9-2023

Dynamic scheduling with uncertain job types

Zuo-Jun Max SHEN
University of California - Berkeley

Jingui XIE
Technische Universitat Munchen

Zhichao ZHENG
Singapore Management University, DANIELZHENG@smu.edu.sg

Han ZHOU
Technische Universitat Munchen

Citation

SHEN, Zuo-Jun Max; XIE, Jingui; ZHENG, Zhichao; and ZHOU, Han. Dynamic scheduling with uncertain job types. (2023). *European Journal of Operational Research*. 309, (3), 1047-1060.
Available at: https://ink.library.smu.edu.sg/lkcsb_research/7180

Dynamic Scheduling with Uncertain Job Types

Zuo-Jun Max Shen
Department of Industrial Engineering and Operations Research, University of California, Berkeley, CA 94720-1777, maxshen@berkeley.edu

Jingui Xie
School of Management, Technical University of Munich, 74072 Heilbronn, Germany, jingui.xie@tum.de

Zhichao Zheng
Lee Kong Chian School of Business, Singapore Management University, Singapore 178899, danielzheng@smu.edu.sg

Han Zhou*
School of Management, Technical University of Munich, 74072 Heilbronn, Germany, han.zhou@tum.de

Uncertain job types can arise as a result of predictive or diagnostic inaccuracy in healthcare or repair service systems and unknown preferences in matching service systems. In this paper, we study systems with multiple types of jobs, in which type information is imperfect and will be updated dynamically. Each job has a prior probability of belonging to a certain type which may be predicted by data, models, or experts. A job can only be processed by the right machine, and a job assigned to the wrong machine must be rescheduled. More information is learned from the mismatch, and job type probabilities are updated. The question is how to dynamically schedule all jobs so that they can be processed in a timely fashion. We use a novel coupling and inductive method to conduct optimality analysis. We obtain the near-optimal policy regarding completion time, named the less-uncertainty-first policy, when there are two types of jobs; the insights it yields are used to develop heuristic algorithms for more general cases. We also consider other objectives, including the number of mismatches and the total amount of time jobs spend in the system. In our numerical study, we examine the performance of the proposed heuristics when there are more than two types of jobs under two learning schemes: dedicated learning and exclusive learning. In the extension, we also analyze the problem when jobs are assigned online and find similar insights. It is essential that managers dynamically schedule services by leveraging predictive information and mismatches. Our proposed less-uncertainty-first policy can be used to improve system efficiency in a variety of contexts, because the policy accounts for system dynamics to avoid mismatches and resource idling.

Key words: Scheduling; Uncertain Job Types; Predictive Information; Learning; Mismatch and Rescheduling

* Corresponding author: H. Zhou, han.zhou@tum.de

1. Introduction

Service and manufacturing systems with multiple types of jobs under *perfect information*—i.e., the job types are known exactly before making scheduling decisions—have been studied extensively during the past 60 years (e.g., **???????**). However, in many manufacturing or service systems, job or customer types are uncertain due to *imperfect predictive information*, and scheduling decisions must be made before knowing the exact types. For example, an emergency health care response to a mass casualty incident, such as a hurricane, tsunami, terrorist attack, etc., may involve medical personnel who have a range of capabilities and skills for treating patients with trauma, burns, crush, respiratory impact, submersion injury or infected wounds, etc. Under such conditions, because the numbers of providers and patients on the scene are relatively large, patient treatment can be performed by separate medical teams with different skill sets in parallel (**?**). Patient assessment and classification, however, may not be accurate due to limited information or time and resource constraints (**?**). The question is then how to dynamically assign patients with uncertain conditions to different health care providers and learn patients' true conditions after wrong assignments so that all patients can be treated promptly.

Uncertain job types can also be the result of diagnostic inaccuracy in many other healthcare settings (**?????**). For example, **?** reviewed diagnostic accuracy studies from 1946 to 2016 and found that diagnostic accuracy ranged from 0.32 to 1. Out of the eleven studies reviewed, **?** further selected eight studies that were deemed to have a low risk of bias and conducted a meta-analysis that showed a diagnostic accuracy of 0.74 (0.62 to 0.82). In repair service systems, it is usually uncertain which component causes the product to be faulty (**?**), which leads to uncertain job types. Faulty products must be dynamically assigned to service engineers with different expertise for diagnosis and repair so that all jobs can be completed as soon as possible. Uncertain job types also arise naturally in many matching service systems. On online dating websites such as eHarmony, the service providers usually suggest matches from a pool of registered women and men when their preferences are not exactly known (**?**). Any unsatisfied customers will return to the system after a trial period. From these failures, service providers can learn and update their information and seek to propose better matches in the future.

More specifically, as above, we can consider a case in the mass-casualty scenario. An emergency healthcare response to such a mass-casualty incident may involve medical personnel with various capabilities and skills for treating patients with different conditions. In such emergency healthcare settings, we can assume that all patients and physicians are available initially. Patients' condition type probabilities are available after triage, and then they are assigned to a physician. If there is a wrong assignment, the patient will be transferred to other physicians, and his or her condition type probability will be updated. The main goal is to treat all patients in a timely fashion.

Motivated by these examples, in this paper we study a scheduling problem with multiple types of jobs in which the information on the job types is imperfect. We explicitly consider the process of dynamic learning and rescheduling after mismatches, unlike prior literature, in which jobs types are associated with different cost implications (e.g., different waiting costs) and mismatches do not require rescheduling (e.g., **???**). To isolate the effect of type uncertainty, we consider a stylized model in which there is no other source of uncertainties, and the decision maker knows beforehand the probability that each job will belong to a certain type. Since job types are uncertain, mismatches always occur. When a mismatch arises, we update the job type probabilities and consider when and how to assign the job to another machine. We explicitly capture such dynamics in our model and demonstrate that this significantly complicates the problem through a series of examples, even without other sources of uncertainty. To the best of our knowledge, this paper is the first to consider learning and rescheduling after mismatches in the scheduling problem with imperfect information on job types. We characterize the optimal policy to minimize the expected number of mismatches and propose a scheduling rule called the less-uncertainty-first (LUF) policy to minimize the expected completion time of the last job and the expected total amount of time that all jobs will spend in the system. We prove the near optimality of the LUF policy when there are two types of jobs and exploit insights obtained from analyzing special cases to design appropriate algorithms for more complicated problems. In addition, we analyze an online version of the problem in which job information is revealed sequentially. We propose three policies and derive their competitive ratios against the stochastic offline adversary model in which all jobs' type probabilities and processing times are given in advance. Our numerical result again shows the trade-off between idling and mismatch when designing algorithms. Our analysis and counterexamples reveal important insights regarding the challenges of the problem, which will shed some light on future research in this direction.

The paper's main contributions of this paper are as follows.

- First, we introduce imperfect information on job type in the scheduling problem and explicitly capture the dynamic rescheduling process, motivated by the diagnostic and predictive inaccuracy in service and manufacturing operations.

- Second, we characterize the optimal policy to minimize the expected number of mismatches and prove the near optimality of the LUF policy to minimize both the expected completion time of the last job and the expected total amount of time that all jobs spend in the system when there are two types of jobs.
- We also demonstrate numerically that the insights from our analytical results for special cases can be generalized to develop heuristic algorithms that perform well in a variety of settings when the number of job types is more than two and when the jobs are assigned online.

The paper is organized as follows. In Section 2, we review the related literature. In Section 3 we formulate the model and, in Section 4, characterize different scheduling policies. We present the numerical analysis of the proposed heuristic algorithms for more general settings with multiple job types in Section 5, discuss the online setting and analyze three policies in Section 6, and conclude in Section 7. All the technical proofs are relegated to Appendix A.

2. Literature Review

Our problem is closely related to classical job shop scheduling problems, in which the literature is generally divided into two streams: online and offline scheduling. In an offline scheduling problem, all job information, especially job release times, is given at the beginning so that scheduling decisions can be made by taking the job information into account. On the other hand, in an online scheduling problem, job information is broken into pieces and gradually presented to the decision maker; this can even include the existence of the job. Scheduling decisions must be made on the go whenever new information is available. Some recent developments in stochastic online scheduling problems in which each job's completion time is uncertain include work by **??** and **?**. In our problem, we first assume that at the beginning, all jobs have given probabilities to be certain types. In this sense, our problem can be viewed as an offline scheduling problem. However, our problem is inherently different due to the rich dynamics involved: Not only are the scheduling decisions dynamic, but the service sequence for each job is also uncertain and depends on the scheduling decisions. In other words, the policy depends on the revealed scenario, which is completely adaptive to the process. This makes our problem much harder than the flow shop problem in the conventional scheduling literature, in which each job must be processed by a series of machines. In addition, we discuss the stochastic online model, i.e., once a job is released, the scheduler knows the information about the job. For a comprehensive overview of these scheduling problems, see **?** and **?**.

In a related vein, scheduling for queueing systems with multiple types of jobs has been studied extensively during the past 60 years. Among those studies, the well-known $c\mu$-rule has proved to be optimal under a variety of conditions. That is, if type i jobs have service rate μ_i and waiting cost c_i per unit of time, the rule to minimize the expected total waiting cost is to process the type of jobs with the highest $c_i\mu_i$ first. This scheduling rule appears to have first been studied by **?** under a deterministic setting; **?** extended it to multi-class M/G/1 queues. **?** analyzed the discrete time version of the $c\mu$ rule, and **?** further extended it to generalized $c\mu$-rule under convex cost functions. Since **?**, this body of work has included **??????**, and **?**, to name a few. Unlike these articles, we introduce the imperfect information on job types into scheduling systems and explicitly model the dynamics of mismatch and rescheduling.

In particular, our problem is closely related to healthcare stochastic scheduling. In the healthcare scheduling literature, our problem is related to the well-known appointment sequencing problem in which service times are uncertain and the smallest-variance-first (SVF) rule has been widely conjectured to be optimal (cf. **?**, **?**, and **?**). Under the SVF rule, the job with the smallest variance in service duration is scheduled to be the first to arrive in the system, followed by the one with the second smallest variance, and so on. Recently, **?** have shown that the SVF rule is optimal in the worst-case distribution when only the means and variances of the service durations are known under some technical conditions. **?** provided an analysis of cases in which the SVF rule may not be optimal under some conditions and presented several counterexamples in more general settings. For more comprehensive and updated reviews of this line of research, see **????????**, and **?**. While this literature may not explicitly consider the existence of a correct job "type", part of the uncertainty in processing time is likely due to the need to perform numerous tests, essentially to determine the correct job type. The primary goal of our work and the literature is similar: How to optimize the allocation of resources given the incomplete information on jobs (either uncertainty regarding service time or job type). Several recent studies also discuss the uncertainty entailed in diagnosing repair needs in maintenance operations. **?** provide an example at MTU AeroEngines, which is a leading engine maintenance provider in Germany. Workers' main jobs are to diagnose an engine's problem and decide whether to repair or replace expensive parts. A similar example in aircraft maintenance, in which engine repair requires disassembling and reassembling engines, is discussed in **?**. Other examples can be found in remanufacturing processes in which returned parts are of uncertain quality (Guide and Wassenhove 2001).

The problem with imperfect job type information has gained more interest recently due to its applications in various domains, including healthcare and maintenance. **?** study assignment priority under imperfect information on job type identities. They considered a service system in which jobs are one of two possible types, and customers' type identities are not directly available to the service provider. However, each job provides a signal, which equals the probability that the job belongs to a particular type. The service provider uses these signals to determine priority levels for jobs with the objective of minimizing the long-run average waiting cost. **?** also consider the possibility of misclassification in the context of patient prioritization in emergency departments. **?** study a congested system in which the service provider conducts a sequence of imperfect tests to determine the job's type. They analyze how to manage this accuracy/congestion trade-off dynamically and find that the service provider should continue to perform the diagnosis as long as her current belief falls into an interval that depends on the congestion level and the number of tests performed thus far. **?** study the priority scheduling of patients with unknown types and consider a service system with finite patients, all available at time zero, who belong to one of two possible types. Each type is characterized by its waiting cost and expected service time. Jobs' type identities are initially unknown, but the service provider has the option to spend some time investigating to determine a job's type, albeit with the possibility of making an incorrect determination. The objective is to identify policies that balance the time spent on information extraction and the time spent on service. **?** study a similar but more general scheduling problem, in which the service provider has to serve a collection of jobs that have a priori uncertain attributes and must decide how to dynamically allocate resources between testing (diagnosing) jobs, to learn more about their respective uncertain attributes, and processing jobs.

Similar to these recent studies, we also consider imperfect job type information as an important feature in operations. However, our work differs from these studies in the following respects.

Key trade-off. Both **?** and **?** consider the trade-off between exploration (testing or diagnosis) and exploitation (service or treatment), which has also been studied in the context of revenue management and supply chain management (see, e.g., **??**). In contrast, we consider the trade-off between load balance (among multiple servers) and service mismatch. **?** study the exploration aspect (diagnostic service) and dynamically manage the accuracy/congestion trade-off, whereas we focus on the exploitation aspect and dynamically manage the utilization/mismatch trade-off.

Multiple types. Both **?** and **?** considered the single server model, in which the server has the skills needed to both classify or test jobs and serve the jobs. The question is how to manage testing and service dynamically. The single-server assumption is appropriate when the resource is scarce, e.g., when there is only a single physician on an isolated battlefield who must simultaneously classify injuries and deliver proper treatment. However, as discussed by **?**, a typical emergency response to a mass casualty event may involve a number of medical personnel who have a range of capabilities. Therefore, a multiserver model may be more appropriate in real settings, which **?** also suggest is an important future research direction. When there are multiple servers, a group of servers (e.g., triage nurses) may conduct the classification separately and the other group of servers may focus on delivering service. **?** study diagnostic service and argue that there is a trade-off between diagnostic accuracy and system congestion. In other words, it is suboptimal for the server to spend sufficient time to generate accurate diagnoses. Given this observation, we study scheduling service under inaccurate job type information. We study the model with multiple servers and explicitly characterize the near-optimal policy for the two-server case.

Inaccurate information. Different from the problems studied previously, in our problem incomplete job type information is given at the beginning. This means that an imperfect classification has been carried out for all jobs separately. In **?** and **?**, complete information is disclosed after testing, while in **?** diagnosis is done separately and the diagnosis is inaccurate. Our problem is to dynamically schedule the service under diagnostic inaccuracy, i.e., uncertain type probabilities.

Rescheduling. Another key difference between our problem and the papers discussed above, with respect to job type uncertainty, is that we require that a mismatched job be rescheduled such that it can leave the system only after it has been processed by the right type of machine. However, in previous papers the consequence of incorrect determination of job type is handled by cost-based approaches without introducing further dynamics into the system, i.e., a mismatched job carries a penalty either in terms of either longer processing time or higher waiting/processing cost but still leaves the system without returning to it. Our analysis shows that such dynamic movement in jobs creates a significant challenge with respect to making sound scheduling decisions, even in the simplest setting.

3. Model Formulation

In this section we first consider a simplified model with two types of jobs and later extend the discussion to multiple types in Section 5. Let $\mathcal{N} = \{1, 2, \dots, n\}$ denote the set of jobs and $\mathcal{M} = \{1, 2\}$ the set of machines. For each job $i \in \mathcal{N}$, its type independently follows a Bernoulli distribution with parameter p_i, i.e., the job has a probability of p_i being type 1, and $(1 - p_i)$ being type 2. A type j job can only be served by machine j. Suppose job i is type j. If the job is assigned to machine j, the job will leave the system after being served. If instead the job is referred to machine j' such that $j' \neq j$, a mismatch occurs and the job will have to be rescheduled to machine j. Processing times differ depending on whether the assignment is correct or incorrect. Without loss of generality, we assume the processing time is one unit for the wrong assignment and the processing time for a right assignment, i.e., service time, is $\tau \geq 1$ units, which indicates that it takes no less time to serve a job than to detect a mismatch.

In this study, we introduce two objectives and find policies that optimize at least one of them. The objective may be to minimize the completion time of the last job, or the manager may want to minimize the number of mismatches. Instead of optimizing one of these objectives, we aim to find a policy that can balance the performance measures.

3.1. Mismatch

If every job is assigned to the machine with a higher type probability, the expected total number of mismatches given a set of jobs $\mathcal{N} = \{1, 2, \ldots, n\}$ is minimized. In contrast, when all jobs are assigned to the machines with lower type probabilities, the expected number of mismatches is maximized. Therefore, the expected number of mismatches has a lower bound,

$$\sum_{i=1}^{n} (1-p_i)\mathbb{1}_{\{p_i \geq 0.5\}} + p_i \mathbb{1}_{\{p_i < 0.5\}},$$

and an upper bound,

$$\sum_{i=1}^{n} (1-p_i)\mathbb{1}_{\{p_i < 0.5\}} + p_i \mathbb{1}_{\{p_i \geq 0.5\}}.$$

3.2. Makespan

The objective may be to minimize the completion time of the last job, which is called *makespan*.

Let C_π be the makespan and C_π^j be the makespan on machine $j \in \mathcal{M}$. By definition, we have $C_\pi = \max\{C_\pi^j\}$. According to Jensen's inequality, we know that the expected makespan is longer than the expected makespan of any machine, i.e.,

$$\mathbb{E}[C_\pi] \geq \max_{j \in \mathcal{M}} \{\mathbb{E}[C_\pi^j]\}. \tag{1}$$

Then inequality (1) provides a lower bound of the expected makespan, which is frequently used in the following performance analysis.

The performance measure is directly or indirectly related to the system's workload, which has two parts: service-related workload and mismatch-related workload. For each machine, the expected service workload is inherent. If one machine has more "right" jobs to serve than the other machine, there exists an unbalanced workload in the system: One machine is busy, and the other may be idle for some time. The service workload does not depend on the assignment and scheduling policy.

The expected number of type 1 jobs is $n_1 := \sum_{i=1}^{n} p_i$, while the expected number of type 2 jobs is $n_2 := \sum_{i=1}^{n}(1-p_i)$. Hence, the expected service workload for machine 1 is $n_1\tau$, while the expected service workload for machine 2 is $n_2\tau$. If all jobs could be processed correctly at the first time, we have the expected makespan $\tau \max\{n_1, n_2\}$ which is the lower bound of any policies.

4. Policy Analysis

Given the system workload analysis, we know that a good policy should be able to decrease the number of mismatches and, at the same time, utilize idle machines to balance workload. In this section, we first analyze two typical deterministic policies that might be used in practice and then propose our dynamic policy. By deterministic policies, we mean the policies that fix job-to-machine assignments at the beginning; a job will be rescheduled to another machine only if the initial assignment is a mismatch. On the other hand, dynamic policies do not fix the assignments at the beginning; instead, an assignment will be determined dynamically when a machine becomes idle.

4.1. Equal-distribution Policy

Given n jobs and two machines, a natural policy that considers load balance or fairness is to divide all the jobs into two equal groups. Assume that $p_1 \geq \cdots \geq p_n$ and n is an even number for easier expression. Thus, the first $n/2$ jobs are assigned to machine 1, and the rest are assigned to machine 2. If a job is found to be the wrong type, it will be transferred to the other machine and join the queue in the last position. The policy is called the equal-distribution (ED) policy and is illustrated in Figure 1 when $n=6$ and $\boldsymbol{p} = (0.9, 0.8, 0.7, 0.6, 0.4, 0.2)$.

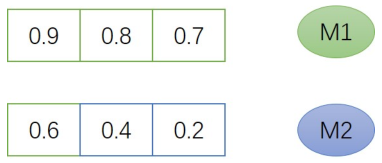


Figure 1 **Illustration of the equal-distribution policy**

Under the ED policy, the expected number of mismatches for machine 1 is $\sum_{i=1}^{n/2}(1-p_i)$, while the expected number of mismatches for machine 2 is $\sum_{i=n/2+1}^{n} p_i$. Hence, the total expected number of mismatches $\mathbb{E}[M_{ED}]$ is given by

$$\mathbb{E}[M_{ED}] = \sum_{i=1}^{\frac{n}{2}}(1-p_i) + \sum_{i=\frac{n}{2}+1}^{n} p_i.$$

The expected makespan $\mathbb{E}[C_{ED}]$ is bounded by

$$\mathbb{E}[C_{ED}] \geq \max\{\mathbb{E}[C_{ED}^1], \mathbb{E}[C_{ED}^2]\} \geq \max\{n_1\tau + \sum_{i=1}^{\frac{n}{2}}(1-p_i), n_2\tau + \sum_{i=\frac{n}{2}+1}^{n} p_i\}.$$

4.2. Likelihood-based Policy

An alternative method is to process each job in order from the most likely machine to the less likely machine, i.e., divide jobs into two groups according to their type likelihood: Jobs with probabilities larger than or equal to 0.5 will be assigned to machine 1, while jobs with probabilities less than 0.5 will be assigned to machine 2. We can refine the policy by assigning jobs with probabilities equal to 0.5 more carefully. But this is not the focus of this paper, and it only works when there are many jobs with probabilities equal to 0.5.[1] If a job is found to be the wrong type, it will be transferred to the other machine and join the queue in the last position. This is the likelihood-based (LB) policy and is illustrated in Figure 2 when $n = 6$ and $\boldsymbol{p} = (0.9, 0.8, 0.7, 0.6, 0.4, 0.2)$. The main objective of this policy is to minimize the expected number of mismatches. If job i is assigned to machine 1, the expected number of mismatches is $1 - p_i$. Similarly, if job i is assigned to machine 2, the expected number of mismatches is p_i. If we assign a job to the machine with a type probability larger than 0.5, the expected number of mismatches will be minimized for the job. Thus, the total expected number of mismatches is minimized. A general formal statement is provided later in Theorem 4.

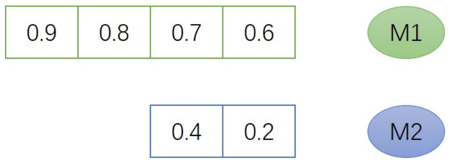


Figure 2 **Illustration of the likelihood-based policy**

Under the LB policy, the expected number of mismatches for machine 1 is $\sum_{i=1}^{n}(1-p_i)\mathbb{1}_{\{p_i \geq 0.5\}}$, while the expected number of mismatches for machine 2 is $\sum_{i=1}^{n} p_i \mathbb{1}_{\{p_i < 0.5\}}$. Hence, the total number of mismatches $\mathbb{E}[M_{LB}]$ is given by

$$\mathbb{E}[M_{LB}] = \sum_{i=1}^{n}(1-p_i)\mathbb{1}_{\{p_i \geq 0.5\}} + p_i \mathbb{1}_{\{p_i < 0.5\}}.$$

[1] For example, jobs with probabilities equal to 0.5 can be assigned based on the principle of keeping the initial queues of both machines as balanced as possible. Specifically, for each job with a probability equal to 0.5, the job can be assigned to the machine with fewer assigned jobs to process. If the numbers of jobs to be served from both machines are the same, the job will be assigned randomly. It should be noted that the assignment of jobs with probabilities equal to 0.5 is performed at the beginning of service.

The expected total number of mismatches is minimized under this policy. The expected makespan $\mathbb{E}[C_{LB}]$ is bounded by

$$\mathbb{E}[C_{LB}] \geq \max\{\mathbb{E}[C^1_{LB}], \mathbb{E}[C^2_{LB}]\} \geq \max\{n_1\tau + \sum_{i=1}^{n}(1-p_i)\mathbb{1}_{\{p_i \geq 0.5\}}, n_2\tau + \sum_{i=1}^{n} p_i \mathbb{1}_{\{p_i < 0.5\}}\}.$$

4.3. The Less-uncertainty-first Policy

Under the above policies, initial job assignments are determined before any job is processed. As the uncertain unfolds as the jobs are processed, the queues for the two machines may become imbalanced, resulting in a less ideal split of workload and even the idling of one machine. In this section, we consider a group of dynamic policies that only determine a job allocation when a machine becomes idle. Specifically, we propose *priority list policies*, which are easy to implement and control. Under a priority list policy, jobs are sorted on a priority list. At the beginning, the first job on the list is assigned to machine 1 and removed from the list; meanwhile, the last job on the list is assigned to machine 2 and removed from the list. If the first job is revealed to be type 1, it leaves the system after one period. Otherwise, we know its type, and the job is returned to the last place on the list. In other words, it will gain the highest priority to be scheduled on machine 2. The same procedure applies to the last job that is assigned to machine 2. The policy is non-idling unless all remaining jobs in the system are of known types (i.e., with probability 1) for one machine. In this case, it is possible that the other machine will be idle. Whenever machine 1 finishes a job, it takes the first job from the list; similarly, whenever machine 2 finishes a job, it takes the last job from the list. The process continues until all jobs are finished. When the initial priority list is sorted according to the probability of being type 1, this is called the less-uncertainty-first (LUF) policy and is illustrated in Figure 3. In case there is only one job left and both machines are idle, it is assigned to the machine that is more likely to be the right type.

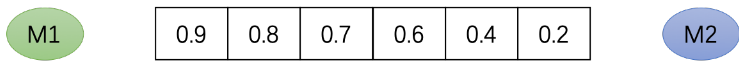


Figure 3 Illustration of the less-uncertainty-first (LUF) policy

Intuitively, we might think that a non-idling policy that processes the job with the least uncertainty (or highest matching probability) at any time whenever a machine is idle (i.e., the LUF policy) should be optimal in terms of makespan. However, the intuition fails to lead to the optimal policy. To illustrate, we present two counterexamples in Appendix B when $\tau = 1$. The insights gained from the counterexamples are crucial for analyzing and bounding the LUF policy's performance. These examples demonstrate one of the key challenges of the problem, whereby the optimal policy needs to consider many details and interactions among all the remaining jobs in the system. Furthermore, it becomes impossible to prove that the LUF policy is optimal given the counterexamples. Therefore, in what follows, we restrict our attention to the set of priority list policies, denoted Π^P, and analyze the best priority list policy.

For analytical tractability, we consider a special case in which $\tau = 1$; i.e., it takes one period to process a job regardless of whether the assignment is correct or not. This case fits settings in which the machine can not detect the job type before service completion. For example, in healthcare settings, when a patient is admitted to a hospital, she is treated according to clinical guidelines and pathways. It is unknown whether the patient could be cured until completion of the service. If the patient does not recover, she may be transferred to another hospital (**?**). Based on insights from analysis of special cases, we design a heuristics policy for more general problems, which performs well in our numerical studies.

Note that the LUF rule is generally not optimal even among Π^P. In particular, the problems in the above examples arise when only one job with an undetermined type is left in the system and both machines are idle. Hence, proving the optimality of any priority list policy becomes nontrivial, since the size of the sample space expands exponentially depending on the procedure of the last operation. However, based on insights from the examples, we are able to show a near-optimality result for the LUF policy.

Theorem 1. *The gap between the expected makespan from the LUF policy and that from any optimal priority list policy is at most 1.*

To prove Theorem 1, we first prove that the LUF policy is optimal among all priority list policies under Assumption 1 stated below, which "speeds up" the last step. This assumption is inspired by the counterexamples presented in Appendix B. Then we know that the gap between the expected makespan from the LUF policy and that from the optimal priority list policy is at most 1 by relaxing the assumption.

Assumption 1. *We assume that when only one job is left and both machines are idle, the two machines can process the final job together and finish it within one period.*

First, one can easily verify that under this assumption, the LUF rule will be optimal for the counterexamples in Appendix B. Next, we introduce some more notation before proving the optimality of the LUF rule under this assumption. Since there are only two machines, we can simplify the job type probabilities to p_i, which denotes the probability that job i is type 1 (i.e., p_{i1} in our previous notation). Then the probability that job i belongs to type 2 is $(1-p_i)$. With a little abuse of notation, we use x_i to represent the possible realization of job i's type, i.e., $x_i \in \{1,2\}$. Without loss of generality, let the sequence of jobs in the priority list be $(1,2,\dots,n)$, and recall that the priority list policy will assign the first job in the priority list to machine 1 and the last job to machine 2 in every period. Note that any priority list policy is completely characterized by its priority list. We prove later that the optimal priority list policy under Assumption 1 should have the property of $p_1 \geq p_2 \geq \cdots \geq p_n$. Define $X = x_1 x_2 \dots x_n$ as a realization of job types for the n jobs on the list, which is a string of ordered n 1-2 characters; for example, $X = 12\dots1$. We call X a scenario of these n jobs in an ordered priority list. Since all of the job type uncertainties are mutually independent of each other, the probability of a scenario $X = x_1 x_2 \dots x_n$ is

$$\begin{aligned} p_X &:= \textstyle\prod_{i=1}^n \left[p_i \mathbb{1}_{\{x_i=1\}} + (1-p_i)\mathbb{1}_{\{x_i=2\}}\right] \\ &= \textstyle\prod_{i=1}^n \left[p_i(2-x_i) + (1-p_i)(x_i-1)\right] \\ &= \textstyle\prod_{i=1}^n \left(3p_i - 2x_i p_i + x_i - 1\right). \end{aligned} \tag{2}$$

For example, if $X = 11\dots1$, then the probability of this scenario is $\prod_{i=1}^n p_i$. If $X = 22\dots2$, then the probability of this scenario is $\prod_{i=1}^n (1-p_i)$. Denote the cardinality of a scenario X as $|X|$; then $|X| = n$ for $X = \{x_1, x_2, \dots, x_n\}$. For any given scenario X, let C_X be the makespan of this scenario under the priority list policy. Note that C_X is not a random variable. Then the expected makespan for n jobs is $(p_1, p_2, \dots, p_n)$ is $\sum_X p_X C_X$. To prove the optimality of the LUF policy, we first establish the following two lemmas.

LEMMA 1. *Under Assumption 1, $C_{12X} \leq C_{21X}$ for any scenario X.*

Here, with a little abuse of notation, C_{12X} represents the makespan of a scenario in which the actual types of the first two jobs on the priority list are type 1 and type 2, respectively; X indicates the scenario of the remaining $n-2$ jobs. To better understand the above lemma and clarify the notation, we illustrate the case in detail using two jobs. There are two jobs of unknown types in the system, and they are ordered as job 1 and job 2 on the priority list. In the first scenario, the actual type of job 1 is type 1 and the actual type of job 2 is type 2, which is represented by scenario 12 in the notation C_{12}. According to the priority list policy, job 1 will be assigned to machine 1, and job 2 will be assigned to machine 2. Then both jobs will be successfully processed in one period and leave the system, so $C_{12} = 1$. In the other scenario, the actual type of job 1 is type 2, and the actual type of job 2 is type 1, which is represented by scenario 21 in the notation C_{21}. However, the decision maker does not know the exact types of the jobs and assigns job 1 to machine 1 and job 2 to machine 2 according to the priority list policy. After one period, the mismatches are realized and both jobs are returned to the list with the sequence changed, which becomes C_{12}. Next, both jobs will be assigned to the right machines and successfully processed within one period, so we have $C_{21} = 2$.

The case of the two jobs presented above is rather straightforward and does not require Assumption 1. However, to show that $C_{12X} \le C_{21X}$ for any X is not trivial. This requires careful analysis of all possible cases, and Assumption 1 plays a critical role in establishing this result, which clears the boundary case when there is only one job left in the system. It is not straightforward to think of Assumption 1, which happens to address several obstacles when we try to prove this lemma and several other results. Next, we show a similar result for the order of the last two jobs.

Lemma 2. *Under Assumption 1, $C_{X12} \le C_{X21}$ for any scenario X.*

Here, C_{X12} denotes the makespan of a scenario in which that actual type of the last job is type 2 and the actual type of the penultimate job is type 1. Note that Lemma 2 can be derived from Lemma 1. We can reverse the priority lists $12X$ and $21X$ into $X21$ and $X12$, and assign the first job on the list to machine 2 and the last job to machine 1. According to Lemma 1, $C_{X21} \le C_{X12}$. Then by swapping the notations for machine 1 and machine 2, we get $C_{X12} \le C_{X21}$ when the first job on the list is assigned to machine 1, and the last job on the list is assigned to machine 2.

Note that Lemmas 1 and 2 hold for any possible scenario X, i.e., any realization of job types. The probabilities of the scenarios have not yet been taken into consideration in these lemmas. Therefore, there is no additional condition related to probabilities for Lemmas 1 and 2. Next, based on these two lemmas, we consider the probabilities of different possible scenarios (realizations of job types) and prove Theorem 2, which says that the optimal priority list should sort p_i's in decreasing order. In other words, the LUF policy is optimal among all priority list policies under Assumption 1, which leads to our main result whereby the gap between the LUF policy and any optimal priority list policy is at most one.

Theorem 2. *Under Assumption 1, the LUF policy is optimal among all the priority list policies, in terms of minimizing the expected makespan.*

Note that Assumption 1 only "speeds up" the last job in some scenarios. Hence, the impact of this assumption on the makespan is bounded above by 1. That is, the expected makespan of any priority list policy will be increased by less than one period without this assumption. Hence, if the LUF policy beats any other priority list policy under Assumption 1, its performance gap from the optimal priority list policy without Assumption 1 is also bounded above by one. Therefore, we prove our main result, Theorem 1.

Another typical objective in the scheduling literature is to minimize the expected total amount of time jobs spend in the system, i.e., the expected total sojourn time, defined as

$$T^{\pi} = \sum_{t} X^{\pi}(t), \tag{3}$$

where $X^{\pi}(t)$ is the number of unprocessed jobs in the system at period t under a scheduling policy π, which is the summation of the sojourn time for each job. The analysis of this objective is similar to the case of minimizing the expected makespan. Therefore, we only state the main results below for two machines (i.e., $m = 2$) but relegate the details to Appendix A.3.

THEOREM 3. *The gap between the expected total sojourn time from the LUF policy and that from the optimal priority list policy is less than one period.*

As shown above, the LUF policy works well under both objectives, minimizing the expected makespan and the expected total sojourn time when there are two machines in the system. Such superior performance of the LUF policy comes from its properties: 1) *reducing mismatch*—the LUF policy assigns jobs to maximize the overall matching probability; 2) *utilizing idle machines*—when a machine is idle, it will help process a job even if the job is more likely to be the other type. It is worth mentioning that it is very challenging to extend the analytical results to more general cases, due to mismatch and rescheduling and because the policy is dynamic. Nevertheless, this simplified case's insights guide us in developing easy-to-implement heuristic policies for general problems.

5. Multiple Types with Learning: A Numerical Study

In this section, we numerically discuss the model with multiple types of jobs. For two types, learning job types is rather simple; i.e., type 1 or type 2. However, when there are more than two types, the learning process is much more complicated and not unique. For example, for complicated diseases with highly uncertain treatments, the information learned from a mismatch is limited for diagnosing the true disease types. Thus, patients will likely visit two or more physicians until the correct treatment is found. However, for mild diseases, the true type can be found after one mismatch because the information gained from diagnostic tests ordered by any given physician may be enough to confirm the disease type. Based on this idea, we discuss two learning processes as follows.

- Exclusive learning (or Bayesian learning): If there is a mismatch, the only thing we learn is to exclude this type and update the remaining probabilities.
- Dedicated learning (or fast learning): If there is a mismatch, the machine will detect the true type of the job.

We can see that the above learning schemes are at two extremes. Exclusive learning is a situation in which after a mismatch, little information is learned except for the exclusion of a mismatched type. For dedicated learning, in contrast, the actual job type can be learned after a mismatch. There is no difference between exclusive and dedicated learning when there are only two job types. In practice, other situations can exist between these two extremes; e.g., learning partial information after mismatch, which will not be discussed in detail here. However, we believe that our results could be applied to partial learning schemes if similar patterns are observed under both exclusive and dedicated learning.

In the following, we first formally introduce the model and these two learning schemes. Then, based on the previous analysis, we introduce the benchmark policy and propose our heuristic policy. Finally, we study the performance of our proposed policy in settings of both homogeneous service time and heterogeneous service time, which may cover various scenarios, including mass-casualty incidents. The results show that our proposed policy performs better than the benchmark in minimizing completion time, with a small sacrifice on the objective of minimizing mismatch. An application in a mass-casualty scenario is described in Appendix C.

5.1. Model Formulation and Learning Schemes

We consider a system with n jobs and m machines. Let $\mathcal{N}=\{1,2,\dots,n\}$ denote the set of jobs and $\mathcal{M}=\{1,2,\dots,m\}$ the set of machines. For each job $i\in\mathcal{N}$, the job has an initial probability p_{ij} of being type j, $j\in\mathcal{M}$. A type j job can only be served by machine j. We assume that the time needed to detecting a mismatch is one period[2]. Each job spends a random amount of service time before being successfully processed by the machine. We assume a discrete-time setting and the service time of a type j job τ_j follows a geometric distribution with the parameter q_j, i.e.,

$$\mathbb{P}(\tau_j=k)=(1-q_j)^{k-1}q_j,\ k\geq 1.$$

[2] In our model, we normalize the mismatch times to one period under both learning schemes. We focus on comparing different policies under each learning scheme, and we do not intend to compare the two learning schemes. Hence, the actual time unit for one period could be different under different learning schemes. For example, one period for the model under the exclusive learning scheme can be 30 minutes, while one period under the dedicated learning case can be 1 hour. The service times can be scaled, too.

This assumption is fairly commonly used in the literature (e.g., **??**); **?** showed that the empirical length-of-stay distribution for inpatients is close to a geometric distribution. The model under these two learning schemes can be formulated as the following dynamic problem.

State: Let $\mathcal{S}$ be the state space. Define the state $s := ((p_{ij})_{n\times m}, (b_j)_{1\times m}) \in \mathcal{S}$, where p_{ij} represents the probability that job i is type j, $b_j \in \{0,1\}$, where $b_j = 0$ denotes that machine j is idle and $b_j = 1$ denotes that machine j is busy.

Action: Let $\mathcal{A}$ be the action space and $a = (a_{ij})_{n\times m} \in \mathcal{A}$ denotes an action. If $a_{ij} = 1$, job i is assigned to machine j. Otherwise, $a_{ij} = 0$. We assume that the assignment can only occur between jobs on the list and idle machines. Let $\boldsymbol{p_i} = (p_{i1}, p_{i2}, \ldots, p_{im})$ represents the state of job i. We let $\mathcal{N}' = \{i \in \mathcal{N} : \boldsymbol{p_i} = \mathbf{0}\}$ denote the set of jobs not on the list and $\mathcal{M}' = \{j \in \mathcal{M} : b_j \neq 0\}$ the set of busy machines. The state-dependent action space is therefore

$$\mathcal{A}(s) = \left\{ (a_{ij})_{n\times m} : \begin{array}{c} \sum_{j=1}^m a_{ij} \le 1, \forall i \in \mathcal{N}; \sum_{i=1}^n a_{ij} \le 1, \forall j \in \mathcal{M}; \\ \sum_{j=1}^m a_{ij} = 0, \forall i \in \mathcal{N}'; \sum_{i=1}^n a_{ij} = 0, \forall j \in \mathcal{M}'. \end{array} \right\}. \tag{4}$$

State transition probabilities: We consider the state transition probabilities under different learning schemes. Since the service process of each job is independent, we discuss the marginal transition probability for simplicity. The state transition probability can easily be obtained by multiplying these independent marginal transition probabilities. If machine j is busy serving a job, then the machine has a probability of q_j to complete the job in the current period and probability $(1-q_j)$ to continue the service process in the next period. If there is an assignment $a_{ij} = 1$, then job i ($p_{ij} \neq 0$) is assigned to machine j ($b_j = 0$). If the job is the right type and is being served by the machine (with probability p_{ij}), then the job has a probability of q_j to be served in one period and leave the system, and probability $(1-q_j)$ to continue the service in the next period. If the job is found to be a wrong type (with probability $1 - p_{ij}$), then the machine will detect the true type of the job as type $k(\neq j)$ with probability p_{ik} under the dedicated learning. In contrast, under exclusive learning, the machine can only exclude type j and the belief of job type is updated accordingly.

Thus, under dedicated learning, the *marginal* transition probability is given as follows:

$$\mathbb{P}_d(s' \mid s, a) = \begin{cases} p_{ij}q_j, & \text{if } p_{ij} \neq 0, b_j = 0, a_{ij} = 1, \text{ and } \boldsymbol{p}'_i = \mathbf{0}, b'_j = 0; \\ p_{ij}(1-q_j), & \text{if } p_{ij} \neq 0, b_j = 0, a_{ij} = 1, \text{ and } \boldsymbol{p}'_i = \mathbf{0}, b'_j = 1; \\ p_{ik}, & \text{if } p_{ij} \neq 0, b_j = 0, a_{ij} = 1, \text{ and } \boldsymbol{p}'_i = \boldsymbol{e}_k, b'_j = 0, k \neq j; \\ q_j, & \text{if } b_j = 1, \text{ and } b'_j = 0; \\ 1 - q_j, & \text{if } b_j = 1, \text{ and } b'_j = 1, \end{cases} \tag{5}$$

where $s' = ((p'_{ij})_{n\times m}, (b'_j)_{1\times m})$ denotes the state in the next period.

For exclusive learning, the marginal transition probability is the same except

$$\mathbb{P}_e(s' \mid s, a) = 1 - p_{ij}, \text{ if } p_{ij} \neq 0, b_j = 0, a_{ij} = 1, \text{ and } \boldsymbol{p}'_i = (\boldsymbol{p}_i - p_{ij}\boldsymbol{e}_j)/(1 - p_{ij}), b'_j = 0. \tag{6}$$

Objective function: The cost function $c(s,a)$ equals 0 if the system is empty. Otherwise, it is 1. Our objective is to minimize the service completion time of all jobs (i.e., makespan) given the initial state s_1, i.e.,

$$\min_{a_t \in \mathcal{A}(s_t)} \mathbb{E}\left[\sum_{t=1}^{\infty} c(s_t, a_t) \mid s_1\right].$$

Under this model formulation, our problem can be considered a Markov decision problem (MDP) with high dimensional state space. Note that the geometric service time distribution is crucial for the model to be considered an MDP. Computationally, our problem suffers severely from the curse of dimensionality. For the problem with n jobs and m machines, the number of states for this MDP is $(m+2)^n 2^m$ under dedicated learning and even much larger under exclusive learning; for example, when $m = 2$, $n = 5$, the number of states can be 4,096 under dedicated learning. In general, our problem is technically and computationally challenging to solve, and we believe that it is useful to develop heuristics based on our analysis of a simplified model.

5.2. Heuristic Policies

In this part, we first introduce a benchmark policy that always assigns a job to the machine with the highest probability of serving the job. This policy will likely be implemented in practice, since most physicians prefer to treat patients correctly right the first time. However, such a policy is not optimal because system utilization is not taken into consideration. Based on our analysis of the special case we develop our heuristic policy, which accounts for system dynamics to avoid mismatches as well as resource idling. To see the performance of our proposed heuristic policy, we conduct a series of numerical studies.

In systems in which mismatch incurs a considerable cost, the manager may want to minimize the number of mismatches. We generalize the likelihood-based policy in Section 4 to the multiple-type setting, and refer to it as the highest-probability-first (HPF) policy. Specifically, in each period, jobs must be assigned to the machines with the highest probabilities. If more than one job is waiting for the same machine, the machine will first serve the job with the highest probability of being its type. [3] We can easily show that this policy is optimal for minimizing the expected number of mismatches.

[3] If equal probabilities exist, the assignment will be determined randomly. This case is rare if the probabilities are continuously distributed.

Theorem 4. *If jobs are always assigned according to the HPF policy, the expected total number of mismatches is minimized under both learning schemes.*

We set the HPF policy in our numerical analysis as a benchmark to see whether our proposed heuristic policy can achieve a better trade-off between mismatch and makespan.

Based on our analysis of the LUF policy for the simplified model, we propose a generalization, in which the assignment decisions are made by solving the following optimization problem in each period when there are jobs on the list and idle machines:

$$\max_{a\in\mathcal{A}(s)}\sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{M}}p_{ij}a_{ij},$$

where $\mathcal{A}(s)$ is defined in (4).

We refer to this policy as the generalized less-uncertainty-first (GLUF) policy. Note that it reduces to the LUF policy when there are only two machines.

5.3. Numerical Experiments

To study the performance of our proposed GLUF policy, we perform a series of numerical experiments when service time and detection time are either the same or different. We compare the performances of the GLUF policy and the benchmark HPF policy to demonstrate how the GLUF policy can benefit the scheduling problem with uncertain-type jobs under different learning schemes. To further illustrate the applicability of the GLUF policy to general settings, we also compare the performances of the GLUF policy and the optimal policy when the service time is stochastic. We examine three performance measures as discussed above: expected makespan, expected total sojourn time, and expected number of mismatches.

5.3.1. Equal Service and Detection Time To focus on the effect of system capacity and workload, we first conduct numerical experiments in the setting of equal service and detection time, which are both set to one unit without loss of generality. For any job i, p_{ij} is generated uniformly between 0 and 1 ($j=1,\dots,m$) and then normalized. Assume that we have $n=20$ jobs and $m=5$ machines. We generate 100 instances of job type probabilities and 100 samples for each instance. The result is presented in Table 1. Under exclusive learning, for the HPF policy, which is designed to minimize the expected number of mismatches, the average makespan is 13.54; the average total sojourn time is 123.57; the average number of mismatches is 25.70. If we apply the GLUF policy, the average makespan decreases to 10.93, a 19.30% reduction compared with the benchmark policy. There is an 11.78% reduction in the average total sojourn time but a 3.67% increase in the expected number of mismatches. The performance comparison between the GLUF and HPF policies under dedicated learning is similar but more significant. Compared with exclusive learning, there is a higher reduction in makespan and total sojourn time under dedicated learning (22.50% and 11.79%, respectively), while there is less increase in the number of mismatches (1.74%). We can see that although HPF performs better with respect to mismatch, the advantage is small. In contrast, the GLUF policy performs much better with respect to makespan and total sojourn time.

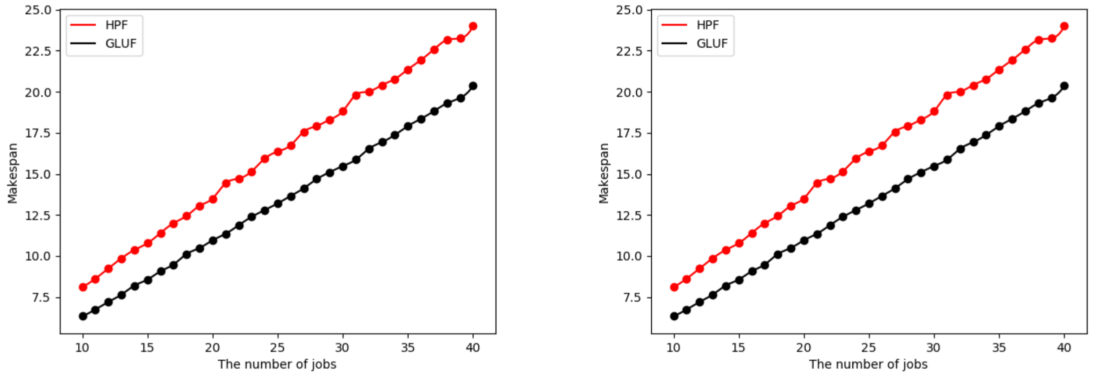
Table 1 **System performance under different policies when $m = 5$ and $n = 20$.**

	Dedicated Learning			Exclusive Learning		
	HPF	GLUF	(gap%)	HPF	GLUF	(gap%)
Makespan	10.09	7.82	(↓22.50%)	13.54	10.93	(↓19.30%)
Total Sojourn Time	94.22	83.12	(↓11.79%)	123.37	108.84	(↓11.78%)
Mismatch	13.02	13.25	(↑1.74%)	25.70	26.64	(↑3.67%)

Effect of the Number of Jobs. To see the effect of system workload on performance comparison, we change the number of jobs, n, from 10 to 40 while keeping the number of machines at $m = 5$. For each n, we generate 100 instances of job type probabilities and 100 samples for each instance. We plot the makespan comparison between the two polices in Figure 4; similar figures on total sojourn time and number of mismatches are provided in Figures 9 and 10 in Appendix D, along with the details reported in Table 5. As the number of jobs increases, performance improvement in terms of makespan and total sojourn time under the GLUF policy relative to the HPF policy is significant: more than 14% for makespan and 6% for total sojourn time. However, performance improvement (proportionally) in terms of makespan decreases. One explanation is that one of the advantages of the GLUF policy is its non-idling feature; i.e., it can better utilize idle machines. When the number of jobs grows, the probability of machines being idle decreases. Therefore, the advantage of the GLUF policy is diminished.

Regarding the number of mismatches, the GLUF policy always performs worse than the HPF policy although the gap is small at less than 6% (see Table 5 in Appendix D). Considering the similar results of two extreme learning schemes in terms of three measures, we believe that partial learning schemes also yield similar results.

Effect of the Number of Job Types. It should be noted that the number of machines equals the number of job types in our setting. To avoid ambiguity, we use the term "job type" in this section. In our setting, on the one hand, as the number of job types increases and as system service capacity increases, the number of jobs being processed simultaneously can be larger; this is beneficial



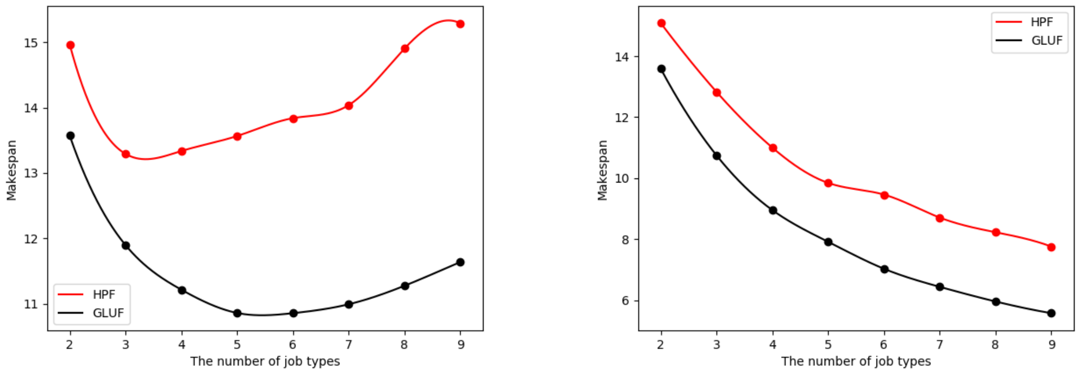
(a) Exclusive learning (b) Dedicated learning

Figure 4 Makespan comparison between GLUF and HPF policies under different numbers of jobs ($m = 5$).

for decreasing system completion time. On the other hand, as the number of job types increases, uncertainty increases, which lead to a higher number of mismatches. There could be a trade-off between type uncertainty and system service capacity. Therefore, it is meaningful to see the effect of the number of job types on performance measure comparison.

We plot the makespan comparison between the two polices in Figure 5; similar figures for total sojourn time and number of mismatches are provided in Figures 11 and 12 in Appendix D, along with the details reported in Table 6. As observed in Figure 5, under exclusive learning, as the number of job types increases, the average makespan drops at the beginning and then rises. One possible explanation is that as the number of job types increases, although it will be faster for jobs to be assigned, the probability of being mismatched can also rise. At first, the impact of service capacity dominates the impact of job type uncertainty. Later, the latter gradually dominates the former. In contrast, for dedicated learning the average makespan is decreasing continuously. Dedicated learning takes at most two steps to serve a job successfully; i.e., there is at most one mismatch for one job. Therefore, the impact of high job-type uncertainty on the number of mismatches as well as makespan is limited, and the influence of service capacity on makespan is remarkable. Hence, there is an overall decrease trend under dedicated learning.

Observation of the total sojourn time measure is similar to makespan. Regarding the measure of mismatch, the performance of the GLUF policy is quite close to the performance of the HPF policy, with a difference of less than 6% (see Table 6 in Appendix D). Since a larger number of job types indicates higher type uncertainty, it is possible that the uncertainty of job types influences performance. The GLUF policy seems to perform better for solving problems in which job types



(a) Exclusive learning (b) Dedicated learning

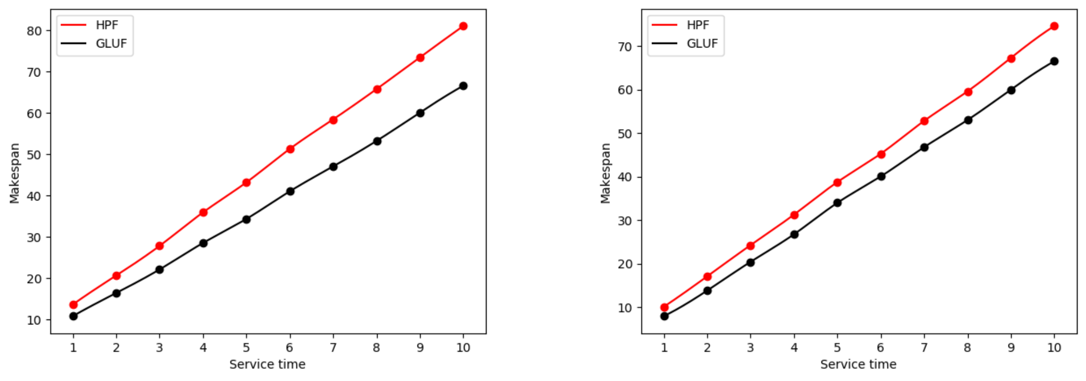
Figure 5 Makespan comparison between GLUF and HPF policies under different numbers of machines ($n = 20$).

are uncertain. This observation affirms our initial motivation for designing the GLUF policy, which is to handle the scheduling problem with job type uncertainty.

5.3.2. Unequal Service and Detection Time In practice, treating a disease can take longer than diagnosing or finding a mismatch. Hence, we now explore a more general case in which the service time differs from the detection time. We first study the situation in which the service time is deterministic and then examine the situation in which the service time is stochastic.

Deterministic Service Time. Assume that we have $n = 20$ jobs and $m = 5$ machines. In this part, we perform a numerical experiment by changing the service time, i.e., the processing time when the assignment is correct. Without loss of generality, we set the time spent on one mismatch as one period and vary the service time from 1 to 10 periods. The complete results are provided in Table 7 in Appendix D. The comparison of average makespan is depicted in Figure 6, and similar comparisons of total sojourn time and number of mismatches are plotted in Figures 13 and 14 in Appendix D. We observe that the average makespan and total sojourn time increase as the service time increases. The number of mismatches under the HPF policy is not affected by the service time, which has been proved in Theorem 4. The average number of mismatches under the GLUF policy rises with the increase in service time. When the service time increases, more machines are occupied in each period and unavailable, so the number of choices for jobs is smaller. Under this situation, the GLUF policy will always greedily make assignments that may not be good enough for jobs. Therefore, the number of mismatches can be large.

The performance improvement in terms of the average makespan under the GLUF policy is significant. For example, when service time equals 10 units, i.e., ten times the detection time, the



(a) Exclusive learning (b) Dedicated learning

Figure 6 Makespan comparison between GLUF and HPF policies under different service times ($m = 5, n = 20$).

performance improvement can be 17.85% under exclusive learning and 9.31% under dedicated learning. To understand this result, we can refer to Figure 14 in Appendix D. As the service time increases, the performance of the GLUF policy in terms of mismatch compared with the HPF policy worsens; perhaps the increasing number of mismatches can explain the reduction in makespan performance improvement. However, the negative impact of a mismatch on system makespan drops as the service time increases, because the relative time needed to identify a mismatch compared with the correct service gets smaller. Therefore, there is still a significant improvement in makespan under the GLUF policy despite its downward trend.

Stochastic Service Time. We numerically compare the GLUF policy with the optimal policy, which minimizes the expected makespan, with the service time following a geometric distribution. As demonstrated in our previous analysis, this problem suffers from the curse of dimensionality when using recursive algorithms. Thus, we limit the experiments to small-size examples. We use the backward recursion algorithm to obtain the optimal makespan, When $m = 2, n = 5$, we set the initial job type probability distributions to (0.2 0.8), (0.3 0.7), (0.9 0.1), (0.6 0.4), and (0.05 0.95) respectively. We vary n from 2 to 5 by choosing the first n jobs from the above example. Suppose the detection time for a mismatch is one period. The expected service time for type 1 job μ_1 is set to two periods and the expected service time for type 2 job μ_2 is varied from 4 to 6 and 8 periods. We generate 100 samples in the simulation under the GLUF and the HPF policy. Comparison of expected makespan under different policies is presented in Table 2, where “Opt” indicates the optimal expected makespan. From Table 2, we can observe that the gap is relatively small, which affirms the efficiency of the proposed GLUF policy. The computational time for $n = 2, 3, 4, 5$ is

0.65s, 18.76s, 788.54s, and 25107.19s, respectively. A case with $m = 3$ is presented in Appendix E to further verify the performance of the GLUF policy. All experiments are performed on a PC with Intel Core i5 2.90 GHz CPU and 8 GB of RAM.

Table 2 **Expected makespan under different policies with service time following geometric distributions ($m = 2$).**

	$(\mu_1 = 2, \mu_2 = 4)$					$(\mu_1 = 2, \mu_2 = 6)$					$(\mu_1 = 2, \mu_2 = 8)$				
n	Opt	GLUF	(gap%)	HPF	(gap%)	Opt	GLUF	(gap%)	HPF	(gap%)	Opt	GLUF	(gap%)	HPF	(gap%)
2	6.48	6.69	3.24%	7.22	11.42%	9.42	9.56	1.49%	9.92	5.31%	12.38	12.41	0.24%	12.88	4.04%
3	7.38	7.59	2.84%	8.05	9.08%	10.43	10.75	3.07%	10.81	3.64%	13.54	13.57	0.22%	14.27	5.39%
4	9.27	9.43	1.73%	10.28	10.89%	13.01	13.40	2.99%	13.55	4.15%	16.87	16.90	0.18%	18.07	7.11%
5	12.43	12.51	0.64%	13.27	6.76%	18.09	18.10	0.06%	19.01	5.09%	23.91	24.21	0.83%	24.66	3.14%

Next, we compare the expected number of mismatches from the GLUF and the HPF policies. The settings are the same as in the above experiment for makespan comparison. Detailed results are presented in Table 8 in Appendix D. We first observe that the mismatch performance of the HPF policy does not depend on service time, which is confirmed in Theorem 4. Under the GLUF policy, the expected number of mismatches increases with μ_2. This is because the longer machine 2 is occupied, the more likely a type 2 job will be assigned to machine 1, which leads to a higher chance of mismatch. We can see a gap in the number of mismatches between these two policies. While the HPF policy minimizes the number of mismatches, the GLUF policy is better for reducing makespan (see Table 2). These observations are consistent with cases in which service times are homogeneous and deterministic. We also perform additional numerical experiments in which the service time follows different distributions. We observe that the GLUF policy always outperforms the HPF policy in terms of makespan. The results are shown in Tables 9–11 in Appendix D.

In summary, we demonstrate numerically that the insights from our analytical results for a simplified model can be generalized to more practical settings. Our proposed GLUF policy performs well with respect to both makespan and mismatch in a variety of settings.

6. Online Stochastic Scheduling: An Extension

In some systems, jobs' information is revealed one by one in sequence. In this section, we consider an extension to a type of online scheduling models, which arise in communication networks (**?**) and many other applications (**??**). A sequence of independent jobs $I = (J_1, J_2, \dots)$ that are ordered on a list must be scheduled irrevocably on one of the machines $\mathcal{M} = \{1, 2\}$ in the order of their arrivals. We denote a job by $J_i = (p_i, \tau_i, T_i)$, where p_i is the probability of being type 1 and $(1 - p_i)$ is the probability of being type 2. τ_i is the service time if the job is scheduled to the right machine;

otherwise, it takes time T_i to detect the mismatch, and the job must be rescheduled to the right machine. We assume $\tau_i > T_i$, which means it takes no more time to detect a mismatch. For instance, if job J_i is of type j and is assigned to machine j, the job will leave the system after service time τ_i. If the job is referred to the other machine j' $(j' \neq j)$, a mismatch occurs and the job will have to be rescheduled to machine j after mismatch time T_i. We perform the assignment of a job once any machine is empty. After the assignment, the next job $J_i = (p_i, \tau_i, T_i)$ can be seen. We continue the assignment until both machines are busy. Next, the machines will process jobs until at least one is empty, and then perform the assignment again. The whole process is complete when both machines are empty and all jobs have been served. The objective is to minimize the expected completion time of the last job, i.e., the makespan. In this section, to avoid confusion, we use "list" to denote the sequence of jobs that have not been assigned, and "queue" refers to jobs that have been assigned and are waiting to be processed by a specific machine.

Due to the handicap of not knowing the entire input, computing the optimal solutions is generally intractable and we have to resort to approximations. The standard approach to evaluating algorithms when working online is competitive analysis. In this study, we propose online policies and measure their competitive ratios against a stochastic offline adversary model. The stochastic offline adversary model can be defined similarly as in Section 3, in which the service time τ_i, time to detect the mismatch T_i, and type probability p_i for each job i are given in advance. Note that the realization of a job's type is still unknown, subject to the randomness following the independent Bernoulli distribution specified by the job's type probability. An online policy A is compared with an optimal stochastic offline policy, denoted OPT. For any input $I_k = (J_1, J_2, \cdots, J_k), k \in \mathbb{N}^+$, let $A(I_k)$ be the makespan achieved by policy A on I_k and let $OPT(I_k)$ be the makespan of the optimal solution for I_k. Due to the uncertainty of job types, $A(I_k)$ and $OPT(I_k)$ are both random variables. An online algorithm is called c-competitive if there exists a constant b such that for any problem input I_k, the inequality $\mathbb{E}[A(I_k)] \leq c \cdot \mathbb{E}[OPT(I_k)] + b$ holds. Note that the competitive ratio (c defined above) is a worst-case performance measure.

6.1. Heuristic Policies and Their Competitive Ratios

Following the insight from the previous analysis, we introduce and analyze three related policies accordingly in this section. First, we also consider the HPF policy as before, which assigns each job to the more likely machine to minimize mismatch, i.e., jobs with probabilities no less than 0.5 will be assigned to machine 1 and jobs with probabilities less than 0.5 will be assigned to machine 2. The following proposition gives an upper bound on the competitive ratio of the HPF policy.

Proposition 1. *The HPF policy achieves a competitive ratio no greater than 2.*

Under the HFP policy, it is possible that a new job is assigned to the busy machine while continuing to leave the empty machine be empty. This assignment may cause unwanted idleness in the system. Hence, to avoid such a situation, we consider a policy that always assigns the new job on the list to the empty machine. Specifically, once a machine becomes empty, we assign the new job to this machine; if both machines become empty at the same time, we assign the job to the machine with a higher matching probability. We refer to this policy as the avoid idling (AI) policy. To analyze the performance of the AI policy, we specifically define the idle time of a machine as the time intervals during which the machine is empty between its busy periods. Under this definition, the final period in which one machine is empty and "waiting" for the other machine to finish processing the remaining assigned jobs is not considered to be the machine's idle time; similarly, the makespan of a machine is the time interval between its first and last busy periods (inclusive). Lemma 3 below gives an upper bound on the total idle time for any machine under the AI policy.

Lemma 3. *The total idle time for any machine under the AI policy is bounded above by* $\max\{T_1, T_2, \dots\}$.

The makespan of each machine contains two parts: total processing time (including service time and detection time for mismatches) and total idle time. Note that for any job, its service time and detection time, if both occur, can only occur in two different machines. Hence, for any specific machine, its total processing time is bounded above by $\sum_{i=1}^{n} \tau_i$, because the detection time is smaller than the service time for any job. Together with the bound on idle time given in Lemma 3, we can derive an upper bound on the makespan of each machine, which is $\sum_{i=1}^{n} \tau_i + \max\{T_1, T_2, \dots, T_n\}$. Since the makespan of the system is the maximum of the makespans of the two machines, the same bound applies to the expected makespan under the AI policy. Therefore, the competitive ratio of the AI policy can be bounded above by

$$\frac{2\sum_{i=1}^{n} \tau_i + 2\max\{T_1, T_2, \dots, T_n\}}{\sum_{i=1}^{n} \tau_i + \sum_{i=1}^{n}(1-p_i)T_i \mathbb{1}_{\{p_i > 0.5\}} + \sum_{i=1}^{n} p_i T_i \mathbb{1}_{\{p_i \le 0.5\}}}.$$

Finally, in attempting to balance the trade-off between mismatch and idling, we consider a policy that achieves less expected makespan when assigning any new job. As before, once any of the machines becomes empty, the system performs the assignment of the first job on the list if not empty. Given the job assignment before the new job, we compute the expected makespan if the new job is assigned to machine 1, which can be approximated by simulation. Specifically, past makespan has been realized at the time of assigning this new job, so we only need to simulate the type of the new job or jobs (if any) that have not been processed to estimate the expected remaining makespan. In our numerical experiments, we generate 100 instances for the job types to compute the approximated makespan. Similarly, we also compute the expected makespan if the new job is allocated to machine 2. The new job will be assigned to the machine that achieves a lower expected makespan for the system. This policy is called the less-expected-makespan first (LEMF) policy. Note that this is a myopic greedy policy that optimizes makespan at each time of assigning a job. Similar to the HPF policy, we can prove the same upper bound on the competitive ratio of the LEME policy.

PROPOSITION 2. *The LEMF policy achieves a competitive ratio no greater than 2.*

6.2. Numerical Study

In this section, we perform several numerical experiments to investigate the performance of our proposed policies in various settings to demonstrate the trade-off between avoiding idling and mismatch. We use the makespan as the main performance measure. In this numerical experiment, job i's type probability p_i is generated from a uniform distribution, $U(0,1)$, or beta distributions, $Beta(2,2)$, and $Beta(0.5,0.5)$. The service time of job i, τ_i is generated from a uniform distribution $U[2,11]$, and its mismatch time T_i is generated from a uniform distribution $U[1,\tau_i]$ to ensure that the mismatch time is always less than the service time for a given job. We consider $m=2$ machines. We obtain the expected makespan under different job type probability distributions and different numbers of jobs n. For each case, we generate 100 numerical samples and run 50 replications for each sample. The results are summarized in Table 3.

Table 3 Average makespan under different job type probability distributions with service time following uniform distribution.

	$U(0,1)$			$Beta(0.5,0.5)$			$Beta(2,2)$		
n	AI	HPF	LEMF	AI	HPF	LEMF	AI	HPF	LEMF
5	22.96	23.63	22.79	21.91	22.44	21.64	22.69	23.69	22.63
10	41.73	42.37	41.19	40.71	41.21	39.99	41.29	42.36	40.95
15	60.26	60.54	59.20	59.99	58.93	58.64	59.22	60.19	58.48
20	79.73	79.67	78.77	78.84	77.57	77.30	78.52	79.76	78.01
25	97.31	96.99	96.89	99.79	97.64	97.42	98.48	99.13	97.25
30	115.86	115.24	115.19	117.58	114.82	114.72	116.37	116.74	115.14

When p_i is generated from $Beta(2,2)$, the expected makespan under the AI policy is less than that under the HPF policy and the gap decreases as n increases. When p_i is close to 0.5, the benefit of assigning a job to the highest probability machine can be small. That is why the AI policy always outperforms the HPF policy. As n increases, both machines become less likely to be idle. Once a machine is idle, a new job will be ready for assignment. Therefore, the benefit of the AI policy could diminish, which explains why the gap decreases as n increases. However, the LEMF policy always performs the best because the policy takes into account the trade-off between idling and mismatch. When p_i is generated from $Beta(0.5, 0.5)$, p_i could be either very small or very large. When $n = 5$ or 10, the AI policy performs better than the HPF policy. However, as n increases, the HPF policy outperforms the AI policy. On the one hand, once the job is assigned to the "right" machine, the possibility of being mismatched is small, which means that the benefit of avoiding mismatch could be large. On the other hand, with an increase in n, the benefit of the AI policy is decreasing. Therefore, the AI policy is better than the HPF policy when n is small and becomes worse when n is large. Similarly, we can observe that the LEMF policy performs the best. Under the uniform distribution $U(0, 1)$, the spread of p_i is neither too extreme nor too centralized. The results show that when $n \leq 15$, the AI policy performs better than the HPF policy. Then, as n increases, the expected makespan under the HPF policy is smaller than that under the AI policy. We can observe that the LEMF policy always performs the best in minimizing the expected makespan. Our result highlights the importance of balancing the trade-off between mismatch and idling.

We further explore other distributions to generate the parameters. In particular, the service time τ_i is generated from geometric distribution $G(2/13)$ or Erlang distribution $Erlang(2, 2/13)$, while the mismatch time T_i is generated from the uniform distribution $U[1, \tau_i]$. The main insights are consistent with what is presented above, and the detailed results are provided in Tables 12 and 13 in Appendix D.

7. Conclusion

In this paper, we study a new scheduling problem with uncertain types of jobs, i.e., the probability that a job belongs to a certain type can be less than 1. We explicitly model the dynamics whereby a mismatched job must be rescheduled so that a job can leave the system only after it has been processed by the right machine, which is required in many service systems, including most healthcare systems. The problem turns out to be much harder, since the service sequence for each job becomes stochastic and depends on the dynamic scheduling decisions. We investigate the optimal policies under different objectives: makespan, total sojourn time, and the number of mismatches. In the numerical study, we find that the GLUF policy performs relatively well under all objectives. The

GLUF policy performs quite well in minimizing makespan by slightly sacrificing the number of mismatches. With a larger number of job types or shorter service time, the GLUF policy performs much better than the benchmark policies. This is because the GLUF policy accounts for system dynamics to avoid both mismatches and resource idling. Our results imply that managers should design policies not only from a clinical perspective to minimize mismatches but also from an operational perspective to balance utilization.

Since this is a preliminary study of the scheduling problems with uncertain job types, there are several limitations. Although we have demonstrated that the LUF policy is optimal when there are two types of jobs under Assumption 1, it is still unclear what the optimal policies are in scheduling systems with multiple job types. This is a challenging future research direction, and the methodology used in this paper may no longer be applicable. Another future research direction would be to analyze the performance of the LUF policy when p_i's have a certain pattern or follow a specific distribution. Given such additional information, it may be possible to derive more analytic results. It would also be interesting to apply the approximate dynamic programming or reinforcement learning to solve the high-dimensional MDP problems (**??**). For example, the temporal-difference learning or Q-learning methods may be feasible for our problem, which is worth exploring.

Online Appendices:

Appendix A: Technical Proofs

A.1. Proof of Lemma 1

We prove the lemma using induction. We start with boundary conditions.

First of all, under Assumption 1, it takes only one period to complete a single job, i.e., $C_1 = C_2 = 1$.

When $|X| = 0$, we have $C_{12} = 1$, and $C_{21} = 1 + C_{12} = 2 > C_{12}$. When $|X| = 1$, we have $C_{121} = 1 + C_{12} = 2$, $C_{211} = 1 + C_{112} = 2 + C_1 = 3 > C_{121}$, and $C_{122} = 1 + C_2 = 2$, $C_{212} = 1 + C_{12} = 2 = C_{122}$.

When $|X| = 2$, we have the following four possible cases:

Case 1. $C_{1211} = 1 + C_{121} = 2 + C_{12} = 3$, $C_{2111} = 1 + C_{1112} = 2 + C_{11} = 3 + C_1 = 4 > C_{1211}$;

Case 2. $C_{1212} = 1 + C_{21} = 2 + C_{12} = 3$, $C_{2112} = 1 + C_{112} = 2 + C_1 = 3 = C_{1212}$;

Case 3. $C_{1221} = 1 + C_{122} = 2 + C_2 = 3$, $C_{2121} = 1 + C_{1122} = 2 + C_{12} = 3 = C_{1221}$;

Case 4. $C_{1222} = 1 + C_{22} = 2 + C_2 = 3$, $C_{2122} = 1 + C_{122} = 2 + C_2 = 3 = C_{1222}$.

Therefore, the lemma holds when $|X| = 0, 1, 2$.

Using induction, to prove Lemma 1, i.e., $C_{12X} \leq C_{21X}$ for any scenario X, assume that the lemma holds for all $|X| \leq k+1$ $(k \geq 1)$. We only need to prove that it still holds for $|X| = k+2$. We need to show the following four cases: $C_{12Y11} \leq C_{21Y11}$, $C_{12Y12} \leq C_{21Y12}$, $C_{12Y21} \leq C_{21Y21}$, and $C_{12Y22} \leq C_{21Y22}$, where $|Y| = k \geq 1$.

Case 1. $C_{12Y11} \leq C_{21Y11}$.

We decompose this case into two subcases: $Y = W1$ and $Y = W2$.

- Subcase 1.1. $Y = W1$.

  $C_{12W111} = 1 + C_{12W11} = 2 + C_{12W1} \leq 2 + C_{21W1} = 3 + C_{11W2} = 4 + C_{1W}$.

  $C_{21W111} = 1 + C_{11W112} = 2 + C_{1W11} = 3 + C_{1W1} = 4 + C_{1W} \geq C_{12W111}$.

- Subcase 1.2. $Y = W2$.

  $C_{12W211} = 1 + C_{12W21} = 2 + C_{12W2} \leq 2 + C_{21W2} = 3 + C_{1W2} = 4 + C_W$.

  $C_{21W211} = 1 + C_{11W212} = 2 + C_{1W21} = 3 + C_{1W2} = 4 + C_W \geq C_{12W211}$.

Case 2. $C_{12Y12} \leq C_{21Y12}$.

We decompose this case into two subcases: $Y = 1W$ and $Y = 2W$.

- Subcase 2.1. $Y = W1$.

  $C_{121W12} = 1 + C_{21W1} = 2 + C_{11W2} = 3 + C_{1W}$.

  $C_{211W12} = 1 + C_{11W12} = 2 + C_{1W1} = 3 + C_{1W}$.

  When $|W| = 0$, $C_{121W12} = C_{211W12} = 4$. When $|W| \geq 1$, $C_{1W} = C_{1W}$, so $C_{121W12} = C_{211W12}$.

- Subcase 2.2. $Y = W2$.

  $C_{122W12} = 1 + C_{22W1} = 2 + C_{12W2} \leq 2 + C_{21W2} = 3 + C_{1W2} = 4 + C_W$.

  $C_{212W12} = 1 + C_{12W12} = 2 + C_{2W1} = 3 + C_{1W2} = 4 + C_W \geq C_{122W12}$.

Case 3. $C_{12Y21} \leq C_{21Y21}$.

$C_{12Y21} = 1 + C_{12Y2} \leq 1 + C_{21Y2} = 2 + C_{1Y2} = 3 + C_Y$.

$C_{21Y21} = 1 + C_{11Y22} = 2 + C_{1Y2} = 3 + C_Y \geq C_{12Y21}$.

Case 4. $C_{12Y22} \le C_{21Y22}$.

$C_{12Y22} = 1 + C_{2Y2} = 2 + C_{Y2}$.

$C_{21Y22} = 1 + C_{1Y22} = 2 + C_{Y2}$.

Since $|Y| \ge 1$, $C_{Y2} = C_{Y2}$. Thus, $C_{21Y22} = C_{12Y22}$.

This completes the proof.

A.2. Proof of Theorem 2

Assume we have a job list with probabilities $p_1, p_2, \dots, p_n$. If there exist two consecutive jobs $(j-1)$ and j such that $p_{j-1} < p_j$, we show that the expected makespan can be reduced if we switch these two jobs.

Note that before the machines process job $(j-1)$ or j, the makespan is the same for any coupled sample paths. Thus, we only need to compare the makespan between $(j-1, j, \dots)$ and $(j, j-1, \dots)$, or between $(\dots, j-1, j)$ and $(\dots, j, j-1)$. We show the former case here, and the latter one follows a similar argument. Let X be a realization (scenario) of the job types of those unprocessed jobs after $j-1$ and j on the list, with probability p_X. We have

$$\begin{aligned}
&\mathbb{E}[C_{(p_j, p_{j-1}, \dots)}] - \mathbb{E}[C_{(p_{j-1}, p_j, \dots)}] \\
=& \sum_X [p_j(1-p_{j-1})p_X(C_{12X} - C_{21X}) + p_{j-1}(1-p_j)p_X(C_{21X} - C_{12X})] \\
=& (p_j - p_{j-1}) \sum_X p_X(C_{12X} - C_{21X}) \le 0.
\end{aligned}$$

The last inequality is due to Lemma 1 that $C_{12X} \le C_{21X}$ and the fact that $p_j > p_{j-1}$.

The above analysis shows that switching job $(j-1)$ with job j when $p_{j-1} < p_j$ decreases the expected makespan.

Repeating such an operation, the job with the highest probability will be the first job on the list. All jobs will be re-ordered according to their probabilities of being type-1 from high to low. The process works exactly the same as the idea in the bubble sort. In this way, we show that the LUF policy is optimal among all the priority list policies under Assumption 1, in the sense that it minimizes the expected makespan.

A.3. Proof of Theorem 3

We prove Theorem 3 following the same steps as Theorem 1. We first provide two lemmas. Lemma 4 below says that if the first two jobs in the list happen to be type-2 and type-1, then by switching these two jobs, the total sojourn time can be reduced. Lemma 5 says that if the last two jobs in the list are type-2 and type-1, then by switching these two jobs, the total sojourn time can be reduced. Following the same notation defined in the analysis of expected makespan, for any given scenario X, let T_X be the total sojourn time of this scenario. Then the expected total sojourn time under policy is $\sum_X p_X T_X$.

Lemma 4. *Under Assumption 1, $T_{12X} \le T_{21X}$ for any scenario X.*

Proof. We prove the lemma using induction. We start with boundary conditions.

First of all, under Assumption 1, the sojourn time of a single job satisfies $T_1 = T_2 = 1$.

When $|X| = 0$, we have $T_{12} = 2$, $T_{21} = 2 + T_{12} = 4 > T_{12}$.

When $|X| = 1$, we have $T_{121} = 3 + T_{12} = 5$, $T_{211} = 3 + T_{112} = 6 + T_1 = 7 > T_{121}$, and $T_{122} = 3 + T_2 = 4$, $T_{212} = 3 + T_{12} = 5 > T_{122}$.

When $|X| = 2$, we have the following four possible cases:

Case 1. $T_{1211} = 4 + T_{121} = 7 + T_{12} = 9$, $T_{2111} = 4 + T_{1112} = 8 + T_{11} = 10 + T_1 = 11 > T_{1211}$;

Case 2. $T_{1212} = 4 + T_{21} = 6 + T_{12} = 8$, $T_{2112} = 4 + T_{112} = 7 + T_1 = 8 = T_{1212}$;

Case 3. $T_{1221} = 4 + T_{122} = 7 + T_2 = 8$, $T_{2121} = 4 + T_{1122} = 8 + T_{12} = 10 > T_{1221}$;

Case 4. $T_{1222} = 4 + T_{22} = 6 + T_2 = 7$, $T_{2122} = 4 + T_{122} = 6 + T_2 = 7 = T_{1222}$.

Therefore, the lemma holds when $|X| = 0, 1, 2$.

Assume that the lemma holds for all $|X| \le k+1$ ($k \ge 1$). We prove that it still holds for $|X| = k+2$. We need to show the following four cases: $T_{12Y11} \le T_{21Y11}$, $T_{12Y12} \le T_{21Y12}$, $T_{12Y21} \le T_{21Y21}$, and $T_{12Y22} \le T_{21Y22}$, where $|Y| = k \ge 1$.

Case 1. $T_{12Y11} \le T_{21Y11}$.

We decompose this case into two subcases: $Y = W1$ and $Y = W2$.

- Subcase 1.1. $Y = W1$.

  $T_{12W111} = (k+4) + T_{12W11} = (k+4) + (k+3) + T_{12W1} \le (k+4) + (k+3) + T_{21W1} = (2k+7) + (k+2) + T_{11W\dot{2}} = (3k+9) + (k+2) + T_{1W} = (4k+11) + T_{1W}$.

  $T_{21W111} = (k+4) + T_{11W112} = (k+4) + (k+4) + T_{1W11} = (2k+8) + (k+2) + T_{1W1} = (3k+10) + (k+1) + T_{1W} = (4k+11) + T_{1W} \ge T_{12W111}$.

- Subcase 1.2. $Y = W2$.

  $T_{12W211} = (k+4) + T_{12W21} = (k+4) + (k+3) + T_{22W2} \le (k+4) + (k+3) + T_{21W2} = (2k+7) + (k+2) + T_{1W2} = (3k+9) + (k+2) + T_W = (4k+11) + T_W$.

  $T_{21W211} = (k+4) + T_{11W212} = (k+4) + (k+4) + T_{1W21} = (2k+8) + (k+2) + T_{1W2} = (3k+10) + (k+1) + T_W = (4k+11) + T_W \ge T_{12W211}$.

Case 2. $T_{12Y12} \le T_{21Y12}$.

We decompose this case into two subcases: $Y = 1W$ and $Y = 2W$.

- Subcase 2.1. $Y = W1$.

  $T_{121W12} = (k+4) + T_{21W1} = (k+4) + (k+2) + T_{11W2} = (2k+6) + (k+2) + T_{1W} = (3k+8) + T_{1W}$.

  $T_{211W12} = (k+4) + T_{11W12} = (k+4) + (k+3) + T_{1W1} = (2k+7) + (k+1) + T_{1W} = (3k+8) + T_{1W}$.

  When $|W| = 0$, $T_{121W12} = T_{211W12} = (3k+9)$. When $|W| \ge 1$, $T_{1W} = T_{1W}$, so $T_{121W12} = T_{211W12}$.

- Subcase 2.2. $Y = W2$.

  $T_{122W12} = (k+4) + T_{22W1} = (k+4) + (k+2) + T_{12W2} \le (k+4) + (k+2) + T_{21W2} = (2k+6) + (k+2) + T_{1W2} = (3k+8) + (k+1) + T_W = (4k+9) + T_W$.

  $T_{212W12} = (k+4) + T_{12W12} = (k+4) + (k+3) + T_{2W1} = (2k+7) + (k+1) + T_{1W2} = (3k+8) + (k+1) + T_W = (4k+9) + T_W \ge T_{122W12}$.

Case 3. $T_{12Y21} \le T_{21Y21}$.

$T_{12Y21} = (k+4) + T_{12Y2} \le (k+4) + T_{21Y2} = (k+4) + (k+3) + T_{1Y2} = (2k+7) + (k+2) + T_Y = (3k+9) + T_Y$.

$T_{21Y21} = (k+4) + T_{11Y22} = (k+4) + (k+4) + T_{1Y2} = (2k+8) + (k+2) + T_Y = (3k+10) + T_Y > T_{12Y21}$.

Case 4. $T_{12Y22} \leq T_{21Y22}$.

$T_{12Y22} = (k+4) + T_{2Y2} = (k+4) + (k+2) + T_{Y2} = (2k+6) + T_{Y2}$.

$T_{21Y22} = (k+4) + T_{1Y22} = (k+4) + (k+3) + T_{Y2} = (2k+7) + T_{Y2}$.

Since $|Y| \geq 1$, $T_{Y2} = T_{Y2}$. Thus, $T_{21Y22} > T_{12Y22}$.

This completes the proof.

Following the same argument of proving Lemma 2 using Lemma 1, we can prove the following lemma using Lemma 4.

LEMMA 5. *Under Assumption 1, $T_{X12} \leq T_{X21}$ for any scenario X.*

Now, we are ready to prove the results of Theorem 5, which immediately implies Theorem 3.

THEOREM 5. *Under Assumption 1, the LUF policy is optimal among all the priority list policies, in terms of minimizing the expected total sojourn time.*

Proof. Assume we have a job list with probabilities $p_1, p_2, \dots, p_n$. If there exist two consecutive jobs $(j-1)$ and j such that $p_{j-1} < p_j$, we show that the expected total sojourn time can be reduced if we switch these two jobs.

Note that before the machines process job $j-1$ or j, the total sojourn time is the same for any coupled sample paths. Thus, we only need to compare the total sojourn time between $(j-1, j, \dots)$ and $(j, j-1, \dots)$, or between $(\dots, j-1, j)$ and $(\dots, j, j-1)$. We show the former case here, and the latter one follows a similar argument. Let X be a realization (scenario) of the job types of those unprocessed jobs after $j-1$ and j on the list, with probability p_X. We have

$$\begin{aligned}
&\mathbb{E}[T_{(p_j, p_{j-1}, \dots)}] - \mathbb{E}[T_{(p_{j-1}, p_j, \dots)}] \\
= &\sum_X p_j(1-p_{j-1}) p_X (T_{12X} - T_{21X}) + p_{j-1}(1-p_j) p_X (T_{21X} - T_{12X}) \\
= &(p_j - p_{j-1}) \sum_X p_X (T_{12X} - T_{21X}) \leq 0.
\end{aligned}$$

The last inequality is due to Lemma 4 and the fact that $p_j > p_{j-1}$.

The above analysis shows that switching job $(j-1)$ with job j when $p_{j-1} < p_j$ decreases the expected total sojourn time. Repeating such an operation, the job with the highest probability will be the first job on the list. All jobs will be re-ordered according to their probabilities of being type-1 from high to low. The process works exactly the same as the idea in the bubble sort. In this way, we show that the LUF policy is optimal among all the priority list policies under Assumption 1, in the sense that it minimizes the expected total sojourn time.

Based on Theorem 5, we know that without Assumption 1, the gap between the expected total sojourn time from the LUF policy and that from the optimal priority list policy is less than one period. Therefore, we complete the proof of Theorem 3.

A.4. Proof of Theorem 4

We first prove the theorem under dedicated learning. For job i, suppose its service sequence is $(m_1, m_2, \ldots, m_m)$, which denote an ordering of $\{1, 2, \ldots, m\}$. Given this service sequence, we will first assign job i to machine m_1. Note that job i's probabilities of being type $m_1, m_2, \ldots, m_m$ are $p_{im_1}, p_{im_2}, \ldots, p_{im_m}$, respectively. Then with probability p_{im_1}, job i will be successfully served by machine m_1, and the number of mismatches of job i is 0. Otherwise, there is a mismatch, and machine m_1 will detect the job's true type and send job i to the right machine so that the number of mismatches of job i is 1. Denote the number of mismatches of job i as M_i, then the expected number of mismatches of job i under the service sequence$(m_1, m_2, \ldots, m_m)$ is

$$\mathbb{E}\left[M_i \middle| (m_1, \ldots, m_m)\right] = 1 - p_{im_1}.$$

Clearly, if we design the service sequence according to p_{ij} from highest to lowest for job i, then $\mathbb{E}[M_i]$ is minimized. Under the HFP policy, the service sequences of the jobs are independent. Therefore, if the expected number of mismatches is minimized for each job, the expected number of mismatches in the system is minimized.

Then, we prove the theorem under exclusive learning. Similarly, for job i, suppose its service sequence is $(m_1, m_2, \ldots, m_m)$, which denote an ordering of $\{1, 2, \ldots, m\}$. Given this service sequence, if job i belongs to type m_1, there will be no mismatch; if job i belongs to type m_2, the number of mismatches will be 1; and so on. Thus, the expected number of mismatches of job i is

$$\mathbb{E}\left[M_i \middle| (m_1, \ldots, m_m)\right] = \sum_{k=2}^{m} (k-1)\, p_{im_k}.$$

Clearly, if we design the service sequence according to p_{ij} from highest to lowest for job i, then $\mathbb{E}[M_i]$ is minimized. Under the HFP policy, the service sequences of the jobs are independent. Therefore, if the expected number of mismatches is minimized for each job, the expected number of mismatches in the system is minimized.

A.5. Proof of Proposition 1

Before proving the proposition, we present a lower bound on the expected optimal offline makespan. For an input $I_k = (J_1, J_2, \ldots, J_k)$, the total expected processing time of all jobs includes the correct service time $\sum_{i=1}^{k} \tau_i$ and mismatch time, which is at least $\sum_{i=1}^{k}(1-p_i)T_i \mathbb{1}_{\{p_i > 0.5\}} + \sum_{i=1}^{k} p_i T_i \mathbb{1}_{\{p_i \le 0.5\}}$. A lower bound on the expected optimal offline makespan can be the total processing service time divided by 2, i.e.,

$$\mathbb{E}[OPT(I_k)] \ge L_{I_k} \triangleq \frac{\sum_{i=1}^{k} \tau_i + T_i \left[\sum_{i=1}^{k}(1-p_i)\mathbb{1}_{\{p_i > 0.5\}} + \sum_{i=1}^{k} p_i \mathbb{1}_{\{p_i \le 0.5\}}\right]}{2}.$$

We prove the property using induction. First of all, we consider the assignment of one job, J_1, i.e., $I_1 = (J_1)$. According to the HPF policy, the job will be assigned to the machine with a success matching probability greater than 0.5. The expected makespan of this job satisfies $\mathbb{E}[HPF(I_1)] = 2L_{I_1} \le 2\mathbb{E}[OPT(I_1)]$.

By induction, we assume $\mathbb{E}[HPF(I_k)] \le 2\mathbb{E}[OPT(I_k)]$ for all job sequence $I_k = (J_1, J_2, \ldots, J_k)$. We need to show the inequity holds for $I_k = (J_1, J_2, \ldots, J_k, J_{k+1})$, i.e., $\mathbb{E}[HPF(I_{k+1})] \le 2\mathbb{E}[OPT(I_{k+1})]$. We first prove the following induction property for any scheduling policy.

Lemma 6. *For any policy A, if $\mathbb{E}[A(I_k)]/L_{I_k} \le 2, \forall k \le n-1$, and $\mathbb{E}[A(I_{k+1})] - \mathbb{E}[A(I_k)] \le \tau_{k+1} + T_{k+1}[(1-p_{k+1})\mathbb{1}_{\{p_{k+1}>0.5\}} + p_{k+1}\mathbb{1}_{\{p_{k+1}\le 0.5\}}]$, then, $\mathbb{E}[A(I_{k+1})]/L_{I_{k+1}} \le 2$.*

Proof. According to the definition of L_{I_k}, we have

$$L_{I_{k+1}} - L_{I_k} = \frac{\tau_{k+1} + T_{k+1}\left[(1-p_{k+1})\mathbb{1}_{\{p_{k+1}>0.5\}} + p_{k+1}\mathbb{1}_{\{p_{k+1}\le 0.5\}}\right]}{2}.$$

Since $\mathbb{E}[A(I_k)]/L_{I_k} \le 2$, it follows that

$$\frac{\mathbb{E}[A(I_k)] + 2(L_{I_{k+1}} - L_{I_k})}{L_{I_k} + (L_{I_{k+1}} - L_{I_k})} \le 2.$$

Given the assumption in the lemma, $\mathbb{E}[A(I_{k+1})] \le \mathbb{E}[A(I_k)] + 2(L_{I_{k+1}} - L_{I_k})$, we have

$$\frac{\mathbb{E}[A(I_{k+1})]}{L_{I_{k+1}}} \le 2.$$

This completes the proof of the lemma. □

According to Lemma 6, it suffices to prove that $\mathbb{E}[HPF(I_{k+1})] - \mathbb{E}[HPF(I_k)] \le \tau_{k+1} + T_{k+1}[(1-p_{k+1})\mathbb{1}_{\{p_{k+1}>0.5\}} + p_{k+1}\mathbb{1}_{\{p_{k+1}\le 0.5\}}]$. At the time to assign job J_{k+1}, which we denote as a_{k+1}, we have the following three possible cases. Denote the remaining makespan for the first k jobs at time a_{k+1} as r_k, which is equivalent to $HPF(I_k) - a_{k+1}$.

Case 1. Two machines are both empty. Under the HPF policy, we choose machine j so that the probability of job J_{k+1} belonging to type j is larger than 0.5. In this case, the expected makespan will be increased by $\tau_{k+1} + T_{k+1}[(1-p_{k+1})\mathbb{1}_{\{p_{k+1}>0.5\}} + p_{k+1}\mathbb{1}_{\{p_{k+1}\le 0.5\}}]$.

Case 2. One machine is empty, and the other is busy; the type(s) of the job(s) being processed by or waiting for the busy machine is (are) known, i.e., there will not be any mismatch in the busy machine. Without loss of generality, suppose machine 1 is empty, and machine 2 is busy.

- Subcase 2.1. When $p_{k+1} \ge 0.5$, job J_{k+1} is assigned to the empty machine, i.e., machine 1.
  - —With probability $(1-p_{k+1})$, J_{k+1} is mismatched, and it will be rescheduled to machine 2 to process after a duration of T_{k+1}. If $T_{k+1} > r_k$, job J_{k+1} will be immediately processed by machine 2 for a duration of τ_{k+1}. If $T_{k+1} \le r_k$, machine 2 will be busy when job J_{k+1} is rescheduled to machine 2. Job J_{k+1} has to wait for a duration of $r_k - T_{k+1}$, and it will be processed for a duration of τ_{k+1}. The increased makespan can be written as $\max\{T_{k+1}, r_k\} + \tau_{k+1} - r_k$.
  - —With probability p_{k+1}, job J_{k+1} is successfully processed for a duration of τ_{k+1}. The increased makespan is $\max\{\tau_{k+1} - r_k, 0\}$.

  Therefore, in this subcase, the expected makespan will be increased by

$$\begin{aligned}
&(1-p_{k+1})\left(\max\{T_{k+1}, r_k\} + \tau_{k+1} - r_k\right) + p_{k+1}\max\{\tau_{k+1} - r_k, 0\}\\
=&\tau_{k+1} + (1-p_{k+1})\max\{T_{k+1} - r_k, 0\} + p_{k+1}\max\{\tau_{k+1} - r_k, 0\} - p_{k+1}\tau_{k+1}\\
=&\tau_{k+1} + (1-p_{k+1})\max\{T_{k+1} - r_k, 0\} + p_{k+1}\max\{-r_k, -\tau_{k+1}\}\\
\le&\tau_{k+1} + (1-p_{k+1})T_{k+1},
\end{aligned}$$

  where the last inequality follows since $r_k, T_{k+1}, \tau_{k+1} \ge 0$.

- Subcase 2.2. When $(1-p_{k+1})>0.5$, job J_{k+1} is assigned to the busy machine, i.e., machine 2.
  - With probability $(1-p_{k+1})$, job J_{k+1} is processed successfully for a duration of τ_{k+1}.
  - With probability p_{k+1}, job J_{k+1} is mismatched, and it will be rescheduled to machine 1 to process after a duration of T_{k+1}. Since machine 1 is empty, job J_{k+1} will be processed immediately for a duration of τ_{k+1}.

  Therefore, in this subcase, the expected makespan increases by $(1-p_{k+1})\tau_{k+1}+p_{k+1}(T_{k+1}+\tau_{k+1})=\tau_{k+1}+p_{k+1}T_{k+1}$.

Combing the two subcases, the increase in expected makespan is bounded by $\tau_{k+1}+T_{k+1}[(1-p_{k+1})\mathbb{1}_{\{p_{k+1}>0.5\}}+p_{k+1}\mathbb{1}_{\{p_{k+1}\leq 0.5\}}]$.

Case 3. One machine is empty, and the other is busy; the type(s) of (some) job(s) being processed by or waiting for the busy machine is (are) uncertain, i.e., there will probably be (some) mismatch(es) in the busy machine. Without loss of generality, suppose machine 1 is empty, and machine 2 is busy. There are two possible scenarios in this case, depending on which machine processes the last job in I_k or, in other words, works until time $HPF(I_k)=a_{k+1}+r_k$.

- Subcase 3.1. Suppose it is the busy machine at time a_{k+1}, i.e., machine 2, that works until time $HPF(I_k)$.
  - Subsubcase 3.1.1. When $p_{k+1}\geq 0.5$, job J_{k+1} is assigned to the empty machine at time a_{k+1}, i.e., machine 1.
    - With probability $(1-p_{k+1})$, job J_{k+1} is mismatched, and it will be rescheduled to machine 2 to process after a duration of T_{k+1}. Note that the process of job J_{k+1} may disrupt the process of some jobs in I_k if they are rescheduled to machine 1 after time a_{k+1}. However, since machine 2 is the one that works until time $HPF(I_k)$, machine 1's working duration will be increased by at most T_{k+1}. For machine 2, its working duration will be increased by $\max\{T_{k+1},r_k\}+\tau_{k+1}-r_k$, which is the same as Subcase 2.1 before. It is obvious that the increased working duration on machine 2 is more than that on machine 1 as $\tau_{k+1}>T_{k+1}$. Thus, the system's makespan will be increased by $\max\{r_k,T_{k+1}\}+\tau_{k+1}-r_k$.
    - With probability p_{k+1}, job J_{k+1} is successfully processed for a duration of τ_{k+1}. The makespan will be increased by at most τ_{k+1}.

    Therefore, in this subsubcase, the expected makespan will be increased by at most

$$\begin{aligned}&(1-p_{k+1})\left(\max\{T_{k+1},r_k\}+\tau_{k+1}-r_k\right)+p_{k+1}\tau_{k+1}\\ =&\tau_{k+1}+(1-p_{k+1})\max\{T_{k+1}-r_k,0\}\\ \leq&\tau_{k+1}+(1-p_{k+1})T_{k+1},\end{aligned}$$

    where the last inequality follows since $r_k,T_{k+1}\geq 0$.
  - Subsubcase 3.1.2. When $(1-p_{k+1})>0.5$, job J_{k+1} is assigned to the busy machine at time a_{k+1}, i.e., machine 2.
    - With probability $(1-p_{k+1})$, job J_{k+1} is successfully processed. The makespan will be increased by τ_{k+1}.

  - $*$ With probability p_{k+1}, job J_{k+1} is mismatched after a duration of T_{k+1} and rescheduled to machine 1. Since machine 1 will be empty at time $HPF(I_k)+T_{k+1}$, it will immediately process job J_{k+1} for a duration of τ_{k+1}. The makespan will be increased by at most $\tau_{k+1}+T_{k+1}$.

  Therefore, in this subsubcase, the expected makespan will be increased by at most $(1-p_{k+1})\tau_{k+1}+p_{k+1}(\tau_{k+1}+T_{k+1})=\tau_{k+1}+p_{k+1}T_{k+1}$.

Combing the two subsubcases, the increase in expected makespan is bounded by $\tau_{k+1}+T_{k+1}[(1-p_{k+1})\mathbb{1}_{\{p_{k+1}>0.5\}}+p_{k+1}\mathbb{1}_{\{p_{k+1}\leq 0.5\}}]$.

- Subcase 3.2. Suppose it is the empty machine at time a_{k+1}, i.e., machine 1, that works until time $HPF(I_k)$, which means that one mismatched job at machine 2 is rescheduled to machine 1 and occupies it until time $HPF(I_k)$.

  —Subsubcase 3.2.1. When $p_{k+1}\geq 0.5$, job J_{k+1} is assigned to the empty machine at time a_{k+1}, i.e., machine 1.

  - $*$ With probability $(1-p_{k+1})$, job J_{k+1} is mismatched, and it will be rescheduled to machine 2 to process after a duration of T_{k+1}. Note that the process of job J_{k+1} may disrupt the process of the job(s) in I_k that is (are) rescheduled to machine 1 after time a_{k+1}. Machine 1's working duration will be increased by at most T_{k+1}. When job J_{k+1} is rescheduled to machine 2, machine 2 may be busy or empty. If machine 2 is busy, its working duration will be increased by τ_{k+1}. If machine 2 is empty, this implies that T_{k+1} is longer than the remaining processing time at machine 2 for I_k, which is at most r_k; then machine 2's working duration will be increased by at most $T_{k+1}+\tau_{k+1}$. Overall, the system's makespan will be increased by at most $T_{k+1}+\tau_{k+1}$.
  - $*$ With probability p_{k+1}, job J_{k+1} is successfully processed for a duration of τ_{k+1}. The makespan will be increased by at most τ_{k+1}.

  Therefore, in this subsubcase, the expected makespan will be increased by at most $(1-p_{k+1})(T_{k+1}+\tau_{k+1})+p_{k+1}\tau_{k+1}=\tau_{k+1}+(1-p_{k+1})T_{k+1}$.

  —Subsubcase 3.2.2. When $(1-p_{k+1})>0.5$, job J_{k+1} is assigned to the busy machine at time a_{k+1}, i.e., machine 2. In this subcase, job J_{k+1} has to wait until machine 2 is empty before it can be processed, for a duration of at most r_k.

  - $*$ With probability $(1-p_{k+1})$, job J_{k+1} is successfully processed. The makespan will be increased by at most τ_{k+1}.
  - $*$ With probability p_{k+1}, job J_{k+1} is mismatched, and it will be rescheduled to machine 1 to process after a duration of T_{k+1}. The makespan will be increased by at most $\tau_{k+1}+T_{k+1}$.

  Therefore, in this subsubcase, the expected makespan will be increased by at most $(1-p_{k+1})\tau_{k+1}+p_{k+1}(\tau_{k+1}+T_{k+1})=\tau_{k+1}+p_{k+1}T_{k+1}$.

  Combing the two subsubcases, the increase in expected makespan is bounded by $\tau_{k+1}+T_{k+1}[(1-p_{k+1})\mathbb{1}_{\{p_{k+1}>0.5\}}+p_{k+1}\mathbb{1}_{\{p_{k+1}\leq 0.5\}}]$.

Overall, in this case, the expected makespan will be increased by at most $\tau_{k+1} + T_{k+1}[(1 - p_{k+1})\mathbb{1}_{\{p_{k+1}>0.5\}} + p_{k+1}\mathbb{1}_{\{p_{k+1}\leq 0.5\}}]$.

Therefore, under all cases, the expected makespan will be increased by at most $\tau_{k+1} + T_{k+1}[(1 - p_{k+1})\mathbb{1}_{\{p_{k+1}>0.5\}} + p_{k+1}\mathbb{1}_{\{p_{k+1}<0.5\}}]$, i.e., $\mathbb{E}[HPF(I_{k+1})] - \mathbb{E}[HPF(I_k)] \leq \tau_{k+1} + T_{k+1}[(1 - p_{k+1})\mathbb{1}_{\{p_{k+1}>0.5\}} + p_{k+1}\mathbb{1}_{\{p_{k+1}<0.5\}}]$. This completes the proof.

A.6. Proof of Lemma 3

Under our settings of the online stochastic scheduling problem, after an assignment of a new job, the next job in the list (if any) can be seen. After the assignment of any job, we have the following possible scenarios.

Case 1. The list is not empty and the next job can be seen. Once a machine becomes empty, this new job will be assigned to it. In this case, no idleness will occur.

Case 2. The list is empty, i.e., there are no more jobs to be assigned. There are two possible scenarios depending on whether the machines are busy or not.

- Subcase 2.1. Only one machine is busy; specifically, the machine is busy processing the newly assigned job. If the job is successfully processed, there will be no idleness. if the job is mismatched, it will be rescheduled to another machine, which results in the other machine being idle for the duration of detecting the mismatch. This idle time is the detection time of the newly assigned job, which is bounded by $\max\{T_1, T_2, \dots\}$.
- Subcase 2.2. Both machines are busy; one is processing the newly assigned job, and the other is processing a previously assigned job. Suppose job J_x and J_y ($x \geq 1$, $y \geq 1$) are processed on machine 1 and machine 2, respectively. Without loss of generality, assume job J_x is the newly assigned job. Note that under the AI policy, a new job will always be assigned to the empty machine, which means that the job will be processed immediately. A job will be rescheduled to another machine if the first assignment turns out to be a mismatch. Thus, under the AI policy, any jobs that are queueing to be served must be rescheduled jobs with known types. Therefore, in this case, there are at most two jobs with unknown types on the machines, one on each, being processed. Furthermore, if there are any jobs queueing in the system, they must belong to type 2 and are waiting for machine 2 to finish processing job J_y.

  —Subcase 2.2.1. Both jobs J_x and J_y have not revealed their types.

  * If both jobs are processed successfully, no idleness will occur.
  * If job J_x is processed successfully but job J_y is mismatched. Job J_y will be rescheduled to machine 1 after a duration no longer than T_y, since machine 2 may have started processing job J_y before job J_x is assigned. Then an idle time may occur for machine 1 if machine 1 finishes processing job J_x first, and the idle time is at most T_y.
  * If job J_x is mismatched but job J_y is processed successfully. Job J_x will be rescheduled to machine 2 after a duration of T_x. Then an idle time may occur for machine 2 if machine 2 finishes processing job J_y (and any other queueing jobs) first, and the idle time is at most T_x.

- $*$ If both jobs are mismatched, there may be idleness on one of the machines depending on which one finishes detection first. Following the same argument as before, the total idle time is bounded above by either T_x or T_y.

— Subcase 2.2.2. Only one of the jobs has not revealed its type, and it must be the newly assigned job, J_x.

- $*$ If job J_x is processed successfully, no idleness will occur.
- $*$ If job J_x is mismatched, it will be rescheduled to machine 2, waiting for machine 2 to finish processing job J_y and any other jobs queueing to be processed (if any). In this scenario, again, no idleness will occur.

To summarize, in this subcase, the idle time for any machine is at most $\max\{T_x, T_y\}$.

Therefore, the total idle time for any machine is bounded above by $\max\{T_1, T_2, \dots\}$.

A.7. Proof of Proposition 2

We prove the property using induction. First of all, we consider the assignment of one job, J_1, i.e., $I_1 = \{J_1\}$. According to the LEMF policy, the job will be assigned to the machine with a success matching probability greater than 0.5. The expected makespan of this job satisfies $\mathbb{E}[LEMF(I_1)] = 2L_{I_1} \leq 2\mathbb{E}[OPT(I_1)]$.

By induction, we assume $\mathbb{E}[LEMF(I_k)] \leq 2\mathbb{E}[OPT(I_k)]$ for all job sequence $I_k = (J_1, J_2, \dots, J_k)$. We need to show the inequity holds for $I_k = (J_1, J_2, \dots, J_k, J_{k+1})$, i.e., $\mathbb{E}[LEMF(I_{k+1})] \leq 2\mathbb{E}[OPT(I_{k+1})]$. To prove this inequality, according to Lemma 6, it suffices to prove that $\mathbb{E}[LEMF(I_{k+1})] - \mathbb{E}[LEMF(I_k)] \leq \tau_{k+1} + T_{k+1}[(1-p_{k+1})\mathbb{1}_{\{p_{k+1}>0.5\}} + p_{k+1}\mathbb{1}_{\{p_{k+1}\leq 0.5\}}]$. At the time to assign job J_{k+1}, which we denote as a_{k+1}, we have the following three possible cases. Denote the remaining makespan for the first k jobs at time a_{k+1} as r_k, which is equivalent to $LEMF(I_k) - a_{k+1}$.

Case 1. Two machines are both empty. Under the LEMF policy, we choose machine j so that the probability of job J_{k+1} belonging to type j is larger than 0.5. In this case, the expected makespan increases by $\tau_{k+1} + T_{k+1}[(1-p_{k+1})\mathbb{1}_{\{p_{k+1}>0.5\}} + p_{k+1}\mathbb{1}_{\{p_{k+1}\leq 0.5\}}]$.

Case 2. One machine is empty, the other is busy; the type(s) of the job(s) being processed by or waiting for the busy machine is (are) known, i.e., there will not be any mismatch in the busy machine. Without loss of generality, suppose machine 1 is empty, and machine 2 is busy. The analysis here will be almost the same as two subcases under Case 2 in the proof of Proposition 1. The only difference is that instead of assigning the job according to the value of p_{k+1}, the job will be assigned to the machine with less increase in expected makespan.

- Subcase 2.1. If job J_{k+1} is assigned to the empty machine, i.e., machine 1, following Subcase 2.1 in the proof of Proposition 1, the expected makespan will be increased by at most $\tau_{k+1} + (1-p_{k+1})T_{k+1}$.
- Subcase 2.2. If job J_{k+1} is assigned to the busy machine, i.e., machine 2, following Subcase 2.1 in the proof of Proposition 1, the expected makespan will be increased by $\tau_{k+1} + p_{k+1}T_{k+1}$.

Under the LEMF policy, the job will be assigned to the machine with less increase in expected makespan. Hence, the expected makespan, in this case, will be increased by at most

$$\begin{aligned}&\min\{\tau_{k+1} + (1-p_{k+1})T_{k+1}, \tau_{k+1} + p_{k+1}T_{k+1}\}\\ =&\tau_{k+1} + T_{k+1}\left[(1-p_{k+1})\mathbb{1}_{\{p_{k+1}>0.5\}} + p_{k+1}\mathbb{1}_{\{p_{k+1}\leq 0.5\}}\right].\end{aligned}$$

Case 3. One machine is empty, and the other is busy; the type(s) of some job(s) being processed by or waiting for the busy machine is (are) uncertain, i.e., there will probably be (some) mismatch(es) in the busy machine. Without loss of generality, suppose machine 1 is empty, and machine 2 is busy. There are two possible scenarios in this case, depending on which machine processes the last job in I_k or, in other words, works until time $LEMF(I_k) = a_{k+1} + r_k$.

- Subcase 3.1. Suppose it is the busy machine at time a_{k+1}, i.e., machine 2, that works until time $LEMF(I_k)$.
  - —Subsubcase 3.1.1. If job J_{k+1} is assigned to the empty machine, i.e., machine 1, following Subsubcase 3.1.1 in the proof of Proposition 1, the expected makespan will be increased by at most $\tau_{k+1} + (1 - p_{k+1})T_{k+1}$.
  - —Subsubcase 3.1.2. If job J_{k+1} is assigned to the busy machine, i.e., machine 2, following Subsubcase 3.1.2 in the proof of Proposition 1, the expected makespan will be increased by at most $(1 - p_{k+1})\tau_{k+1} + p_{k+1}(\tau_{k+1} + T_{k+1}) = \tau_{k+1} + p_{k+1}T_{k+1}$.

  Under the LEMF policy, the job will be assigned to the machine with less increase in expected makespan. Hence, the expected makespan, in this subcase, will be increased by at most

$$\begin{aligned}&\min\{\tau_{k+1} + (1 - p_{k+1})T_{k+1}, \tau_{k+1} + p_{k+1}T_{k+1}\}\\ =&\tau_{k+1} + T_{k+1}\left[(1 - p_{k+1})\mathbb{1}_{\{p_{k+1}>0.5\}} + p_{k+1}\mathbb{1}_{\{p_{k+1}\le 0.5\}}\right].\end{aligned}$$

- Subcase 3.2. Suppose it is the empty machine at time a_{k+1}, i.e., machine 1, that works until time $LEMF(I_k)$, which means that one mismatched job at machine 2 is rescheduled to machine 1 and occupies it until time $LEMF(I_k)$.
  - —Subsubcase 3.2.1. If job J_{k+1} is assigned to the empty machine, i.e., machine 1, following Subsubcase 3.2.1 in the proof of Proposition 1, the expected makespan will be increased by at most $\tau_{k+1} + (1 - p_{k+1})T_{k+1}$.
  - —Subsubcase 3.2.2. If job J_{k+1} is assigned to the busy machine, i.e., machine 2, following Subsubcase 3.2.2 in the proof of Proposition 1, the expected makespan will be increased by at most $\tau_{k+1} + p_{k+1}T_{k+1}$.

  Under the LEMF policy, the job will be assigned to the machine with less increase in expected makespan. Hence, the expected makespan, in this subcase, will be increased by at most

$$\begin{aligned}&\min\{\tau_{k+1} + (1 - p_{k+1})T_{k+1}, \tau_{k+1} + p_{k+1}T_{k+1}\}\\ =&\tau_{k+1} + T_{k+1}\left[(1 - p_{k+1})\mathbb{1}_{\{p_{k+1}>0.5\}} + p_{k+1}\mathbb{1}_{\{p_{k+1}\le 0.5\}}\right].\end{aligned}$$

Overall, in this case, the expected makespan will be increased by at most $\tau_{k+1} + T_{k+1}[(1 - p_{k+1})\mathbb{1}_{\{p_{k+1}>0.5\}} + p_{k+1}\mathbb{1}_{\{p_{k+1}\le 0.5\}}]$.

Therefore, under all cases, the expected makespan will be increased by at most $\tau_{k+1} + T_{k+1}[(1 - p_{k+1})\mathbb{1}_{\{p_{k+1}>0.5\}} + p_{k+1}\mathbb{1}_{\{p_{k+1}<0.5\}}]$, i.e., $\mathbb{E}[LEMF(I_{k+1})] - \mathbb{E}[LEMF(I_k)] \le \tau_{k+1} + T_{k+1}[(1 - p_{k+1})\mathbb{1}_{\{p_{k+1}>0.5\}} + p_{k+1}\mathbb{1}_{\{p_{k+1}<0.5\}}]$. This completes the proof.

Appendix B: Counterexamples to the Optimality of the LUF Policy

EXAMPLE 1. Suppose there are 3 jobs and 2 machines ($\tau = 1$). The initial job-type probability matrix is given by

$$P(1) = \begin{pmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \\ p_{31} & p_{32} \end{pmatrix} = \begin{pmatrix} 0.2 & 0.8 \\ 0.3 & 0.7 \\ 1 & 0 \end{pmatrix}.$$

According to the LUF policy, the whole process can be presented by a decision tree, as shown in Figure 7. The scheduling decisions in the first period are $a_{1,2}(1) = a_{3,1}(1) = 1$, i.e., job 1 is assigned to machine 2, and job 3 is assigned to machine 1. After one period, job 3 leaves the system, and with probability 0.8 job 1 also leaves the system. However, there may be a mismatch for job 1 with probability 0.2, and job 1 will be assigned to machine 1 in the second period. In either case, job 2 will be assigned to machine 2 in the second period, and with probability 0.7, it will be served successfully and leave the system. In the other case, job 2 has to stay for one more period to be processed by machine 1. The expected makespan under the LUF policy is $0.7 \times 2 + 0.3 \times 3 = 2.3$. Consider an alternative policy that keeps job 3 unassigned in the first period. Its

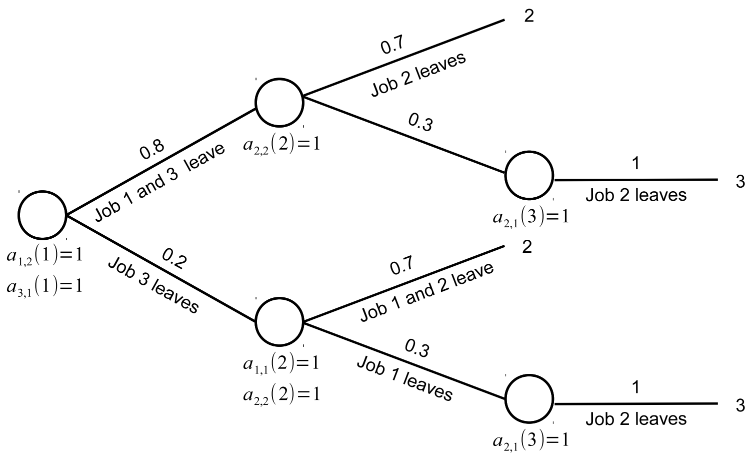


Figure 7 Graphical representation of the LUF policy in Example 1

decision tree is presented in Figure 8. One can verify that after the first period, the types of all the remaining jobs in the system are explicitly known, and the decisions become straightforward. The expected makespan under this policy is $0.8 \times 2 + 0.2 \times 3 = 2.2 < 2.3$.

Example 1 has a job with a known type, so one may suspect the presence of certainty could affect the execution of the LUF policy. However, we can modify Example 1 by changing the probabilities of job 3 to obtain the case when all jobs have uncertain types but the LUF policy is still suboptimal. We present the modified example next.

EXAMPLE 2. Suppose there are 3 jobs and 2 machines ($\tau = 1$). The initial job-type probability matrix is given by

$$P(1) = \begin{pmatrix} 0.2 & 0.8 \\ 0.3 & 0.7 \\ 1-\epsilon & \epsilon \end{pmatrix},$$

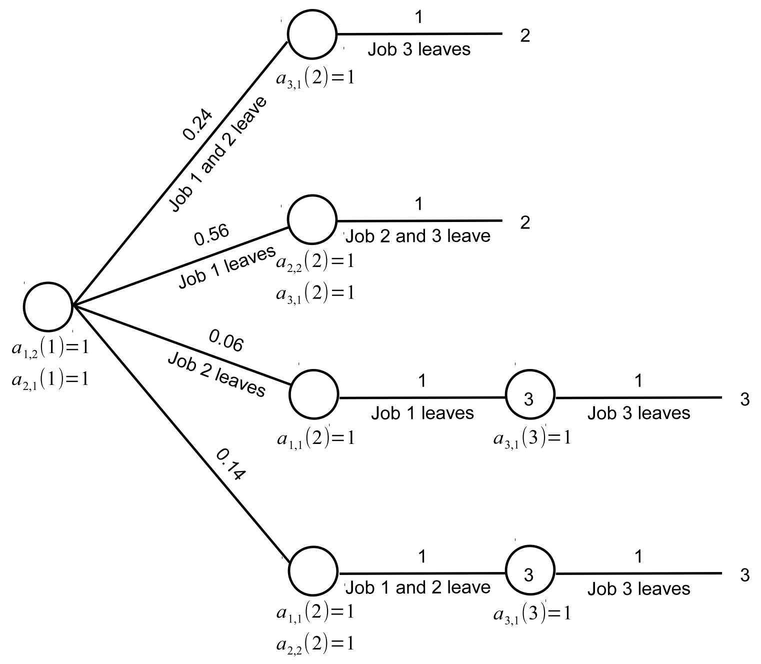


Figure 8 Graphical representation of the alternative policy in Example 1

where $0 < \epsilon << 1$ is a very small constant. One can easily compute the expected makespan under the LUF policy, which equals $2.3 + 0.52\epsilon$. On the other hand, the alternative priority list policy that assigns job 1 to machine 2 and job 2 to machine 1 gives the expected makespan $2.2 + 0.88\epsilon$. Therefore, when ϵ is small enough, the LUF policy is still worse than the alternative policy.

Appendix C: Description of a Mass-casualty Scenario

Let's consider a mass-casualty scenario, for example, nuclear detonation, terrorist attack, hurricane, tsunami, etc. The emergency health care response to such a mass-casualty incident may involve medical personnel that have a wide range of capabilities and skills for treating patients with trauma, burns, respiratory impact, submersion injury, infected wounds, etc. In such emergency healthcare settings, we can assume that all patients are available at time zero and the types of care required are unknown initially.

Emergency care usually consists of two phases of service: diagnosis and treatment, which are typically carried out by separate teams. A team of nurses quickly assesses patients' symptoms and conditions. We assume that initial probabilities are available at this time. We can also assume a predictive model, together with nurse assessments, that can estimate the likelihood of a patient's condition type. With the accumulation of medical data and the development of technology, such a scenario may not be impossible. We believe that predictive information, which indicates the probability distribution of patients' conditions, can become increasingly available in the era of big data. In an emergency healthcare setting, the type probability information could be present at the beginning or sequentially assessed and learned.

Given the initial probability, we have different heuristic policies for assigning patients to physicians. The service time is usually longer than the diagnosis time. In **?**'s study, the diagnosis time was about 7 minutes, and the service time was about 13 minutes. **?** set the diagnosis time to 0.5 minute and the service time to at least 4 minutes. We assume that if there is a wrong assignment, the physician will detect the right type of condition for the patient.

Using the procedure described above, it is not difficult to apply our proposed model given these parameters in a disaster scenario. The system manager can leverage the data and the predictive model to make efficient scheduling based on our proposed policy. In the following, we present a tsunami example to illustrate how to use our framework and compare our proposed policy with the benchmark policy.

Example 3. After a tsunami, there can be an outbreak of respiratory virus disease when the victims aspirate seawater and are infected by contamination from mud and microorganisms (**?**). Patients with infectious diseases (e.g., influenza) should be treated in the infectious disease department, while others can be treated in general internal medicine. We can use a predictive model (e.g., classification and regression trees) to predict the likelihood of infectious disease for each patient based on the data collected, such as symptoms (**?**). Given the disease likelihood of each patient, they will accordingly be assigned to the infectious disease department or general internal medicine accordingly.

We assume the proportion of infectious cases is about 32.7% and generate the initial probability accordingly (**?**). We normalize the processing time for a wrong assignment to be one period. We assume that the service time for a noninfectious disease takes one period, while the service time for an infectious disease is longer; e.g., two periods. We randomly generate 30 examples and run 100 replications for each example. Using the procedure described above, we apply our proposed policy to this example and compare it with the benchmark policy. Table 4 shows the numerical results.

When the number of patients is 50, compared with the HPF policy, the GLUF policy can reduce 4.84 periods of expected makespan, with a sacrifice of 1.17 mismatch cases. When $n = 100$, there is a reduction of

Table 4 System performance under different policies in the tsunami example.

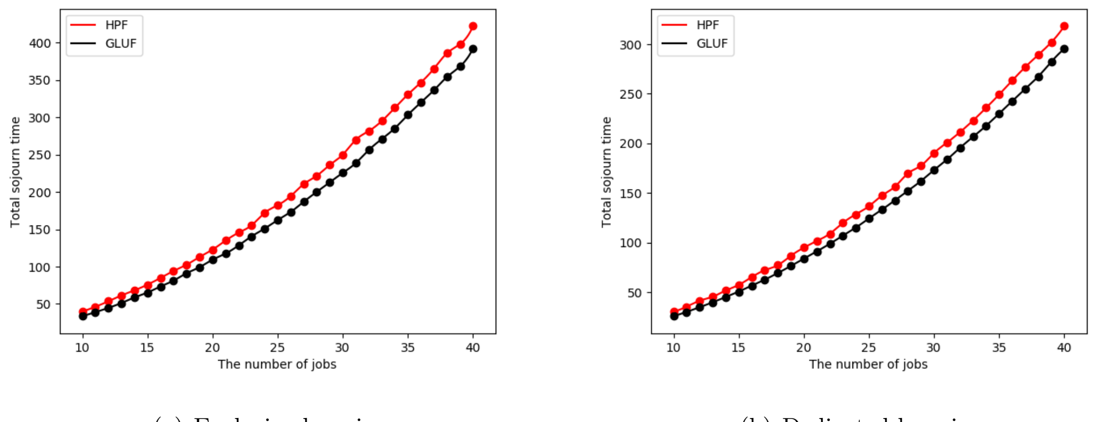
	$(n = 50)$			$(n = 100)$		
	HPF	GLUF	(gap%)	HPF	GLUF	(gap%)
Makespan	46.76	41.92	(↓10.35%)	90.14	83.23	(↓7.67%)
Total Sojourn Time	1084	1017	(↓6.18%)	4152	3988	(↓3.95%)
Mismatch	14.69	15.86	(↑7.96%)	28.63	30.75	(↑7.40%)

6.91 periods in the expected makespan and an increase of 2.12 in the number of mismatches. If the primary objective of the disaster manager is to treat all patients as soon as possible, our results indicate that the GLUF may be a better scheduling policy.

Appendix D: Additional Numerical Results

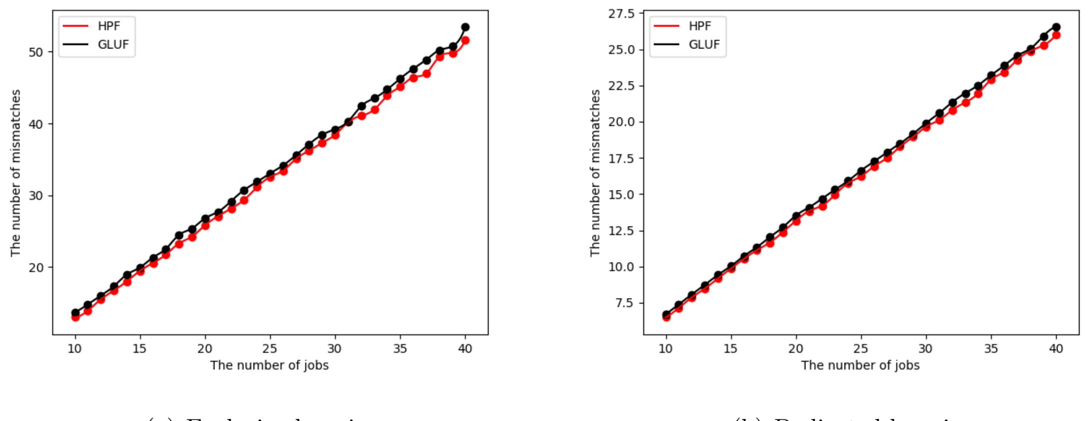
Table 5 System performance under different numbers of jobs ($m = 5$).

	Makespan				Total Sojourn Time				Mismatch Workload			
	Exclusive		Dedicated		Exclusive		Dedicated		Exclusive		Dedicated	
n	HPF	GLUF	HPF	GLUF	HPF	GLUF	HPF	GLUF	HPF	GLUF	HPF	GLUF
10	8.1	6.3	5.9	4.5	39.8	33.6	30.7	25.9	13.0	13.6	6.5	6.7
11	8.6	6.7	6.4	4.8	46.1	38.9	35.5	30.2	13.9	14.8	7.1	7.4
12	9.1	7.2	6.8	5.2	52.5	44.7	40.5	34.9	15.5	16.0	7.9	8.1
13	9.8	7.7	7.2	5.5	60.0	51.2	45.9	39.9	16.7	17.3	8.5	8.7
14	10.4	8.2	7.6	5.8	68.1	58.4	51.7	45.2	18.0	18.9	9.2	9.4
15	10.8	8.6	8.0	6.1	75.8	65.2	57.8	50.7	19.5	19.9	9.9	10.0
16	11.4	9.1	8.4	6.5	84.8	73.3	64.5	56.7	20.6	21.3	10.5	10.7
17	12.0	9.6	8.9	6.9	93.1	81.2	71.2	62.7	21.7	22.5	11.1	11.3
18	12.4	10.0	9.3	7.2	103.0	90.1	78.7	69.4	23.3	24.5	11.7	12.0
19	13.1	10.5	9.7	7.5	112.8	99.2	86.5	76.4	24.2	25.4	12.4	12.7
20	13.5	10.9	10.1	7.9	123.0	108.5	95.0	84.0	25.8	26.8	13.2	13.5
21	14.2	11.5	10.5	8.2	132.9	117.5	102.7	91.3	27.1	27.7	13.8	14.1
22	14.7	11.9	10.9	8.5	145.0	128.3	111.0	98.9	28.1	29.2	14.2	14.7
23	15.3	12.4	11.3	8.9	158.4	140.7	120.0	107.0	29.3	30.7	15.0	15.3
24	15.9	12.9	11.7	9.2	170.0	151.0	128.6	115.2	31.1	31.9	15.8	15.9
25	16.4	13.3	12.1	9.5	182.3	162.4	137.9	124.4	32.5	33.0	16.2	16.6
26	16.9	13.8	12.5	9.8	194.0	173.2	146.9	133.3	33.4	34.1	16.9	17.2
27	17.5	14.2	12.9	10.2	208.7	187.0	157.2	142.7	35.1	35.6	17.5	17.9
28	17.9	14.7	13.3	10.5	221.8	200.2	167.4	152.3	36.2	37.1	18.3	18.5
29	18.4	15.1	13.7	10.8	236.0	213.3	177.6	162.2	37.3	38.4	18.9	19.2
30	18.8	15.5	14.0	11.2	249.2	225.6	189.4	173.2	38.4	39.2	19.7	19.9
31	19.4	16.0	14.4	11.5	263.6	238.9	200.9	183.9	40.2	40.3	20.1	20.6
32	20.0	16.5	14.7	11.9	281.8	256.4	212.5	195.7	41.1	42.4	20.8	21.3
33	20.4	16.9	15.1	12.2	296.5	271.1	224.0	206.7	41.9	43.5	21.3	22.0
34	20.9	17.4	15.5	12.5	311.9	285.6	236.0	217.9	43.9	44.7	22.0	22.5
35	21.5	17.9	15.9	12.9	330.8	303.5	248.5	230.1	45.1	46.3	22.9	23.2
36	22.0	18.4	16.3	13.2	346.9	320.0	262.0	242.7	46.4	47.6	23.4	23.9
37	22.6	18.8	16.7	13.5	363.8	336.5	275.0	255.0	47.0	48.9	24.3	24.6
38	23.1	19.3	17.1	13.8	382.4	354.3	288.3	267.5	49.3	50.2	24.9	25.0
39	23.5	19.8	17.4	14.2	396.6	368.4	304.0	282.3	49.8	50.7	25.3	25.9
40	24.0	20.4	17.8	14.5	422.5	392.5	317.3	295.3	51.7	53.4	26.0	26.5



(a) Exclusive learning (b) Dedicated learning

Figure 9 **Total sojourn time comparison between GLUF and HPF policies under different numbers of jobs ($m = 5$).**

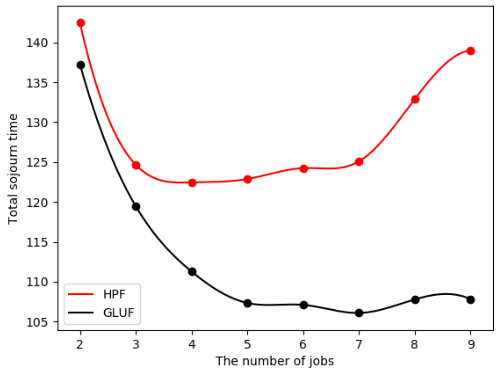


(a) Exclusive learning (b) Dedicated learning

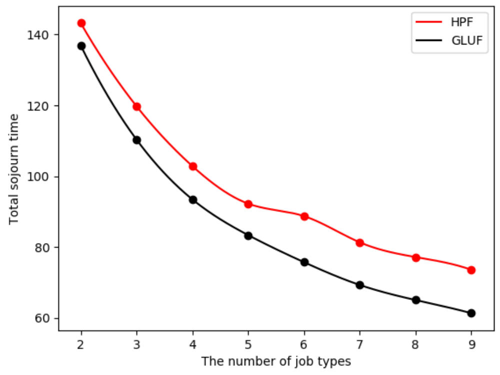
Figure 10 **Mismatch comparison between GLUF and HPF policies under different numbers of jobs ($m = 5$).**

Table 6 **System performance under different numbers of machines ($n = 20$).**

	Makespan				Total Sojourn Time				Mismatch Workload			
	Exclusive		Dedicated		Exclusive		Dedicated		Exclusive		Dedicated	
m	HPF	GLUF	HPF	GLUF	HPF	GLUF	HPF	GLUF	HPF	GLUF	HPF	GLUF
2	15.0	13.7	15.1	13.6	143.0	137.8	143.6	137.7	6.3	6.5	6.1	6.5
3	13.9	11.9	12.8	10.8	128.4	119.3	119.6	110.9	12.7	13.1	9.6	9.9
4	13.4	11.2	11.0	9.0	121.9	112.2	103.2	95.1	19.0	20.0	11.6	12.1
5	13.5	11.0	10.1	7.9	122.1	108.4	94.1	83.6	25.6	26.6	13.0	13.3
6	13.9	11.0	9.3	7.1	125.7	107.7	86.5	75.4	32.7	33.5	14.1	14.3
7	14.3	11.1	8.5	6.4	128.9	108.2	79.4	69.6	38.9	40.6	14.8	15.0
8	14.6	11.4	8.1	6.1	133.2	109.1	76.5	65.2	46.7	46.9	15.5	15.6
9	15.1	11.6	7.7	5.7	134.8	110.1	73.1	61.7	52.5	54.0	15.9	16.1
10	15.7	12.0	7.3	5.3	140.0	111.4	69.9	58.7	59.0	60.5	16.3	16.4

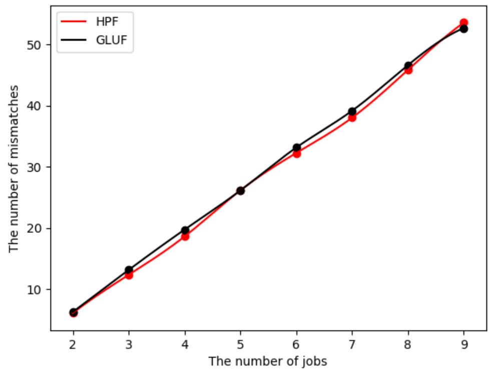


(a) Exclusive learning

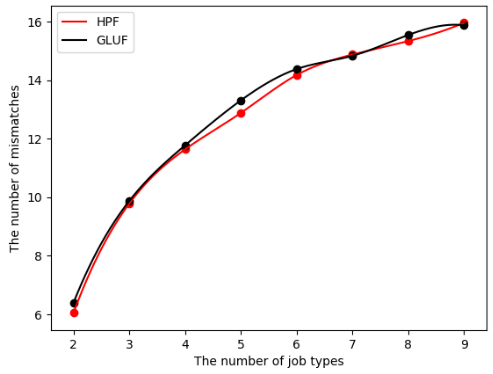


(b) Dedicated learning

Figure 11 **Total sojourn time comparison between GLUF and HPF policies under different numbers of machines ($n = 20$).**



(a) Exclusive learning



(b) Dedicated learning

Figure 12 **Mismatch comparison between GLUF and HPF policies under different numbers of machines ($n = 20$).**

Table 7 **System performance under different service times ($m = 5, n = 20$).**

	Makespan				Total Sojourn Time				Mismatch Workload			
	Exclusive		Dedicated		Exclusive		Dedicated		Exclusive		Dedicated	
t	HPF	GLUF	HPF	GLUF	HPF	GLUF	HPF	GLUF	HPF	GLUF	HPF	GLUF
1	13.6	10.8	10.0	7.9	123.4	108.8	94.2	83.1	25.7	26.6	13.0	13.2
2	20.6	16.4	17.0	13.8	190.3	164.5	158.6	140.0	25.9	28.2	13.0	13.6
3	27.8	22.1	24.2	20.4	261.5	220.9	225.1	197.8	25.7	28.6	13.0	13.9
4	35.9	28.5	31.2	26.7	328.7	280.1	292.2	255.9	26.0	29.8	13.0	14.3
5	43.1	34.4	38.7	34.0	405.4	337.1	363.4	311.7	25.7	30.6	13.0	15.1
6	51.2	41.0	45.2	40.0	477.8	396.6	426.7	366.1	25.7	31.6	12.9	15.5
7	58.4	47.0	52.8	46.8	547.2	452.4	492.3	420.9	25.8	32.1	12.9	16.2
8	65.7	53.2	59.6	53.0	614.7	508.3	564.4	478.8	25.8	32.6	13.0	17.7
9	73.4	60.1	67.3	60.0	680.4	566.9	626.2	538.7	26.0	33.1	13.0	18.4
10	81.0	66.6	74.6	66.5	762.3	625.8	698.2	592.3	25.7	34.2	13.0	19.8

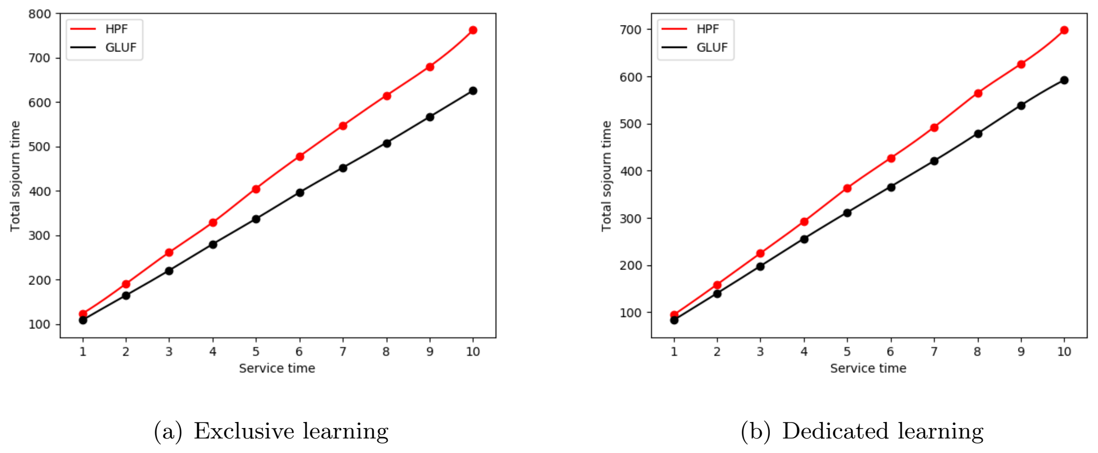


Figure 13 **Total sojourn time comparison between GLUF and HPF policies under different service times ($m = 5, n = 20$).**

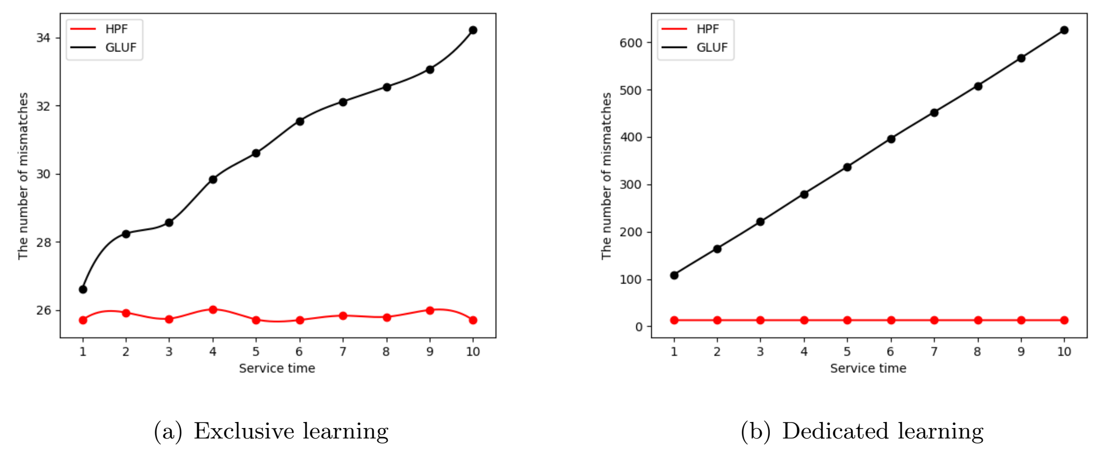


Figure 14 **Mismatch comparison between GLUF and HPF policies under different service times ($m = 5, n = 20$).**

Table 8 **Average mismatch under different policies with service time following geometric distributions ($m = n$).**

	$(\mu_1 = 2, \mu_2 = 4)$		$(\mu_1 = 2, \mu_2 = 6)$		$(\mu_1 = 2, \mu_2 = 8)$	
n	HPF	GLUF	HPF	GLUF	HPF	GLUF
2	0.474	0.897	0.485	0.909	0.469	0.919
3	0.616	0.787	0.586	0.841	0.629	0.873
4	0.998	1.152	0.979	1.188	1.021	1.260
5	1.085	1.642	1.015	1.762	1.069	1.835

Table 9 Average makespan under different policies with service time following geometric distributions ($m = 3$).

	Dedicated Learning			Exclusive Learning		
n	HPF	GLUF	(gap%)	HPF	GLUF	(gap%)
3	9.83	8.46	(↓13.94%)	10.53	8.82	(↓16.23%)
6	17.21	14.88	(↓13.54%)	17.84	15.30	(↓13.68%)
9	23.67	20.70	(↓12.55%)	24.72	21.09	(↓14.68%)
12	30.42	26.58	(↓12.62%)	31.22	26.84	(↓14.02%)
15	32.91	32.22	(↓5.14%)	38.60	33.2	(↓13.98%)

Table 10 Expected makespan under different policies with service time following lognormal distributions, $Lognormal(1, 0.5^2)$, $Lognormal(2, 0.5^2)$, and $Lognormal(1, 0.5^2)$ ($m = 3$).

	Dedicated Learning			Exclusive Learning		
n	HPF	GLUF	(gap%)	HPF	GLUF	(gap%)
3	11.04	10.03	(↓9.15%)	11.81	10.54	(↓10.75%)
6	20.09	17.89	(↓10.95%)	19.78	17.54	(↓11.32%)
9	28.85	25.55	(↓11.43%)	28.73	26.38	(↓8.18%)
12	37.52	33.43	(↓10.90%)	37.92	35.02	(↓7.65%)
15	44.24	40.10	(↓9.36%)	45.61	41.44	(↓9.14%)

Note. We have to discretize the service time in our setting and we round it down.

Table 11 Expected makespan under different policies with service time following discrete uniform distributions $\mathcal{U}$[1,3], $\mathcal{U}$[1,4], and $\mathcal{U}$[1,5] ($m = 3$).

	Dedicated Learning			Exclusive Learning		
	HPF	GLUF	(gap%)	HPF	GLUF	(gap%)
3	5.55	4.75	(↓14.41%)	5.94	4.92	(↓17.17%)
6	9.74	8.08	(↓14.99%)	10.13	8.26	(↓18.46%)
9	13.32	11.40	(↓14.41%)	13.98	11.69	(↓16.38%)
12	16.87	14.56	(↓13.69%)	18.06	15.04	(↓16.72%)
15	20.36	17.58	(↓13.65%)	21.37	18.25	(↓14.60%)

Table 12 Expected makespan under different job type probability distributions with service time following geometric distribution.

	$U(0,1)$			$Beta(0.5, 0.5)$			$Beta(2, 2)$		
n	AI	HPF	LEMF	AI	HPF	LEMF	AI	HPF	LEMF
5	26.50	27.42	26.11	25.05	25.18	24.31	24.55	26.00	24.29
10	48.38	48.82	47.37	48.11	47.73	46.71	46.73	48.30	46.11
15	69.37	68.71	67.52	65.29	63.86	63.79	71.96	74.24	71.33
20	87.62	87.19	86.06	91.15	89.16	88.42	91.88	93.31	90.76
25	108.90	107.72	106.46	110.71	107.56	107.47	108.43	109.92	106.87
30	127.89	126.87	125.39	129.63	126.24	126.24	136.41	137.61	134.73

Appendix E: MDP Solution for Small-size Examples When $m = 3$

When $m = 3, n = 4$, we set the initial job type probability distribution to

$$\begin{pmatrix} 0.2 & 0.5 & 0.3 \\ 0.3 & 0.6 & 0.1 \\ 0.8 & 0.1 & 0.1 \\ 0.6 & 0.1 & 0.3 \end{pmatrix}.$$

Table 13 Expected makespan under different job type probability distributions with service time following Erlang distribution.

	$U(0,1)$			$Beta(0.5,0.5)$			$Beta(2,2)$		
n	AI	HPF	LEMF	AI	HPF	LEMF	AI	HPF	LEMF
5	27.44	28.51	26.95	27.28	27.36	26.41	26.67	28.04	26.42
10	50.36	51.54	49.80	48.12	47.85	47.09	49.04	50.61	48.44
15	71.50	71.27	70.17	71.52	70.80	70.27	72.31	73.71	71.45
20	93.48	91.61	91.58	93.55	91.10	91.07	93.28	93.84	92.01
25	114.65	113.35	112.41	115.65	113.07	113.04	117.18	117.91	115.95
30	138.83	136.59	136.58	139.12	135.90	135.89	138.04	138.28	136.49

We vary n from 2 to 4 by choosing the first n jobs from the above example. Suppose the detection time for a mismatch is one period, and the expected service time of type-1, type-2, and type-3 jobs μ_1, μ_2, μ_3 are two periods, four periods, and five periods, respectively. The comparison between the optimal expected makespan and the expected makespan using the GLUF policy under dedicated learning can be seen in Table 14.

Table 14 Makespan comparison among different policies ($m = 3$).

	$(\mu_1 = 2, \mu_2 = 4, \mu_3 = 5)$						
n	Opt	GLUF	(gap%)	HPF	(gap%)	Computing time
2	6.43	6.86	6.69%	8.03	24.88%	10.61s
3	7.42	7.99	7.68%	8.51	14.69%	511.71s
4	8.98	9.40	4.68%	11.07	23.27%	34764.75s