Integrated Online Learning and Adaptive Control in Queueing Systems with Uncertain Payoffs

Wei-Kang Hsu
School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47906, wkhsu@purdue.edu

Jiaming Xu
The Fuqua School of Business, Duke University, Durham, NC 27708, jiaming.xu868@duke.edu

Xiaojun Lin
School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47906, linx@ecn.purdue.edu

Mark R. Bell
School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47906, mrb@ecn.purdue.edu

We study task assignment in online service platforms, where un-labeled clients arrive according to a stochastic process and each client brings a random number of tasks. As tasks are assigned to servers, they produce client/server-dependent random payoffs. The goal of the system operator is to maximize the expected payoff per unit time subject to the servers' capacity constraints. However, both the statistics of the dynamic client population and the client-specific payoff vectors are unknown to the operator. Thus, the operator must design task-assignment policies that integrate adaptive control (of the queueing system) with online learning (of the clients' payoff vectors). A key challenge in such integration is how to account for the non-trivial closed-loop interactions between the queueing process and the learning process, which may significantly degrade system performance. We propose a new utility-guided online learning and task assignment algorithm that seamlessly integrates learning with control to address such difficulty. Our analysis shows that, compared to an oracle that knows all client dynamics and payoff vectors beforehand, the gap of the expected payoff per unit time of our proposed algorithm in a finite T horizon is bounded by $\beta_1/V + \beta_2\sqrt{\log N/N} + \beta_3 N(V+1)/T$, where V is a tuning parameter of the algorithm, and $\beta_1, \beta_2, \beta_3$ only depend on arrival/service rates and the number of client classes/servers. Through simulations, we show that our proposed algorithm significantly outperforms a myopic matching policy and a standard queue-length based policy that does not explicitly address the closed-loop interactions between queueing and learning.

Key words: Online learning, online service platforms, convex optimization, decentralized algorithms

History: This paper was first submitted on December 10, 2018.

1. Introduction

Driven by advances in communication and computing, new generations of online service platforms have emerged in many domains, from the online labor market of freelance work (e.g., Upwork (Upw 2017)), online hotel rental (e.g., Airbnb (Air 2017)), online education (e.g., Coursera (Cou 2017)), crowd-sourcing (e.g., Amazon Mechanical Turk (Ipeirotis 2010)), to online advertising (e.g., AdWords (AdW 2017)), and many more. By bringing an unprecedented number of clients and service providers together, these online service platforms greatly increase access to service, lower the barrier-to-entry for competition, improve resource utilization, reduce cost and delay, and thus have transformed or even disrupted the existing business models in many industries (Kenney and Zysman 2016, Evans and Gawer 2016, van Dijck et al. 2018).

One of the key control decisions in operating such online service platforms is how to assign clients to servers to attain the maximum system benefit. However, such decisions are challenging because there often exists significant uncertainty in both client payoffs and client dynamics. First, there is significant uncertainty in the quality, i.e., payoff, of a particular assignment between a client (needing service) and a server (providing service). These true payoff parameters are often unknown a priori and the online service platforms can only observe noisy payoff feedback from each past assignment. Thus, the platforms face a fundamental exploration-exploitation trade-off. Should the platforms "explore" to learn more about the payoff parameters from feedback, or "exploit" existing knowledge to maximize payoffs? Second, the population of clients is often highly dynamic. The statistics of such arrivals and departures are often unknown beforehand, resulting in queueing dynamics with unknown statistics. Further, when a new client arrives, it brings a new set of payoff parameters, which may need to be learned from scratch. As a result, the platform operator must not only continuously learn the payoffs associated with each new client, but also adaptively control the assignment and resource allocation in response to the uncertain arrival/departure dynamics. As we will demonstrate throughout the paper, this combination of uncertainty in payoff and uncertainty in client arrival/departure statistics leads to unique challenges, complicating both learning and control. Thus, the goal (and main contribution) of this paper is to design new algorithms that integrate both online learning (of uncertain payoffs) and adaptive control (of queueing dynamics with unknown statistics) in a unified framework to achieve provably-optimal performance in such a dynamic and uncertain environment.

1.1. High-level Model

To address the aforementioned challenges in operating online service platforms, we study the following queueing system model similar to Johari et al. (2016, 2017) (although there are a number of critical differences between our model and that of Johari et al. (2016, 2017), which will be explained shortly in Section 1.3 with additional details in Section EC.1). There are a fixed number of servers with known service rates. (This assumption of known service rates can be relaxed. See Section EC.7.) Clients arrive at the system with unknown arrival rates. Each client brings a random (unknown) number of tasks. Associated with each new client, there is a payoff vector over all servers. We assume that this payoff vector is randomly chosen from a discrete set of possible values according to a certain probability distribution. *However, both the random payoff vector of a new client and the distribution are unknown to the operator.* As the tasks of a client are assigned to different servers, they produce random payoff feedback with the mean given by the underlying payoff vector. Thus, the operator has to learn the payoff vector of a new client from the random payoff feedback, and then decide how to match clients with servers.

To illustrate our model, we provide three concrete examples:

1. *Online shopping*, e.g., Stitch Fix (Johari et al. 2016). Clients are customers who are interested in buying clothes, and servers are stylists who suggest clothes for customers. Customers arrive and depart constantly in the online shopping platforms. The payoff is the satisfaction of a customer when clothing is recommended by a stylist. Different customers may have different tastes for clothing. For example, some customers may prefer more formal attire, while others prefer more casual attire. Similarly, a particular stylist may be good at recommending a specific type of clothing. On the one hand, the features of new customers are unknown when they arrive. Thus, when a customer is

assigned to a stylist, there will be significant uncertainty in the payoff, i.e., whether a customer will buy a piece of clothing suggested by a stylist. On the other hand, the successes (clothes bought) or failures provide payoff feedback, which can only be observed after the customers are served and can then be used for learning.

2. *Online advertisement* (Combes et al. 2015, Tan and Srikant 2012). Clients are advertisers, and servers are keyword searches from different groups of users. Advertisers arrive and depart constantly in online ad platforms. The payoff is the click-through rate when an ad from a particular advertiser is displayed in response to a given keyword search request. An advertisement may have different appeals to different groups of users, even if they are searching for the same keyword. On the one hand, such an appeal may be unknown for a new advertisement. On the other hand, the clicks provide the payoff feedback, which can only be observed after displaying the ads and can then be used for learning.

3. *Crowdsourcing* (Karger et al. 2011a, 2013, 2011b, Khetan and Oh 2016). Clients are posted jobs, and servers are workers. Posted jobs arrive and depart constantly in crowdsourcing platforms. The payoff corresponds to the efficiency of completing a particular task from a posted job by a given worker. On the one hand, different jobs receive different completion efficiency when assigned to a group of workers. However, such completion efficiency is unknown for new jobs. On the other hand, the ratings/completion-times provide the payoff feedback, which can only be observed after completing the tasks and can then be used for learning.

1.2. Main Results

The main contribution of this paper is the development of new schemes that integrate online learning with adaptive control to address uncertainty in both payoff parameters and client dynamics. As we will explain shortly, a key finding is that the learning part and the control part of the algorithm must be designed together. This is because of the non-trivial closed-loop interactions between queueing and learning. Specifically, excessive *queueing*, which can be caused by both agent arrivals/departures and task assignment/completion, delays the collection of payoff feedback, and consequently, delays the *learning* process of the uncertain payoffs. Ineffective *learning*, in turn, leads to suboptimal assignment decisions based on inaccurate payoff estimates, which increases *queueing* delay and lowers system payoff under uncertain agent dynamics. As a result, simply putting together an algorithm for online learning (such as UCB (Auer et al. 2002a)) and an algorithm for control (such as queue-length based control) will not lead to good results because they fail to account for such closed-loop interactions (see Section EC.1 for details).

Instead, in this paper, we propose a new utility-guided online learning and task assignment algorithm that can achieve near-optimal system payoff even compared to an "oracle" that knows the statistics of client dynamics and all clients' payoff vectors beforehand. In particular, to address the aforementioned closed-loop interactions, our new algorithm carefully controls the number of tasks assigned to each server at each time to be no greater than the server capacity. As a result, it eliminates server-side queues and thus eliminates the delay in collecting payoff feedback. Further, a logarithmic utility function of the assignment probabilities of each client is added to the objective function of the policy decision. Such a utility function not only promotes fairness so that every client can learn, but also balances the current and future payoffs so that the whole system can adapt to unknown client dynamics. (Note that here we use the term "utility function" for the purpose of defining the decision of the algorithm rather than treating clients as strategic. In other words, it is independent of each client's own valuation.)

Analytically, we prove that, by employing these two ideas (i.e., eliminating server-side queues and adding a logarithmic utility function), our proposed algorithm successfully deals with the closed-loop interactions between queueing and learning. First, we show that our algorithm stabilizes the system in the sense that the mean number of backlogged clients in the system at any time is finite. Second and more importantly, we show that the gap between the expected payoff per unit time achieved by our proposed algorithm and that achieved by the "oracle" in a finite T horizon is at most the sum of three terms with natural physical interpretations (see details in Section 3.2):

1. The first term is of the order $1/V$, where V is a parameter of the proposed algorithm that can be taken to a large value. This term is due to the uncertainty in client arrival process.

2. The second term is of the order $\sqrt{\log N/N}$, where N is the average number of tasks per client. This term is related to the notion of "regret" in typical MAB problems and captures the payoff loss due to the uncertainty in payoffs as a function of the total number of tasks per client, i.e., the tradeoff between exploration and exploitation (Lai and Robbins 1985, Auer et al. 2002a).

3. The third one is of the order $N(V+1)/T$, which decreases as the time T increases. This term accounts for the payoff loss incurred by the backlogged clients over a finite time horizon.

The above payoff-gap bound is quite appealing as it separately captures the impact of the uncertainty in client dynamics and the impact of the uncertainty in payoffs for any finite time horizon T. If N and T are known in advance, we can tune V to be $\min\{T/N, \sqrt{T/N}\}$, so that the payoff-gap upper-bound is minimized and becomes of the order $\max\{N/T, \sqrt{N/T}\} + \sqrt{\log N/N}$. As a consequence, as T increases, the payoff gap eventually gets saturated at $\sqrt{\log N/N}$.

On the technical end, our proof builds upon the standard Lyapunov drift technique (Neely et al. 2005, Lin et al. 2008, De Veciana et al. 2001, Bonald and Massoulie 2001) and finite-time regret analysis (Auer et al. 2002a); however, we make two major innovations, which could be of independent interest. First, to obtain $\sqrt{\log N/N}$ loss in payoff learning, we combine a martingale argument and a comparison argument to capture both the impact of the payoff estimation errors on the queueing dynamics and the impact of congestion on the rate of payoff learning. Second, to derive the NV/T loss in a finite time horizon, we upper bound the mean number of backlogged clients in the system by establishing a coupling between the current system and a Geom/Geom/m queue.

Through simulations, we show that our proposed algorithm significantly outperforms a myopic matching policy and a standard queue-length based policy that does not explicitly address the aforementioned closed-loop interactions between queueing and learning. While this utility-guided algorithm is designed for the case when the service time at each server is deterministic, we can further derive a low-complexity and virtual-queue (dual) based algorithm (see Section EC.7 of the e-companion), which performs well empirically even when the service time is random and the mean service time is unknown. This algorithm is particularly appealing from a practical standpoint: The platforms only need to maintain a virtual queue for each server, and the task assignment and payoff learning for every client are completely decoupled given the virtual queues at the servers.

1.3. Comparison to Related Work

Note that our work deviates significantly from both classical online learning problems and adaptive control problems in the literature. On the one hand, while online learning has often been studied as a MAB (multi-armed bandit) problem, most existing studies ignore the client dynamics and focus on either one client (Lai and Robbins 1985, Auer et al. 2002a, Gittins et al. 2011, Whittle 1988) or a fixed set of clients (Kalathil et al. 2014, Anandkumar et al. 2011, Lai et al. 2011, Combes and Proutiere 2015, Buccapatnam et al. 2015). Our model is also different from open-bandit processes (where arms are dynamic and follow a Markov chain with known statistics (Lai and Ying 1988)), and contextual bandits (which assume known label, i.e., context, for each incoming client (Badanidiyuru et al. 2014)). On the other hand, although adaptive control and online matching policies under uncertain dynamics have been provided, e.g., for online ads (Tan and Srikant 2012, Feldman et al. 2009) or queueing networks in general (Tassiulas and Ephremides 1992, Neely et al. 2005, Lin et al. 2006, 2008), these studies usually assume that the payoff vectors of the clients are known. In contrast, our work aims to integrate both online learning (of uncertain payoffs) and adaptive control (under uncertain dynamics).

There are a few recent studies in the literature that also attempt to integrate learning and queueing (Johari et al. 2016, Massoulie and Xu 2016, Johari et al. 2017, Bimpikis and Markakis 2015, Shah et al. 2017). Compared to these related studies, our work makes a number of new contributions. First, recall that a key challenge in integrating learning with queueing is the non-trivial closed-loop interactions between the queueing process and the learning process, which, if not properly managed, can produce a vicious cycle that severely degrades the overall system performance (see further discussions in Section 2.1). While prior work in (Johari et al. 2016, Massoulie and Xu 2016, Johari et al. 2017) attempts to deal with these closed-loop interactions, their policies explicitly divide exploration (where learning occurs) and exploitation (where queueing and control occur) into two stages. In order to perform such an explicit division, these policies have to assume perfect knowledge of the various uncertainties, such as the class-dependent payoff vectors, the number of tasks per client, and/or the "shadow prices" (which in turn require knowledge of the statistics of the client arrival dynamics). Although (Johari et al. 2017) provides a heuristic policy that can replace the shadow prices with queue lengths (to learn underlying statistics and class-dependent payoff vectors), no performance guarantees are provided. On the contrary, our algorithm requires no such prior knowledge and seamlessly integrates exploration, exploitation, and dynamic assignment at all times.

Second, the analyses in (Johari et al. 2016, 2017) focus exclusively on the stationary setting. In contrast, our work carefully characterizes the transient behavior of the system and obtains a payoff gap that holds for any finite time horizon, providing important design insight when real systems either operate at early phases or experience recent changes of statistics.

Finally, our proposed algorithm can scale to large systems with many clients and servers, which is in sharp contrast to the previous work in both (Bimpikis and Markakis 2015) (which only deals with two types of clients) and (Shah et al. 2017) (which uses an exponential number of virtual queues).

1.4. Organization

The rest of the paper is organized as follows. The system model is defined and the main challenges are highlighted in Section 2. We present our utility-guided algorithm and the main analytical results in Section 3. The key proofs are sketched in Section 4. Simulation results of our utility-guided algorithm are shown in Section 5. Finally, we conclude and discuss possible future directions. Due to the page limits, we present our virtual-queue (dual) based algorithm, as well as other remaining proofs, in the e-companion to this paper.

2. System Model

We model an online service platform as a multi-server queueing system that can process tasks from a dynamic population of clients, as shown in Fig. 1. Note that on many online service platforms, the operator often has enough aggregate information on the servers' features, and servers arrive at and depart from the system at a slower time scale than clients (Johari et al. 2016, Massoulie and Xu 2016, Johari et al. 2017). Thus, we focus on the setting where there are a fixed number of servers with known service rates. (In Section EC.7, we extend our algorithm to more general settings with unknown service rates.)

Time is slotted, and the system is empty at time 0. At the beginning of each time-slot $t \geq 1$, a new client arrives with probability $\lambda/N < 1$, independently of other time-slots. Upon arrival, the client carries a random number of tasks that is geometrically distributed with mean N. Note that in this way, the total rate of arriving tasks is λ. A client leaves the system once all of her tasks have been served. Let $\mathcal{S} = \{1, 2, \dots, J\}$ denote the fixed set of servers. Each server $j \in \mathcal{S}$ can serve exactly μ_j tasks in a time-slot. Let $\mu = \sum_{j=1}^{J} \mu_j$ and assume $\lambda < \mu$. We assume that each new client l is associated with a payoff vector $\vec{C}^{l,*} = [C_1^{l,*}, \dots, C_J^{l,*}]$, where $C_j^{l,*}$ is the expected payoff when a task from client l is served by server j. However, this payoff vector is unknown to the operator. Instead, we assume that $\vec{C}^{l,*}$ is chosen randomly from a finite set of possible values, $\vec{C}_1^*, \dots, \vec{C}_I^*$, where the probability of $\vec{C}^{l,*} = \vec{C}_i^*$ is ρ_i, $i = 1, \dots, I$, independently of other clients. However, the operator does not know this probability distribution (nor the set of possible values). Further, the operator does not know the offered load λ and the expected number N of tasks per client.

What does the system operator know? The system does know the identity of the servers and their service rates μ_j. Another important quantity that the system can learn from is the noisy payoff feedback observed by the system after a task from a client is served by a server. Conditioned on the client's payoff vector being $\vec{C}^{l,*}$, this noisy payoff feedback is assumed to be a Bernoulli random variable with mean $\vec{C}^{l,*}$, (conditionally) independent of other tasks and servers.

The goal of the system is then to use the noisy payoff feedback to learn the payoff vectors of the clients and to assign their tasks to servers in order to maximize the total system payoff.

Such decisions must be adaptively made without knowing the statistics of the client arrivals and departures. At each time-slot t, let $n(t)$ be the total number of clients in the system (including any new arrival at the beginning of the time-slot). (Note that $n(0)=0$.) Let $p_j^l(t)$ be the expected number of tasks from the l-th client that are assigned to server j in this time-slot, $l=1,...,n(t)$. Then, the decision at time t of a policy Π maps the current state of the system (including all payoff feedback observed before time t) to the quantities $p_j^l(t)$. Such decisions lead to the evolution of the following two types of queues.

Queueing dynamics: First, let $q_j(t)$ denote the number of tasks waiting to be served at server j at the beginning of time-slot t. Let $Y_j^l(t)$ be the actual number of tasks from the l-th client that are assigned to server j at time-slot t, with mean given by $p_j^l(t)$. The dynamics of the task-queue $q_j(t)$ can then be described as

$$q_j(t+1)=\max\left[q_j(t)+\sum_{l=1}^{n(t)}Y_j^l(t)-\mu_j,0\right]. \tag{1}$$

Second, let $n_i(t)$ be the number of clients in the system at the beginning of time t whose payoff vector is equal to $\vec{C}_i^*$. For ease of exposition, in the rest of paper we will say that a client is of class i if its payoff vector is $\vec{C}_i^*$. Then, $n_i(t)$ is the number of class-i clients in the system at time t. (We caution that, while the operator sees $n(t)$, $n_i(t)$ cannot be directly observed because both the payoff vectors and the class labels of the clients are unknown.) The dynamics of the client-queue $n_i(t)$ can be described as

$$n_i(t+1)=n_i(t)+U_i(t+1)-D_i(t), \tag{2}$$

where $U_i(t+1)$ is the number of client arrivals of class i at the beginning of time $t+1$ and is Bernoulli with mean λ_i/N, and $D_i(t)$ is the number of client departures of class i at time-slot t. (Recall that a client leaves the system once all of her tasks have been served. We again caution that, while the operator sees the total number of departures $D(t)=\sum_{i=1}^{I}D_i(t)$, $D_i(t)$ cannot be directly observed because the class labels of the clients are unknown.) Because the total number of tasks per client is assumed to be geometrically distributed, conditioned on $[Y_j^l(t)]$ and the underlying class of the l-th client for $l = 1, \dots, n(t)$, the departure $[D_i(t), i \in I]$ is independent of everything else.

The expected payoff per unit time of such a policy Π for a given time horizon T is defined as

$$R_T(\Pi) = \frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{J}\mathbb{E}\left[\sum_{l=1}^{n(t)} p_j^l(t) C^*_{i(l),j}\right], \tag{3}$$

where $i(l)$ denotes the underlying (but unknown) class of the l-th client. In contrast, if an "oracle" knew in advance the statistics of the arrival and departure dynamics (i.e., λ, ρ_i and N), as well as the class label and payoff vector of each client, one can formulate the following linear program:

$$R^* = \max_{[p_{ij}]\geq 0} \quad \lambda \sum_{i=1}^{I} \rho_i \sum_{j=1}^{J} p_{ij} C^*_{ij} \tag{4}$$

$$\text{subject to} \quad \lambda \sum_{i=1}^{I} \rho_i p_{ij} \leq \mu_j \text{ for all servers } j, \tag{5}$$

$$\sum_{j=1}^{J} p_{ij} = 1 \text{ for all classes } i, \tag{6}$$

where p_{ij} is the probability that a task from a class-i client is assigned to server j. It is not difficult to show that R^* provides an upper bound for the expected payoff per unit time $R_T(\Pi)$ under any policy Π (Johari et al. 2016, 2017). Thus, our goal is to achieve a provably small gap $R^* - R_T(\Pi)$ for any given time T.

Remark 1. We note that our model assumes that the payoff vectors (and thus the class labels) of new clients are completely unknown to the operator. In some applications (e.g., online advertisement), the clients may possess known features that can be used to group the clients into types (e.g., clothes ad vs. electronics ad). However, even in these cases, there will still be significant uncertainty in the payoff vectors. Thus, our model captures the most challenging scenario where the prior type-information is either unavailable or not informative in reducing the uncertainty in the payoff vectors. On the other hand, we expect that our algorithm can also be extended to the scenarios when the type of a new client is known, while its payoff vector is randomly chosen from a set that is dependent on the type. We leave this extension for future work.

Remark 2. When there are both client dynamics and payoff uncertainty, a key new challenge for our algorithm design and analysis is the closed-loop interactions between the queueing dynamics and learning dynamics:

1. In one direction, *queueing* degrades *learning* in at least two ways. First, because the operator only learns the noisy payoff of a task *after* the task is served, excessive task-queues at servers not only delay the service of the tasks, but also delay the payoff feedback. As a result, the learning of the clients' uncertain payoff vectors is also delayed. (We will refer to this effect as the "learning delay.") Second, because all clients compete for the limited server capacity, the learning process of some clients will inevitably be slowed down, which also results in a poor estimate of their payoff vectors. (We will refer to this second effect as the "learning slow-down.")

2. In the opposite direction, ineffective *learning* in turn affects *queueing* because assignment decisions have to be made based on delayed or inaccurate payoff estimates. Such sub-optimal decisions not only lower system payoff, but also increase queueing delay.

Together, the above closed-loop interactions between queueing and learning may produce complex system dynamics that can severely degrade system performance. (See also the discussions in Section EC.1 under "Queue-Length Based Control.") Our main contribution is thus to develop new algorithms and proof techniques to address such interactions. (See also the discussions at the beginning of Section 3 and Section 4.)

2.1. Limitations of myopic matching and queue-length based control policies

Before we present our solution to this problem, we outline the limitations of some related approaches in order to motivate our new algorithm. We first illustrate that a myopic-matching strategy, where clients and servers are matched to maximize the instantaneous system payoff, is sub-optimal even when the system payoff vectors are known in advance.

Example (Myopic matching is suboptimal): Assume two servers, each of service rate $\mu_j = 1$ as shown in Fig. 2. There are two classes, and a new client belongs to one of the two classes with an equal probability of 0.5. The payoff vectors of the two classes are [0.9 0.1] and [0.9 0.3], respectively.

Assume $N = 100$ and $\lambda/N = 1.2/N$. It is easy to see that the optimal solution should assign all class-1 tasks to server 1, 2/3 of class-2 tasks to server 1, and the rest 1/3 of class-2 tasks to server 2. The resulting optimal payoff is $0.6*0.9+0*0.1+0.4*0.9+0.2*0.3 = 0.96$. However, if a myopic matching strategy is used, with close-to-1 probability two tasks will be assigned at each time, one of which will have to be assigned to server 2. Thus, the overall payoff is approximately upper-bounded by $1.2*(0.9+0.3)/2 = 0.72$. As we can see, the myopic matching strategy focuses too much on maximizing the current payoff, and as a result sacrifices future payoff opportunities. Naturally, if the class labels and payoff vectors are unknown and need to be learned, a similar inefficiency will incur.

Another type of algorithm that is closely related to our problem setting is the queue-length based policy. When the payoff vector of each client is known in advance, the queue-length based control policy can be shown to attain near-optimal system payoff (by building large server-side queues) (Tan and Srikant 2012). However, when the payoff vector of each client is unknown, such a queue-length based control will no longer produce high long-term payoff. The underlying reason is explained in Remark 2 above, where the large server-side queues leads to a vicious cycle between queueing and learning. Indeed, as we will demonstrate in Section 5, a straightforward version of such queue-length based control will lead to a lower system payoff when combined with online learning. Readers can refer to Section EC.1 for more discussions of myopic matching and queue-length based control policies in related work.

3. Dynamic Assignment with Online Learning under Uncertainties

In this section, we present a new algorithm that seamlessly integrates online learning with dynamic task assignment to address the aforementioned closed-loop interactions. Unlike classical queue-length based control policies that rely on the task-queues as the "shadow prices," we instead eliminate the task-queues at the servers, and thus eliminate the "payoff-feedback delay" altogether. This is achieved by controlling the number of tasks assigned to each server j at each time-slot to be always no greater than its service rate μ_j. Therefore, the length of the server-side task-queue

$q_j(t)$ given by (1) is trivially zero at all times. However, without the server-side task-queues or "shadow prices," we need another congestion indicator to guide us in the dynamic assignment of tasks. Here, we propose to rely on the number of backlogged clients in the system. Specifically, we use the solution to a utility-maximization problem (that is based on the current number of backlogged clients in the system) to help us trade off between the current and future payoffs. The parameters of the utility-maximization problem are carefully chosen to also control the *learning slow-down* of all clients in a fair manner. We remark that while the structure of our proposed utility-guided algorithm bears some similarity to that of flow-level congestion control in communication networks (Lin et al. 2008, De Veciana et al. 2001, Bonald and Massoulie 2001), the focus therein was only on the system stability (but not the system payoff), and there was no online learning of uncertain parameters. *To the best of our knowledge, our work is the first in the literature that uses utility-guided adaptive control policies with online learning to optimize system payoff.*

3.1. Utility-Guided Dynamic Assignment with Online Learning

We present our algorithm in Algorithm 1. At each time-slot t, the algorithm operates in three steps. Step 1 generates an estimate $[C_j^l(t)]$ of the payoff vector for each client l based on previous payoff feedback (Lines 4-7). Note that $n(t)$ is the current number of clients in the system, including any newly-arriving client. For each client $l = 1, ..., n(t)$, if no task of this client has been assigned to server j yet, we set $C_j^l(t) = 1$ (Line 6); otherwise, we use a truncated Upper-Confidence-Bound (UCB) estimate (Auer et al. 2002a) to generate $C_j^l(t)$ (Line 7):

$$C_j^l(t) = \min\left\{\overline{C}_j^l(t-1) + \sqrt{\frac{2\log h^l(t-1)}{h_j^l(t-1)}},\ 1\right\}, \tag{9}$$

where $h_j^l(t-1)$ is the number of tasks from client l that have previously been assigned to server j before the end of the $(t-1)$-th time-slot and $h^l(t-1) = \sum_{j=1}^J h_j^l(t-1)$; $\overline{C}_j^l(t-1)$ is the empirical average payoff of client l based on the received noisy payoff feedback for server j until the end of the $(t-1)$-th time-slot. Since the true payoffs C_{ij}^*'s are within $[0,1]$, we truncate the UCB estimate at threshold 1 in (9). Then, Step 2 solves a maximization problem (Line 9), subject to the capacity

Algorithm 1: Utility-Guided Dynamic Assignment with Online Learning

```
1  For every time slot $t$:
2  Update the total number of clients $n(t)$ (including newly arriving clients)
3  Step 1: Form truncated UCB payoff estimates
4  for $l = 1 : n(t)$ do
5      for $j = 1 : J$ do
6          if $h_j^l(t-1) = 0$ or client $l$ is new then $C_j^l(t) \leftarrow 1$; $h_j^l(t) \leftarrow 0$;
7          else Set $C_j^l(t)$ according to (9) ;
8  Step 2: Solve $[p_j^l(t)]$ for the optimization problem
```

$$\max_{[p_j^l] \geq 0} \sum_{l=1}^{n(t)} \left\{ \frac{1}{V} \log \left(\sum_{j=1}^{J} p_j^l \right) + \sum_{j=1}^{J} p_j^l (C_j^l(t) - \gamma) \right\} \tag{7}$$

$$\text{sub to} \quad \sum_{l=1}^{n(t)} p_j^l \leq \mu_j, \quad \text{for all servers } j \in \mathcal{S}. \tag{8}$$

```
9  Step 3: Assign tasks and obtain noisy payoff feedback
10 Initialize to zero the number of tasks from client $l$ to be assigned to server $j$, i.e., $Y_j^l(t) = 0$.
11 for $j = 1 : J$ do
12     for $\nu = 1 : \mu_j$ do
13         Choose a client $l^*$ randomly such that the probability of choosing client $l$ is equal to
            $p_j^l(t)/\mu_j$. Assign one task from client $l^*$ to server $j$ and let $Y_j^{l^*}(t) \leftarrow Y_j^{l^*}(t) + 1$ ;
14 for $l = 1 : n(t)$ do
15     for $j = 1 : J$ do
16         Observe $Y_j^l(t)$ number of Bernoulli payoffs $X_j^l(t,1), \dots, X_j^l\left(t, Y_j^l(t)\right) \overset{\text{i.i.d.}}{\sim} \text{Bern}\left(C^*_{i(l),j}\right)$;
17         Update $h_j^l(t)$ and $\overline{C}_j^l(t)$ according to $Y_j^l(t)$ and $X_j^l(t,\cdot)$:
18         $h_j^l(t) = h_j^l(t-1) + \sum_{j=1}^J Y_j^l(t)$ ;
19         $\overline{C}_j^l(t) = \left(h_j^l(t-1)\overline{C}_j^l(t-1) + \sum_{k=1}^{Y_j^l(t)} X_j^l(t,k)\right)/h_j^l(t)$ ;
20     $h^l(t) = \sum_{j=1}^J h_j^l(t)$ ;
21 Clients with no remaining tasks leave the system.
```

constraint of each server, to obtain the values of $p_j^l(t)$, which corresponds to the expected number of new tasks of client l to be assigned to server j in time-slot t. The objective (7) can be viewed as the sum of some utility functions over all clients. The parameter γ in (7) is chosen to be strictly larger than 1, and V is a positive parameter. Finally, Step 3 determines the exact number $Y_j^l(t)$

of tasks from client l that are assigned to server j (Lines 12-14). The values of $Y_j^l(t)$ are randomly chosen in such a way that (i) each server j receives at most μ_j tasks, i.e., $\sum_{l=1}^{n(t)} Y_j^l(t) \le \mu_j$; and (ii) the expected value of $Y_j^l(t)$ is equal to $p_j^l(t)$. The tasks are then sent to the servers, and new noisy payoffs are received (Lines 15-19).

Before we present our analytical results, we make several remarks on the design of our proposed algorithm.

Seamless Integration of Learning and Control: First, recall that the policies in (Johari et al. 2016, Massoulie and Xu 2016, Johari et al. 2017) separate exploration and exploitation into distinct stages by assuming perfect knowledge of class-dependent payoff vectors, the total number of tasks per client, or "shadow prices." In contrast, our algorithm does not require such prior knowledge at all and seamlessly integrates exploration, exploitation, and dynamic assignment at all times.

Zero Payoff-Feedback Delay: Second, note that by Step 3 of the algorithm, the number of assigned tasks to each server j is no more than the service rate μ_j. Since server j can serve exactly μ_j tasks per time-slot, there is no longer any "payoff-feedback delay," i.e., all payoff feedback will be immediately revealed at the end of the time-slot.

The Importance of Fairness: Third, if we removed the logarithmic term in (7) and set $\gamma = 0$, then the maximization problem in (7) would have become a myopic matching policy that maximizes the total payoff in time-slot t based on the current payoff estimates. However, such a myopic matching policy focuses too much on the current payoff, and as a result, underperforms in terms of the long-term average payoff. Instead, the logarithmic term in (7) serves as a concave utility function that promotes fairness (Lin et al. 2008, De Veciana et al. 2001, Bonald and Massoulie 2001), so that even clients with low payoff estimates can still receive some service (i.e., $\sum_{j=1}^{J} p_j^l(t)$ is always strictly positive). This fairness property is desirable in two ways. First, it has an eye for the future (which is somewhat related to the fact that fair congestion-control ensures long-term system stability (Bonald and Massoulie 2001)), so that we can strike the right balance between current and future payoffs. Second, it also controls the *learning slow-down* of all clients in a fair manner. Specifically, note that the rate with which tasks of client l are assigned to the servers at time t is given by $\sum_{j=1}^{J} p_j^l(t)$, which determines how quickly (or slowly) the system can receive payoff feedback for client l and improve her payoff estimate. We can thus refer to this value as the "learning rate" of client l. Thanks to the aforementioned fairness property, we can show that the "learning rate" of each client based on imprecise payoff estimates will not be too far off from the "learning rate" of the client based on its true payoff vector, which constitutes a crucial step in our analysis (see Section 4.4 and the first paragraph in Appendix EC.2.4).

Parameter Choice: Fourth, the choice $\gamma > 1$ in (7) and the truncation of the UCB estimates in (9) also play a crucial role in achieving near-optimal system payoff. If $\gamma = 0$, then the system will tend to increase $\sum_{j=1}^{J} p_j^l(t)$ for every client l by utilizing as many available servers as possible, *even when the payoff from the server is low.* Such an aggressive approach typically leads to suboptimal performance because, in many settings, the optimal policy must assign some clients to only the high-payoff servers. Instead, by choosing $\gamma > 1 \geq C_j^l(t)$, the second term inside the summation in (7) becomes negative. As a result, the operator assigns clients to low-payoff servers only if the derivative of the logarithmic term, which is equal to $1/[V \sum_{j=1}^{J} p_j^l(t)]$, is sufficiently large. This feature leads to an inherent "conservativeness" of our algorithm in choosing low-payoff servers. As the parameter V increases, on the one hand, our algorithm becomes more and more conservative in choosing low-payoff servers, which benefits the long-term payoff. On the other hand, the number of clients backlogged in the system increases, leading to extra payoff loss for a finite time horizon T. Our theoretical results below will capture this trade-off.

Complexity: Last but not least, the maximization problem in Step 2 is a convex program and can be effectively solved. Thus, our proposed algorithm can scale to large systems with many clients and servers, which is in contrast to the work in both (Bimpikis and Markakis 2015) (which only deals with two types of clients) and (Shah et al. 2017) (which uses an exponential number of virtual queues).

3.2. Main Results

We now present our main results for the performance guarantees of the proposed algorithm. For simplicity, we denote $\lambda_i = \lambda\rho_i$, and hence $\lambda = \sum_{i=1}^{I}\lambda_i$. Recall that $\mu = \sum_{j=1}^{J}\mu_j$ and $n(0) = \sum_{i=1}^{I} n_i(0) = 0$. Further, the service rates are assumed to be fixed and independent of the client classes. Our first result is regarding the stability of the system. Note that even if we assume that the arrival rate λ is strictly less than the total service capacity μ, the stability of our proposed policy is not automatic. This is because our proposed policy is not work-conserving. In particular, it may not use some low-payoff servers even if they are available. Indeed, as shown by the Example in Section 2, such avoidance of low-payoff servers is essential for achieving low regret in the long run. Fortunately, Theorem 1 below shows that our policy, albeit not always work-conserving, still maintains the system stability. In fact, the result directly gives an upper bound on the average number of backlogged clients in the system at any time t under our proposed Algorithm 1. A key idea underlying our proof is to show that, when the number of backlogged clients in the system is large enough, our policy indeed becomes work-conserving, i.e., all servers must be busy (See Lemma EC.3).

THEOREM 1. *Suppose that the arrival rate λ is strictly less than the total service capacity, i.e., $\lambda < \mu$. Then for any time t,*

$$\mathbb{E}\left[n(t)\right] \leq \frac{2\mu}{\mu-\lambda}\left(1+\frac{\mu^2\gamma}{\gamma-1}\right)+\mu\gamma V. \tag{10}$$

Theorem 1 immediately implies the system is stable in the mean (Kumar and Meyn 1995). i.e.,

$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}[n(t)] < \infty.$$

The upper bound in (10) characterizes the effect of V. As V increases, our algorithm becomes more conservative in choosing low-payoff servers, leading to more clients backlogged in the system.

Departing from the standard Lyapunov technique in proving system stability (Lin et al. 2008, De Veciana et al. 2001, Bonald and Massoulie 2001), our proof of Theorem 1 relies on a careful coupling between $n(t)$ and a Geom/Geom/μ queue with Bernoulli arrivals and Binomial departures (details in Appendix EC.3).

The second result below further characterizes the payoff gap of the proposed algorithm compared to the oracle.

THEOREM 2. *Suppose $N \log N \geq 1$. The gap between the upper bound (4) and the expected payoff of Algorithm 1 at any time horizon T is bounded by:*

$$\sum_{i=1}^{I}\sum_{j=1}^{J}\lambda_i p_{ij}^* C_{ij}^* - \frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{J}\mathbb{E}\left[\sum_{l=1}^{n(t)} p_j^l(t) C_{i(l),j}^*\right] \leq \frac{\beta_1}{V} + \beta_2\sqrt{\frac{\log N}{N}} + \beta_3\frac{N(V+1)}{T}, \tag{11}$$

where $[p_{ij}^*]$ is the optimal solution to (4), and

$$\begin{aligned}
\beta_1 &= \frac{I}{2} + \frac{\mu^2}{2}\left(1 + \frac{\gamma^2}{(\gamma-1)^2}\right)\sum_{i=1}^{I}\frac{1}{\lambda_i},\\
\beta_2 &= 4\sqrt{2}\lambda\left(\sqrt{J}+\mu\right) + 3\lambda\left(1 + \frac{J\gamma^2}{(1-\gamma)^2}\right)\mu,\\
\beta_3 &= \frac{2\mu\gamma}{\mu-\lambda} + \frac{2\mu^3\gamma^2}{(\mu-\lambda)(\gamma-1)} + \mu\gamma^2.
\end{aligned} \tag{12}$$

To the best of our knowledge, this is the first result in the literature that characterizes the payoff gap for this type of utility-guided adaptive controller under payoff uncertainty. The above result is quite appealing as it separately captures the impact of the uncertainty in client dynamics and the impact of the uncertainty in payoffs for any finite time horizon T. In (11), the first term on the right-hand-side is of the order $1/V$, capturing the impact of the uncertainty in client dynamics (e.g., we do not know what the values of λ_i and N are). The second term in (11) is of the order $\sqrt{\log N/N}$ and related to the notion of "regret" in typical MAB problems. It captures the payoff loss due to the uncertainty in payoffs as a function of the total number of tasks per client, i.e., the trade-off between exploration and exploitation (Lai and Robbins 1985, Auer et al. 2002a). The third term in (11) is of the order $N(V+1)/T$, characterizing the payoff loss incurred by clients backlogged in the system. Given T, the first term and the third term reveal an interesting tradeoff as V increases. On the one hand, the first term will approach zero at the speed of $1/V$, indicating that the policy adapts to the unknown client dynamics; on the other hand, the third term will increase linearly with V due to the payoff loss incurred by clients backlogged in the system. If N

and T are known in advance, we can tune V to be $\min\{T/N, \sqrt{T/N}\}$, so that the payoff-gap upper-bound is minimized and becomes of the order $\max\{N/T, \sqrt{N/T}\} + \sqrt{\log N/N}$. As a consequence, as T increases, the payoff gap first decreases at a rate of T^{-1} for $T \leq N$ and then at a rate of $T^{-1/2}$ for $N \leq T \leq N^2/\log N$ and finally gets saturated at $\sqrt{\log N/N}$ for $T \geq N^2/\log N$.

The $\sqrt{\log N/N}$ regret here may seem inferior compared to the $\log N/N$ regret in (Johari et al. 2016, 2017). However, the regret in (Johari et al. 2016, 2017) only holds under the stationary setting $T \to \infty$ and assuming knowledge of various uncertainty, such as the class-dependent payoff vectors (being given and fixed), the number of tasks per client, or "shadow prices" (which in turn require knowledge of the statistics of the client arrival dynamics). In contrast, our payoff gap characterizes the transient behavior of the system and holds for any finite time T and finite $N \geq 2$ without any such prior knowledge. Moreover, our payoff gap holds in the minimax setting, where the payoff vector can be adversarially chosen from any distribution with a finite support. In fact, our payoff reget $\sqrt{\log N/N}$ matches the optimal minimax regret bound $\sqrt{1/N}$ up to a $\sqrt{\log N}$ factor (Auer et al. 2002b). It remains open whether our $\sqrt{\log N/N}$ regret can be further reduced to $\log N/N$ in the instance-dependent setting where the payoff vector is chosen from a given and fixed distribution.

In addition, we remark that our proof of Theorem 2 (see eq. (24)) in fact derives a smaller β_2 than that reported in Theorem 2, i.e.,

$$\beta_2 = 4\lambda\sqrt{2J} + 4\sqrt{2}\lambda\mu\frac{1}{\sqrt{N}} + 3\lambda\left(1 + \frac{J\gamma^2}{(1-\gamma)^2}\right)\mu\frac{1}{\sqrt{N\log N}}.$$

(The value of β_2 reported in Theorem 2 follows from invoking $N\log N \geq 1$.) As a consequence, when N is large, β_2 is dominated by $4\lambda\sqrt{2J}$, and hence our regret bound $\beta_2\sqrt{\log N/N}$ grows as the square root of the number of servers/arms, which coincides with the dependency of the optimal minimax regret bound $\sqrt{J/N}$ on J (Auer et al. 2002b).

Our proof of Theorem 2 builds upon the standard Lyapunov drift technique (Neely et al. 2005, Lin et al. 2008, De Veciana et al. 2001, Bonald and Massoulie 2001) and finite-time regret analysis (Auer et al. 2002a); however, our proof employs new techniques to carefully account for the closed-loop interactions between the queueing dynamics and the learning processes (see Section 4).

4. Main Proofs

Due to the space constraint, here we sketch the proof of Theorem 2. The omitted details, along with the proof of Theorem 1, will be given in the e-companion to this paper.

We first give an overview of the main steps of our proof, highlighting the key innovations:

1. Use *Lyapunov drift plus payoff gap* analysis to show that the payoff gap of Algorithm 1 is upper bounded by roughly $1/V + N(V+1)/T + \frac{1}{TV}\sum_{t=1}^{T}\mathbb{E}[A_1(t)]$ (cf. Lemma 1 and (23)). While similar Lyapunov drift analysis has been used for other stochastic optimization problems, the rest of the steps will focus on bounding $\mathbf{E}[A_1(t)]$, which is our new contribution.

2. Use the concavity of the objective function in Algorithm 1 to show $A_1(t) \leq A_2(t) + A_3(t)$, where $A_2(t)$ captures the payoff loss due to the *payoff estimation errors*, and $A_3(t)$ captures the payoff loss due to the *task assignment errors* (cf. Lemma 2).

3. Show that $\frac{1}{TV}\sum_{t=1}^{T}\mathbb{E}[A_2(t)]$ is upper bounded by roughly $\sqrt{\log N/N}$ (cf. Lemma 3 and (31)). This part extends the finite-time regret analysis for a standard MAB problem, but with a key difference. In standard MAB problems, exactly one arm is pulled at each time. In our setting, the servers/arms are randomly chosen according to probabilities, which themselves are random and depend on the system states. Therefore, we have to use a new and delicate martingale argument to obtain the regret bound.

4. Show that $\frac{1}{TV}\sum_{t=1}^{T}\mathbb{E}[A_3(t)]$ is upper bounded by roughly $1/N$ (cf. Lemma 4 and (32)). Compared to step (c), this step is even more difficult because the rate with which regret is accumulated is different from the rate with which the servers are chosen (see Remark 3 in Section 4.4). The key to resolve this difficulty is to exploit the fairness property of Algorithm 1: no client's task assignment probabilities can be heavily skewed by UCB payoff estimates (cf. Lemma EC.2).

4.1. Equivalent Reformulation

We will use a Lyapunov-drift analysis with special modifications to account for the learning of uncertain payoffs. Note that for the purpose of this drift analysis, we can use the underlying class label of each client to keep track of the system dynamics, even though our algorithm does not know this underlying class label. Thus, we re-label the $n(t)$ clients at time-slot t as follows. Recall that $n_i(t)$ is the number of clients at time-slot t whose underlying class is i. Let $\mathcal{I}(t) = \{i : n_i(t) \geq 1\}$. We have $\sum_{i\in\mathcal{I}(t)} n_i(t) = n(t)$. For each class $i \in \mathcal{I}(t)$, we use $k = 1, ..., n_i(t)$ to index the clients of this class at time t. Similar to the notations $C^*_{i(l),j}, \overline{C}^l_j(t)$ and $C^l_j(t)$, we denote $C^*_{ij}, \overline{C}^k_{ij}(t)$ and $C^k_{ij}(t)$ as the true expected value, empirical average of past payoffs at time t, and the UCB estimate at time t, respectively, for the payoff of server j serving tasks from the k-th client of the underlying class i. We also define $h^k_{ij}(t)$, $h^k_i(t) = \sum_{j=1}^J h^k_{ij}(t)$, and $p^k_{ij}(t)$ analogously to $h^l_j(t)$, $h^l(t)$ and $p^l_j(t)$. Thus, the UCB estimate (9) in Step 1 of our proposed Algorithm 1 is equivalent to

$$C^k_{ij}(t) \leftarrow \min\left\{\overline{C}^k_{ij}(t-1) + \sqrt{\frac{2\log h^k_i(t-1)}{h^k_{ij}(t-1)}}, 1\right\}, \tag{13}$$

and the maximization problem (7) in Step 2 of our proposed Algorithm 1 is equivalent to:

$$\max_{[p^k_{ij}]\geq 0} \sum_{i\in\mathcal{I}(t)} \sum_{k=1}^{n_i(t)} \left\{ \frac{1}{V} \log\left(\sum_{j=1}^J p^k_{ij}\right) + \sum_{j=1}^J p^k_{ij}\left(C^k_{ij}(t) - \gamma\right) \right\} \tag{14}$$

$$\text{s.t.} \sum_{i\in\mathcal{I}(t)} \sum_{k=1}^{n_i(t)} p^k_{ij} \leq \mu_j, \text{ for all } j \in \mathcal{S}. \tag{15}$$

Our proof below will use these equivalent forms of the proposed Algorithm 1.

In the rest of the analysis, we will use boldface variables (e.g., $\mathbf{n}$, $\mathbf{p}$, and $\mathbf{C}$) to denote vectors, and use regular-font variables (e.g., n_i, p^k_{ij}, and C^k_{ij}) to denote scalars. A list of notations is provided in Table 1.

4.2. Handling Uncertain Client Dynamics

Recall that $\lambda_i = \lambda\rho_i$, $\lambda = \sum_{i=1}^I \lambda_i$ and $\mu = \sum_{j=1}^J \mu_j$. Define the vector $\mathbf{n}(t) = [n_i(t), 1 \leq i \leq I]$ and $\mathbf{p}(t) = [p^k_{ij}(t), i \in \mathcal{I}(t), 1 \leq j \leq J, 1 \leq k \leq n_i(t)]$. Define the Lyapunov function $L(\mathbf{n}(t))$ as

$$L(\mathbf{n}(t)) = \frac{1}{2}\sum_{i=1}^I \frac{n_i^2(t)}{\lambda_i}. \tag{16}$$

Recall that $[p^*_{ij}]$ is the optimal solution of the upper bound in (4). Note that if we replace each C^*_{ij} in (4) by $C^*_{ij} - \gamma$, this will result in the same optimal solution. Thus, $[p^*_{ij}]$ is also the optimal solution to the following optimization problem:

$$\max_{[p_{ij}]\geq 0} \sum_{i=1}^I \lambda_i \sum_{j=1}^J p_{ij}(C^*_{ij} - \gamma), \text{ subject to (5) and (6)}. \tag{17}$$

Next, we will add a properly-scaled version of the following term to the drift of the Lyapunov function $L(\mathbf{n}(t+1)) - L(\mathbf{n}(t))$:

$$\Delta(t) = \sum_{i=1}^{I}\sum_{j=1}^{J} \lambda_i p_{ij}^*(C_{ij}^* - \gamma) - \sum_{i\in\mathcal{I}(t)}\sum_{k=1}^{n_i(t)}\sum_{j=1}^{J} p_{ij}^k(t)(C_{ij}^* - \gamma). \tag{18}$$

The value of $\Delta(t)$ captures the gap between the achieved payoff and the upper bound in (17), both adjusted by γ. The following lemma bounds the Lyapunov drift plus this payoff gap.

Lemma 1. *The expected drift plus payoff gap is bounded by*

$$\begin{aligned}&\mathbb{E}\left[L(\mathbf{n}(t+1)) - L(\mathbf{n}(t)) + \frac{V}{N}\Delta(t) \mid \mathbf{n}(t), \mathbf{p}(t)\right]\\ &\le \frac{A_1(t)}{N} + \frac{V}{N}\sum_{i\notin\mathcal{I}(t)}\sum_{j=1}^{J}\lambda_i p_{ij}^*(C_{ij}^* - \gamma) + \frac{c_1+c_2}{N},\end{aligned} \tag{19}$$

where $c_1 = \frac{I}{2}$, $c_2 = \frac{\mu^2}{2}\left(1 + \frac{\gamma^2}{(\gamma-1)^2 N}\right)\sum_{i=1}^{I}\frac{1}{\lambda_i}$, *and*

$$A_1(t) \triangleq \sum_{i\in\mathcal{I}(t)}\sum_{k=1}^{n_i(t)}\sum_{j=1}^{J}\left(\frac{n_i(t)}{\lambda_i} + V(C_{ij}^* - \gamma)\right)\left(\frac{\lambda_i p_{ij}^*}{n_i(t)} - p_{ij}^k(t)\right). \tag{20}$$

The proof is given in the e-companion to this paper.

4.3. Proof of Theorem 2

With Lemma 1, we are ready to present the proof of Theorem 2. Recall that in Step 2 of our proposed algorithm, we have chosen $\gamma > 1 \ge C_{ij}^*$. Hence, the second term on the right-hand-side of (19) is always less than 0. Then, taking expectations of (19) over $\mathbf{n}(t)$ and $\mathbf{p}(t)$, summing over $0 \le t \le T-1$, and divided by T, we have

$$\frac{1}{T}\mathbb{E}\left[L(\mathbf{n}(T)) - L(\mathbf{n}(0))\right] + \frac{V}{TN}\sum_{t=0}^{T-1}\mathbb{E}\left[\Delta(t)\right] \le \frac{1}{NT}\sum_{t=0}^{T-1}\mathbb{E}\left[A_1(t)\right] + \frac{c_1+c_2}{N}.$$

Since the system at time $t=0$ is empty, i.e., $n(t)=0$, it follows that $L(\mathbf{n}(0)) = 0$. In view of $L(\mathbf{n}(T)) \ge 0$, the last displayed equation gives

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\Delta(t)\right] \le \frac{1}{TV}\sum_{t=0}^{T-1}\mathbb{E}\left[A_1(t)\right] + \frac{c_1+c_2}{V}. \tag{21}$$

Let R_T denote the expected payoff per unit time achieved by Algorithm 1 for a given time T as defined in (3). By definitions of $\Delta(t)$, R_T, and R^*, we get that

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[\Delta(t)\right]=R^*-R_T-\gamma\lambda+\frac{\gamma}{T}\sum_{t=0}^{T-1}\sum_{j=1}^{J}\mathbb{E}\left[\sum_{i\in I(t)}\sum_{k=1}^{n_i(t)}p_{ij}^k(t)\right]. \tag{22}$$

We show in Appendix EC.4 that the total number $D(t)$ of departures at time t satisfies

$$\mathbb{E}\left[D(t)\mid \mathbf{n}(t),\mathbf{p}(t)\right]\le\frac{1}{N}\sum_{i\in I(t)}\sum_{j=1}^{J}\sum_{k=1}^{n_i(t)}p_{ij}^k(t).$$

It follows that

$$\begin{aligned}\lambda-\frac{1}{T}\sum_{t=0}^{T-1}\sum_{j=1}^{J}\mathbb{E}\left[\sum_{i\in I(t)}\sum_{k=1}^{n_i(t)}p_{ij}^k(t)\right]&\le\lambda-\frac{N}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[D(t)\right]=\frac{N}{T}\sum_{t=0}^{T-1}\mathbb{E}\left[U(t+1)-D(t)\right]\\&=\frac{N}{T}\mathbb{E}\left[n(T)\right]\le\frac{\beta_3N(V+1)}{\gamma T},\end{aligned}$$

where $U(t)$ denotes the total number of arrivals at time t, and the last inequality holds in view of Theorem 1 and the definition of β_3 in (12). By combining the last displayed equation with (22) and (21), we get that

$$R^*-R_T\le\frac{\beta_3N(V+1)}{T}+\frac{1}{TV}\sum_{t=1}^{T}\mathbb{E}\left[A_1(t)\right]+\frac{c_1+c_2}{V}. \tag{23}$$

In the rest of this section, we show that for all T,

$$\frac{1}{TV}\sum_{t=1}^{T}\mathbb{E}\left[A_1(t)\right]\le\frac{\lambda}{N}\left[4\sqrt{2JN\log N}+\mu\big(4\sqrt{2\log N}+3+\frac{3J\gamma^2}{(1-\gamma)^2}\big)\right]=\beta_2\sqrt{\frac{\log N}{N}}, \tag{24}$$

where

$$\beta_2=4\lambda\sqrt{2J}+4\sqrt{2}\lambda\mu\frac{1}{\sqrt{N}}+3\lambda\left(1+\frac{J\gamma^2}{(1-\gamma)^2}\right)\mu\frac{1}{\sqrt{N\log N}}.$$

Substituting (24) into (23), we get that

$$R^*-R_T\le\frac{\beta_3N(V+1)}{T}+\beta_2\sqrt{\frac{\log N}{N}}+\frac{c_1+c_2}{V}.$$

Finally, invoking the assumption $N\log N\ge1$ and the fact that $N\ge1$, the result of Theorem 2 readily follows. The remainder of the section focuses on proving (24).

4.4. Bounding $A_1(t)$: Handling Payoff Uncertainty

As we will see soon, if there were no errors in the payoff estimates $[C_{ij}^k(t)]$, we would have obtained $A_1(t) \leq 0$ (see discussions after Lemma 2). Thus, the key in bounding $A_1(t)$ is to account for the impact of the errors of the payoff estimates.

In the rest of this subsection, we fix $\mathbf{n} = \mathbf{n}(t)$. Recall that $\mathcal{I}(t) = \{i | n_i(t) \geq 1\}$, and $\mathbf{p}(t) = [p_{ij}^k(t)]$ is the solution to the optimization problem (14). Denote vectors $\mathbf{W} = [W_{ij}^k, i \in \mathcal{I}(t), 1 \leq j \leq J, 1 \leq k \leq n_i(t)]$ and $\boldsymbol{\pi} = [\pi_{ij}^k, i \in \mathcal{I}(t), 1 \leq j \leq J, 1 \leq k \leq n_i(t)]$. We define function

$$f(\boldsymbol{\pi}|\mathbf{n}, \mathbf{W}) \triangleq \sum_{i \in \mathcal{I}(t)} \sum_{k=1}^{n_i(t)} \left[\log \left(\sum_{j=1}^{J} \pi_{ij}^k \right) + V \sum_{j=1}^{J} (W_{ij}^k - \gamma) \pi_{ij}^k \right]. \tag{25}$$

Let $\mathbf{C}(t)$ denote the vector $[W_{ij}^k]$ such that $W_{ij}^k = C_{ij}^k(t)$ for all $i \in \mathcal{I}(t), 1 \leq j \leq J, k = 1, ..., n_i(t)$. Then $f(\boldsymbol{\pi}|\mathbf{n}, \mathbf{C}(t))$ is the objective function of (14) multiplied by V, and hence $\mathbf{p}(t)$ is the maximizer of $f(\boldsymbol{\pi}|\mathbf{n}, \mathbf{C}(t))$ over the constraint (15).

Now, let $\mathbf{C}^*$ denote the vector $[W_{ij}^k]$ such that $W_{ij}^k = C_{ij}^*$ for all $i \in \mathcal{I}(t), 1 \leq j \leq J, k = 1, ..., n_i(t)$. Denote $\widehat{\mathbf{p}}(t) = [\widehat{p}_{ij}^k(t)]$ as the maximizer of $f(\boldsymbol{\pi}|\mathbf{n}, \mathbf{C}^*)$ over the constraint (15).

Using the concavity of function f in π_{ij}^k, we can prove the following lemma.

Lemma 2. *For each time-slot t, $A_1(t)$ given in (20) satisfies*

$$A_1(t) \leq f(\widehat{\mathbf{p}}(t)|\mathbf{n}, \mathbf{C}^*) - f(\mathbf{p}(t)|\mathbf{n}, \mathbf{C}^*) \tag{26}$$

$$\leq A_2(t) + A_3(t), \tag{27}$$

where

$$A_2(t) = V \sum_{i \in \mathcal{I}(t)} \sum_{k=1}^{n_i(t)} \sum_{j=1}^{J} \left(C_{ij}^k(t) - C_{ij}^* \right) p_{ij}^k(t),$$

$$A_3(t) = V \sum_{i \in \mathcal{I}(t)} \sum_{k=1}^{n_i(t)} \sum_{j=1}^{J} \left(C_{ij}^* - C_{ij}^k(t) \right) \widehat{p}_{ij}^k(t). \tag{28}$$

The proof is given in the e-companion to this paper.

To appreciate the difference between the two terms in (26), recall that $\widehat{\mathbf{p}}(t)$ maximizes $f(\boldsymbol{\pi}|\mathbf{n}, \mathbf{C}^*)$, while $\mathbf{p}(t)$ maximizes $f(\boldsymbol{\pi}|\mathbf{n}, \mathbf{C}(t))$. Clearly, if there were no errors in the payoff estimates $\mathbf{C}(t)$, i.e., if $\mathbf{C}(t) = \mathbf{C}^*$, we would have obtained $\mathbf{p}(t) = \widehat{\mathbf{p}}(t)$ and $A_1(t) \le 0$ follows immediately from (26). Lemma 2 further bounds the difference even when $\mathbf{C}(t) \ne \mathbf{C}^*$.

It remains to upper bound $(1/T)\sum_{t=1}^{T}\mathbb{E}[A_2(t)]$ and $(1/T)\sum_{t=1}^{T}\mathbb{E}[A_3(t)]$ for a given time T. We define Λ as a particular realization of the sequence of client arrival-times up to time slot T. We use $\mathbb{E}_\Lambda$ to denote the conditional expectation given Λ. Given Λ, we slightly abuse notation and use the index k now to denote the k-th client of class i that arrives to the system. Let $t_1(k)$ denote the arrival time of this client, and $t_2(k)$ denote the *minimum* of her departure time and T.

Lemma 3. *Suppose that the k-th arriving client of the underlying class i brings a total number a_k of tasks into the system. Then it holds that*

$$\mathbb{E}_\Lambda\left[\sum_{j=1}^{J}\sum_{s=t_1(k)}^{t_2(k)}(C_{ij}^k(s) - C_{ij}^*)p_{ij}^k(s)\right] \le 4\sqrt{2Ja_k\log a_k} + 4\mu\sqrt{2\log a_k} + 3\mu. \tag{29}$$

Lemma 4. *Suppose that the k-th arriving client of class i arrives at time $t_1(k)$. Then it holds that*

$$\mathbb{E}_\Lambda\left[\sum_{j=1}^{J}\sum_{s=t_1(k)}^{t_2(k)}\left(C_{ij}^* - C_{ij}^k(s)\right)\widehat{p}_{ij}^k(s)\right] \le 3J\left(\frac{\gamma}{\gamma-1}\right)^2\mu. \tag{30}$$

By summing (29) and (30), respectively, over all users k, we can then obtain that (See Remark EC.2 for a detailed derivation):

$$\frac{1}{TV}\sum_{t=1}^{T}\mathbb{E}\left[A_2(t)\right] \le \frac{\lambda}{N}\left(4\sqrt{2JN\log N} + \left(4\sqrt{2\log N} + 3\right)\mu\right) \tag{31}$$

$$\frac{1}{TV}\sum_{t=1}^{T}\mathbb{E}[A_3(t)] \le \frac{3\lambda\mu J}{N}\left(\frac{\gamma}{1-\gamma}\right)^2. \tag{32}$$

The proofs of Lemma 3 and Lemma 4 are given in the e-companion to this paper. Below, we briefly illustrate the underlying intuition; see Remark EC.1 and Remark EC.3 for more detailed explanation.

Remark 3. The proofs of both lemmas involve finite-time regret analysis for UCB (Auer et al. 2002a). However, there is a major innovation. In standard MAB problems, exactly one arm is pulled at each time. In our system, the rate that servers are chosen is determined by $p_{ij}^k(t)$, which in turn depends on previous values of $C_{ij}^k(s)$, $s < t$. Our proofs use a delicate martingale argument to take care of such dependency. Between these two lemmas, the proof of Lemma 4 is even trickier. To see this, note that the loss in Lemma 3 is accumulated at the same rate as the rate that each server is chosen. Thus, we may view the time as being "slowed down" (compared to a standard MAB problem), and expect Lemma 3 to hold. In contrast, the loss in Lemma 4 is accumulated at the rate $\widehat{p}_{ij}^k(s)$, which is different from the rate $p_{ij}^k(t)$ that each server is chosen. Thus, a direct "slowing-down" argument will not work. Fortunately, by exploiting the fairness property of Algorithm 1, we can show that $\sum_{j=1}^J \widehat{p}_{ij}^k(t)$ and $\sum_{j=1}^J p_{ij}^k(t)$ cannot differ by more than a constant factor. We can then prove Lemma 4 using a similar martingale argument.

Finally, we remark that Lemma 3 is the main reason for the $\sqrt{\log N/N}$ regret in the second term of (11). In the instance-dependent setting of the MAB problem in (Auer et al. 2002a), there is a non-zero gap δ between the best arm and the second-best arm. Thus, once the estimation error is within δ, which takes $\Theta(\log N)$ time slots, learning can stop. Hence, the regret is on the order of $\log N/N$. In contrast, in our problem, the notions of "best/second-best arms" are more fluid because they depend on the number of clients in the system. Thus we do not have such a fixed gap δ, which is why we have a larger $\sqrt{\log N/N}$ loss due to learning. Also, our upper bound on the payoff gap holds in the instance-independent (minimax) setting, where the payoff vector can be adversarially chosen from any distribution with a finite support. In this setting, our payoff reget $\sqrt{\log N/N}$ matches the optimal minimax regret bound $\sqrt{1/N}$ up to a $\sqrt{\log N}$ factor (Auer et al. 2002b).

5. Simulation Study of Algorithm 1

We first consider the same setup as described in the counterexample in Section 2.1 and illustrated in Fig. 2. The code for Algorithm 1 implementation is publicly available on Github at `https://github.com/waycan/QueueLearning`. In particular, there are two servers and two classes of clients. Define the key parameters as follows: $\mu_j = 1$ for each server $j = 1, 2$; $\lambda_i = 0.6$ for each class $i = 1, 2$, and hence $\lambda = \lambda_1 + \lambda_2 = 1.2$. The expected number of tasks per client is initially set to be $N = 100$, but we will vary the value later. The true payoff vectors for class 1 and class 2 are given by [0.9 0.1] and [0.9 0.3], respectively, although they are unknown to the operator. Note that server 1 has larger expected payoffs for both class 1 and class 2. However, its service rate is insufficient to support all clients. Hence, this contention must be carefully controlled when the system aims to maximize the payoff. For a given policy Π and simulation time T, we report the average system payoff $R_T(\Pi)$ per unit time.

5.1. Two Other Policies for Comparison

We compare our proposed Algorithm 1 with the myopic matching policy and a queue-length based policy. These two policies also use UCB to estimate unknown payoff vectors. The myopic matching policy aims at maximizing the total payoff for the current time-slot. Specifically, at each time slot t, the policy solves a modified maximization problem of the form (7)–(8), but with the logarithmic term removed and with $\gamma = 0$. Then, based on the solution to this modified maximization problem, the tasks are assigned to each server in the same way as described in Step 3 in Algorithm 1. We expect that the myopic matching policy incurs a relatively large payoff loss because it does not look at the future.

The queue-length based policy maintains a queue of tasks at each server j and uses the length of this task-queue $q_j(t)$ at time-slot t to indicate the congestion level at server j. At each time-slot t and for each client l, the operator finds the server $j^*(l)$ with the highest queue-adjusted payoff, i.e., $j^*(l) = \arg\max_j \{C^l_j(t) - q_j(t)/V\}$. The operator then adds one task from each client l to the end of the task queue at the server $j^*(l)$. Every server j then processes the $\mu_j = 1$ task from the head of its own queue and observes the random payoff generated. As discussed in Section EC.1, one weakness of such a queue-length based policy is that, when tasks are waiting in the queues, the system cannot observe their payoff feedback right away. Hence, there is significant payoff feedback delay, which in turn leads to suboptimal assignment decisions for subsequent tasks.

5.2. Performance Comparisons

We fix $\gamma = 1.1$ for our proposed Algorithm 1 in all experiments, but we may vary N and V. The first set of experiments give a general feel of the dynamics of different policies. In Fig. 3(a), we fix $N = 100$ and plot the evolution of the time-averaged system payoff up to time T under the three policies (with different V), as the simulation time T advances. We can see that, even when N is not very large ($N = 100$), the system payoff under our proposed Algorithm 1 with $V = 21$ (the solid curve with marker ▲) approaches the upper bound (4) (the horizontal dashed line). Further, comparing $V = 2$ (marker ▲, dashed curve) with $V = 21$ (marker ▲, solid curve), we observe significantly higher system payoff under Algorithm 1 when V increases. In comparison, the payoff achieved by the myopic matching policy (the lowest dotted curve) is significantly lower. The performance of the queue-length based policy (the two curves with marker ▼) also exhibits a noticeable gap from that of the proposed Algorithm 1. Further, even when V increases from $V = 2$ (marker ▼, dashed curve) to $V = 100$ (marker ▼, solid curve), the improvement of the queue-length based policy is quite limited. This result suggests that, due to the increase in payoff-feedback delay, controlling V is not effective in improving the performance of the queue-length based policy.

In Fig. 3(b), we fix $N = 100$, increase V, and plot the payoff gap (compared to the upper bound (4)) of our proposed Algorithm 1 over $T = 7 \times 10^5$ time slots. Each plotted data point represents the average of 5 independent runs. We can observe that initially, the payoff gap decreases significantly with V, but eventually it saturates at $V \geq 50$. This is because the regret due to small N, i.e., the second term of (11), eventually dominates the payoff gap when V is large.

In Fig. 3(c), we set $V = N/\log N$. In this way, the first term in (11) is of the order $\log N/N$, and hence our payoff loss upper bound (11) is dominated by the second term, which is of the order $\sqrt{\log N/N}$ when T is sufficiently large. Thus we vary N and simulate the payoff gap of our Algorithm 1 versus N on a log-log scale. The result indicates that the payoff gap scales with a rate between $\Theta(\sqrt{\log N/N})$ and $\Theta(\log N/N)$, which agrees with our payoff loss upper bound.

5.3. Infinite Number of Classes

Although our analytical results have assumed that clients are from a finite number I of classes, Algorithm 1 can be used even if the number of classes is infinite. In Fig. 3(d), we simulate the performance of Algorithm 1 under such a setting. In particular, each client l has a payoff vector

$(C^*_{l,1}, C^*_{l,2})$ independently and uniformly distributed over the lower triangular part of the unit square $\Omega = \{(x,y) \in [0,1]^2 : x \geq y\}$. In other words, let X_1 and X_2 denote two independent random variables uniformly distributed over $[0,1]$. Then $(C^*_{l,1}, C^*_{l,2})$ has the same distribution as $(X_{(1)}, X_{(2)})$, where $X_{(1)} = \max\{X_1, X_2\}$ and $X_{(2)} = \min\{X_1, X_2\}$. To maximize the total expected payoff, it is preferable to assign tasks from those clients l with a small gap $C^*_{l,1} - C^*_{l,2}$ to server 2. We show in EC.6 that the oracle upper bound for the payoff is around 0.79, while the expected payoff under myopic matching is approximately upper bounded by 0.6. As we can see from Fig. 3(d), Algorithm 1 outperforms myopic matching, achieving a higher expected payoff around 0.69.

6. Conclusion

In this paper, we propose a new utility-guided task assignment algorithm for queueing systems with uncertain payoffs. Our algorithm seamlessly integrates online learning and adaptive control, without relying on any prior knowledge of the statistics of the client dynamics or the payoff vectors. We show that, compared to an oracle that knows all client dynamics and payoff vectors beforehand, the gap of the expected payoff per unit time of our proposed algorithm at any finite time T is on the order of $1/V + \sqrt{\log N/N} + N(V+1)/T$, where V is a parameter controlling the weight of a logarithmic utility function, and N is the average number of tasks per client. Our analysis carefully accounts for the closed-loop interactions between the queueing and learning processes. Numerical results indicate that the proposed algorithm outperforms a myopic matching policy and a standard queue-length based policy that does not explicitly address such closed-loop interactions. Further, we derive a virtual-queue heuristic from the proposed algorithm, which incurs low-complexity for large systems and is capable of handling server-side uncertainty.

There are a number of future research directions. First, while our payoff regret bound $\sqrt{\log N/N}$ matches the optimal minimax regret bound $1/\sqrt{N}$ up to a $\sqrt{\log N}$ factor (Auer et al. 2002b), it remains open whether our $\sqrt{\log N/N}$ regret bound can be further reduced to $\log N/N$ (Auer et al. 2002a) in the instance-dependent setting where the payoff vector is chosen from a given and fixed distribution. Second, although our model does not assume any prior knowledge of the class-dependent payoff vectors, it would be interesting to study whether the idea from this paper can be used to improve the policies in (Johari et al. 2016, Massoulie and Xu 2016, Johari et al. 2017) when such knowledge is available. Third, although we show via simulation that Algorithm 1 works for an infinite number of classes, our bound on the payoff gap established in Theorem 2 scales linearly in the number of client classes I and thus blows up to infinity when I goes to infinity. It would be of great interest to prove that Algorithm 1 achieves a payoff gap independent of I.

Acknowledgments

This work has been partially supported by the National Science Foundation through grants IIS-1838124, CCF-1850743, CCF-1856424, CNS-1457137, and ECCS-1509536.

References

(2017) Adwords. `https://adwords.google.com/`, accessed: 2017-08-30.

(2017) Airbnb. `https://www.airbnb.com/`, accessed: 2017-08-30.

(2017) Coursera. `https://www.coursera.org/`, accessed: 2017-08-30.

(2017) Upwork. `https://www.upwork.com/`, accessed: 2017-08-30.

Alaei S, Hajiaghayi M, Liaghat V (2013) The online stochastic generalized assignment problem. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, 11–25 (Springer).

Alfa AS (2016) *Applied discrete-time queues* (Springer).

Anandkumar A, Michael N, Tang AK, Swami A (2011) Distributed algorithms for learning and cognitive medium access with logarithmic regret. *IEEE Journal on Selected Areas in Communications* 29(4):731–745.

Auer P, Cesa-Bianchi N, Fischer P (2002a) Finite-time analysis of the multiarmed bandit problem. *Machine learning* 47(2-3):235–256.

Auer P, Cesa-Bianchi N, Freund Y, Schapire RE (2002b) The nonstochastic multiarmed bandit problem. *SIAM journal on computing* 32(1):48–77.

Badanidiyuru A, Langford J, Slivkins A (2014) Resourceful contextual bandits. *Conference on Learning Theory*, 1109–1134.

Bimpikis K, Markakis MG (2015) Learning and hierarchies in service systems. *under review* .

Bonald T, Massoulie L (2001) Impact of Fairness on Internet Performance. *Proceedings of ACM SIGMETRICS*, 82–91 (Cambridge, MA).

Buccapatnam S, Tan J, Zhang L (2015) Information sharing in distributed stochastic bandits. *IEEE INFOCOM*, 2605–2613.

Chakrabarty D, Zhou Y, Lukose R (2008) Online knapsack problems. *Workshop on Internet and Network Economics (WINE)*.

Chiang M (2004) To Layer or Not to Layer: Balancing Transport and Physical Layers in Wireless Multihop Networks. *IEEE INFOCOM* (Hong Kong).

Combes R, Jiang C, Srikant R (2015) Bandits with budgets: Regret lower bounds and optimal algorithms. *ACM SIGMETRICS Performance Evaluation Review* 43(1):245–257.

Combes R, Proutiere A (2015) Dynamic rate and channel selection in cognitive radio systems. *IEEE Journal on Selected Areas in Communications* 33(5):910–921.

De Veciana G, Lee TJ, Konstantopoulos T (2001) Stability and Performance Analysis of Networks Supporting Elastic Services. *IEEE/ACM Trans. on Networking* 9(1):2–14.

Eryilmaz A, Srikant R (2005) Fair Resource Allocation in Wireless Networks Using Queue-length-based Scheduling and Congestion Control. *IEEE INFOCOM* (Miami, FL).

Evans PC, Gawer A (2016) The rise of the platform enterprise: a global survey Available at https://www.thecge.net/archived-papers/the-rise-of-the-platform-enterprise-a-global-survey/.

Feldman J, Korula N, Mirrokni VS, Muthukrishnan S, Pál M (2009) Online ad assignment with free disposal. *WINE*, volume 9, 374–385 (Springer).

Gai Y, Krishnamachari B, Jain R (2012) Combinatorial network optimization with unknown variables: Multi-armed bandits with linear rewards and individual observations. *IEEE/ACM Transactions on Networking (TON)* 20(5):1466–1478.

Ganesh AJ, O'Connell N, Wischik DJ (2004) *Big queues* (Springer).

Gittins J, Glazebrook K, Weber R (2011) *Multi-armed bandit allocation indices* (John Wiley & Sons).

Hajek B (2015) *Random Processes for Engineers* (Cambridge University Press).

Ipeirotis PG (2010) Analyzing the amazon mechanical turk marketplace. *XRDS* 17(2):16–21, ISSN 1528-4972.

Johari R, Kamble V, Kanoria Y (2016) Know your customer: Multi-armed bandits with capacity constraints. *arXiv preprint arXiv:1603.04549v1* .

Johari R, Kamble V, Kanoria Y (2017) Matching while learning. *Proceedings of the 2017 ACM Conference on Economics and Computation (EC '17)* (Cambridge, MA).

Kalathil D, Nayyar N, Jain R (2014) Decentralized learning for multiplayer multiarmed bandits. *IEEE Transactions on Information Theory* 60(4):2331–2345.

Karger DR, Oh S, Shah D (2011a) Budget-optimal crowdsourcing using low-rank matrix approximations. *49th Annual Allerton Conference on Communication, Control, and Computing*, 284–291.

Karger DR, Oh S, Shah D (2011b) Iterative learning for reliable crowdsourcing systems. *Advances in Neural Information Processing Systems*, 1953–1961.

Karger DR, Oh S, Shah D (2013) Efficient crowdsourcing for multi-class labeling. *ACM SIGMETRICS Performance Evaluation Review* 41(1):81–92.

Kenney M, Zysman J (2016) The rise of the platform economy. *Issues in science and technology* 32(3):61.

Khetan A, Oh S (2016) Achieving budget-optimality with adaptive schemes in crowdsourcing. *Advances in Neural Information Processing Systems*, 4844–4852.

Kumar PR, Meyn SP (1995) Stability of Queueing Networks and Scheduling Policies. *IEEE Transactions on Automatic Control* 40:251–260.

Lai L, El Gamal H, Jiang H, Poor HV (2011) Cognitive medium access: Exploration, exploitation, and competition. *IEEE Transactions on Mobile Computing* 10(2):239–253.

Lai TL, Robbins H (1985) Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics* 6(1):4–22.

Lai TL, Ying Z (1988) Open bandit processes and optimal scheduling of queueing networks. *Advances in Applied Probability* 20(2):447–472.

Lelarge M, Proutiere A, Talebi MS (2013) Spectrum bandit optimization. *IEEE Information Theory Workshop (ITW)*, 1–5.

Lin X, Shroff NB, Srikant R (2006) A Tutorial on Cross-Layer Optimization in Wireless Networks. *IEEE Journal on Selected Areas in Communications (JSAC)* 24(8):1452–1463.

Lin X, Shroff NB, Srikant R (2008) On the Connection-Level Stability of Congestion-Controlled Communication Networks. *IEEE Trans. on Inf. Theory* 54(5):2317–2338.

Massoulie L, Xu K (2016) On the capacity of information processing systems. *Conference on Learning Theory*, 1292–1297.

Nayyar N, Kalathil D, Jain R (2016) On regret-optimal learning in decentralized multi-player multi-armed bandits. *IEEE Transactions on Control of Network Systems* .

Neely MJ, Modiano E, Li C (2005) Fairness and Optimal Stochastic Control for Heterogeneous Networks. *IEEE INFOCOM* (Miami, FL).

Neuts MF (1981) *Matrix-geometric solutions in stochastic models: an algorithmic approach* (Courier Corporation).

Shah V, Gulikers L, Massoulie L, Vojnovic M (2017) Adaptive matching for expert systems with uncertain task types. *arXiv preprint arXiv:1703.00674* .

Shahrampour S, Rakhlin A, Jadbabaie A (2017) Multi-armed bandits in multi-agent networks. *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2786–2790 (IEEE).

Tan B, Srikant R (2012) Online advertisement, optimization and stochastic networks. *IEEE Transactions on Automatic Control* 57(11):2854–2868.

Tassiulas L, Ephremides A (1992) Stability Properties of Constrained Queueing Systems and Scheduling Policies for Maximum Throughput in Multihop Radio Networks. *IEEE Trans. on Automatic Control* 37(12):1936–1948.

van Dijck J, Poell T, de Waal M (2018) *The platform society: Public values in a connective world* (Oxford University Press).

Whittle P (1988) Restless bandits: Activity allocation in a changing world. *Journal of Applied Probability* 25(A):287–298.

Zheng F, Cheng Y, Xu Y, Liu M (2013) Competitive strategies for an online generalized assignment problem with a service consecution constraint. *European Journal of Operational Research* 229(1):59–66.

7. Tables

Table 1 List of Notations

Symbol	Meaning
$C_{ij}^k(t)$	UCB estimate for server j by client k of class i
$C_j^l(t)$	UCB estimate for server j by client l
C_{ij}^*	True expected payoff for server j and class i
$\overline{C}_{ij}^k(t), \overline{C}_j^l(t)$	Empirical payoff average
$\gamma > 1$	Parameter in (7) and (14)
p_{ij}^*	Solution to upper bounds (4) or (17)
$p_{ij}^k(t)$, $p_j^l(t)$	Solution to (7) or (14) using UCB payoff estimates
$\widehat{p}_{ij}^k(t)$	Solution to (14) with $C_{ij}^k(t)$ replaced by C_{ij}^*

8. Figures

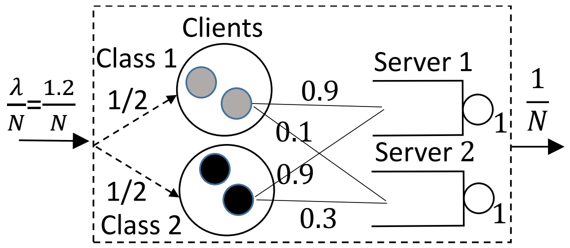

Figure 1 Uncertain client dynamics and payoffs.

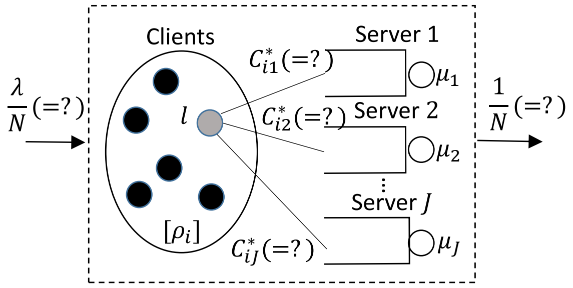

Figure 2 Myopic matching is suboptimal.

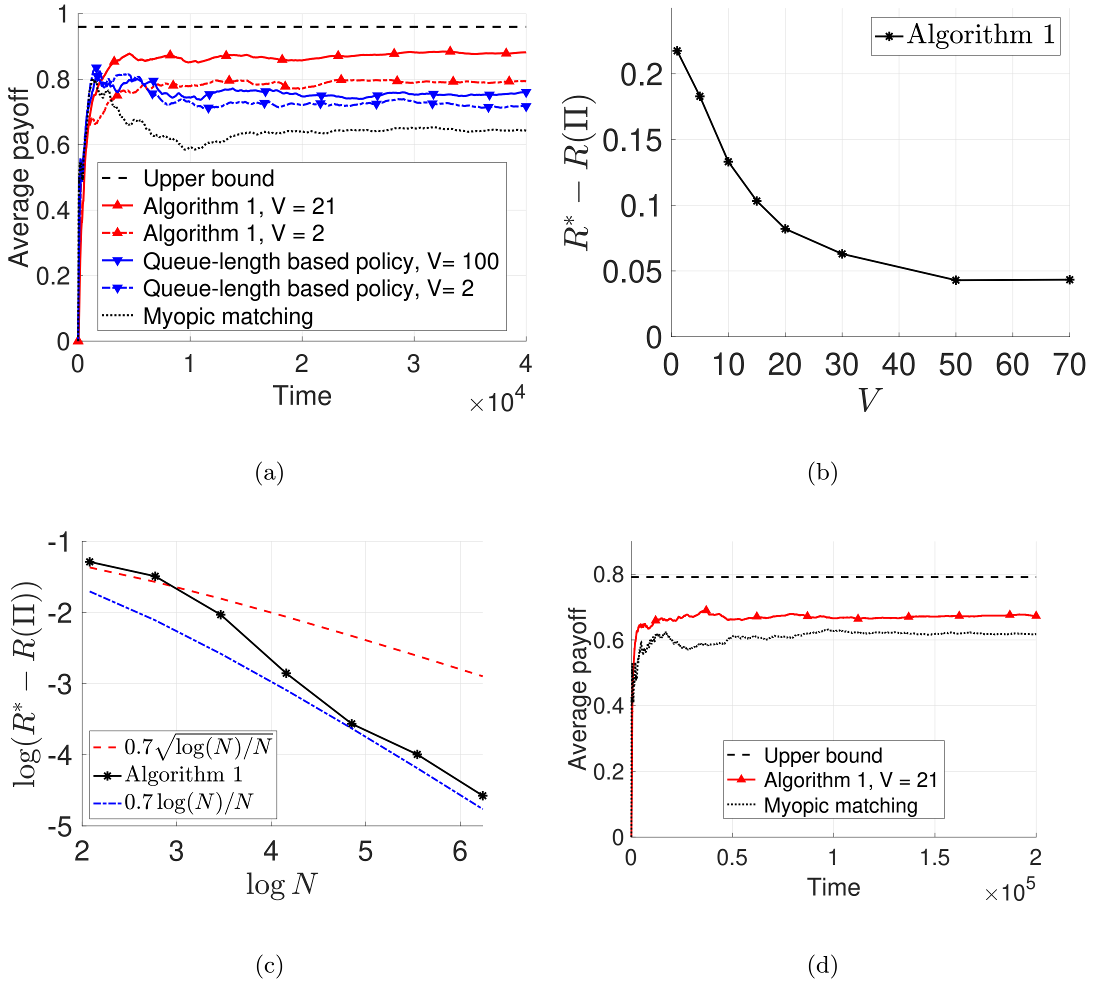

Figure 3 Plot (a): The evolution of the time-averaged payoff under the three policies as simulation time advances. The upper bound (4) is 0.96 (the horizontal line). Plot (b): With a fixed $N = 100$, the payoff gap of Algorithm 1 decreases and eventually saturates as V increases. Plot (c):With $V = N/\log N$, the payoff gap of Algorithm 1 scales as $\Theta(\log N/N)$. Plot (d): Algorithm 1 applies to infinite number of client classes and still achieves good performance.