0.00	1.32	0.00	0.00
2	100	1	1	1	80.56	90.44	79.99	89.80	80.38	90.51	80.00	90.04
3	1	100	1	1	0.00	0.01	0.00	0.00	0.00	1.62	0.00	0.00
4	1	1	100	1	0.00	0.09	0.00	0.00	0.00	1.99	0.00	0.00
5	1	1	1	100	0.00	0.06	0.00	0.00	0.00	1.88	0.00	0.00
6	100	100	1	1	0.00	0.05	0.00	0.00	0.00	3.29	0.00	0.00
7	100	1	100	1	80.83	90.31	79.93	90.07	80.82	90.50	80.31	89.88
8	100	1	1	100	0.00	0.15	0.00	0.00	0.00	1.47	0.00	0.00
9	1	100	100	1	0.00	0.11	0.00	0.00	0.00	2.03	0.00	0.00
10	1	100	1	100	0.00	0.08	0.00	0.00	0.00	2.17	0.00	0.00
11	1	1	100	100	0.00	0.03	0.00	0.00	0.00	2.05	0.00	0.00
12	100	100	100	1	0.00	0.07	0.00	0.00	0.00	2.63	0.00	0.00
13	100	100	1	100	0.00	0.12	0.00	0.00	0.00	1.87	0.00	0.00
14	100	1	100	100	0.00	0.13	0.00	0.00	0.00	1.68	0.00	0.00
15	1	100	100	100	0.00	0.15	0.00	0.00	0.00	1.92	0.00	0.00

From rejection rates in the Table 3.4, we can see that the system tends to have a high rejection rate when the running cost is high (see, for example, cost settings

2 and 7). This is reasonable because the optimal control tends to run the server at a lower service rate when the effort cost is high (see, for example, controls under cost settings 2 and 7 in Figures 3.3 - 3.10), and the system is likely to be at a high congestion level. Therefore, it is more likely to observe rejections. However, when the holding cost or rejection penalty increases, the rejection rate decreases even when the effort cost is high (see, for example, cost settings 6, 8, 12, 13, and 14). This is justifiable as an increased holding cost could result in optimal control policies that prevent the system from entering a high congestion level. And when the rejection penalty is high, the optimal control would decrease the likelihood of rejections. Hence, rejections are rarely observed.

In addition, scenarios 2, 4, 6 and 8 have relatively higher rejection rates compared to other scenarios. The arrival rate is set to 10 in scenarios 2, 4, 6 and 8, whereas in other scenarios, the arrival rate is set to 5. Since the system is much more busy when it has a higher arrival rate, it makes sense to observe more rejections in these cases.

For the linear cost setting, the optimal service rate switches at most once in the control policy, and the service rate remains the same after the switch. In all the parameter combinations of the cost setting, the service rate remains the same for $x \geq 10$.

By comparing the policy under the cost setting 2 to other policies, we can observe that the optimal policy will only use the smallest service rate when the effort cost C_r is the highest cost. When both C_r and C_h (or p) are high (the cost setting 6 and 8), the optimal control still uses the highest service rate. However, when C_r and C_m are high (the cost setting 7), only the smallest service rate is used in the optimal policy.

The optimal controls under the cost setting 4, in which $C_r = C_h = p = 1$ and $C_m = 100$, switch to the maximum service rate when $x = 1$ in scenarios 4, 6 and 8.

However, the switch is delayed to $x = 2$ in scenarios 1, 2, 3, 5, and 7. The delay may be caused by a high downtime penalty as C_m is set to 100 in the cost setting. Because the arrival rate is relatively high ($\lambda = 10$) in scenarios 4, 6 and 8, a delay in the switch might result in a system congestion, therefore, the controls in these scenarios choose to switch to $\bar{\mu}$ when $x = 1$.

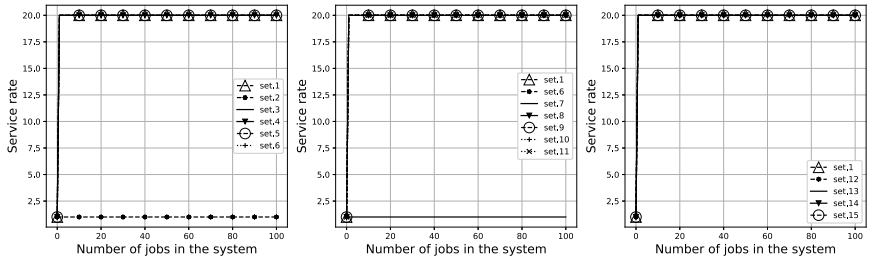


Figure 3.3: Optimal Controls of Scenario 1 (Linear Costs)

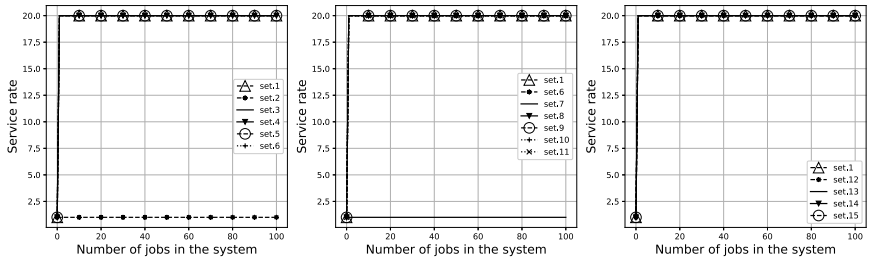


Figure 3.4: Optimal Controls of Scenario 2 (Linear Costs)

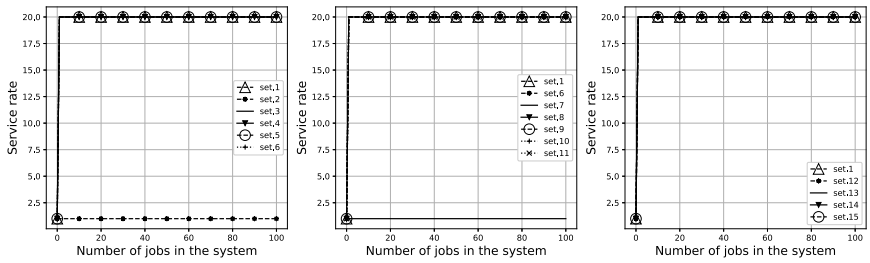


Figure 3.5: Optimal Controls of Scenario 3 (Linear Costs)

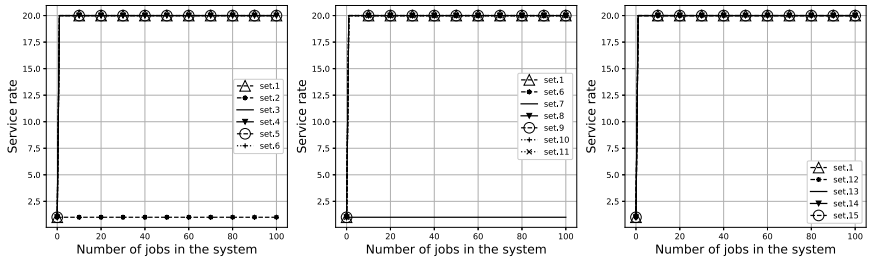


Figure 3.6: Optimal Controls of Scenario 4 (Linear Costs)

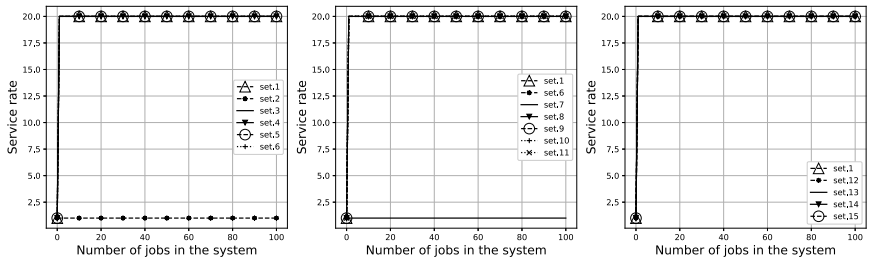


Figure 3.7: Optimal Controls of Scenario 5 (Linear Costs)

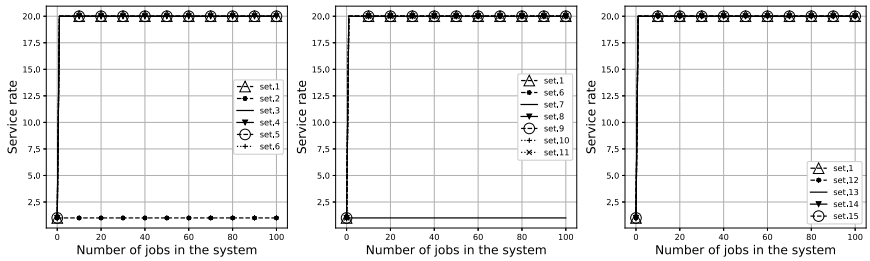


Figure 3.8: Optimal Controls of Scenario 6 (Linear Costs)

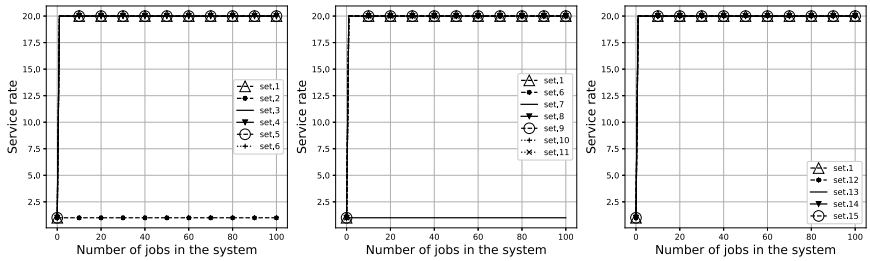


Figure 3.9: Optimal Controls of Scenario 7 (Linear Costs)

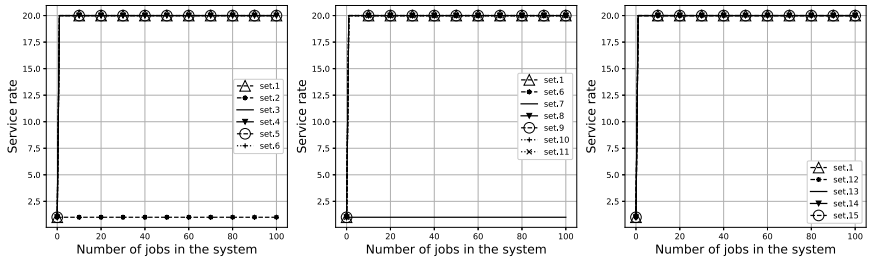


Figure 3.10: Optimal Controls of Scenario 8 (Linear Costs)

3.6.0.2 Quadratic Costs

Similar to the previous subsection, the optimal controls are obtained by solving equations (3.25) to (3.30), but the cost functions are set to quadratic functions. The optimal controls are shown in Figures 3.11 - 3.18. The server utilization and rejection rate from implementing the optimal policy across all the scenarios are listed in the Table 3.5 and 3.6, which are obtained via simulating 500,000 events.

Table 3.5: Server Utilization (%) across Scenarios 1 - 8 (Quadratic Costs)

Setting	C_r	C_h	C_m	p	S1	S2	S3	S4	S5	S6	S7	S8
1	1	1	1	1	87.48	78.95	97.49	95.42	77.63	58.20	95.36	91.85
2	100	1	1	1	88.11	96.33	97.66	99.18	79.13	94.50	95.55	98.78
3	1	100	1	1	87.42	79.00	97.42	95.60	77.04	59.78	94.99	91.22
4	1	1	100	1	87.77	78.20	97.45	95.57	78.04	58.43	95.59	91.74
5	1	1	1	100	88.12	77.43	97.69	95.76	76.68	57.71	95.52	91.49
6	100	100	1	1	88.44	78.37	97.68	95.57	77.19	60.75	95.61	91.65
7	100	1	100	1	88.17	96.52	97.69	99.25	78.64	94.35	95.65	98.78
8	100	1	1	100	88.70	96.63	97.44	98.88	78.27	94.46	95.71	98.76
9	1	100	100	1	87.14	78.10	97.43	95.42	76.50	60.71	95.07	91.50
10	1	100	1	100	86.79	78.62	97.51	95.33	76.01	60.58	95.29	91.33
11	1	1	100	100	87.77	77.67	97.55	95.11	77.51	59.59	95.37	91.82
12	100	100	100	1	87.96	78.34	97.55	95.68	76.97	59.39	95.49	91.81
13	100	100	1	100	87.17	78.58	97.45	95.49	77.70	59.64	95.27	91.68
14	100	1	100	100	88.52	96.14	97.63	99.24	78.09	93.96	95.49	98.76
15	1	100	100	100	87.19	77.80	97.23	95.61	76.46	59.29	94.98	91.09

The rejection rates in Table 3.6 indicate that when the effort cost is high, the system tends to have a high rejection rate (see, for example, cost settings 2, 7, 8, and 14). This is reasonable because the optimal control tends to run the server at a lower service rate when the effort cost is high (see, for example, controls under cost settings 2, 7, 8, and 14 in Figures 3.11 - 3.18), and the system is likely to be at

Table 3.6: Rejection Rates (%) across Scenarios 1 - 8 (Quadratic Costs)

Setting	C_r	C_h	C_m	p	S1	S2	S3	S4	S5	S6	S7	S8
1	1	1	1	1	0.0	0.1	0.0	0.0	0.0	1.7	0.0	0.0
2	100	1	1	1	0.0	90.4	0.0	89.6	0.0	90.2	0.0	90.0
3	1	100	1	1	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0
4	1	1	100	1	0.0	0.1	0.0	0.0	0.0	2.2	0.0	0.0
5	1	1	1	100	0.0	0.2	0.0	0.0	0.0	2.0	0.0	0.0
6	100	100	1	1	0.0	0.2	0.0	0.0	0.0	1.2	0.0	0.0
7	100	1	100	1	0.0	90.2	0.0	89.9	0.0	90.4	0.0	90.0
8	100	1	1	100	0.0	90.2	0.0	82.4	0.0	90.5	0.0	89.9
9	1	100	100	1	0.0	0.0	0.0	0.0	0.0	1.4	0.0	0.0
10	1	100	1	100	0.0	0.1	0.0	0.0	0.0	1.5	0.0	0.0
11	1	1	100	100	0.0	0.0	0.0	0.0	0.0	1.9	0.0	0.0
12	100	100	100	1	0.0	0.1	0.0	0.0	0.0	2.8	0.0	0.0
13	100	100	1	100	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0
14	100	1	100	100	0.0	90.0	0.0	88.5	0.0	90.8	0.0	90.0
15	1	100	100	100	0.0	0.1	0.0	0.0	0.0	1.9	0.0	0.0

a high congestion level. Therefore, it is more likely to observe rejections. When the holding cost increases, the rejection rate decreases even when the effort cost is high (see, for example, cost settings 6, 12, and 13). This is justifiable as an increased holding cost could result in optimal control policies that prevent the system from entering a high congestion level.

However, increasing the rejection cost when the effort cost is high does not have a significant impact on the rejection rate (see, for instance, cost settings 8 and 14). From the optimal controls under cost settings 2 and 8 in Figures 3.11 - 3.18, we can see that an increased rejection penalty does not change the policy significantly when the effort cost is high. Therefore, the system has a high likelihood of entering a high congestion level. Hence, the rejection rate remains high. This is different from the linear case, it is likely due to the high running cost in the quadratic case has much larger impact on the optimal control.

Scenarios 2, 4, 6 and 8 have higher rejection rates compared to other scenarios similar to the linear case. However, when the arrival rate is relatively low, the rejection rate is near 0, which is different from the linear case.

From the optimal policies in Figures 3.11 - 3.18, we can see that policies under cost settings 2, 7, 8 and 14 are not monotonic. In the all 4 cases, the running cost

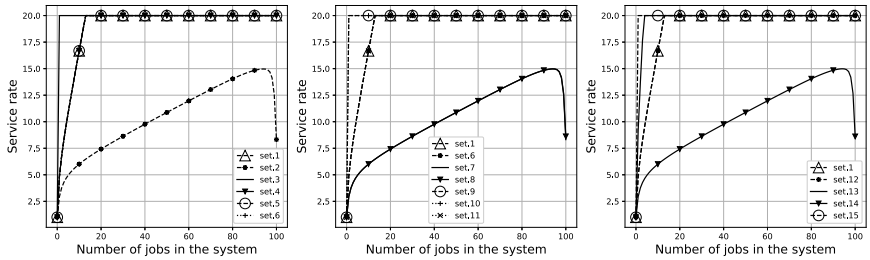


Figure 3.11: Optimal Controls of Scenario 1 (Quadratic Costs)

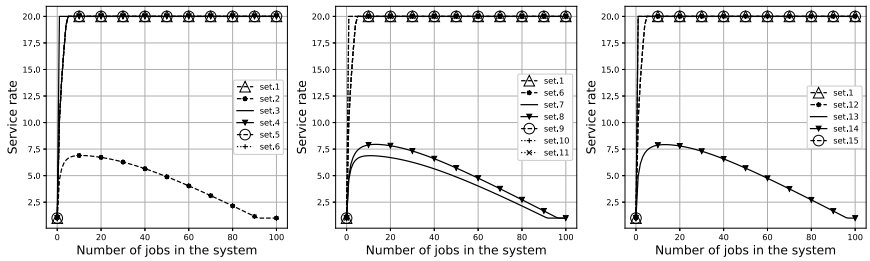


Figure 3.12: Optimal Controls of Scenario 2 (Quadratic Costs)

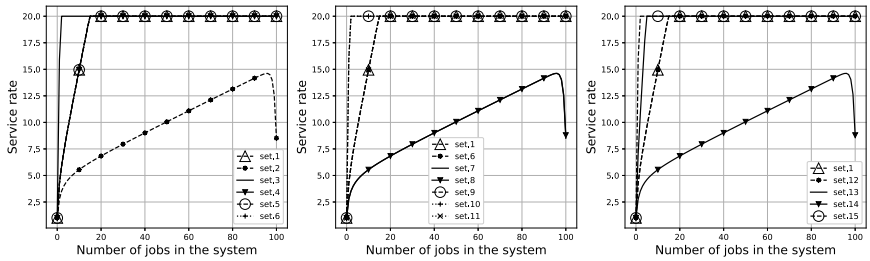


Figure 3.13: Optimal Controls of Scenario 3 (Quadratic Costs)

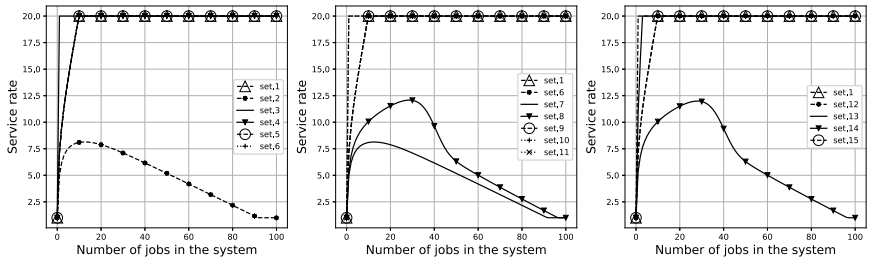


Figure 3.14: Optimal Controls of Scenario 4 (Quadratic Costs)

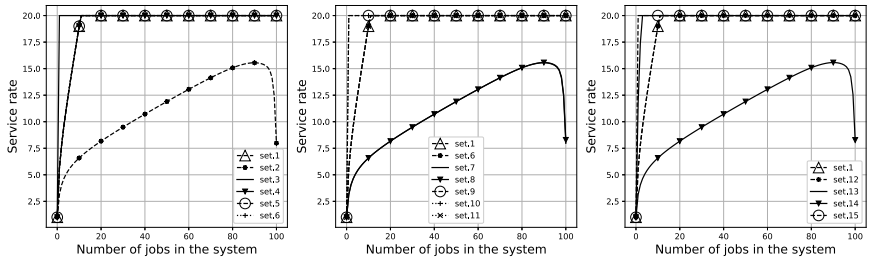


Figure 3.15: Optimal Controls of Scenario 5 (Quadratic Costs)

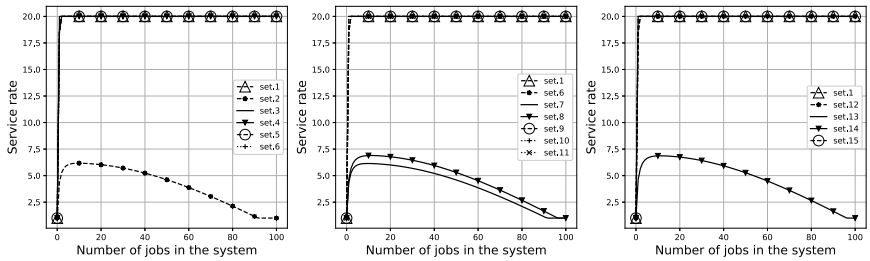


Figure 3.16: Optimal Controls of Scenario 6 (Quadratic Costs)

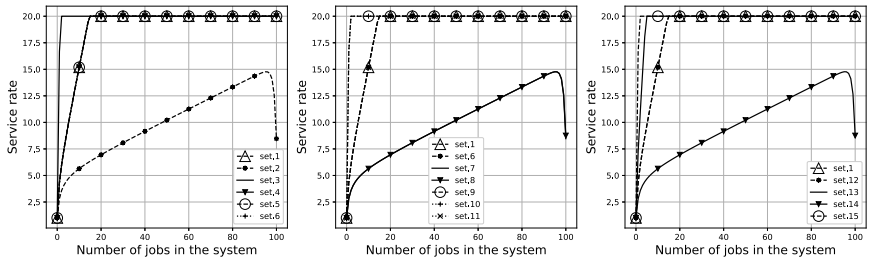


Figure 3.17: Optimal Controls of Scenario 7 (Quadratic Costs)

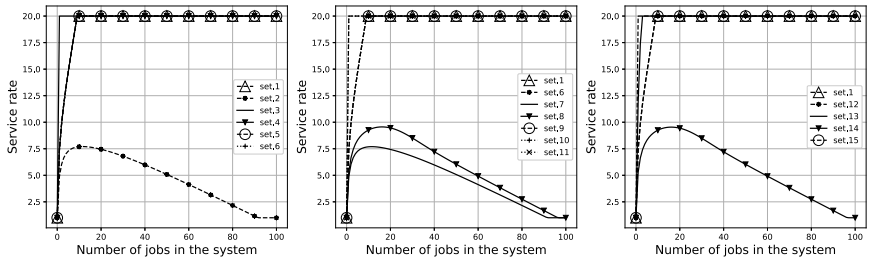


Figure 3.18: Optimal Controls of Scenario 8 (Quadratic Costs)

is set to the highest value. In other cases in which the running cost is high, if the holding cost is also high, the optimal control is still monotonic. The non-monotone policies can be further classified into two groups. In scenarios 1, 3, 5, and 7, the service rate decreases only when the number of jobs approaches the system capacity. However, in the remaining scenarios, the service rate decreases when $x = 20$, which is far away from the system capacity. Given that the arrival rate is relatively high ($\lambda = 10$) in scenarios 2,4, 6 and 8, the optimal controls are likely to avoid a further congestion by operating at a lower service rate, thereby reducing the likelihood of machine breakdowns. In scenarios 1, 3, 5, and 7, the optimal controls indicate that it is better to reduce the chance of machine breakdown by operating at a lower service rate when the system is at a high congestion level.

3.7 Learning

In practice, it might be costly to fully observe the system dynamics, and the dynamics may change. In such cases, errors in the estimation of system dynamics might result in a suboptimal control policy, which may incur additional costs over time. Because it requires knowledge of the system dynamics to obtain an optimal control, unknown parameters must be learnt.

Online learning algorithms are commonly used in this class of problems. The quality of an online learning algorithm is typically measured in terms of its regret, where the regret is the difference between the aggregate performance of the algorithm compared to the best decision made in hindsight. An algorithm is deemed to have a good performance if its regret approaches zero at a fast rate.

Recall that the breakdown rate is assumed to be a linear function of the service rate satisfying $\beta_u(\mu) = a_u\mu + b_u$ for $\mu \in [\underline{\mu}, \bar{\mu}]$ where a_u, b_u are positive constants. In the first subsection, we propose online algorithms for obtaining the optimal

policy when a_u and b_u are unknown by solving a set of quasi-likelihood equations.

The second subsection describes an online reinforcement learning algorithm that can obtain a good control policy when the system dynamics are entirely unknown.

3.7.1 Partially Unknown MDP

In this subsection, we assume that a_u and b_u are unknown positive constants. We use $T(\mu)$ to denote the sojourn time when the system is in the up state and the server operates with a service rate of $\mu \in [\underline{\mu}, \bar{\mu}]$. Assuming that the CTMC is governed by a control policy $\pi \in \Pi_s$, the probability density function of the sojourn time $T(\pi(x,1))$ is defined as

$$f^{\pi}(T(\pi(x,1))) = (\lambda + \pi(x,1) + \beta_u(\pi(x,1)))e^{-(\lambda+\pi(x,1)+\beta_u(\pi(x,1)))T(\pi(x,1))},$$

where $x = 1, 2, 3, ..., N-1$. Since $\beta_u(\mu) = a_u\mu + b_u$, it's evident that

$$f^{\pi}(T(\pi(x,1))) = (\lambda + \pi(x,1) + a_u\pi(x,1) + b_u)e^{-(\lambda+\pi(x,1)+a_u\pi(x,1)+b_u)T(\pi(x,1))}.$$

Since a_u and b_u are unknown, we can use the maximum log-likelihood estimation to estimate the unknown parameters based on the observed service rate μ_i and its sojourn time $T(\mu_i)$ for $i = 1, 2, 3, ..., n$, where n is the number of observations. Namely,

$$\max_{\hat{a}_u, \hat{b}_u} L(\hat{a}_u, \hat{b}_u; T(\mu_1), T(\mu_2), ..., T(\mu_n)) = \sum_{i}^{n} \ln((\lambda + \mu_i + \hat{a}_u\mu_i + \hat{b}_u)e^{-(\lambda+\mu_i+\hat{a}_u\mu_i+\hat{b}_u)T(\mu_i)}),$$

which can be rewritten as

$$\max_{\hat{a}_u, \hat{b}_u} \sum_{i}^{n} \ln(\lambda + \mu_i + \hat{a}_u\mu_i + \hat{b}_u) - \sum_{i}^{n} (\lambda + \mu_i + \hat{a}_u\mu_i + \hat{b}_u)T(\mu_i). \tag{3.41}$$

The first derivatives of the function with respect to $\hat{a}_u, \hat{b}_u$ are

$$\frac{\partial L}{\partial \hat{a}_u} = \sum_i^n \frac{\mu_i}{\lambda + \mu_i + \hat{a}_u \mu_i + \hat{b}_u} - \sum_i^n \mu_i T(\mu_i),$$

$$\frac{\partial L}{\partial \hat{b}_u} = \sum_i^n \frac{1}{\lambda + \mu_i + \hat{a}_u \mu_i + \hat{b}_u} - \sum_i^n T(\mu_i).$$

And the best estimate $\hat{a}_u^*, \hat{b}_u^*$ should take the form

$$\begin{cases} \sum_i^n \frac{\mu_i}{\lambda + \mu_i + \hat{a}_u^* \mu_i + \hat{b}_u^*} - \sum_i^n \mu_i T(\mu_i) = 0 \\ \sum_i^n \frac{1}{\lambda + \mu_i + \hat{a}_u^* \mu_i + \hat{b}_u^*} - \sum_i^n T(\mu_i) = 0 \end{cases} \tag{3.42}$$

Since the (3.42) is clearly non-convex, the solution of (3.42) may not be unique. Therefore, we choose the root with the lowest mean-least square error, see, for example [76, Chapter 13.3]. We define the error terms $\{\varepsilon_i\}_{i \in \mathbb{N}}$ by

$$\varepsilon_i := T(\mu_i) - \frac{1}{(\lambda + \mu_i + a_u \mu_i + b_u)}.$$

As the second equation of (3.42) show, the average of ε_i is 0 for the true values of a_u and b_u. Therefore, $\{\varepsilon_i\}_{i \in \mathbb{N}}$ forms a martingale with respect to its natural filtration. Hence, when we are solving equations (3.42), we select the solution that has the lowest $\frac{1}{N} \sum_i^N \varepsilon_i^2$.

The procedure is as follow:

1. Define stopping criterion

2. Initialize $\hat{a}_u$ and $\hat{b}_u$ with random positive numbers such that the ergodic condition 3.12 is satisfied.

3. If the stopping criterion is not met, solve the differential value functions (3.16) - (3.21) to obtain a control policy π where the a_u and b_u in β_d are substituted

by $\hat{a}_u$ and $\hat{b}_u$; otherwise, stop.

4. Operate the server using π and collect service rate $\pi(x, 1) \in [\underline{\mu}, \bar{\mu}]$ and its sojourn time $T(\pi(x, 1))$ when $x > 0$ and $K = 1$.

5. Update $\hat{a}_u$ and $\hat{b}_u$ by solving 3.42 and return to step 3.

Algorithm 12 shows the detail of the procedure.

3.7.2 Completely Unknown MDP

System dynamics of queuing systems can change for various reasons. For instance, as a machine ages, a higher speed might cause a more frequent breakdown; the arrival distribution changes completely when an unforeseen pandemic like COVID-19 occurs. Model-free reinforcement learning can adapt to such changes and provide a good quality control policy, although the optimality of the policy is not guaranteed.

The control variable is a continuous variable from a compact set, namely $\mu \in [\underline{\mu}, \bar{\mu}]$, and the problem is a continuous task, namely, no terminal state. A slightly modified (to accommodate the fact that the control problem minimizes the average cost and is a continuous control task) deep deterministic policy gradient method can be applied. Essentially, a parameterized policy is updated to minimize the cost J (3.3).

The DDPG algorithm combines both the DPG [36] and deep Q-network [31] (DQN) methods. It utilizes the DPG to update the actor (policy). Both experience replay and the target network from DQN are used in DDPG to stabilize the learning process of the critic (evaluator of the policy). The target Q-network is periodically frozen for a certain period before the update in the DQN; however, the parameters of the target networks are updated via a soft update procedure in DDPG. A Gaussian noise $\mathcal{N}$ is added when the action is drawn from the deterministic policy to ensure exploration.

Algorithm 12: Optimal Control with Estimated Parameters

Result: $\pi \approx \pi^*$
Input: $\lambda, \underline{\mu}, \bar{\mu}, C_r, C_h, C_m, \beta_d, N$
Parameters: $Iter_1, Iter_2$
Initialize $\hat{a}_u > 0, \hat{b}_u > 0$ randomly
Initialize $c_1 = 0, c_2 = 0$, $S = (0, 1)$, $\mu_{history} = [], t_{history} = [], n = 0$
while $c_1 \leq Iter_1$ **do**
  Update the estimation $\hat{a}_u$, $\hat{b}_u$ in β_u, solve the following system via the value iteration to obtain π

$$
\begin{aligned}
v(0,0) =& \frac{1}{q}[C_m - J^* + \beta_d v(0,1) + \lambda v(1,0) + (q - \beta_d - \lambda)v(0,0)] \\
v(0,1) =& \frac{1}{q}[R(\underline{\mu}) - J^* + \beta_u(\underline{\mu})v(0,0) + \lambda v(1,1) + (q - \beta_u(\underline{\mu}) - \lambda)v(0,1)] \\
v(x,0) =& \frac{1}{q}[H(x) + C_m - J^* + \beta_d v(x,1) + \lambda v(x+1,0) + (q - \beta_d - \lambda)v(x,0)] \\
v(x,1) =& \min_{\mu} \frac{1}{q}[H(x) + R(\mu) - J^* + \beta_u(\mu)v(x,0) + \lambda v(x+1,1) + \mu v(x-1,1) \\
& + (q - \beta_u(\mu) - \mu - \lambda)v(v,1)] \\
v(N,0) =& \frac{1}{q}[H(N) + C_m - J^* + \lambda p + \beta_d v(N,1) + (q - \beta_d)v(N,0)] \\
v(N,1) =& \min_{\mu} \frac{1}{q}[R(\mu) + H(N) - J^* + \lambda p + \beta_u(\mu)v(N,0) + \mu v(N-1,1) \\
& + (q - \beta_u(\mu) - \mu)v(N,1)]
\end{aligned}
$$

  $x = 1, 2, 3, ...N - 1$
  $c_1 \leftarrow c_1 + 1$
  if $c_1 > Iter_1$ **then**
    break
  $c_2 = 0$
  while $c_2 \leq Iter_2$ **do**
    if $S[1] = 1$ **then**
      $\mu = \pi(S)$
      Take action μ and observe S',sojourn time t
      $\mu_{history}.append(\mu)$
      $t_{history}.append(t)$
      $n \leftarrow n + 1$
    else
      observe S'
    $S \leftarrow S'$
    $c_2 \leftarrow c_2 + 1$
  Solve the following equations to find $\hat{a}_u^*$ and $\hat{b}_u^*$
  $\sum_i^n \frac{\mu_{history}[i]}{\lambda + \mu_{history}[i] + \hat{a}_u^* \mu_{history}[i] + \hat{b}_u^*} - \sum_i^n \mu_{history}[i] t_{history}[i] = 0$
  $\sum_i^n \frac{1}{\lambda + \mu_{history}[i] + \hat{a}_u^* \mu_{history}[i] + \hat{b}_u^*} - \sum_i^n t_{history}[i] = 0$
  $\hat{a}_u \leftarrow \hat{a}_u^*$
  $\hat{b}_u \leftarrow \hat{b}_u^*$
return π

In this section, the same MDP defined in Section 3.3 is used. Namely, the state space is defined as $\mathcal{S} \equiv [0, N] \times \{0, 1\}$ and the action space is defined as $\mathcal{A} \equiv [\underline{\mu}, \bar{\mu}]$.

The modified DDPG learns and improves the policy following this procedure:

1. Initialize a differentiable policy parameterization μ_Θ and a target policy $\mu_{\Theta_{targ}} := \mu_\Theta$. Policy μ_Θ outputs the action, and the target policy is used to estimate the long-term reward of the policy (see steps 9 and 10).

2. Initialize a differentiable state-action value function parameterization $Q_{\Theta'}$, and a target state-action value function $Q_{\Theta'_{targ}}$. For the state-action value functions, $Q_{\Theta'}$ is used to derive the gradient Θ. Since $Q_{\Theta'}$ takes the output of μ_Θ as an input, the cost can be reduced by improving $Q_{\Theta'}$ with respect to Θ using gradient-based methods. The target value function $Q_{\Theta'_{targ}}$ is used to estimate the long-term reward of the policy (see step 9 and 10).

3. Define stopping criterion, soft-update parameter $\rho \in [0, 1]$, learning rate of actor $\alpha_A \in (0, 1]$, critic $\alpha_C \in (0, 1]$, and average reward estimator α, and initialize an empty memory buffer $\mathcal{D}$, average reward estimator $\hat{J} = 0$, an exploration rate $\sigma > 0$ and an exploration rate decay factor $\gamma \in (0, 1)$.

4. Observe the current state S and generate an action A (service rate) from the policy, namely,

$$A = \min\{\bar{\mu}, \max\{\mu_\Theta(s) + \epsilon, \underline{\mu}\}\}$$

   where ϵ is a Gaussian noise from $\mathcal{N}(0, \sigma)$ to ensure the exploration. Since no explicit restriction is defined on the image of the parameterized policy μ_Θ, we adjust the output policy to ensure $A \in [\underline{\mu}, \bar{\mu}]$.

5. Take the action A and observe next on-state S', and the reward R (in this context, the reward is negative of the sum of holding cost, running cost and maintenance cost). Since the agent needs to provide an action only when the machine is in an up-state, if the machine is in a down-state, then the simulation will continue to run and accumulate the costs until the machine is in the up state.

6. Store the transitions (S, A, R, S') in memory buffer $\mathcal{D}$.

7. If the updating criterion is satisfied, go to the next step; else, return to step 4.

8. Sample a batch of transitions $B = (s, a, r, s')$ from memory $\mathcal{D}$.

9. Estimate the long-term rewards of sampled transitions using the target policy and target value function,

$$y(r, s') = r - \hat{J} + Q_{\Theta'_{targ}}(s', \mu_{\Theta_{targ}}(s')) \quad \forall (s, a, r, s') \in B$$

10. Update the estimated average reward

$$\hat{J} = \hat{J} + \alpha \frac{1}{|B|} (\sum_{(s,a,r,s') \in B} y(r, s') - Q_{\Theta'_{targ}}(s', \mu_{\Theta_{targ}}(s')))$$

11. Update Q-function by one step of gradient descent

$$\Theta' \leftarrow \Theta' - \alpha_C \nabla_{\Theta'} \frac{1}{|B|} \sum_{(s,a,r,s') \in B} (Q_{\Theta'}(s, a) - y(r, s'))^2$$

12. Update policy by one step of gradient ascent using

$$\Theta \leftarrow \Theta + \alpha_A \nabla_{\Theta} \frac{1}{|B|} \sum_{s \in B} Q_{\Theta'}(s, \mu_{\Theta}(s))$$

13. Update the target networks with

$$\Theta_{targ} \leftarrow \rho \Theta_{targ} + (1 - \rho)\Theta$$

$$\Theta'_{targ} \leftarrow \rho \Theta'_{targ} + (1 - \rho)\Theta'$$

14. Decrease the exploration rate

$$\sigma \leftarrow \gamma\sigma$$

15. Stop if stopping criterion is satisfied, else go to step 4

Note that the method only requires the ergodicity from the MDP, and it views the underlying queuing system as a black box that returns the next state and cost for an input action. The modified DDPG is described in the Algorithm 13.

Algorithm 13: Modified DDPG for Continuous Control Tasks with Average Reward

Result: $\mu_\Theta \approx \mu^*$
Input: a differentiable policy parameterization μ_Θ, a target policy $\mu_{\Theta_{targ}} := \mu_\Theta$
Input: a differentiable state-action value function parameterization $Q_{\Theta'}$, a target Q-function $Q_{\Theta'_{targ}} := Q_{\Theta'}$
Parameters: α_A, α_C, α, $Iter$, ρ, $freq$, γ, a_{Low},a_{High}, σ, γ
Initialize policy parameter Θ, state-value parameters Θ', episode count $epoch = 0$, step count $step = 0$, an empty experience buffer $\mathcal{D}$, $\hat{J} = 0$
while *True* **do**
  Initialize first state of episode S
  $A = min[max[\mu_\Theta(S) + \epsilon, a_{Low}], a_{High}]$, where $\epsilon \sim \mathcal{N}$
  Take action A and observe S', R
  Store (S, A, R, S') in the replay buffer $\mathcal{D}$
  $S \leftarrow S'$
  $step \leftarrow step + 1$
  if $step\%freq = 0$ **then**
    Sample a batch of transitions, $B = (s, a, r, s')$ from $\mathcal{D}$
    Compute targets

$$y(r, s') = r - \hat{J} + Q_{\Theta'_{targ}}(s', \mu_{\Theta_{targ}}(s'))$$

    Update estimated average cost

$$\hat{J} = \hat{J} + \alpha \frac{1}{|B|}\Big(\sum_{(s,a,r,s')\in B} y(r, s') - Q_{\Theta'_{targ}}(s', \mu_{\Theta_{targ}}(s'))\Big)$$

    Update Q-function by one step of gradient descent using

$$\Theta' \leftarrow \Theta' - \alpha_C \nabla_{\Theta'} \frac{1}{|B|} \sum_{(s,a,r,s')\in B} (Q_{\Theta'}(s, a) - y(r, s'))^2$$

    Update policy by one step of gradient ascent using

$$\Theta \leftarrow \Theta + \alpha_A \nabla_\Theta \frac{1}{|B|} \sum_{s\in B} Q_{\Theta'}(s, \mu_\Theta(s))$$

    Update the target networks with

$$\Theta_{targ} \leftarrow \rho\Theta_{targ} + (1 - \rho)\Theta$$

$$\Theta'_{targ} \leftarrow \rho\Theta'_{targ} + (1 - \rho)\Theta'$$

    Decrease the exploration rate

$$\sigma \leftarrow \gamma\sigma$$

  $epoch \leftarrow epoch + 1$
  if *epoch=Iter* **then**
    break
  return μ_Θ

3.8 Numerical Study: Adaptive Control

In this section, we present the numerical examples of the adaptive control problems.

3.8.1 Partially Unknown MDP

We first show the results of the proposed learning algorithms when a_u and b_u are unknown positive constants. These parameters are used in the breakdown rate $\beta_u(\mu) = a_u\mu + b_u$ for $\mu \in [\underline{\mu}, \bar{\mu}]$.

Recall that $X(t)$ and $K(t)$ are the number of jobs in the system and the status of the server at time $t \geq 0$. Let m_T denote the number of transitions for the process before time t. Let $\tau_0 = 0$, and denote $\tau_i, 1 \leq i \leq m_t$, as the i-th jump time of the process. Let $g^\pi(x,k)$ be the unit cost under an admissible stable policy π.

$$g^\pi(x,k) = R(\pi(x,k)) + H(x) + C_m(1-k). \tag{3.43}$$

We define the cumulative cost function F as follows:

$$\begin{aligned} F(t) := &\sum_{i=1}^{m_t} \int_{\tau_{i-1}}^{\tau_i} g^{\pi_{i-1}}(X(s),K(s))ds + \int_{\tau_{m_t}}^{t} g^{\pi_{m_t}}(X(s),K(s))ds \\ &+ p\sum_{i=1}^{m_t}(A(\tau_i) - A(\tau_{i-1})) \int_{\tau_{i-1}}^{\tau_i} \mathbb{1}(X(s)=N)ds \\ &+ p(A(t) - A(\tau_{m_t})) \int_{\tau_{m_t}}^{t} \mathbb{1}(X(s)=N)ds \end{aligned}$$

for $t \geq 0$, where the π_i denotes the policy used in the i-th jump above and $A(t)$ is the number of arrivals up to time t.

We use the regret to measure the performance of the proposed algorithms

$$R(n) := \frac{1}{t_n}\mathbb{E}[F(t_n)] - J^*$$

for a positive integer $n \leq L$, where L denotes the number of timestamps in the simulation.

We set L=1000, and t_L is chosen to be sufficiently large such that the average regret is near zero as n approaches L. We conduct experiments for both cost function types under the first system dynamic scenario shown in Tables 3.1. We also use the cost parameter setting 1, namely, $C_r = C_h = C_m = p = 1$.

In Figures 3.19 - 3.20, the x-axis represents the timestamps in the simulations with the difference between the timestamps equal to $\frac{t_L}{L}$, and the y-axis corresponds to the average regret $R(n)$ at each timestamp t_n. In each experiment, the expectation is approximated by the average over the values of 300 trajectories. As shown in the figures, the average regret asymptotically converges to 0, which validates the proposed algorithms.

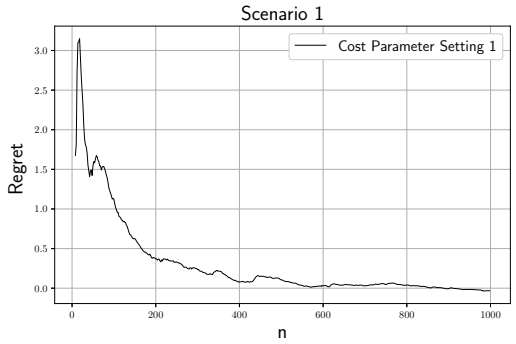


Figure 3.19: Poisson Arrivals with Linear Cost Functions

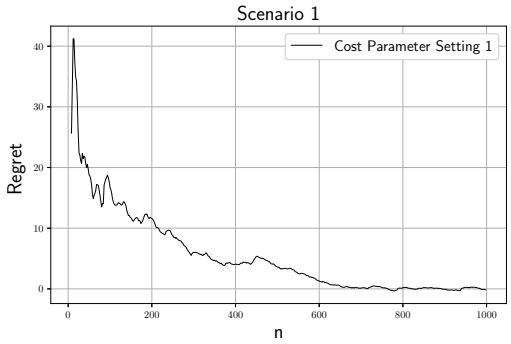


Figure 3.20: Poisson Arrivals with Quadratic Cost Functions

3.8.2 Completely Unknown MDP

In this section, we use the DDPG algorithm to learn the control policy for the problem under two types of cost functions. The configurations of the algorithm are shown in Table 3.7.

Parameters	Value
α_A	0.01
α_C	0.01
α	0.01
σ	3.5
γ	0.99
ρ	0.99
$freq$	10
$Iter$	20000
a_{Low}	1.5
a_{High}	5
Neural network structure (actor)	[16,8]
Activation function (actor)	ReLU,ReLU,Linear
Neural network structure (critic)	[16,8]
Activation function (critic)	ReLU,ReLU,Linear

Table 3.7: DDPG Configurations

Linear Cost

The system dynamics used in this experiment are list in Table 3.8. Figure 3.21 shows the average cost obtained from the training process, and Figure 3.22 compares the policy obtained by the DDPG algorithm to the theoretical optimal policy.

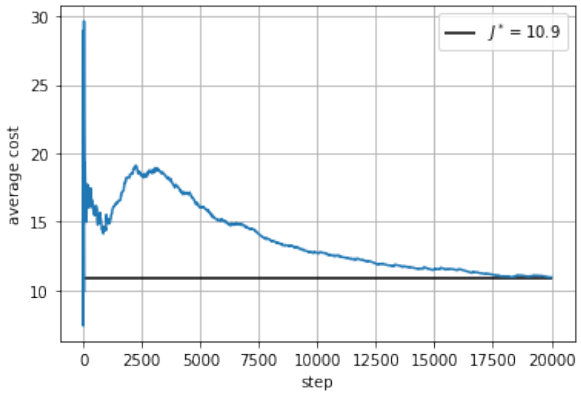


Figure 3.21: Average Cost of DDPG Training

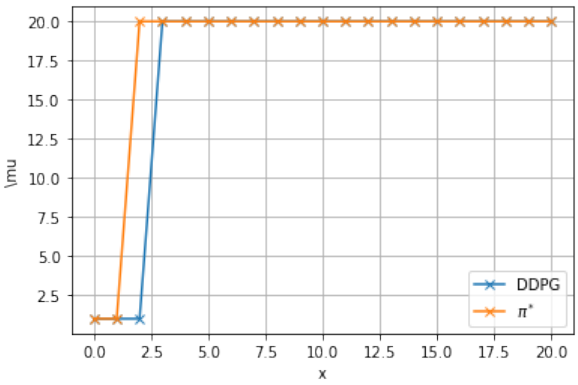


Figure 3.22: Policy Obtained from DDPG Method

50,000 runs of simulations using the policy obtained by the DDPG result in an average cost of 11.1, although it is higher than the theoretical optimal cost (10.9), it is much lower than the average cost of constant policies (21.87 &31.82)

Table 3.8: System Dynamics Parameters

Parameters	Linear Cost	Quadratic Cost
λ	5	5
$\underline{\mu}$	1	1
$\bar{\mu}$	20	20
β_d	5	5
a_u	$\frac{1}{25}$	$\frac{1}{25}$
b_u	$\frac{1}{50}$	$\frac{1}{50}$
N	20	20
C_r	1	1
C_h	1	1
C_m	100	100
p	1	1

and uniformly random policy (17.31).

Quadratic Cost

The system dynamics used in this experiment are shown in Table 3.8. Figure 3.23 shows the average cost obtained during the training process, and Figure 3.24 compare the policy obtained by the DDPG algorithm to the theoretical optimal policy.

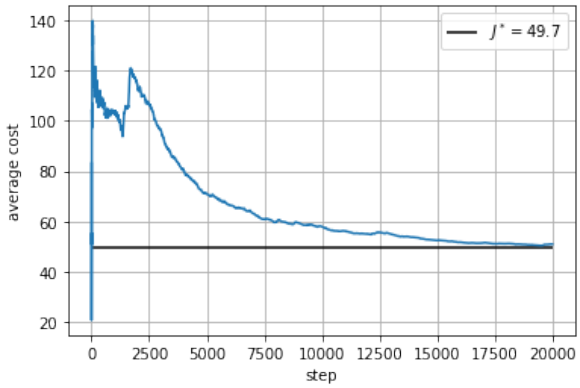


Figure 3.23: Average Cost of DDPG Training

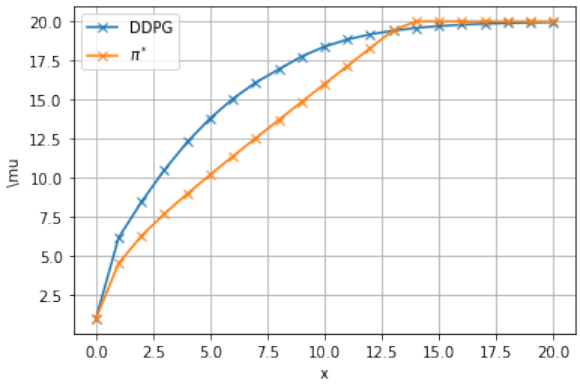


Figure 3.24: Policy Obtained from DDPG Method

50,000 runs of simulations using the policy obtained by the DDPG result in an

average cost of 51.8, it is slightly higher than the theoretical optimal cost (49.7), it is much lower than the average cost of constant policies (172.51&360.07) and uniformly random policy (121.29).

3.9 Conclusions

In this chapter, we investigate a service rate control problem for a finite capacity single-server queue with an unreliable server. The goal is to minimize a combination of effort cost, holding cost, maintenance cost and rejection cost incurred per unit time, and we study the problem under an average cost optimality criterion. We show the conditions under which the stable admissible policy exists. We further prove the optimality of a policy derived from the solution of the Bellman equations. Additionally, an algorithm is proposed to learn the optimal policy based on the estimated downrate parameters when they are unknown. Numerical studies are conducted to validate algorithms in both linear and quadratic cost cases. Furthermore, a model-free RL algorithm, DDPG, is used to learn when all parameters of the queue system are unknown in the case of Poisson arrival, and the results shows that the obtained control policy is similar to the optimal policy. As the algorithm is based on the policy gradient theorem, in a general case, the policy obtained from DDPG is a local optimum.

The learning algorithm for the partially unknown MDP utilizes the underlying system dynamics to estimate the unknown parameters, therefore, it is a model-based method. The DDPG method used in the case of completely unknown MDP, however, is a model-free method, namely, it finds control policies solely from interacting with the environment. From the results, we can see that the model-free method cannot obtain the exact optimal control policy under the configurations. Although the result favors the model-based method, when the underlying MDP

has a high-dimensional state space and action space, the computational cost of the model-based method will be high and a near-optimal control policy from a model-free method will be more practical in general.

This work can be extended in several ways. In this chapter, we assume that the server runs at the lowest speed even when the system is empty, and this is a valid assumption for a twenty-four hours operating floor. However, by adding an additional layer of control, for instance, the policy can switch the machine off and on via a N-policy [103, 112], the problem can be applied to more general settings. In addition, the arrival process is assumed to be stationary and it can be extended to a non-homogeneous Poisson or Markov modulated Poisson arrival process [96].

Bibliography

[1] Herrmann, J. W. (2006) *Handbook of production scheduling*, arXiv:1011.1669v3.

[2] Dequeant, K., P. Vialletelle, P. Lemaire, and M. L. Espinouse (2017) "A literature review on variability in semiconductor manufacturing: The next forward leap to Industry 4.0," in *Proceedings - Winter Simulation Conference*.

[3] Abu-Samah, A., M. K. Shahzad, E. Zamaï, and S. Hubac (2014) "Methodology for Integrated Failure-Cause Diagnosis with Bayesian Approach : Application to Semiconductor Manufacturing Equipment," *European Conference of the Prognostics and Health Management Society*, (i), pp. 1–11.

[4] Aberdeen, "The Rising Cost of Downtime," https://www.aberdeen.com/techpro-essentials/stat-of-the-week-the-rising-cost-of-downtime.

[5] Advanced Technology Services, I., "Downtime Costs Auto Industry $22K/Minute - Survey," https://news.thomasnet.com/companystory/downtime-costs-auto-industry-22k-minute-survey-481017.

[6] Frazee, T. and C. Standridge (2016) "Conwip versus POLCA: A comparative analysis in a high-mix, low-volume (HMLV) manufacturing environment with batch processing," *Journal of Industrial Engineering and Management*, **9**(2), pp. 432–449.

[7] Brockman, G., V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba (2016) "Openai gym," *arXiv preprint arXiv:1606.01540*.

[8] Chaari, T., S. Chaabane, N. Aissani, and D. Trentesaux (2014) "Scheduling under uncertainty: Survey and research directions," in *2014 International Conference on Advanced Logistics and Transport, ICALT 2014*, pp. 229–234.

[9] Al-Hinai, N. and T. Y. Elmekkawy (2011) "Robust and stable flexible job shop scheduling with random machine breakdowns using a hybrid genetic algorithm," *International Journal of Production Economics*, **132**(2), pp. 279–281.

[10] Xiong, J., L.-n. Xing, and Y.-w. Chen (2013) "Robust scheduling for multi-objective flexible job-shop problems with flexible workdays," *International Journal of Production Economics*, **141**, pp. 112—-126.
URL http://dx.doi.org/10.1016/j.ijpe.2012.04.015

[11] Sun, T. C., K. K. Lai, K. Lam, and K. P. So (1994) "A study of heuristics for bidirectional multi-hoist production scheduling systems," *International Journal of Production Economics*, **33**(1-3), pp. 207–214.

[12] Smith, S. (1995) "Reactive scheduling systems," *Intelligent scheduling systems*, p. 38.

[13] Subramaniam, V., A. S. Raheja, and K. Rama Bhupal Reddy (2005) "Reactive repair tool for job shop schedules," *International Journal of Production Research*, **43**(1), pp. 1–23.

[14] Zhang, W. and T. Dietterich (1995) "A reinforcement learning approach to job-shop scheduling," *International Joint Conference on Artificial ...*, pp. 1114–1120.

[15] Aydin, M. E. and E. Öztemel (2000) "Dynamic job-shop scheduling using reinforcement learning agents," *Robotics and Autonomous Systems*, **33**(2), pp. 169–178.

[16] Wang, Y. C. and J. M. Usher (2005) "Application of reinforcement learning for agent-based production scheduling," *Engineering Applications of Artificial Intelligence*, **18**(1), pp. 73–82.

[17] Park, I.-B., J. Huh, J. Kim, and J. Park (2019) "A reinforcement learning approach to robust scheduling of semiconductor manufacturing facilities," *IEEE Transactions on Automation Science and Engineering*, **17**(3), pp. 1420–1431.

[18] Liu, C.-L., C.-C. Chang, and C.-J. Tseng (2020) "Actor-critic deep reinforcement learning for solving job shop scheduling problems," *IEEE Access*, **8**, pp. 71752–71762.

[19] Shahrabi, J., M. A. Adibi, and M. Mahootchi (2017) "A reinforcement learning approach to parameter estimation in dynamic job shop scheduling," *Computers & Industrial Engineering*, **110**, pp. 75–82.
URL http://linkinghub.elsevier.com/retrieve/pii/S0360835217302309

[20] Sutton, R. S. and A. G. Barto (2018) *Reinforcement learning: An introduction*.

[21] Bellman, R. (1957) "A Markovian decision process," *Journal of mathematics and mechanics*, pp. 679–684.

[22] Howard, R. A. (1960) "Dynamic programming and markov processes." .

[23] Puterman, M. L. and M. C. Shin (1978) "Modified policy iteration algorithms for discounted Markov decision problems," *Management Science*, **24**(11), pp. 1127–1137.

[24] Bertsekas, D. P., D. P. Bertsekas, D. P. Bertsekas, and D. P. Bertsekas (1995) *Dynamic programming and optimal control*, vol. 1, Athena scientific Belmont, MA.

[25] Rummery, G. A. and M. Niranjan (1994) *On-line Q-learning using connectionist systems*, vol. 37, University of Cambridge, Department of Engineering Cambridge, UK.

[26] Watkins, C. J. C. H. (1989) "Learning from delayed rewards," .

[27] Watkins, C. J. and P. Dayan (1992) "Q-learning," *Machine learning*, **8**(3-4), pp. 279–292.

[28] Rubinstein, R. (1981) "Simulation and the Monte Carlo Method, Wiley," *New York*.

[29] Kalos, M. and P. Whitlock (1986), "Monte Carlo Methods: Basics, vol. 1," .

[30] Barto, A. G., R. S. Sutton, and C. Watkins (1989) *Learning and sequential decision making*, University of Massachusetts Amherst, MA.

[31] Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. (2015) "Human-level control through deep reinforcement learning," *Nature*, **518**(7540), pp. 529–533.

[32] Broomhead, D. and D. Lowe (1988) "Multivariable functional interpolation and adaptive networks, complex systems, vol. 2," .

[33] Narasimhan, K., T. Kulkarni, and R. Barzilay (2015) "Language understanding for text-based games using deep reinforcement learning," *arXiv preprint arXiv:1506.08941*.

[34] Sutton, R. S., D. A. McAllester, S. P. Singh, and Y. Mansour (2000) "Policy gradient methods for reinforcement learning with function approximation," in *Advances in neural information processing systems*, pp. 1057–1063.

[35] Williams, R. J. (1992) "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, **8**(3-4), pp. 229–256.

[36] Silver, D., G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller (2014) "Deterministic policy gradient algorithms," .

[37] Schaul, T., J. Quan, I. Antonoglou, and D. Silver (2015) "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*.

[38] Ishfaq, K., N. Mufti, J. Ahmad, M. Sajid, and M. Jahanzaib (2018) "Analysis of the Effect of Wire Electric Dischage Machining Process Parameters for the Formation of High Speed Steel Form Tool," *Advances in Science and Technology Research Journal*.

[39] Mikell P. Groover (2014) "Fundamentals of Modern Manufacturing: Materials, Processes, and Systems," *Igarss 2014*, `arXiv:1011.1669v3`.

[40] Srikant, R., D. N. Rao, C. S. Rao, and M. S. Subrahmanyam (2008) "Mathematical Modeling of the Influence of Emulsifier Content on Performance of Cutting Fluids," *Journal of the Brazilian Society of Mechanical Sciences and Engineering*.

[41] Seibt, M., M. Apel, A. Doller, H. Ewe, E. Spiecker, W. Schroter, and A. Zozime (1998) "Modeling and Experimental Verification of Gettering Mechanisms," in *Semi-Conductor Conference, volume 1*, p. 1064.

[42] Luo, Y., H. Guo, X. Sun, M. Mao, and J. Guo (2017) "Effects of Austenitizing Conditions on the Microstructure of AISI M42 High-Speed Steel," *Metals*.

[43] Garey, M. R., D. S. Johnson, and R. Sethi (1976) "The Complexity of Flowshop and Jobshop Scheduling," *Mathematics of Operations Research*, **1**(2), pp. 117–129.
URL `http://pubsonline.informs.org/doi/abs/10.1287/moor.1.2.117`

[44] Allahverdi, A., C. T. Ng, T. C. Cheng, and M. Y. Kovalyov (2008) "A survey of scheduling problems with setup times or costs," *European Journal of Operational Research*.

[45] Sharma, P. and A. Jain (2016), "A review on job shop scheduling with setup times," .

[46] Vela, C. R., R. Varela, and M. A. González (2010) "Local search and genetic algorithm for the job shop scheduling problem with sequence dependent setup times," *Journal of Heuristics*.

[47] Naderi, B., M. Zandieh, and S. M. Fatemi Ghomi (2009) "Scheduling job shop problems with sequence-dependent setup times," *International Journal of Production Research*.

[48] Cheung, W. and H. Zhou (2001) "Using Genetic Algorithms and Heuristics for Job Shop Scheduling with Sequence-Dependent Setup Times," *Annals of Operations Research*.

[49] Naderi, B., M. Zandieh, and S. M. Fatemi Ghomi (2009) "Scheduling sequence-dependent setup time job shops with preventive maintenance," *International Journal of Advanced Manufacturing Technology*.

[50] Shen, L. (2014) "A tabu search algorithm for the job shop problem with sequence dependent setup times," *Computers and Industrial Engineering*.

[51] Shen, L., S. Dauzère-Pérès, and J. S. Neufeld (2018) "Solving the flexible job shop scheduling problem with sequence-dependent setup times," *European Journal of Operational Research*.

[52] Naderi, B., S. M. Ghomi, and M. Aminnayeri (2010) "A high performing metaheuristic for job shop scheduling with sequence-dependent setup times," *Applied Soft Computing Journal*.

[53] Driss, E., R. Mallouli, and W. Hachicha (2018) "Mixed integer programming for job shop scheduling problem with separable sequence-dependent setup times," *American Journal of Mathematical and Computational Sciences*, **3**(1), pp. 31–36.

[54] Moghaddas, R. and M. Houshmand (2008) "Job-shop scheduling problem with sequence dependent setup times," *Proceedings of the International MultiConference of Engineers and Computer Scientists (IMECS), Vol.II, 19-21 March, 2008, Hong Kong*.

[55] Low, C., T. H. Wu, and C. M. Hsu (2005) "Mathematical modelling of multi-objective job shop scheduling with dependent setups and re-entrant operations," *International Journal of Advanced Manufacturing Technology*.

[56] Stecke, K. E. (1983) "Formulation and solution of nonlinear integer production planning problems for flexible manufacturing systems," *Management Science*, **29**(3), pp. 273–288.

[57] ——— (1985) "Design, planning, scheduling, and control problems of flexible manufacturing systems," *Annals of Operations Research*.

[58] Calmels, D. (2018) "The job sequencing and tool switching problem: state-of-the-art literature review, classification, and trends," *International Journal of Production Research*.

[59] Bard, J. F. (1988) "A heuristic for minimizing the number of tool switches on a flexible machine," *IIE Transactions (Institute of Industrial Engineers)*.

[60] Van Hop, N. and N. N. Nagarur (2004) "The scheduling problem of PCBs for multiple non-identical parallel machines," *European Journal of Operational Research*.

[61] Lee, G., H. Sarmadi, and S. Gholami (2012) "Modeling of Tool Switching Problem in a Flexible Manufacturing Cell: with two or More Machines," in *International Conference on Mechanical and Electrical Technology, 3rd, (ICMET-London 2011), Volumes 1–3*.

[62] Özpeynirci, S., B. Gökgür, and B. Hnich (2016) "Parallel machine scheduling with tool loading," *Applied Mathematical Modelling*.

[63] Khan, B. K., B. Gupta, D. S. Gupta, and K. Kumar (2000) "A generalized procedure for minimizing tool changeovers of two parallel and identical CNC machining centres," *Production Planning & Control*, **11**(1), pp. 62–72.

[64] Fathi, Y. and K. Barnette (2002) "Heuristic procedures for the parallel machine problem with tool switches," *International Journal of Production Research*, **40**(1), pp. 151–164.

[65] Gökgür, B., B. Hnich, and S. Özpeynirci (2018) "Parallel machine scheduling with tool loading: a constraint programming approach," *International Journal of Production Research*, **56**(16), pp. 5541–5557.

[66] Manne, A. S. (1960) "On the Job-Shop Scheduling Problem," *Operations Research*.

[67] Sreejith, P. and B. Ngoi (2000) "Dry machining: machining of the future," *Journal of materials processing technology*, **101**(1-3), pp. 287–291.

[68] Groover, M. P. (2020) *Fundamentals of modern manufacturing: materials, processes, and systems*, John Wiley & Sons.

[69] Chong, W. (2004) "Analysis of the Failures of High-Precision Progressive Dies [J]," *Die & Mould Industry*, **10**.

[70] White, H. and L. S. Christie (1958) "Queuing with Preemptive Priorities or with Breakdown," *Operations Research*, **6**(1), pp. 79–95.

[71] Gaver, D. P. (1962) "A Waiting Line with Interrupted Service, including Priorities," *Journal of the Royal Statistical Society. Series B (Methodological)*, **24**(1), pp. 73–90.
URL http://www.jstor.org/stable/2983746%5Cnhttp://about.jstor.org/terms

[72] Avi-Itzhak, B. and P. Naor (1963) "Some Queuing Problems with the Service Station Subject to Breakdown," *Operations Research*, **11**(3), pp. 303–320.
URL http://dl.acm.org/citation.cfm?id=2772421.2772422

[73] Yadin, M. and P. Naor (1963) "Queueing systems with a removable service station," *Journal of the Operational Research Society*, **14**(4), pp. 393–405.

[74] Heyman, D. P. (1968) "Optimal operating policies for M/G/1 queuing systems," *Operations Research*, **16**(2), pp. 362–382.

[75] Mandl, P. (1985) "On self-optimizing control of Markov processes," in *Mathematical control theory*, vol. 14 of *Banach Center Publ.*, PWN, Warsaw, pp. 345–360.

[76] Heyde, C. C. (1997) *Quasi-likelihood and its application*, Springer Series in Statistics, Springer-Verlag, New York, a general approach to optimal parameter estimation.
URL https://doi.org/10.1007/b98823

[77] den Boer, A. V. and B. Zwart (2014) "Mean square convergence rates for maximum quasi-likelihood estimators," *Stoch. Syst.*, **4**(2), pp. 375–403.
URL https://doi.org/10.1214/12-SSY086

[78] Krishnamoorthy, A., P. K. Pramod, and S. R. Chakravarthy (2014) "Queues with interruptions: A survey," *TOP*, **22**(1), pp. 290–320.

[79] Kella, O. and W. Whitt (1990) "Diffusion Approximations for Queues with Server Vacations," *Advances in Applied Probability*, **22**(3), pp. 706–729.
URL http://www.jstor.org/stable/info/1427465

[80] Chen, H. and W. Whitt (1993) "Diffusion approximations for open queueing networks with service interruptions," *Queueing Systems*, **13**(4), pp. 335–359.

[81] Gray, W. J., P. P. Wang, and M. Scott (2004) "A Queueing Model with Multiple Types of Server Breakdowns," *Quality Technology & Quantitative Management*, **1**(2), pp. 245–255.

[82] Madan, K. C. (1973) "A priority queueing system with service interruptions," *Statistica Neerlandica*, **27**(3), pp. 115–123.

[83] ——— (1989) "A single channel queue with bulk service subject to interruptions," *Microelectronics Reliability*, **29**(5), pp. 813–818.

[84] ——— (1992) "A bulk queueing system with random failures and two phase repairs," *Microelectronics Reliability*, **32**(5), pp. 669–677.

[85] Altiok, T. (1989) "Queueing modeling of a single processor with failures," *Performance Evaluation*, **9**(2), pp. 93–102.

[86] Tadj, L. and G. Choudhury (2009) "A quorum queueing system with an unreliable server," *Applied Mathematics Letters*, **22**(11), pp. 1710–1714.

[87] Chang, J. and J. Wang (2017), "Unreliable M/M/1/1 retrial queues with set-up time," .

[88] Jayaraman, D., R. Nadarajan, and M. R. Sitrarasu (1994) "A general bulk service queue with arrival rate dependent on server breakdowns," *Applied Mathematical Modelling*, **18**(3), pp. 156–160.

[89] Li, Q.-L., J.-Y. Ma, R.-N. Fan, and L. Xia (2019) "An overview for Markov decision processes in queues and networks," in *International Conference of Celebrating Professor Jinhua Cao's 80th Birthday*, Springer, pp. 44–71.

[90] Jo, K. and O. Maimon (1991) "Optimal dynamic load distribution in a class of flow-type flexible manufacturing systems," *European journal of operational research*, **55**(1), pp. 71–81.

[91] Wu, C.-H., J. T. Lin, and W.-C. Chien (2010) "Dynamic production control in a serial line with process queue time constraint," *International Journal of Production Research*, **48**(13), pp. 3823–3843.

[92] Okamura, H., S. Miyata, and T. Dohi (2015) "A markov decision process approach to dynamic power management in a cluster system," *IEEE Access*, **3**, pp. 3039–3047.

[93] Crabill, T. B. (1972) "Optimal control of a service facility with variable exponential service times and constant arrival rate," *Management Science*, **18**(9), pp. 560–566.

[94] Lippman, S. A. (1975) "On dynamic programming with unbounded rewards," *Management Science*, **21**(11), pp. 1225–1233.

[95] George, J. M. and J. M. Harrison (2001) "Dynamic control of a queue with adjustable service rate," *Operations research*, **49**(5), pp. 720–731.

[96] Kumar, R., M. E. Lewis, and H. Topaloglu (2013) "Dynamic service rate control for a single-server queue with Markov-modulated arrivals," *Naval Research Logistics (NRL)*, **60**(8), pp. 661–677.

[97] Lu, F. and R. F. Serfozo (1984) "M/M/1 queueing decision processes with monotone hysteretic optimal policies," *Operations Research*, **32**(5), pp. 1116–1132.

[98] Hipp, S. K. and U. D. Holzbaur (1988) "Decision processes with monotone hysteretic policies," *Operations Research*, **36**(4), pp. 585–588.

[99] Carrizosa, E., E. Conde, and M. Munoz-Marquez (1998) "Admission policies in loss queueing models with heterogeneous arrivals," *Management Science*, **44**(3), pp. 311–320.

[100] Ghosh, A. P. and A. P. Weerasinghe (2007) "Optimal buffer size for a stochastic processing network in heavy traffic," *Queueing Systems*, **55**(3), pp. 147–159.

[101] ——— (2010) "Optimal buffer size and dynamic rate control for a queueing system with impatient customers in heavy traffic," *Stochastic processes and their applications*, **120**(11), pp. 2103–2141.

[102] Dimitrakopoulos, Y. and A. Burnetas (2017) "The value of service rate flexibility in an M/M/1 queue with admission control," *IISE Transactions*, **49**(6), pp. 603–621.

[103] Wang, K.-H. (2003) "Optimal control of a removable and non-reliable server in an M/M/1 queueing system with exponential startup time," *Mathematical methods of operations research*, **58**(1), pp. 29–39.

[104] Ke, J.-C. (2004) "Bi-level control for batch arrival queues with an early startup and un-reliable server," *Applied Mathematical Modelling*, **28**(5), pp. 469–485.

[105] ——— (2006) "An M/G/1 queue under hysteretic vacation policy with an early startup and un-reliable server," *Mathematical Methods of Operations Research*, **63**(2), p. 357.

[106] Hsu, W.-K., J. Xu, X. Lin, and M. R. Bell (2018) "Integrating Online Learning and Adaptive Control in Queueing Systems with Uncertain Payoffs," in *2018 Information Theory and Applications Workshop (ITA)*, IEEE, pp. 1–9.

[107] Johari, R., V. Kamble, and Y. Kanoria (2016) "Matching while learning," *arXiv preprint arXiv:1603.04549*.

[108] Liu, B., Q. Xie, and E. Modiano (2019) "Reinforcement Learning for Optimal Control of Queueing Systems," in *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, IEEE, pp. 663–670.

[109] Ross, S. M. (2014) *Introduction to probability models*, Academic press.

[110] He, Q.-M. (2014) *Fundamentals of matrix-analytic methods*, vol. 365, Springer.

[111] Ata, B. (2005) "Dynamic power control in a wireless static channel subject to a quality-of-service constraint," *Operations Research*, **53**(5), pp. 842–851.

[112] Badian-Pessot, P., M. E. Lewis, and D. G. Down (2019) "Optimal control policies for an M/M/1 queue with a removable server and dynamic service rates," *Probability in the Engineering and Informational Sciences*, pp. 1–21.

Juxihong Julaiti

juxihongjulaiti1225@gmail.com
814-321-1711

LinkedIn Profile
Website
Languages: Python, Java

Education

The Pennsylvania State University (PSU) Aug/2015 - Dec/2021
Dual Title Ph.D. in Industrial Engineering and Operations Research GPA: 3.78/4.0

The Pennsylvania State University Jan/2015 - Dec/2016
M.S. in Industrial Engineering GPA: 3.78/4.0

Capital University of Economics and Business (CUEB) Aug/2010-Jul/2014
B.E. in Industrial Engineering GPA: 3.74/4.0

Experience

Data Scientist, TE Connectivity June/2020 - Present

- Developed a simulation-based safety stock optimization model that saves $1M inventory cost.
- Developed a control panel for the safety stock optimization model to help users understand the tool and conduct what-if analysis.
- Optimized a recommendation model and saved 1,800 hours of computational time per year.
- Developed a supervised model to predict stamping machine failures with an AUC score of 90%.
- Developed both front-end and back-end of an anomaly detection model for a plating process and saved cost of poor quality by 23%.
- Developed mathematical models to optimize an existing anomaly detection solution and improved the model performance by 10%.

Instructor, PSU IE405: Deterministic models in operations research Jan/2020 - May/2020

Data Scientist, CPNET May/2019 - Aug/2019

- Developed an agnostic simulation package for manufacturing processes using an LSTM model.
- Developed a control optimization algorithm and improved KPI of a chemical factory by 20%.
- Developed an algorithm to optimize hyper-parameters tuning for reinforcement learning methods and saved computational cost by 80%.

ACI Software Specialist , Institute for Computational and Data Sciences, PSU Jan/2018 - Dec/2018

- Built Singularity container images for scientific projects.
- Optimized and automated clients' code using GNU parallel and CronJobs.
- Developed a classification model to predict the topic of unlabeled tickets with an accuracy of 80%.

Data Scientist Co-op, TE Connectivity May/2017 - Dec/2017

- Developed a mathematical model of sequence-dependent parallel-machine scheduling problems.
- Developed reinforcement learning models (DDQN) to solve the scheduling problem.
- Designed the architecture and data pipelines for deploying the solution on AWS.
- Developed a GUI that calls corresponding modules deployed on AWS.

Teaching Assistant, PSU

- Advanced Engineering Analytics, Department of Industrial Engineering Aug/2019 - Dec/2019
- Seminar of Science BS/MBA Program, Smeal College of Business Jan/2018 - May/2018