Congestion-Aware Matching and Learning for Service Platforms

Xu Sun
Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL 32611, USA, xusun@ufl.edu

Jingtong Zhao
School of Economics, Renmin University of China, Beijing, China, zhaojingtong@ruc.edu.cn

We study dynamic matching in a service platform modeled as a multi-class, multi-server queueing system with Bernoulli rewards dependent on job-server assignments. The service platform, however, is unaware of the job-server-specific mean rewards. Thus, the goal is to minimize regret, defined as the difference between the cumulative payoff over a time horizon and the maximum possible payoff obtained when the platform has complete knowledge of all system parameters while all job arrivals and service completions occur in a deterministic fashion. We propose and analyze a main algorithm for matching and learning based on the idea underpinning the interior-point method, as well as a bandit algorithm for estimating rewards of job-server assignments. For settings where the total service capacity exceeds the total demand, we show that our main algorithm has a sub-linear regret that matches the lower bound of a typical multi-armed bandit problem while maintaining stability for the queue lengths at various servers. We also supply the main algorithm with a queue-length-based matching scheme. Numerical experiments using both synthetic randomly generated data and a real-world data set reveal that both our main algorithm and the queue-length-based matching scheme are quite effective. A major takeaway from our analysis is that, despite not appearing explicitly in the managerial objective, queuing delays can slow down learning, which in turn hurts the profitability of these platforms. As a result, the consideration of reducing service delays may arise not only from the need to achieve high service levels but also from the need to maximize profit.

Key words: service platforms; reward maximization; matching; online learning; multi-armed bandit; queueing

1. Introduction

A slew of online service platforms have emerged in a variety of domains. By connecting a customer (“job”) to a service provider (“server”) using information and communications technology,
these online platforms have greatly improved service access and resource utilization while also reducing search costs and service delays. Among these service platforms, many share the following characteristics, which directly motivate this work. First, jobs are naturally segmented (and thus heterogeneous), and they are served by a group of servers with overlapping capabilities. Second, services are appointment-based, which means that a job must be assigned to a server upon arrival, and since each server has limited capacity, an assigned job may have to wait before his or her service can actually commence. Third, different assignments between a job and a server can result in different expected rewards. However, since the platform does not know how good a match is at the outset, there is a need to learn enough about its job categories and server characteristics.

One notable example is online healthcare platforms, which have grown rapidly in recent years. For patients with minor diseases, online healthcare consultations through photos or phone calls are often more attractive because they are less expensive compared to offline consultations while also allowing patients to consult physicians from home. Online consultations also alleviate hospital congestion, allowing offline medical resources to be reserved for patients who are most in need. For these reasons, the availability of online healthcare platforms improves patient satisfaction by promoting more equitable access to healthcare services and easing the constraints caused by medical resource scarcity. HaoDf and WeiDoctor are two mature online healthcare platforms in China that allow patients to not explicitly select a physician when making a consultation request and instead accept the platform’s assignment. This option is especially appealing to patients who do not know the physicians well and do not have the time to conduct a thorough search. In assigning patients to physicians, the current consideration is to ensure the assigned physician can respond quickly enough so that the patient does not have to wait a long time. A major drawback of this approach, however, is that a patient may be assigned to a less suitable physician, producing a less desirable clinical outcome.

Thus, the problem that these platforms face is matching so as to optimize system payoffs subject to server capacity constraints without prior knowledge of various reward parameters. The problem
is challenging because it requires resolving an exploration-exploitation trade-off. Specifically, making high-quality matches necessitates accurate estimates of the unknown reward parameters. However, how the matching is performed will determine whether the unknown parameters can be learned efficiently. The task is further compounded by the fact that job arrivals are subject to probabilistic uncertainty. This gives rise to rich and complex queueing dynamics, the consequences of which cannot be ignored. Indeed, as we demonstrate shortly, overlooking the stochastic variability could lead to algorithms that force certain servers to operate at full utilization, resulting in large job backlogs at those servers, thereby delaying the collection of reward feedback and leaving a large number of jobs unattended at the end of a given time horizon.

In this paper, we consider a model that can be perceived as a mathematical abstraction of the aforementioned problem. Specifically, we view the platform as a queueing network with $I$ job types and $J$ servers. Upon arrival, jobs are assigned to one of the servers. Therefore, each server keeps its own queue and processes jobs in a first-come, first-served manner. Each job, regardless of the type, takes one period for a single server to complete. Each service completion generates a reward that follows a probability distribution dependent on both the job type and the server. (We assume a Bernoulli distribution for the rewards.) All realized rewards are observable to the platform, which has system-level knowledge of the arrival rates of each job type as well as the precise type of each job when it arrives. However, the platform is unaware of the values of the expected rewards. Therefore, the platform must learn about those values through the rewards obtained when jobs complete services in their respective queues and exit the system.

We wish to point out that, even when the expected reward values are known to the platform, it seems not immediately clear how to build a good matching algorithm from scratch. Under this circumstance, one may attempt to solve a deterministic relaxation of the matching problem (thereby ignoring all probabilistic uncertainty), which is essentially a linear program whose solution determines the long-run proportion of jobs of a specific type that will be sent to a specific server. It turns out that such a naive approach can be problematic. The reason is that the optimal solution
of a linear program is typically located at one of the vertices. Interpreting this in our context would mean that certain servers may inevitably have a utilization rate of one. Hence, if the time horizon is long enough, the queues at those servers could potentially grow very large, causing a significant loss of payoff at the end of the time horizon as well as poor service levels over time. As a result, a well-designed algorithm ought to ensure that the capacity constraints are not only met but also non-binding (i.e., utilization rate being strictly less than one).

A major contribution of this paper is the development of an algorithm that meaningfully integrates learning, a component needed to efficiently estimate the unknown mean reward values, and routing, another component seeking to assign incoming jobs to the most suitable servers to maximize payoff generation while ensuring that all queues are stable.

To develop some intuition into our solution approach, consider again the situation where all mean reward values are known. Recall that keeping the utilization rate of all servers strictly less than one is necessary to achieve good performance. For this purpose, we modify the deterministic relaxation of the matching problem (assuming all mean rewards are known) by adding a penalty term to the objective and refer to the resulting formulation as the penalty problem. The penalty term is chosen such that its value will be small at points away from the capacity constraint boundaries and will tend to infinity as the constraint boundaries are approached. This is reminiscent of the well-known interior-point method in linear programming. Unlike the motive behind the creation of the interior-point method, however, which was to accelerate computation for solving large-scale linear programs, our goal is to keep capacity constraints strictly non-binding. In the penalty problem, we use $1/V$ to fine-tune the penalty weight, similar to how the interior-point method is typically executed, where there is a penalty parameter that can be adjusted adaptively. Intuitively, the higher the value of $V$, the higher the payoff rate and the closer the servers’ utilization rate to one (hence more pending jobs in the system). In this regard, the choice of $V$ strikes a delicate balance between service payoff and service level. When the mean reward values are unknown, as we have assumed in this study, we surrogate the (unknown) true mean rewards in the penalty problem.
with their respective estimates obtained from an optimistic algorithm (a variant of the celebrated upper confidence bound (UCB) algorithm), giving rise to what we call the main algorithm. Because rewards are only collected after jobs are completed, a large queue would entail prolonged delays in receiving the rewards, which in turn slows down learning. In this respect, the choice of $V$ also influences the learning rate.

On the theoretical side, our main result (Theorem 2) reveals that our main algorithm essentially achieves the optimal regret bound as the length of the time horizon grows to infinity, where the regret is defined as the loss in total payoff accrued in the planning time horizon relative to the maximum payoff that can be achieved when mean rewards are completely known and all random variables are replaced by their respective expected values. The regret bound comprises of four dominating terms. The first term, a quantity of order $T/V$, is due to the “barrier” created to keep server utilization rate from reaching one; intuitively, this term reflects the payoff sacrificed to ensure the queues are stable. The second term, which is order $\sqrt{T \log T}$, captures the loss of payoff caused by the platform’s agnostic attitude toward mean rewards. The third term, a quantity of order $V (\log T)^{3/2}$, results from the learning slowdown caused by longer queues. The fourth term, which is of order $V$, accounts for the payoff loss due to the job backlog at the end of the planning horizon. Overall, our regret bound exposes the inherent tension between the goal of learning the unknown reward parameters and reward maximization.

Our theoretical results are substantiated by numerous numerical explorations, which demonstrate that our main algorithm outperforms a myopic algorithm in a wide range of realistic scenarios. We also propose a queue-length-based matching algorithm derived using the same idea underpinning our main algorithm. This algorithm creates a closed feedback loop by penalizing rewards based on real-time queue lengths. In addition to performing well empirically (it achieves nearly the same reward maximization performance as our main algorithm), this queue-length-based matching algorithm is also easy to implement, sharing the same computational complexity as a very simple greedy algorithm.
The rest of the paper is organized as follows. Section 2 goes over the relevant literature. Section 3 lays out the problem formulation and sets the stage for our analysis. Sections 4 and 5, respectively, present and analyze our main algorithm. We present the results of our numerical experiments in Section 6. In Section 7, we conclude.

2. Literature Review

This paper pertains to several streams of research, which we will survey in turn.

Early applications such as public-housing allocation and the rise of the sharing economy have given rise to a number of interesting models for matching in two-sided queueing systems. Caldentey et al. (2009) introduce a bipartite matching model, which has since been further developed by Adan and Weiss (2012), Bušić et al. (2013), Adan and Weiss (2014), Adan et al. (2018). A design problem seeking optimal matching typologies (i.e., compatibility structures) between customer classes and servers is considered by Afeche et al. (2022). Driven by ride-sharing applications, Hu and Zhou (2022) study a stochastic matching problem, where the goal is to maximize the sum of discounted match-dependent rewards. A few papers have considered multipartite matching. For example, Gurvich and Ward (2015) study a model where items from multiple component queues are to be matched instantly and the goal is to minimize system-wide holding costs subject to compatibility constraints. Moyal and Perry (2017) investigate instability of the model considered in Gurvich and Ward (2015) and Nazari and Stolyar (2019) consider maximizing match-dependent reward rates subject to queue stability constraints. Kerimov et al. (2021) study centralized dynamic matching markets with finite agent types and heterogeneous match values and show that suitably designed greedy policies can be near-optimal. Aveklouris et al. (2021) consider a matching model with heterogeneous impatient supply and demand. Based on fluid analysis, they show that a greedy static priority policy is near-optimal. Other papers that consider bipartite or multipartite matching include, among others, Anderson et al. (2017), Ashlagi et al. (2019a,b).

The present work is related to a large body of research that studies matching for one-sided queueing systems. The vast majority of papers in this stream focus on scheduling optimization
under compatibility constraints, with a particular emphasis on minimizing congestion-related costs. Typically, asymptotically optimal scheduling policies are identified under either the conventional heavy traffic regime (Harrison and López 1999, Mandelbaum and Stolyar 2004) or the so-called many-server limiting regime (Atar 2005, Dai and Tezcan 2008, Ward and Armony 2013). More recently, motivated by ride-hailing systems, Özkan and Ward (2020) analyze a problem of maximizing the number of matches for a variant of the parallel server model with probabilistic compatibility graph. In contrast to these studies, where some compatibility graph and some version of the so-called complete resource pooling are specified, we impose no external compatibility constraints. The suitability of matching is encoded through mean rewards. Among studies of matching in one-sided queues, there is a long line of investigations considering match-dependent rewards in one-sided queues. Problems of this kind arise routinely from studies of organ allocation (Zenios et al. 2000, Akan et al. 2012, Ding et al. 2021) as well as housing allocation (Bloch and Cantala 2017, Arnosti and Shi 2020). While we also consider match-dependent rewards, the quality of different matches is not known to the system a priori, a key feature that necessitates integrating learning with job assignment decisions.

Our work builds on a stream of research on resource allocation under uncertainty that combines learning with optimization of resource allocation, focusing on scheduling and matching problems (Massoulié and Xu 2018, Levi et al. 2019, Shah et al. 2020, Krishnasamy et al. 2021, Johari et al. 2021). The problem that we study is different in that we consider the objective of reward maximization subject to queueing stability. Our work is most closely related to Hsu et al. (2022), in which the authors investigate the problem in an online service platform where each job brings a randomly generated service requirement (number of tasks) that can be distributed across multiple servers. They assume the assignment rewards and other job-related statistics are unknown, and propose a utility-guided online learning and assignment algorithm. Kim and Vojnovic (2021) extend the preceding setting to allow for bi-linear rewards. Liu et al. (2020) investigates a problem in which $M$ types of jobs must be dispatched to $N$ servers while adhering to capacity, fairness, and resource
budget constraints, without knowing various input parameters in advance. A key distinction between these three papers and ours is that they allow jobs to be backlogged instead of being routed to a dedicated server immediately upon arrival, and as a result, a job can be partly served by different servers at different times. In contrast, jobs in our model must be assigned to a server immediately after they arrive, and they will wait in a queue to be served by this server. In short, the foregoing papers consider queueing-upon-arrival, which effectively eliminates server-side queues and thus delays in collecting rewards altogether, whereas we consider server-side queues. As a result, our algorithm is different and is designed to accommodate the learning slowdown caused by delays in collecting rewards. Accordingly, our regret analysis employs novel proof techniques to quantify the impact of the learning slowdown on the total payoff.

Last but not the least, our work draws on a vast body of literature on bandit problems utilizing confidence bounds, an idea that can trace back to Lai et al. (1985), Lai (1987), Agrawal (1995). The version of UCB adopted here is most similar to that analysed by Auer et al. (2002); see also (Lattimore and Szepesvári 2020, Chapter 7).

3. Model

Consider an online service platform with $I$ types of jobs and $J$ servers. Time is slotted and there is a finite planning horizon of length $T$. At each time slot $t = 1, \ldots, T$, the number of newly arriving type $i$ jobs follows a Poisson distribution with mean $\lambda_i$. The arrival of jobs is assumed to be independent across job types and time steps. Servers are labeled and indexed by $j = 1, \ldots, J$. We assume each job requires one time period to complete, so that regardless of the job and which server is performing the service, the service times are all deterministic and equal to one. For many relevant applications, such as online healthcare platforms, where service providers typically offer 10-minute consultations, the assumption of deterministic service times is reasonable. Thus, the decision maker can choose this time window to be the length of a time slot. Let $\lambda = \sum_{i=1}^{I} \lambda_i$, and we assume $\lambda < J$. A matrix $r := [r_{ij}]$ can be used to summarize the expected matching rewards, where $r_{ij}$ represents the expected reward of assigning a type $i$ job to server $j$. The realized reward
of matching a type $i$ job with server $j$ is assumed to follow a Bernoulli distribution with mean $r_{ij}$. We assume $\min_{i,j} r_{ij} > 0$. For technical reasons, we also assume that the platform knows the value of $\min_{i,j} r_{ij}$ is lower bounded by a positive number $r_*$, which results in a lower truncation of the reward estimates (cf. Eq. (6)). This assumption is innocuous in our opinion because the value of $r_*$ can be chosen arbitrarily small. Moreover, our extensive numerical experiments indicate that even without the lower truncation, the algorithm remains very effective in practice.

Jobs arrive at the start of each time period, and the platform must make an irreversible matching decision immediately, routing each of these new jobs to one of the $J$ servers. If there are multiple jobs waiting to be served by server $j$, then the jobs will be processed in a first-come-first-serve order. Let $\Gamma_{ij}(t)$ denote the actual number of type $i$ jobs assigned to server $j$ in time slot $t$ for any matching policy $\Pi$. Then, under probabilistic matching, each $\Gamma_{ij}(t)$ will be a Poisson random variable due to the thinning property of the Poisson distribution.

Let $Q_{ij}(t)$ denote the number of jobs of type $i$ that are waiting in queue $j$ at the beginning of time slot $t$. Also, we denote by $Q_j(t)$ the number of jobs awaiting to be served by server $j$ at the start of time slot $t$. Clearly, we have $Q_j(t) = \sum_i Q_{ij}(t)$. Because we do not explicitly incorporate waiting costs in the decision-making and all jobs share the same service time, each job, once routed in a queue, will be the same as all existing jobs in the queue. This observation motivates us to concentrate on $Q_j$, whose dynamics can then be described as

$$Q_j(t + 1) = \max \left\{ Q_j(t) + \sum_{i=1}^I \Gamma_{ij}(t) - 1, 0 \right\}. \tag{1}$$

In addition, we define $B_{ij}(t)$ as the number of type $i$ jobs served by server $j$ in time slot $t$. Because each server can only process one job at a time, it follows that

$$\sum_{i=1}^I B_{ij}(t) = \min \left\{ Q_j(t) + \sum_{i=1}^I \Gamma_{ij}(t), 1 \right\}. \tag{2}$$

The platform is aware of the type of each arriving job, as well as the value of $\lambda_i$, but is unaware of the expected matching rewards, which are encoded in the matrix $r$. Thus, the platform would
need to learn these reward parameters from the observed noisy feedback while trying to maximize the total payoff over the planning horizon of length $T$. More formally, the platform seeks to

$$\max_{\Pi} R^{\Pi}(T) := \sum_{t=1}^{T} \sum_{i=1}^{I} \sum_{j=1}^{J} \mathbb{E}[B_{ij}(t)r_{ij}], \quad (3)$$

where each $\Pi$ ought to be understood as an integrating learning and job routing scheme.

Consider a perfect scenario in which not only is the expected reward matrix $r$ known, but all stochastic variability is removed. In this case, the task of maximizing rewards boils down to a static planning problem, which can be expressed as a linear program given below:

$$\max_{p_{ij}} \sum_{i=1}^{I} \sum_{j=1}^{J} \lambda_ip_{ij}r_{ij}$$

subject to

$$\sum_{i=1}^{I} \lambda_ip_{ij} \leq 1 \text{ for all } j \quad (4)$$

$$\sum_{j=1}^{J} p_{ij} = 1 \text{ for all } i$$

$$p_{ij} \geq 0 \text{ for all } i, j$$

In plain words, the first set of constraints in (4) requires that the expected number of jobs assigned to a server should be less than the number of jobs that server can complete in one time slot. The second set of constraints in (4) stipulates that the assignment probabilities of each job type need to sum to 1; that is, we cannot leave a job unassigned. We will henceforth denote the optimal solution to the preceding program as $p^* := [p^*_{ij}]$ and the corresponding objective value as $R^*$. It is also straightforward to formulate the dual problem of the linear program (4). To that end, let $\alpha := [\alpha_j]$ denote the corresponding dual variables of the first set of constraints in (4) and $\beta := [\beta_i]$ the dual variables of the second set of constraints. Then the dual problem can be formulated as

$$\min_{\alpha_j, \beta_i} \sum_{j=1}^{J} \alpha_j + \sum_{i=1}^{I} \beta_i$$

subject to $\lambda_i\alpha_j + \beta_i \geq \lambda_ir_{ij}$ for all $i, j$

$$\alpha_j \geq 0 \text{ for all } j$$

The following result establishes useful properties of the optimal dual variables, which will be useful in the regret analysis for our main algorithm.
**Lemma 1.** Let $\beta^* := [\beta^*_i]$ denote the optimal dual variables associated with the $I$ equality constraints. Then under the condition $\lambda < J$, we have that $\beta^*_i > 0$ for all $i = 1, \ldots, I$.

Lemma 1 shows that all of the optimal dual variables associated with the equality constraints are strictly positive. This means that the platform has no incentive to leave any job unassigned even if it is permitted to do so. In other words, we can relax all of the equality constraints to “less than or equal to” without affecting the optimal objective value of the primal. To develop some intuition, suppose, for the sake of contradiction, that the optimal solution to the primal requires that a portion of jobs be rejected. Since the total capacity exceeds the demand, re-directing some or all of the rejected jobs to some underutilized servers can strictly improve the objective value, contradicting the optimality hypothesis.

We will adopt the concept of *regret* to measure the performance of policy $\Pi$, namely,

$$\text{Reg}^\Pi(T) := R^*T - R^\Pi(T),$$

where $R^*T$ is to be seen as an upper bound for $R^\Pi(T)$.

### 4. Main Algorithm

We now introduce our main algorithm. As mentioned in the Introduction section, the core idea of our algorithm is to include a penalty term in the objective. In this way, the solution we obtain will be away from the constraint boundaries, which in other words means that we can avoid utilizing some servers at full capacity.

Similar to the algorithm in Hsu et al. (2022), our algorithm proceeds in three steps. As the mean rewards are unknown to the platform at the beginning, the first step is to use past observations to form UCB estimates of the rewards. We use $h_{ij}(t-1)$ to denote the number of type $i$ jobs that have been served by server $j$ up to time $t-1$, and the average payoff of these observations is denoted as $\bar{r}_{ij}(t-1)$. The UCB estimates at time $t$ are then calculated from equation (6).

The second step is to use the UCB estimates of the rewards in solving the optimization problem defined by equation (7), which aims to both maximize the total payoff and reduce queueing delays.
As in the usual case, the second term of our objective function simply seeks to maximize the total payoff based on our current estimates. The first term in the objective serves as a penalty term. More specifically, we do not want to assign all jobs to a certain server $j$, because this will lead to a long queue at server $j$, and we do not want the size of any queue to blow up. Because we assume service rate to be one, $\log(1 - \sum_{i=1}^{I} \lambda_i p_{ij}(t))$ aims to ensure that no server is always working at full capacity, resulting in more equitable and even assignments. The constant $V$ controls the magnitude of the penalty term, and the penalty decreases as $V$ increases.

The last step is to assign jobs according to the optimal solution of our optimization problem. We also observe rewards from the current period, which can then be used to update our reward estimates. The complete algorithm is given in Algorithm 1.

5. Analysis of the Main Algorithm

In this section, we analyze the stability of queue lengths and provide a regret bound for our main algorithm. Throughout this section, we will assume that $T$ is large enough that all the conditions pertaining to the time horizon in the theoretical results hold.

The first question that naturally arises is whether the main algorithm maintains system stability. This is addressed by the following theorem, which guarantees that all queues are stable.

**Theorem 1.** Under the condition $\lambda < J$, the following states are true: (i) for every $t \geq 0$,
\[
\mathbb{E}[Q_j(t)] \leq V + 1,
\]
for all $V$ large enough; and (ii) $\mathbb{E}[Q_{j_{\text{max}}}(T)] \leq \gamma V \log T$ for all sufficiently large $V$, where $Q_{j_{\text{max}}}(T)$ denotes the running maximum of $Q_j$ up to $T$ and $\gamma$ is some constant independent of anything else.

Having verified that all queues are stable, we now turn to establishing the regret bound for our main algorithm. Our starting point is to note that
\[
\text{Reg}_{\text{alg}}(T) \leq \mathbb{E} \left[ \sum_{t=1}^{T} \sum_{i=1}^{I} \sum_{j=1}^{J} \lambda_i (p_{ij}^\ast - p_{ij}(t)) r_{ij} \right] + \mathbb{E} \left[ \sum_{j=1}^{J} Q_j(T) \right].
\]
\[ (8) \]

The first term on the right-hand side can be understood as the payoff gap between the payoff achieved by our algorithm and the benchmark value $R^\ast T$ if all jobs were served at the end of time.
Algorithm 1: Main Algorithm

Data: $\lambda_i, \forall i = 1, 2, ..., I$ and $V$

1 For every time slot $t = 1, ..., T$:

2 Step 1: Form truncated UCB mean reward estimates

3 for $i = 1, 2, ..., I$ do

4 for $j = 1, 2, ..., J$ do

5 if $h_{ij}(t-1) = 0$ then

6 \hspace{1cm} $r_{ij}(t) = 1.$

7 end

8 else

9 \hspace{1cm} Set

10 \hspace{1.2cm} $r_{ij}(t) = \max \left\{ \min \left\{ \bar{r}_{ij}(t-1) + \sqrt{\frac{2 \log(t-1)}{h_{ij}(t-1)}}, 1 \right\}, r^*_j \right\}.$ \hspace{1cm} (6)

11 end

12 end

13 Step 2: Solve $p_{ij}(t)$ for the optimization problem

\[
\max_{p_{ij}(t)} \sum_{j=1}^{J} \left\{ \frac{1}{V} \log \left( 1 - \sum_{i=1}^{I} \lambda_i p_{ij}(t) \right) + \sum_{i=1}^{I} \lambda_i p_{ij}(t) r_{ij}(t) \right\}
\]

subject to $\sum_{j=1}^{J} p_{ij}(t) = 1$ for all $i$ \hspace{1cm} (7)

$p_{ij}(t) \geq 0$ for all $i, j$

Step 3: Assign jobs and obtain noisy reward feedback

14 for $i = 1, 2, ..., I$ do

15 Assign each type $i$ job that arrives at time $t$ to the queue at server $j$ with probability

16 $p_{ij}(t)$.

17 end
for $j = 1, 2, ..., J$ do
  if the queue at server $j$ is not empty then
    Observe the type of the first job in the queue $i^*$, and the reward $X_{i^*j} \sim \text{Bern}(r_{i^*j})$.
    Update
    \begin{align*}
    h_{i^*j}(t) &= h_{i^*j}(t - 1) + 1 \\
    \bar{r}_{i^*j}(t) &= (h_{i^*j}(t - 1)\bar{r}_{i^*j}(t - 1) + X_{i^*j}) / h_{i^*j}(t)
    \end{align*}
  end
end

$T_j$. The second term on the right can be viewed as a correction term, capturing the loss of rewards due to jobs in the queue at the end of the considered time horizon.

To get a handle on the first term on the right-hand side of (8), we denote matrices $m := [m_{ij}]$ and $\pi := [\pi_{ij}]$ and define the function
$$f(\pi|m) := \sum_{j=1}^{J} \left[ \log \left( 1 - \sum_{i=1}^{I} \lambda_i \pi_{ij} \right) + V \sum_{i=1}^{I} \lambda_i \pi_{ij} m_{ij} \right].$$

Note that $f(\pi|r(t))$ is the objective of (7) multiplied by $V$. Thus, $p(t) := [p_{ij}(t)]$ is the maximizer of $f(\pi|r(t))$ subject to the constraints in (7). Also, let $\tilde{p} := [\tilde{p}_{ij}]$ denote the maximizer of $f(\pi|r(t))$ over the constraints in (7). That is, $\tilde{p}$ is the solution to the following optimization problem:
$$\max_{p_{ij}} \sum_{j=1}^{J} \left\{ \frac{1}{V} \log \left( 1 - \sum_{i=1}^{I} \lambda_i p_{ij} \right) + \sum_{i=1}^{I} \lambda_i p_{ij} r_{ij} \right\}$$
subject to $\sum_{j=1}^{J} p_{ij} = 1$ for all $i$
$$p_{ij} \geq 0$$ for all $i, j$.

With these preparations, we have from (8) that
$$\text{Reg}_{\text{alg}}(T) \leq T \sum_{i=1}^{I} \sum_{j=1}^{J} \lambda_i (p_{ij}^* - \tilde{p}_{ij}) r_{ij} + \mathbb{E} \left[ \sum_{t=1}^{T} \sum_{i=1}^{I} \sum_{j=1}^{J} \lambda_i (\tilde{p}_{ij} - p_{ij}(t)) r_{ij} \right] + \mathbb{E} \left[ \sum_{j=1}^{J} Q_j(T) \right].$$

(10)
From (10) one can see that a regret bound can be obtained if each term on the right-hand side can be properly bounded. We start with the first term, which leads to the following proposition.

**Proposition 1.** For \( \tilde{p} \) as defined above, we have that

\[
\sum_{i=1}^{I} \sum_{j=1}^{J} \lambda_i (p_{ij}^* - \tilde{p}_{ij}) r_{ij} \leq \frac{J}{V}.
\]

Proposition 1 shows that the logarithmic barrier, which is added to ensure queue stability, results in a payoff loss of the order of \( 1/V \) (per unit time), when compared to the offline static problem.

**Proposition 2.** For all \( V \) large enough, we have that

\[
\mathbb{E} \left[ \sum_{t=1}^{T} \sum_{i=1}^{I} \sum_{j=1}^{J} \lambda_i (\tilde{p}_{ij} - p_{ij}(t)) r_{ij} \right] \leq 2K \sqrt{2 \log T} \left( 2IJ + 2J \sqrt{T} + \sum_{i,j} \mathbb{E} [Q_{ij}^{\text{max}}(T)] \right) + \frac{14}{3} KJ \lambda,
\]

where \( K \) is some positive constant dependent on the model primitives. 

Notably, the bound in Proposition 2 involves the running maxima of queue lengths. To put this result into perspective, we remark that the notion of maximum values in queueing processes does not appear in the regret bound established in Hsu et al. (2022) or Kim and Vojnovic (2021). We attribute the difference to the fact that by requiring jobs to queue first before being dispatched, both the aforementioned papers remove server-side queues and thus eliminate the delay in collecting reward feedback, whereas our model has server-side queues and delays in queue also influence the learning rate. This fundamental difference also manifests itself through the mathematical proofs. To quantify the payoff loss due to learning, the corresponding proof in Hsu et al. (2022) ingeniously employs a martingale argument while also relying on a key observation that each job assignment allows the UCB algorithm to update the reward estimate. Our proof, despite using a similar martingale argument, cannot exploit the “instant feedback” mechanism as in Hsu et al. (2022). To overcome this technical challenge, we developed alternative proof techniques, which we believe carry some methodological novelty. At a high level, our proof of Proposition 2 employs the following strategy: First act as if reward feedback is instantly collected, then make a correction to compensate for
the learning slowdown. It turns out that the correction produces an error term that is linked to maximum values in queueing processes.

The main result of this section is Theorem 2, which follows as a direct consequence of (10), Propositions 1 and 2, as well as Theorem 1.

**Theorem 2.** Suppose \( \lambda < J \). Then

\[
\text{Reg}^{a_{10}}(T) \leq \frac{J T}{V} + 2J K \sqrt{\frac{2 \log T}{I}} + 2J K \sqrt{2 \log T + \gamma IV \log T} + \frac{14}{3} K J \lambda + J(V + 1)
\]

for all \( V \) large enough.

There are four dominating terms in the above regret bound that come from four different sources. The first term \( JT/V \) is the price we pay for keeping the queues stable. The second term \( 4JK \sqrt{2IT \log T} \) is due to the fact that the platform does not know the mean rewards at the outset. The third term \( 2\gamma IJKV \sqrt{2(\log T)^3} \) comes from the learning slowdown due to longer queues. The last term \( J(V + 1) \) is the payoff loss caused by the backlogged jobs at the end of the planning horizon.

To see that this bound is competitive, we let \( V = \sqrt{T} \), yielding a bound of order \( \sqrt{T}(\log T)^{3/2} \).

Recall that a large \( V \) gives preference to allocating jobs to servers with higher rewards but also promotes longer queues. Thus, the regret bound exposes the inherent tension between the goal of learning the unknown reward parameters and reward maximization.

**6. Numerical Experiments**

In this section, we conduct numerical experiments to verify the efficiency of our main algorithm. We first introduce our benchmarks, where we propose a simple queue-length-based benchmark algorithm inspired by methods commonly used in online matching problems. We then show that the regret of our algorithm from simulation is in line with our theoretical guarantee and is better than the regrets of the benchmarks. We also show how the regret of our algorithm is affected by the parameters of the model, such as the congestion level of the system, and the tuning parameter \( V \). We then consider scenarios where the reward-generating process or the arrival process differs from our assumptions to show that our algorithm is robust to such deviations. Finally, we test our algorithm with real-world data crawled from a large Chinese online healthcare platform.
6.1. Benchmark Algorithms

To test the performance of our algorithm, we compare it to several popular benchmarks.

First, we introduce a simple queue-length-based benchmark algorithm, which we call ‘Inventory Balancing (IB)’. As the name suggests, this algorithm is inspired by a type of algorithm commonly used in online matching problems, where resources of limited inventories are assigned to customers; see, e.g., Mehta et al. (2007) and Golrezaei et al. (2014). In the original setting, each reward is multiplied by a penalty function, which is non-decreasing of the current inventory level. This way, if a resource has a large reward but a low inventory level, then it might not be assigned because the inventory-adjusted reward will be low. As a result, this resource can be saved for better use later. In our case, the penalty function will be related to the length of the queue at server $j$, which we will denote by $q_j(t)$. Since IB also does not know the true rewards, we assume it will also form UCB estimates. The complete algorithm is given in Algorithm 2. In particular, the constant $d$ in equation (12) is used to adjust the magnitude of the penalty term related to the queue length, and the greater $d$ is, the greater the penalty.

We also include the plain greedy algorithm as another benchmark. To be fair, the greedy algorithm will also form UCB estimates of the unknown rewards, and the procedure is exactly the same as in Algorithm 2, except that line 15 will be replaced by

“Assign each type $i$ job that arrives at time $t$ to the queue at server argmax $r_{ij}(t)$.”

6.2. Simulation Results

In the simulation experiments, we let $I = 2$, $J = 6$, and $\lambda = [2, 3]$. We also set $T = 1000$, $V = 1000$, and $d = 0.01$. Note that we are using these specific values of the tuning parameters $V$ and $d$ in our algorithm and the IB algorithm because it turns out they do not have a large influence on the performance, and we will return to this point in more detail later. The expected reward of each possible assignment is randomly generated from $Unif(0, 1)$ at the beginning. The cumulative reward and average regret of each algorithm are shown in the left and right plot of Figure 1 respectively. In both plots, the orange solid line, the yellow dotted line, and the blue dashed line represent the
Algorithm 2: Inventory Balancing Algorithm

For every time slot $t = 1, \ldots, T$:

Step 1: Form truncated UCB mean reward estimates

for $i = 1, 2, \ldots, I$ do
  for $j = 1, 2, \ldots, J$ do
    if $h_{ij}(t - 1) = 0$ then
      $r_{ij}(t) = 1$.
    else
      Set $r_{ij}(t) = \max \left\{ \min \left\{ \tilde{r}_{ij}(t - 1) + \sqrt{\frac{2 \log(t - 1)}{\tilde{r}_{ij}(t - 1)}}, 1 \right\}, r^* \right\}$.
  end
end

Step 2: Assign jobs and obtain noisy reward feedback

for $i = 1, 2, \ldots, I$ do
  Assign each type $i$ job that arrives at time $t$ to the queue at server
  \[
  \arg \max_j r_{ij}(t)e^{-dq_j(t)}
  \]
end

for $j = 1, 2, \ldots, J$ do
  if the queue at server $j$ is not empty then
    Observe the type of the first job in the queue $i^*$, and the reward $X_{i^*j} \sim \text{Bern}(r_{i^*j})$.
    Update
    \[
    h_{i^*j}(t) = h_{i^*j}(t - 1) + 1 \quad \text{and} \quad \tilde{r}_{i^*j}(t) = (h_{i^*j}(t - 1)\tilde{r}_{i^*j}(t - 1) + X_{i^*j})/h_{i^*j}(t).
    \]
  end
end
greedy, IB, and our algorithm respectively. We first notice from the left plot that both our algorithm and the IB algorithm perform quite well, and they are much better than the greedy algorithm. In fact, the cumulative rewards of these two algorithms are very close to the upper bound represented by the purple circled line.

From the right plot, we again notice that the average regret of the greedy algorithm is quite large, and it is not decreasing properly with time. In contrast, the average regret of our algorithm or the IB algorithm decreases rapidly at first and gradually approaches zero. The purple circled line represents the function $y = \sqrt{\log(t)/t}$ scaled by a constant, which is the regret bound we proved before divided by $t$. We see that the average regret of our algorithm is very close to this line, which is in line with our theoretical results.

![Figure 1](image-url) Cumulative reward (left) and average regret (right) of different algorithms.

Besides the reward and the regret, we will also look at some other interesting aspects of the system. First, we will look at the queue length at different servers under different assigning rules. In our example, there are $J = 6$ servers in total, so we will only look at a few representative servers to avoid making the figure too cluttered.

Figure 2 shows the queue length at different servers under the greedy, IB, and our main algorithm, represented by the orange solid line, the yellow dotted line, and the blue dashed line, respectively. The left plot shows the queue length at the server that has the highest matching reward, which is
server 6, and this server happens to yield the highest reward for both types of jobs. We first notice
that the queue length under the greedy algorithm blows up. This is as expected, because after some
initial learning periods, the greedy algorithm will have formed a rather accurate estimate of each
pairing reward, and it will assign all jobs to server 6 from then on, as this is the most profitable
server. In contrast, the queue length of our algorithm or the IB algorithm is rather stable and does
not show any sharp increase. Hence, neither our algorithm nor the IB algorithm will assign all jobs
to server 6 even after they have learned that this is the most profitable server, because by design,
these two algorithms will try to avoid assigning all jobs to one server.

The right plot shows the queue length at server 5. For type 1 jobs, server 5 has the lowest reward,
and the reward for type 2 jobs is also not very high. Hence, we can treat server 5 as a less desirable
server. For the greedy algorithm, after some initial learning periods, it is rather confident that
server 5 is not the most profitable server for either type of jobs, so it stops assigning any jobs to this
server, which can be seen from the sharp decrease in the queue length approximately starting from
time 220. In contrast, for the other two algorithms, although they have also learned that server 5 is
probably not the best server, they still assign some jobs to this server to help make the queues in
the system more balanced.

![Queue length at server 6](image1)

![Queue length at server 5](image2)

**Figure 2** Queue length at server 6 (left) and server 5 (right).

We also record the average queue length at each server and the average waiting time of each job
type in the following two tables. From the left part of Table 1, we see that compared to the greedy
Table 1 Average queue length of each server and average waiting time of each type of jobs under different algorithms.

<table>
<thead>
<tr>
<th></th>
<th>Server 1</th>
<th>Server 2</th>
<th>Server 3</th>
<th>Server 4</th>
<th>Server 5</th>
<th>Server 6</th>
<th>Type 1 Jobs</th>
<th>Type 2 Jobs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy</td>
<td>45.0090</td>
<td>184.2910</td>
<td>0.2180</td>
<td>187.0040</td>
<td>7.3170</td>
<td>617.4280</td>
<td>203.0911</td>
<td>172.5313</td>
</tr>
<tr>
<td>IB</td>
<td>4.4110</td>
<td>15.2380</td>
<td>0.1770</td>
<td>26.7480</td>
<td>3.6450</td>
<td>31.1600</td>
<td>15.2386</td>
<td>16.7510</td>
</tr>
<tr>
<td>Alg</td>
<td>6.8100</td>
<td>6.1070</td>
<td>0.1060</td>
<td>15.8040</td>
<td>5.9670</td>
<td>25.6780</td>
<td>11.2906</td>
<td>12.3589</td>
</tr>
</tbody>
</table>

algorithm, the average queue length under our algorithm is distributed much more evenly, and it is also slightly better than that under the IB algorithm. Using Little’s law, we can deduce that the average waiting time of jobs under our algorithm will be much smaller than that under the greedy algorithm and slightly smaller than that under the IB algorithm as well. This is supported by the right part of Table 1.

6.3. Sensitivity Analysis

We now explore how the model parameters will influence the performance of our algorithm. To start, an important factor that will influence the performance of queueing systems is the congestion level of the system. This is also known as “traffic intensity”, and in the present context is defined as \( \rho = \frac{\lambda}{J} \). It is a measure of how busy a system is and, in a single-server setting, approximates the fraction of time the server is busy.

The four plots in Figure 3 show how the traffic intensity will influence the performance of different algorithms. In these four experiments, we let all parameters to be the same as before, including \( d \), \( V \), and \( r \), and we only change the arrival rates of the two types of jobs. Specifically, the arrival rates in the four plots are \([1,1], [1,2], [2,2], \) and \([2.99,2.99] \), which corresponds to traffic intensity \( \rho = 0.3333, 0.5, 0.6667, \) and \( 0.9967 \) respectively. From the top left plot, we see that when the traffic intensity is quite low, meaning that servers are idle for most of the time, then the greedy algorithm performs quite well, and the gap between the greedy algorithm and our algorithm is also quite small. But as the traffic intensity increases, we see that the gap between the greedy algorithm and
OPT keeps increasing, but the gap between our algorithm and OPT is actually decreasing, and the gap is actually very small in the bottom right figure, where the traffic intensity is over 99%.

This observation is in line with our intuition because when the system is not busy, only using the most profitable servers will be enough. On the other hand, when the traffic intensity is very close to 1, then we need to utilize every server in order to avoid any queue from blowing up, no matter how large the reward from that server is, and that is exactly what our algorithm or the IB algorithm is trying to achieve.

Figure 3    Cumulative rewards in systems with arrival rates $[1, 1]$ (top left), $[1, 2]$ (top right), $[2, 2]$ (bottom left), $[2.99, 2.99]$ (bottom right).

Next, we investigate how the value of the tuning parameter $V$ affects the performance of our algorithm, and we again let $\lambda = [2, 3]$ as in our main experiment. In Figure 4, we show the performance
of our algorithm using four different values of $V$. The blue solid line, the orange dashed line, the yellow dotted line, and the purple dash-dotted line show the cumulative reward when using $V = 10$, 100, 1000 and 10000, respectively, and the green circled line is the upper bound. We see that the performance is slightly worse when $V = 10$, but there is not much difference between the other values of $V$.

![Cumulative reward for different choices of $V$.](image)

However, from Table 2, we see that the average queue length has an increasing trend, especially at server 6. The average queue length at server 6, the server with the highest reward for both types of jobs, is increasing with $V$. The right part of Table 2 shows that the average waiting time of either type of jobs is also increasing with $V$. These observations are reasonable because as $V$ increases, the magnitude of the penalty term is decreasing, and our algorithm will be more reward-oriented. Hence, although there is not much difference between the cumulative rewards, more jobs are assigned by the algorithm to the most profitable server as $V$ increases, at the cost of each job having to wait a longer time. Thus, if jobs come from actual customers, then from the point of view of a customer, there is a trade-off between “being served promptly” and “being served by the most desirable server”, and the platform can choose $V$ depending on which of the two objectives the customers value more. For instance, if an online healthcare platform wants all patient requests to
be responded to in a timely manner, then it can choose a smaller $V$. As a side note, although this is not the focus of this work, we observe a similar pattern in the tuning parameter $d$ in the IB algorithm. That is, as $d$ decreases, the average waiting time and average queue length will increase, but the cumulative reward does not change much.

### 6.4. Robustness Tests

We now proceed to check the robustness of our algorithm by considering cases where some of our assumptions are violated.

Recall that our model assumes jobs arrive according to a Poisson process, but we expect our algorithm to perform well even if this assumption does not hold in reality. In Figure 5, we consider two cases where the arrival process deviates from Poisson. In the left plot, the number of arrivals at each time step follows a geometric distribution, while it follows a binomial distribution with success probability of 0.1 in the right plot. In both plots, the expected number of arrivals for the two types of jobs are still kept at [2, 3], and all the other parameters remain the same as in our main experiment. We see that despite the arrival process, as long as the expected number of arrivals stays the same, then our algorithm and the IB algorithm still have very good performance.

We also test the influence of the reward distribution on the performance of our algorithm. In Figure 6, instead of assuming the rewards follow a Bernoulli distribution, we simply let the realized reward to be equal to the expected reward plus a random noise term drawn from $Unif(-\epsilon, \epsilon)$. We

---

<table>
<thead>
<tr>
<th>$V$</th>
<th>Server 1</th>
<th>Server 2</th>
<th>Server 3</th>
<th>Server 4</th>
<th>Server 5</th>
<th>Server 6</th>
<th>Type 1 Jobs</th>
<th>Type 2 Jobs</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>45.0090</td>
<td>184.2910</td>
<td>0.2180</td>
<td>187.0040</td>
<td>7.3170</td>
<td>617.4280</td>
<td>203.0911</td>
<td>172.5313</td>
</tr>
<tr>
<td>1000</td>
<td>4.4110</td>
<td>15.2380</td>
<td>0.1770</td>
<td>26.7480</td>
<td>3.6450</td>
<td>31.1600</td>
<td>15.2386</td>
<td>16.7510</td>
</tr>
<tr>
<td>100</td>
<td>6.8100</td>
<td>6.1070</td>
<td>0.1060</td>
<td>15.8040</td>
<td>5.9670</td>
<td>25.6780</td>
<td>11.2906</td>
<td>12.3589</td>
</tr>
<tr>
<td>10</td>
<td>6.8100</td>
<td>6.1070</td>
<td>0.1060</td>
<td>15.8040</td>
<td>5.9670</td>
<td>25.6780</td>
<td>11.2906</td>
<td>12.3589</td>
</tr>
</tbody>
</table>

**Table 2** Average queue length of each server and average waiting time of each type of jobs for different choices of $V$. 
then apply the $\max(\cdot, 0)$ and $\min(\cdot, 1)$ functions to keep the realized reward to be within the range of $[0, 1]$. In the left plot $\epsilon = 0.1$, and $\epsilon = 0.5$ in the right plot. We notice that the performance of our algorithm or the IB algorithm seems to deteriorate a little as the variance of the reward increases, but they are still quite close to the upper bound, and are much better than the greedy algorithm.

6.5. Real-World Data

Last but not least, we present the numerical results using real-world data that we crawled from one of the largest online healthcare platforms in China. More specifically, we collected data for 12
physicians from the cardiology department, and we will focus on the online consultation service through phone calls. The physicians charge different prices for a 10-minute phone call, which are shown in the left plot of Figure 7. We see that the charged price ranges from 30 CNY ($4.5) to 270 CNY ($40.5).

![Figure 7](image)

**Figure 7** Price charged for a 10-minute phone call consultation (left), and recommendation score (right) of each physician.

Each physician also has a recommendation score computed by the platform, which can be seen as a summary statistic, and it’s based on many factors, such as the title of the physician, the hospital the physician currently works in, the reviews of the physician from former patients, and so on. The score of each physician is summarized in the right plot of Figure 7, and the closer the score is to 5, the more recommended the physician is. For the 12 physicians in our data, the recommendation score ranges from 3.1 to 5. Since we would like the reward in the objective to capture not only the price charged by the platform but also some aspect of patient satisfaction, we will include this score in the definition of the reward.

As we only have limited data on the patient side, we will roughly classify them into two broad categories: patients with severe diseases, and patients with minor diseases. Recall that the patients are from the cardiology department, so many of them will eventually need operations, and this online consultation is only part of the initial diagnosis. We will refer to such patients as the “sever” type, and patients that can be treated with medicine are referred to as the “minor” type. Based
on the percentage of each type, we let the arrival rates for “minor” and “severe” patients to be [7,3]. For patients with minor diseases, the average consultation time is typically shorter than 8 minutes, while for patients with severe diseases, the average consultation time can be longer than 15 minutes. Since our model assumes that the service time is fixed, we will include such difference into the reward. Specifically, the expected reward of assigning a type $i$ patient to physician $j$ is defined to be

$$r_{ij} = \begin{cases} 
price_j \cdot \frac{score_j}{5} \cdot 0.8 & \text{if } i \text{ is the “minor” type} \\
price_j \cdot \frac{score_j}{5} \cdot 1.5 & \text{if } i \text{ is the “severe” type}
\end{cases}$$

and the realized reward is equal to the expected reward plus a random noise term drawn from $Unif(-5,5)$.

The results are shown in Figure 8, and we see that the performance of our algorithm or the IB algorithm in the real-world setting is not as good as that in the simulated case presented before. We conjecture that this is due to the use of modified prices without proper normalization, so the variance can be large. Nevertheless, our algorithm and the IB algorithm still significantly outperform the greedy algorithm.

![Figure 8](image)

**Figure 8**  Cumulative rewards using real-world data.
7. Concluding Remarks

We studied the problem of matching jobs to servers in a queueing system where jobs are to be routed to a server upon arrival and completed services result in stochastic rewards, with mean rewards depending on the job types and servers. We proposed and evaluated the performance of an algorithm that combines learning and routing to maximize the total expected reward over a given time horizon while ensuring the stability of all queues. This algorithm estimates the rewards of job-server pairs using a UCB-based learning procedure that accounts for the inherent tradeoff between exploration and exploitation. We demonstrated that the regret of our proposed algorithm scales sub-linearly with the planning time horizon. In particular, our regret bound explicitly quantifies the learning slowdown effect caused by job delays, highlighting the dual role of queueing delays in degrading service levels while also harming system profitability due to inefficient learning. Several numerical experiments using either synthetic randomly generated data or a real-world data set were carried out to validate the effectiveness and robustness of our proposed algorithm in comparison to meaningful benchmarks.

There are several directions for future research. First, the current model assumes all servers are fully flexible in the sense that they can be matched to any job type. For certain applications, there may be matching compatibility constraints. It may thus be worthwhile to investigate how to adapt the central idea underlying our proposed algorithm to those circumstances. Second, in some applications, mean rewards may be determined by the features of jobs and servers according to a bi-linear model, say. Incorporating such features could lead to interesting data-driven decision problems that serve as direct extensions of the model under consideration. Third, this paper, along with a few other closely related papers, has considered a UCB-based learning procedure for estimating rewards. There are other learning procedures, such as Thompson sampling, which not only achieve the same (asymptotic) regret bound as UCB but also show better empirical evidence in practice for multi-armed bandit learning problems. Therefore, future research could look into whether incorporating other learning procedures can result in algorithms with equal or
better performance. Last but no least, we see that the simple queue-length-based IB algorithm has competitive performance in all numerical experiments, so it might be worthwhile to derive theoretical results for this algorithm in the future.

References


E-Companion

The e-companion collects proofs of all mathematical results that are omitted from the main paper.

Proof of Lemma 1. Let \( \alpha^* := [\alpha^*_j] \) denote the optimal dual variables associated with the \( J \) inequality constraints. Given \( \lambda < J \), the \( J \) inequality constraints in (4) cannot be binding at the same time. Therefore, there exists some \( j' \) such that \( \sum_i \lambda_i p_{i,j'}^* < 1 \). Thus, in view of the complementary slackness conditions, we conclude that \( \alpha^*_{j'} = 0 \). Thus,

\[
\beta_i^* \geq \lambda_i r_{ij'} - \lambda_i \alpha^*_{j'} = \lambda_i r_{ij'} \geq \lambda_i r_* > 0 \quad \text{for all} \quad i = 1, \ldots, I,
\]

completing the proof. \( \square \)

Proof of Theorem 1. Consider \( p(t) \), which solves the (convex) optimization problem (7). Let \( \beta(t) := [\beta_i(t)] \) denote the optimal dual variables of the \( I \) equality constraints in (7). A direct application of Karush–Kuhn–Tucker (KKT) optimality conditions (Boyd et al. 2004, §5.5.3) yields

\[
-\frac{1}{V} \frac{\lambda_i}{1 - \sum_i \lambda_i p_{ij}(t)} + \lambda_i r_{ij}(t) - \beta_i(t) = 0 \quad \text{for all} \quad (i,j). \tag{EC.1}
\]

Note that \( \sum_j (1 - \sum_i \lambda_i p_{ij}(t)) = J - \lambda > 0 \), where the inequality follows from our model assumption on system load. It follows that there must exist some \( j_0 \) such that

\[
1 - \sum_i \lambda_i p_{ij_0}(t) \geq \frac{J - \lambda}{J}.
\]

Combining the preceding inequality with (EC.1) gives us

\[
\beta_i(t) = \lambda_i r_{ij_0}(t) - \frac{1}{V} \frac{\lambda_i}{1 - \sum_i \lambda_i p_{ij_0}(t)} \geq \lambda_i r_{ij_0}(t) - \frac{1}{V} \frac{\lambda_i J}{J - \lambda} \geq \lambda_i r_* - \frac{1}{V} \frac{\lambda_i J}{J - \lambda} \quad \text{for all} \quad i,
\]

where the last inequality is due to the lower truncation in (6). Because \( r_* \) is strictly positive, we conclude that \( \beta_i(t) \geq 0 \) for all \( V \) large enough. This, in view of (EC.1), implies that

\[
1 - \sum_i \lambda_i p_{ij}(t) = \frac{1}{V} \frac{\lambda_i}{\lambda_i r_{ij} - \beta_i(t)} \geq \frac{1}{V} \frac{1}{r_{ij}} \geq \frac{1}{V} \tag{EC.2}
\]
for all $V$ large enough.

To proceed, we make the following two observations: First, because the algorithm employs probabilistic routing at every time step, each server will receive a number of jobs that follow a Poisson distribution with mean $\sum_i \lambda_i p_{ij}(t)$ due to the thinning and superposition properties of the Poisson distribution. Second, if a batch of jobs received by a server over a time period is considered a “job”, and the number of jobs in the batch is viewed as “service requirement” of that “job”, then the queue length at each server will exhibit the same dynamics as the workload of a Geo/G/1 queue facing Bernoulli arrival with “success probability” 1 and Poisson service times with time-dependent mean given by $\sum_i \lambda_i p_{ij}(t)$. These two observations, together with a standard coupling argument similar to the one used in the proof of Theorem 1 in Hsu et al. (2022) plus (EC.2), allow us to upper-bound $Q_j(t)$ by the steady-state workload of the following Geo/G/1 discrete-time queueing system: a job would arrive (with probability one) at every time slot, and the service requirement follows a Poisson distribution with mean $1 - 1/V$. Denote by $W$ the steady-state workload of this queueing process. Using the formula for the mean number in the system of a Geo/G/1 queue (see, e.g., Eq. (5.71) in (Alfa 2016, Chapter 5)), we have that

$$
E[W] = \left(1 - \frac{1}{V}\right) \left[\left(1 - \frac{1}{V}\right) + \frac{(1 - 1/V)^2 + 1 - 1/V}{2/V}\right] \leq 1 + V.
$$

Thus, for $V$ large enough,

$$
E[Q_j(t)] \leq E[W] \leq V + 1,
$$

completing the proof of part (i).

To prove part (ii), we first note that, by the stochastic ordering relation established in part (i), $Q_{j}^{\text{max}}$ is stochastically dominated by the extreme workload (defined as the maximum amount of work accumulated between two consecutive time epochs at which a server becomes idle) for the Geo/G/1 system described in part (i). Part (ii) thus follows from the classical results on extreme values in a single-server queue; see, e.g., (Iglehart 1972, Corollary 2). □
Proof of Proposition 1. Consider the optimization problem (9) and let $\tilde{\beta} := [\tilde{\beta}_1]$ denote the optimal dual variables of the $I$ equality constraints in (9). For notational convenience, define

$$g_0(\pi) := \sum_{i,j} \lambda_i \pi_{ij} r_{ij} \quad \text{and} \quad g_j(\pi) = 1 - \sum_i \lambda_i \pi_{ij} \quad \text{for} \quad j = 1, \ldots, J.$$  

Also, let $\ell_i(\pi) := 1 - \sum_j \pi_{ij}$ for $i = 1, \ldots, I$. Then $\pi = \tilde{p}$ if

$$\nabla g_0(\pi) + \sum_{j=1}^J \frac{1}{V g_j(\pi)} \nabla g_j(\pi) + \sum_{i=1}^I \tilde{\beta}_i \nabla \ell_i(\pi) = 0.$$  

Therefore, $\tilde{p}$ maximizes the Lagrangian:

$$L(\pi, \tilde{\alpha}, \tilde{\beta}) := g_0(\pi) + \sum_{j=1}^J \tilde{\alpha}_j g_j(\pi) + \sum_{i=1}^I \tilde{\beta}_i \ell_i(\pi)$$

where we have defined $\tilde{\alpha}_j := 1/(V g_j(\tilde{p}))$. By the well-known saddle-point interpretation of Lagrange duality for convex program (Boyd et al. 2004, §5.4), we have that

$$R^* \leq L(\tilde{p}, \tilde{\alpha}, \tilde{\beta}) = g_0(\tilde{p}) + \frac{J}{V}.$$  

This completes the proof.\qed

Proof of Proposition 2. Using the concavity of $f$ in each $\pi_{ij}$, we have that

$$f(\pi|r) - f(\tilde{p}|r) \leq \sum_{i,j} \frac{\partial f}{\partial \pi_{ij}}(\tilde{p}^t r)(\pi_{ij} - \tilde{p}_{ij}). \quad (\text{EC.3})$$

Using the definition of $f$, letting $\pi = p(t)$ in (EC.3) and rearranging terms, we obtain

$$\sum_{i,j} \left( V \lambda_i r_{ij} - \frac{\lambda_i}{1 - \sum_i \lambda_i \tilde{p}_{ij}} \right) (\tilde{p}_{ij} - p_{ij}(t)) \leq f(\tilde{p}|r) - f(p(t)|r). \quad (\text{EC.4})$$

On the other hand,

$$f(\tilde{p}|r) = f(\tilde{p}|r(t)) + V \sum_{i,j} \lambda_i (r_{ij} - r_{ij}(t)) \tilde{p}_{ij}$$

$$\leq f(p(t)|r(t)) + V \sum_{i,j} \lambda_i (r_{ij} - r_{ij}(t)) \tilde{p}_{ij} \quad (\text{EC.5})$$

$$= f(p(t)|r) + V \sum_{i,j} \lambda_i (r_{ij}(t) - r_{ij}) p_{ij}(t) + V \sum_{i,j} \lambda_i (r_{ij} - r_{ij}(t)) \tilde{p}_{ij},$$
where the two equalities result from the definition of $f$, while the inequality follows from the optimality of $p(t)$. Combining (EC.4) and (EC.5), we get
\[
\sum_{i,j} \left( V \lambda_i r_{ij} - \frac{\lambda_i}{1 - \sum_{i} \lambda_i \bar{p}_{ij}} \right) (\bar{p}_{ij} - p_{ij}(t)) \leq V (A_1(t) + A_2(t)),
\]
where
\[
A_1(t) := \sum_{i,j} \lambda_i (r_{ij}(t) - r_{ij}) p_{ij}(t) \quad \text{and} \quad A_2(t) := \sum_{i,j} \lambda_i (r_{ij} - r_{ij}(t)) \bar{p}_{ij}.
\]

For ease of flow, we present the following lemma, whose proof is deferred to after the proof of the current proposition.

**Lemma EC.1.** There exists some $\kappa > 0$ dependent on the model primitives such that for all $V$ large enough, it holds that
\[
\frac{\lambda_i}{1 - \sum_{i} \lambda_i \bar{p}_{ij}} \leq V \lambda_i r_{ij} (1 - \kappa) \quad \text{for all} \quad (i,j).
\]
Combining Lemma EC.1 with (EC.6) yields
\[
\kappa \sum_{i,j} \lambda_i r_{ij} (\bar{p}_{ij} - p_{ij}(t)) \leq A_1(t) + A_2(t).
\]
From the preceding inequality, it follows that
\[
\sum_{i,j} \lambda_i r_{ij} (\bar{p}_{ij} - p_{ij}(t)) \leq K \left( \sum_{t=1}^{T} A_1(t) + \sum_{t=1}^{T} A_2(t) \right) \quad \text{for} \quad K := \kappa^{-1}.
\]
Below, we bound the two terms in the apprentices on the right-hand side separately.

Recall that $h_{ij}(t)$ is the number of jobs of type $i$ assigned to server $j$ up to the end of time $t$. To avoid division by zero, we define $\hat{h}_{ij}(t) := \max\{h_{ij}(t), 1\}$ in the definition of the event $F_{ij}(t)$ below. Specifically, for each $(i,j)$ pair, let
\[
F_{ij}(t) := \left\{ \bar{r}_{ij}(t-1) - r_{ij} \leq \sqrt{2 \log(t-1) \over \hat{h}_{ij}(t-1)} \right\}.
\]
Let $Y_{ij}(t) := (r_{ij}(t) - r_{ij}) \leq 1$. Then
\[
\sum_{t=1}^{T} A_1(t) = \sum_{i,j} \sum_{t=1}^{T} Y_{ij}(t) \lambda_i p_{ij}(t)
= \sum_{i,j} \sum_{t=1}^{T} Y_{ij}(t) \lambda_i p_{ij}(t) 1_{F_{ij}(t)} + \sum_{i,j} \sum_{t=1}^{T} Y_{ij}(t) \lambda_i p_{ij}(t) 1_{\bar{F}_{ij}(t)}
\leq \sum_{i,j} \sum_{t=1}^{T} Y_{ij}(t) \lambda_i p_{ij}(t) 1_{F_{ij}(t)} + \sum_{i,j} \lambda_i p_{ij}(t) 1_{\bar{F}_{ij}(t)}.
\]
We first bound the expectation of the first component on the right-hand side of (EC.8). For this purpose, we use the definition of $F_{ij}(t)$ to get
\[
\sum_{t=1}^{T} Y_{ij}(t) \lambda_i p_{ij}(t) 1_{F_{ij}(t)} \leq 2 \sum_{t=1}^{T} \frac{2 \log(t-1)}{\hat{h}_{ij}(t-1)} \lambda_i p_{ij}(t) \leq 2 \sqrt{2 \log T} \sum_{t=1}^{T} \frac{1}{\hat{h}_{ij}(t-1)} \lambda_i p_{ij}(t). \quad (EC.9)
\]
Recall that $\Gamma_{ij}(t)$ is the actual number of type $i$ jobs assigned to server $j$ in slot $t$. Let $\Lambda$ denote a particular realization of the sequence of job arrival times up to the end of time $T$ and $E_\Lambda$ the conditional expectation operator given $\Lambda$. Then, by applying a martingale argument similar to the proof of Lemma 3 in Hsu et al. (2022), we can deduce that
\[
E_\Lambda \left[ \sum_{t=1}^{T} \sqrt{\frac{1}{\hat{h}_{ij}(t-1)}} \lambda_i p_{ij}(t) \right] = E_\Lambda \left[ \sum_{t=1}^{T} \sqrt{\frac{1}{\hat{h}_{ij}(t-1)}} \Gamma_{ij}(t) \right]. \quad (EC.10)
\]
Clearly,
\[
\sum_{t=1}^{T} \sqrt{\frac{1}{\hat{h}_{ij}(t-1)}} \Gamma_{ij}(t) = \sum_{t=1}^{T} \sqrt{\frac{1}{\hat{h}_{ij}(t-1)}} B_{ij}(t) + \sum_{t=1}^{T} \sqrt{\frac{1}{\hat{h}_{ij}(t-1)}} (\Gamma_{ij}(t) - B_{ij}(t)). \quad (EC.11)
\]
By definition, $B_{ij}(t) = h_{ij}(t) - h_{ij}(t-1)$. It follows that
\[
\sum_{t=1}^{T} \sqrt{\frac{1}{\hat{h}_{ij}(t-1)}} B_{ij}(t) = \sum_{t=1}^{T} \sqrt{\frac{1}{\hat{h}_{ij}(t-1)}} (h_{ij}(t) - h_{ij}(t-1)) \\
\leq 2 + \int_{1}^{h_{ij}(T-1)} \frac{1}{\sqrt{x}} \, dx \\
\leq 2 + 2 \sqrt{h_{ij}(T-1)}. \quad (EC.12)
\]
Using Abel’s summation formula (summation by parts) on the second term on the right-hand side of (EC.11), along with the identity
\[
Q_{ij}(t+1) = \sum_{s=1}^{t} \Gamma_{ij}(s) - \sum_{s=1}^{t} B_{ij}(s),
\]
leads to
\[
\sum_{t=1}^{T} \sqrt{\frac{1}{\hat{h}_{ij}(t-1)}} (\Gamma_{ij}(t) - B_{ij}(t)) = Q_{ij}(T) \sqrt{\frac{1}{\hat{h}_{ij}(T-1)}} + \sum_{t=1}^{T-1} Q_{ij}(t) \left( \sqrt{\frac{1}{\hat{h}_{ij}(t-1)}} - \sqrt{\frac{1}{\hat{h}_{ij}(t)}} \right) \\
\leq Q_{ij}^{\max}(T) \left[ \sqrt{\frac{1}{\hat{h}_{ij}(T-1)}} + \sum_{t=1}^{T-1} \left( \sqrt{\frac{1}{\hat{h}_{ij}(t-1)}} - \sqrt{\frac{1}{\hat{h}_{ij}(t)}} \right) \right] \\
\leq Q_{ij}^{\max}(T), \quad (EC.13)
\]
where we have defined $Q_{ij}^{\text{max}}(T)$ as the running maximum of $Q_{ij}$ up to the end of time $T$. Substituting (EC.12)–(EC.13) into (EC.11), we obtain

$$\sum_{t=1}^{T} \sqrt{\frac{1}{h_{ij}(t-1)}} \Gamma_{ij}(t) \leq 2 + 2 \sqrt{h_{ij}(T-1)} + Q_{ij}^{\text{max}}(T). \quad (\text{EC.14})$$

Next, combining (EC.14) with (EC.9) and (EC.10) yields

$$\mathbb{E} \left[ \sum_{i,j} \sum_{t=1}^{T} Y_{ij}(t) \lambda_{i} p_{ij}(t) 1_{F_{ij}(t)} \right] \leq 2 \sqrt{2 \log T} \left( 2IJ + 2 \sum_{i} \sum_{j} \sqrt{I \sum_{i} h_{ij}(T-1) + \sum_{i,j} \mathbb{E} \left[ Q_{ij}^{\text{max}}(T) \right]} \right) \leq 2 \sqrt{2 \log T} \left( 2IJ + 2J \sqrt{IT} + \sum_{i,j} \mathbb{E} \left[ Q_{ij}^{\text{max}}(T) \right] \right), \quad (\text{EC.15})$$

where the first inequality is due to the Cauchy-Schwartz inequality whereas the second inequality follows from the fact that $h_{j}(T-1) := \sum_{i} h_{ij}(T-1) \leq T$.

We next bound the expectation of the second component on the right-hand side of (EC.8). By apply Chernoff-Hoeffding Inequality (see, e.g., Ex. 7.1 in (Lattimore and Szepesvári 2020, Chapter 7), we have that

$$\mathbb{E} \left[ \sum_{i,j} \sum_{t=1}^{T} \lambda_{i} p_{ij}(t) 1_{F_{ij}(t)} \right] \leq \sum_{i} \sum_{j} \lambda_{i} \mathbb{P}(F_{ij}(t)) \leq \sum_{i,j} \lambda_{i} \left( 1 + \sum_{t=1}^{T-1} \frac{1}{t^4} \right) \leq \frac{7}{3} J \lambda. \quad (\text{EC.16})$$

Finally, combining (EC.8) with (EC.15)–(EC.16) gives us

$$\mathbb{E} \left[ \sum_{t=1}^{T} A_{1}(t) \right] \leq 2 \sqrt{2 \log T} \left( 2IJ + 2J \sqrt{IT} + \sum_{i,j} \mathbb{E} \left[ Q_{ij}^{\text{max}}(T) \right] \right) + \frac{7}{3} J \lambda. \quad (\text{EC.17})$$

Now, for each $(i, j)$ pair, define

$$G_{ij}(t) := \left\{ r_{ij} - \bar{r}_{ij}(t-1) \leq \sqrt{\frac{2 \log(t-1)}{h_{ij}(t-1)}} \right\}.$$

Then

$$\sum_{t=1}^{T} A_{2}(t) = \sum_{i,j} \sum_{t=1}^{T} \lambda_{i} (r_{ij} - r_{ij}(t)) \tilde{p}_{ij} \leq \sum_{i,j} \lambda_{i} (r_{ij} - r_{ij}(t)) \tilde{p}_{ij} 1_{G_{ij}(t)} + \sum_{i,j} \sum_{t=1}^{T} \lambda_{i} (r_{ij} - r_{ij}(t)) \tilde{p}_{ij} 1_{G_{ij}^{c}(t)} \quad (\text{EC.18})$$

$$\leq \sum_{i,j} \sum_{t=1}^{T} \lambda_{i} \tilde{p}_{ij} 1_{G_{ij}^{c}(t)},$$
where the inequality uses the fact that $r_{ij} < r_{ij}(t)$ on $G_{ij}(t)$ for each $(i,j)$ pair. Taking expectations on both sides of (EC.18) and applying Chernoff-Hoeffding Inequality yields

$$
E \left[ \sum_{t=1}^{T} A_2(t) \right] \leq \sum_{i,j} \lambda_i \sum_{t=1}^{T} \mathbb{P}(G_{ij}^c(t)) \leq \sum_{i,j} \lambda_i \left( 1 + \sum_{t=1}^{T-1} \frac{1}{t^4} \right) \leq \frac{7}{3} J \lambda. \quad \text{(EC.19)}
$$

Combining (EC.7), (EC.17) and (EC.19), we reach (11), hence completing the proof. \square

**Proof of Lemma EC.1.** Consider $\tilde{p}$, which solves the optimization problem (9), and recall that $\tilde{\beta}$ denote the optimal dual variables of the equality constraints in (9). By the KKT conditions,

$$
- \frac{1}{V} \frac{\lambda_i}{1 - \sum_i \lambda_i \tilde{p}_{ij}} + \lambda_i r_{ij} - \tilde{\beta}_i = 0 \quad \text{for all (i,j)}. \quad \text{(EC.20)}
$$

This, along with (EC.20), implies that

$$
\frac{\lambda_i}{1 - \sum_i \lambda_i \tilde{p}_{ij}} = V (\lambda_i r_{ij} - \tilde{\beta}_i) = V \lambda_i r_{ij} (1 - \tilde{\beta}_i / (\lambda_i r_{ij})).
$$

Note that (9) can be viewed as a series of optimization problems indexed by $V$, and that the series “converges” to the linear program (4) as $V \to \infty$. In particular, each $\tilde{\beta}_i$ converges to $\beta_i$ as $V \to \infty$. Thus, by letting $\kappa := \min_{i,j} \beta_i / (2 \lambda_i r_{ij})$, we have for sufficiently large $V$ that

$$
\frac{\lambda_i}{1 - \sum_i \lambda_i \tilde{p}_{ij}} \leq V \lambda_i r_{ij} (1 - \kappa).
$$

This completes the proof. \square