Operations Research

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Matching While Learning

Ramesh Johari, Vijay Kamble, Yash Kanoria

To cite this article:
Ramesh Johari, Vijay Kamble, Yash Kanoria (2021) Matching While Learning. Operations Research

Published online in Articles in Advance 07 Jan 2021

. https://doi.org/10.1287/opre.2020.2013

http://pubsonline.informs.org/journal/opre

OPERATIONS RESEARCH
Articles in Advance, pp. 1–27
ISSN 0030-364X (print), ISSN 1526-5463 (online)

Methods

Matching While Learning

Ramesh Johari,[a] Vijay Kamble,[b] Yash Kanoria[c]

[a] Department of Management Science and Engineering, Stanford University, Stanford, California 94305; [b] Department of Information and Decision Sciences, University of Illinois at Chicago, Chicago, Illinois 60607; [c] Columbia Business School, New York, New York 10027

Contact: rjohari@stanford.edu, https://orcid.org/0000-0002-3960-0770 (RJ); kamble@uic.edu, https://orcid.org/0000-0002-9261-1612 (VK); ykanoria@gsb.columbia.edu, https://orcid.org/0000-0002-7221-357X (YK)

Received: June 17, 2017
Accepted: April 1, 2020
Published Online in Articles in Advance: January 7, 2021

Subject Classifications: probability: stochastic model applications, analysis of algorithms; inventory/production: policies: capacity.
Area of Review: Stochastic Models

https://doi.org/10.1287/opre.2020.2013

Abstract. We consider the problem faced by a service platform that needs to match limited supply with demand while learning the attributes of new users to match them better in the future. We introduce a benchmark model with heterogeneous workers (demand) and a limited supply of jobs that arrive over time. Job types are known to the platform, but worker types are unknown and must be learned by observing match outcomes. Workers depart after performing a certain number of jobs. The expected payoff from a match depends on the pair of types, and the goal is to maximize the steady-state rate of accumulation of payoff. Although we use terminology inspired by labor markets, our framework applies more broadly to platforms where a limited supply of heterogeneous products is matched to users over time. Our main contribution is a complete characterization of the structure of the optimal policy in the limit that each worker performs many jobs. The platform faces a tradeoff for each worker between myopically maximizing payoffs (*exploitation*) and learning the type of the worker (*exploration*). This creates a multitude of multiarmed bandit problems, one for each worker, coupled together by the constraint on availability of jobs of different types (*capacity constraints*). We find that the platform should estimate a shadow price for each job type and use the payoffs adjusted by these prices first to determine its learning goals and then for each worker (i) to balance learning with payoffs during the *exploration phase* and (ii) to myopically match after it has achieved its learning goals during the exploitation phase.

Funding: This work was supported by the National Science Foundation [Grants CMMI-1653477, CNS-1343253, CNS-1544548, and CNS-1931696], the Army Research Office [Grant W911NF-14-1-0526], and the Simons Foundation.
Supplemental Material: The online appendices are available at https://doi.org/10.1287/opre.2020.2013.

Keywords: matching • learning • two-sided platform • multiarmed bandit • capacity constraints

1. Introduction

A wide range of online platforms serve as matchmakers between demand and supply; for example, online labor markets match workers to jobs (e.g., Upwork for remote work, Handy for housecleaning, Thumbtack and Taskrabbit for local tasks, etc.); e-commerce platforms match consumers to goods (e.g., eBay, Amazon); and online fashion retailers match clients to clothing items (e.g., Rent The Runway, Stitch Fix). These platforms are characterized by two salient features that motivate our work. First, they have a *limited supply* available; for example, in online labor markets, the supply of jobs is limited, whereas in e-commerce and online fashion platforms, the supply of goods is limited. Second, these platforms need to *learn* enough about their users (the demand side) to be able to match them to the right units of supply. Our paper addresses this twin challenge of matching while learning.

The problem we address is a version of the *exploration-exploitation* tradeoff: on the one hand, efficient operation involves making matches that generate the most value (exploitation); on the other hand, the platform must continuously learn about newly arriving participants, so that they can be efficiently matched (exploration). The task is complicated in our setting because of the fact that supply is limited: matching a unit of supply to one user renders it unavailable to other users, an externality that cannot be ignored, whether exploring or exploiting. In this paper, we develop a structurally simple and nearly optimal approach to resolving the exploration-exploitation tradeoff in settings with limited supply.

For convenience, the terminology in our model will be inspired by online labor markets: we call the demand side of the platform the *workers* and the supply side of the platform the *jobs*. Jobs are in limited supply in the platform. Despite this specific terminology, our model should be viewed as a stylized abstraction of many platforms where supply is matched to users in the presence of limited inventory, for example,

via algorithmic recommendation or matching engines. Examples include online commerce and fashion retail platforms mentioned previously and similar platforms in other industries.

In our model, workers and jobs arrive over discrete time. Workers depart after N periods, whereas jobs each take one period for a single worker to complete (hence each worker performs N jobs over her lifetime). The supply of jobs at each period is limited. Each time a worker and job are matched, a (random) payoff is generated and observed by the platform, where the payoff distribution depends on the worker type and the job type. (We assume a Bernoulli distribution for the payoffs.) To incorporate the limited supply of jobs in the simplest possible way, our model considers a continuum of workers and jobs. As a consequence, in our analysis, we find that for a suitable class of policies, there is stochasticity only at the level of individual workers and not at the level of the overall system.

As our emphasis is on the interaction between matching and learning, our model has several features that focus our analysis on that interaction. First, we assume that the platform centrally controls matching: at the beginning of each time period, the platform matches each worker in the system to an available job. Second, strategic considerations are not modeled; this remains an interesting direction for future work. Finally, we focus on the prototypical goal of maximizing the steady-state rate of payoff generation. (This is a reasonable proxy for the goal of a platform that takes a fraction of the total surplus generated through matches.)

We assume the platform has system-level knowledge of the arrival rates of workers and jobs, as well as the expected payoff generated when workers of a given type are matched to jobs of a given type. However, although we assume job types are known to the platform, we assume the platform is initially *unaware* of any specific worker's type on arrival. (This is consistent with the observation that in most platforms, more is known about one side than the other.)

The platform learns about workers' types through the payoffs obtained when they are matched to jobs. This gives rise to the central learning challenge: because the supply of jobs is limited, using jobs to learn can reduce immediate payoffs and deplete the supply of jobs available to the rest of the marketplace. Thus, the presence of capacity constraints forces us to carefully design both exploration and exploitation in the matching algorithm in order to optimize the rate of payoff generation.

Our main contribution in this paper is the development of a matching and learning policy that is nearly payoff optimal. Our algorithm is divided into two phases in each worker's lifetime: *exploration* (identification of the worker type) and *exploitation* (optimal matching given the worker's identified type). We refer to our policy as *DEEM: Decentralized Explore-then-Exploit for Matching*.

DEEM is an algorithm that assigns jobs to workers over time. We begin by noting that DEEM has a natural decentralization property: it determines the choice of job type for a worker based only on that worker's history and not based on any other workers' histories. (We note, however, that DEEM itself is designed with knowledge of the global system-level statistics described previously.) This decentralization is inspired by the fact that in large-scale online platforms, matching is typically carried out on an individual basis. For example, if a worker searches for jobs on an online labor market platform, the platform will generally display available jobs in a personalized rank order based on metadata about that worker. (In practice, this decentralization arises in part due to the inherent asynchronous nature of these platforms: workers and jobs arrive continuously over time, and batched centralized matching may be infeasible as a product design.)

At a high level, DEEM operates as follows during the lifetime of a given worker. First, DEEM *explores* to make a confident estimate of the type of this worker. This exploration phase consists of two modes: a *guessing* mode, where DEEM initially samples job types uniformly at random to develop a reasonable maximum a posteriori (MAP) estimate of the worker's type; and a *confirmation* mode, when DEEM chooses jobs to confirm the MAP type as efficiently as possible. The exploration phase is followed by an *exploitation* phase, during which jobs are assigned based on the worker type that was confirmed during exploration. Each of these phases is carefully designed to optimize the rate of payoff generation while ensuring that capacity constraints are met.

To develop intuition for our solution, consider a simple example with two types of jobs (Easy and Hard) and two types of workers (Expert and Novice). Experts can do both types of tasks well, but novices can only do easy tasks well. Suppose that there is a limited supply of easy jobs: more than the mass of novices available but less than the total mass of novices and experts. In particular, to maximize payoff, the platform must learn enough to match some experts to hard jobs.

DEEM has several key features, each of which can be understood in the context of this example. *First*, because DEEM operates at the level of a given worker, we must ensure that the algorithm nevertheless does not violate capacity constraints. In particular, it is essential for the algorithm to account for the externality to the rest of the market when a worker is matched to a given job. For example, if easy jobs are relatively scarce, then matching a worker to such a job makes it unavailable to the rest of the market.

Our approach is to *price* this externality: we find *shadow prices* for the capacity constraints and adjust all per-match payoffs downward using these prices.

Second, our algorithm design specifies *learning goals* that ensure an efficient balance between exploration and exploitation. In particular, in our example, we note that there are two kinds of errors possible while exploring: misclassifying a novice as an expert and vice versa. Occasionally mislabeling experts as novices is not catastrophic: some experts need to do easy jobs anyway, and so the algorithm can account for such errors in the exploitation phase. Thus, relatively less effort can be invested in minimizing this error type. However, mistakenly labeling novices as experts *can* be catastrophic: in this case, novices will be matched to hard jobs in the exploitation phase, causing substantial loss of payoff; thus, the probability of such errors must be kept very small. A major contribution of our work is to precisely identify the correct learning goals that determine progression of the algorithm from the exploration phase to the exploitation phase and to then design DEEM to meet these learning goals while maximizing payoff generation.

Third, the exploitation phase in DEEM is carefully constructed to ensure that capacity constraints are met while maximizing payoffs. A naive approach during the exploitation phase would match a worker to any job type that yields the maximum externality-adjusted payoff corresponding to his type label. It turns out that such an approach leads to significant violations of capacity constraints and hence poor performance. The reason is that in a generic capacitated problem instance, one or more worker types are indifferent between multiple job types, and appropriate allocation across multiple optimal job types is necessary to achieve good performance. In our theoretical development, we achieve this by modifying the solution to the static optimization problem with known worker types, whereas our practical implementation of DEEM achieves appropriate allocation via simple but dynamically updated shadow prices.

Our main result (Theorem 1) shows that DEEM achieves essentially optimal regret as the number of jobs N performed by each worker during her lifetime grows, where regret is the loss in payoff accumulation rate relative to the maximum achievable with known worker types. In our setting, a lower bound on the regret is $(C \log N/N)(1+ o(1))$ for some $C \in [0, \infty)$ that is a function of system parameters (we use the technical machinery developed in Agrawal et al. 1989 for a related problem to prove this bound). DEEM achieves this level of regret to leading order when $C > 0$, whereas it achieves a regret of $O(\log \log N/N)$ when $C = 0$.

Situations where $C > 0$ are those in which there is an inherent tension between the goals of learning and payoff maximization. To develop intuition, consider an expanded version of the previous example, where each worker can be either an expert or novice programmer, as well as an expert or novice graphic designer. Suppose that the supply of jobs is such that if worker types were known, only expert graphic designers who are also novice programmers would be matched to graphic design jobs. (This would be the case, e.g., if there were an excess supply of programming jobs, whereas the supply of graphic design jobs was less than the volume of available workers who are both expert graphic designers and novice programmers.) However, if we are learning worker types, then expert graphic designers must be matched to approximately $\Omega(\log N)$ programming jobs to distinguish between novice and expert programmers, so that they can be matched to graphic design and programming jobs, respectively. Thus, $\Omega(\log N/N)$ average regret per period is incurred relative to the optimal solution with known types. DEEM precisely minimizes the regret incurred while these distinctions are made, thus achieving the lower bound on the regret.

Our theory is complemented by a practical heuristic that we call DEEM$^+$, which optimizes performance for small values of N, an implementation leveraging queue-length based shadow prices that demonstrates a natural way of translating our work into practice, and supporting simulations. In particular, our simulations reveal substantial benefit from jointly managing capacity constraints and learning, as we do in DEEM and DEEM$^+$.

The remainder of the paper is organized as follows. After discussing related work in Section 2, we present our model and outline the optimization problem of interest to the platform in Section 3. In Section 4, we discuss the three key ideas in the design of DEEM and present its formal definition. In Section 5, we present our main theorem and discuss the optimal regret scaling. In Section 5.1, we present a sketch of the proof of the main result. In Section 6, we discuss the practical implementation of DEEM and present the heuristic DEEM$^+$. In Section 7, we use simulations to compare the performance of DEEM$^+$ with benchmark multiarmed bandit algorithms. We conclude in Section 8. All proofs are in the online appendices.

2. Related Literature

Here we discuss the relationship between our work and several related threads in the literature on (1) general stochastic multiarmed bandits, (2) dynamic pricing and learning, (3) combinatorial bandits, including bandits with matching constraints, and (4) dynamic stochastic matching models.

Before surveying these threads of literature, we note here that in the period since the initial development

of our results, our paper has inspired a subsequent paper (Hsu et al. 2018), which studies a very similar matching while learning setting and shows near optimality of a *backpressure* algorithm similar to the finite N heuristic DEEM$^+$ that we propose here (see Sections 6 and 7). *Backpressure* is a celebrated methodology that prescribes using current queue-lengths as shadow prices (Tassiulas and Ephremides 1990). Hsu et al. (2018) goes beyond this paper by showing near optimality of backpressure in their setting but at a cost: their bounds on the bandit (learning) problem are loose with a $1/\sqrt{N}$ upper (achievability) bound on the regret, which is much larger than their $\log N/N$ lower bound. By contrast, our theoretical analysis is focused on a tight characterization of regret; we obtain tight $\log N/N$ bounds on the regret, which match even in the constant factor.

2.1. Stochastic Multiarmed Bandits

A foundational model for investigating the exploration-exploitation tradeoff is the stochastic multiarmed bandit (MAB) problem (Lai and Robbins 1985, Gittins et al. 2011, Bubeck and Cesa-Bianchi 2012, Lattimore and Szepesvári 2020). The goal in this problem is to find an adaptive expected-regret-minimizing policy for choosing among arms with unknown payoff distributions, where regret is measured against the expected payoff of the best arm (Lai and Robbins 1985, Auer et al. 2002, Agrawal and Goyal 2012).

The closest work in this literature to the MAB problem we tackle is by Agrawal et al. (1989). In their model, they assume that the joint vector of arm distributions can only take on one of finitely many values. This introduces correlation across different arms. Depending on certain identifiability conditions, the optimal regret is either $\Theta(1/N)$ or $\Theta(\log N/N)$. In our model, the analog is that job types are arms, and for each worker, we solve a MAB problem to identify the true type of a worker from among a finite set of possible worker types. In fact, the model of Agrawal et al. (1989) is a special case of our model with no capacity constraints on jobs. Like us, they study the limit $N \to \infty$ and find a policy that achieves regret that is optimal to leading order as $N \to \infty$. To the best of our knowledge, their result remains the state of the art in their setting. Capacity constraints are of course the innovation and focus of the present paper. Notably, our main result generalizing that of Agrawal et al. (1989) to allow capacity constraints is *as sharp as the result they obtained in their much simpler setting* in the case where there is a tension between learning and exploitation (i.e., the case where regret is $\Theta(\log N/N)$).

As demonstrated in Agrawal et al. (1989), the key to attaining the instance-dependent optimal leading-order regret in such multiarmed bandit problems is the following intuition. Given a potential true model, there is a regret-optimal policy that distinguishes this model from all competing models that entail different optimal decisions (defined as the optimal solution to the optimization problem expressed in (36)). Hence, to minimize regret, the challenge is to use this model-specific regret-optimal policy to learn the true model, without a priori knowing the true model. This is precisely the challenge we tackle using the guess-then-confirm approach in the exploration phase of DEEM. Recently, Modaresi et al. (2020) have addressed a similar challenge in a general combinatorial bandit setting.

On a related note, Massoulié and Xu (2018) study a pure learning problem in a setting similar to ours with capacity constraints on each type of server/expert; although there are some similarities in the style of analysis, that paper focuses exclusively on learning the exact type rather than balancing exploration and exploitation as we do in this paper.

2.2. Dynamic Pricing and Learning

Some of the techniques used in our work have parallels in works on dynamic pricing and learning with a finite inventory of products (for a recent comprehensive survey of dynamic pricing and learning, see den Boer 2015). These are essentially MAB problems where the decisions involve choosing product prices dynamically over a selling horizon, with a capacity constraint arising from the finite inventory. A typical approach in these settings is to consider a regime where both the inventory and the demand grow large (although there are exceptions, notably den Boer and Zwart 2015). This is similar to the regime we consider for our technical results, which is equivalent to having both the job arrival rates and the worker lifetimes simultaneously approach infinity.[1] Such a regime was first analyzed in the case of a single product in Besbes and Zeevi (2009), which proposed algorithms with an explore-then-exploit structure for settings with both parametric and nonparametric uncertainty. A more sophisticated algorithm that mixes exploration and exploitation with an improved regret performance in both settings is presented in Wang et al. (2014). Besbes and Zeevi (2012) and, recently, Ferreira et al. (2018) extend the analysis to network revenue management settings involving multiple products using multiple resources with finite inventories. More generally, a recently proposed formulation for MAB problems with capacity constraints, broadly referred to as *bandits with knapsacks* (Badanidiyuru et al. 2013) and its extensions (Agrawal and Devanur 2014, Badanidiyuru et al. 2014, Agrawal et al. 2016, Agrawal and Devanur 2019), subsume several problems in revenue management under demand uncertainty (see Sauré and Zeevi 2013 and Babaioff et al. 2015 in addition to the models discussed previously).

The algorithms designed in all these works critically leverage the solution to the optimal pricing problem in the full information setting in a deterministic world where stochastic quantities are replaced by their means. Similar to these works, we also crucially use the full information optimal assignment problem (which is a linear program in our case), and in particular, the optimal shadow prices for the jobs from the dual of this optimization problem, in determining the job assignments under DEEM. It is known that simply using the optimal price corresponding to the best model estimate from the obtained information at any step (also known as *certainty equivalent* control) can potentially lead to incomplete learning and hence linear regret (see proposition 1 in den Boer and Zwart 2014). Thus, judicious experimentation with prices is necessary.

In a similar fashion, naively using the optimal shadow prices from the full information optimization problem to greedily assign jobs based on current estimates of the worker type typically leads to linear regret in our setting (see fact 1 in Section 4.1). The problem is twofold in our case: the issue is not only that learning may stop prematurely under such a policy but also that appropriate allocation across *multiple* optimal job types is typically necessary in our setting to satisfy capacity constraints. Thus, a good algorithm in our setting needs to achieve both goals, judicious experimentation and effective allocation across optimal assignments, to achieve low regret. In fact, we go one step further, obtaining a policy that achieves not just sublinear but near-optimal regret.

Another key difference in our work compared with these models is that they consider a single MAB problem over a fixed time horizon. Our setting on the other hand can be seen as a system with an ongoing arriving *stream* of MAB problems, one per worker, that are coupled together by the capacity constraints on arriving jobs.

2.3. Bandits with Matching Constraints and Combinatorial Bandits

Several MAB problems with matching constraints can be seen as instances of a larger class of models typically referred to as combinatorial bandits (Gai et al. 2010, 2012; Liu and Zhao 2012; Chen et al. 2013; Sauré and Zeevi 2013; Kveton et al. 2015). Considering the problem of matching all the workers that exist on a platform to the set of available jobs in a particular time period, one can think of the combinatorial set of all possible matchings as being the arms in a MAB setting (sometimes called superarms); this formulation is the closest to the one in Gai et al. (2010). Several works have looked at exploiting the structure of such problems in various settings to yield efficient learning algorithms (Gai et al. 2010, Liu and Zhao 2012, Sauré and Zeevi 2013).

In our case, there are two key aspects that make such a reduction to combinatorial bandits infeasible. First, the number of workers and jobs on real-world platforms is large, and hence the number of possible matchings is prohibitively large, even when one accounts for limited variety in job types (worker types are unknown and there is a vast heterogeneity in worker histories). Thus, decentralization is critical to obtaining a practically feasible solution, which is a feature rarely seen in combinatorial bandit algorithms. Second, the fact that the workers are arriving and leaving asynchronously means that the set of possible matchings, and hence the set of combinatorial arms, is changing over time, which is another feature that is relatively uncommon in the extant literature. An example is Chakrabarti et al. (2009), who consider this problem in a noncombinatorial setting.

2.4. Other Dynamic Stochastic Matching Models

We briefly discuss a few other directions that are related to this paper. There are a number of recent studies that consider efficient matching in dynamic two-sided matching markets (Damiano and Lam 2005, Das and Kamenica 2005, Akbarpour et al. 2014, Anderson et al. 2015, Kadam and Kotowski 2015, Hu and Zhou 2018, Baccara et al. 2020, Kurino 2020, Ozkan and Ward 2020). A related class of dynamic resource allocation problems, online bipartite matching, is also well studied in the computer science community (see Mehta 2012 for a survey). Similar to the present paper, Fershtman and Pavan (2017) also study matching with learning, mediated by a central platform. Relative to our model, their work does not have constraints on the number of matches per agent while it does consider agent incentives.

3. The Model and the Optimization Problem

In this section we first describe our model. In particular, we describe the primitives of our platform (workers and jobs) and give a formal specification of the matching process we study. We conclude by precisely defining the optimization problem addressed in this paper.

A key aspect of our approach is that we consider a model with a *continuum* of workers in the system. The policies we propose for matching workers to jobs will recommend a job type independently for each worker as a function of the history of that worker alone. In our analysis, we leverage the general framework provided by (Sun 2006, section 2.4), which provides a formal mathematical basis for a continuum of independent stochastic processes, including the exact law of large numbers (ELLN) for cross-sectional averages (theorem 2.16 of Sun 2006). Informally, applying this

framework allows the interchange of worker-level probabilistic statements with population-level statements about the evolution of the cross-sectional worker measure over time, yielding the tractable (though challenging) optimization problem we study in this paper (see Section 3.4). We apply the ELLN throughout our development below to yield such interchanges, as appropriate.

3.1. Preliminaries: A Continuum Model

In this section, we describe the basic model that we work with.

3.1.1. Time. We assume that time is discrete $t = 0, 1, 2, \ldots$.

3.1.2. Probability Space. We fix a probability space $(\Omega, \mathcal{F}, P)$. An element $\omega \in \Omega$ is a state of the world. All randomness throughout our development below is resolved by the state of the world $\omega \in \Omega$. An event is a measurable subset B of Ω (that is, an element of $\mathcal{F}$), whose probability is $P(B)$. Any statements of events occurring "with probability 1" refer to almost sure events with respect to the measure P.

3.1.3. Workers and Jobs. For convenience we adopt the terminology of *workers* and *jobs* to describe the two sides of the market. Each job in the system is of one of a fixed finite set of *job types* $\mathcal{J}$, and each worker in the system is one of a fixed finite set of *worker types* $\mathcal{I}$. We consider a continuum model with infinitesimal workers and jobs and thus refer to *masses* of workers and jobs. Informally, this approach is intended to capture a large market, that is, where many workers and jobs are present at each time step.

We assume a fixed unit mass of workers; we view the space of workers as a measure space, endowed with the Lebesgue measure on $[0, 1]$ and the Borel σ-algebra. Each element $g \in [0, 1]$ represents a worker. Sun (2006) provides a *Fubini extension* of the product measure corresponding to the worker measure space and the probability space $(\Omega, \mathcal{F}, P)$; this extension is, roughly, a rich enough probability measure on the product space such that the Fubini property holds. We leverage this extension in our development.

We wish to model a process by which workers arrive and depart from the system; however, for technical simplicity, we also wish to consider a system where the mass of workers remains finite at all times. To achieve both goals, we consider a system where each worker *regenerates* after every N time periods; we refer to N as the *lifetime* of a worker.[2] We assume the platform knows N.

Formally, fix a distribution ρ over worker types, that is, $\rho_i > 0 \ \forall i \in \mathcal{I}$ such that $\sum_{i \in \mathcal{I}} \rho_i = 1$. We assume that the system initially starts empty prior to $t = 0$, and in each time period $t = 0, \ldots, N-1$, a mass $1/N$ of workers arrives to the system. (In what follows, we ultimately consider a steady-state analysis of the dynamical system, and initial conditions will be irrelevant.) Each worker is of type i with probability ρ_i; these realizations are independent across workers.[3] No further arrivals take place after time period N. Instead, each worker subsequently regenerates every N periods after their arrival: at a regeneration time, the worker type is resampled from the distribution ρ; that is, the new type is i with probability ρ_i, and these regenerations are also independent across workers and across time. The ELLN (theorem 2.16 of Sun 2006) ensures that, at each time t subsequent to time N, the mass of workers of type i in the system is exactly ρ_i. (In what follows, we will consider the scaling regime where ρ_i is held constant and $N \to \infty$.) When the meaning is clear from the context, we sometimes refer to a worker type regeneration as an arrival. Correspondingly, we sometimes refer to ρ_i as the arrival rate of workers of type i.

Each worker has the opportunity to do at most one job during each time period of their lifetime. We assume that in each time period a mass $\mu_j > 0$ of jobs of type j arrive to be matched to workers; each job lives for only a single time period; we call μ_j the *capacity constraint* of job type j. The platform's *matching policy* determines how workers are matched to jobs; we elaborate further on matching policies later.

We assume that type uncertainty exists only for workers; that is, the platform knows the types of arriving jobs exactly, but only knows that each newly arrived worker has independently and identically distributed (i.i.d.) type with distribution ρ and needs to learn the types of workers. We also assume that the arrival rates of jobs $(\mu_j)_{j \in \mathcal{J}}$ and the distribution of worker types $(\rho_i)_{i \in \mathcal{I}}$ are known to the platform.

3.1.4. Matching and the Payoff Matrix. If a worker of type $i \in \mathcal{I}$ is matched to a job of type $j \in \mathcal{J}$, then the resulting match, independent of everything else, generates a Bernoulli reward with success probability $A(i, j) \in [0, 1]$. The matrix A thus characterizes compatibility between workers and jobs. We call the matrix A the *payoff matrix*. Throughout, we assume that no two rows of A are identical. (This mild requirement simply ensures that it is possible, in principle, to distinguish between each pair of worker types.) Because we will only be concerned with the long-run rate of payoff generation, we do not concern ourselves with the division of this payoff between workers and employers. We assume that realized payoffs are observed by the platform.

For ease of exposition, we define an *empty* job type κ, such that all worker types matched to κ generate zero reward, that is, $A(i, \kappa) = 0$ for all i. We view κ as representing the possibility that a worker goes

unmatched, and thus assume that an unbounded capacity of job type κ is available, that is, $\mu_\kappa = \infty$. We assume that κ is included in $\mathcal{J}$.

A key assumption in our work is that the platform *knows* the matrix A. In particular, we are considering a platform that has enough aggregate information to precisely decipher the compatibility between different worker and job types.

We note here that a platform can estimate μ, ρ, and A from data: the job arrival rates μ can be directly estimated empirically because job types are observed, whereas the worker arrival rates ρ and payoff matrix A can be indirectly estimated using the observed outcome data as described in Online Appendix EC.1.1.

3.1.5. Generalized Imbalance. Throughout our technical development, we make a mild structural assumption on the problem instance, defined by the tuple (ρ, μ, A). This is captured by the following definition. We say that arrival rates $\rho = (\rho_i)_{i\in\mathcal{I}}$ and $\mu = (\mu_j)_{j\in\mathcal{J}}$ satisfy the *generalized imbalance condition* if there is no pair of nonempty subsets of worker types and job types $(\mathcal{I}', \mathcal{J}')$, such that the total worker arrival rate of $\mathcal{I}'$ exactly matches the total job capacity of $\mathcal{J}'$. Formally,

$$\sum_{i\in\mathcal{I}'} \rho_i \neq \sum_{j\in\mathcal{J}'} \mu_j \quad \forall \mathcal{I}' \subseteq \mathcal{I}, \mathcal{J}' \subseteq \mathcal{J}, \mathcal{I}' \neq \phi. \tag{1}$$

The generalized imbalance condition holds generically.[4] This condition does not depend on the matrix A. (The condition will ensure that the shadow prices corresponding to capacity constraints under full information are uniquely determined; see Proposition 3.)

3.2. Matching Policies and Platform Objective

A *matching policy* is what the platform uses to match jobs to workers. Informally, we model the following process. The operator knows, at any point in time, the history of each worker in the platform, and also knows the job arrival rates μ_j for $j \in \mathcal{J}$. The matching policy of the platform decides how to match workers and jobs; in particular, it decides which job type each worker is assigned to, while respecting the capacity constraints on job types.

With this intuition in mind, we now formally define a matching policy, and then define the platform's goal: to choose a matching policy that maximizes the long-run average rate of payoff generation.

3.2.1. Worker History. To define the state of the system and the resulting matching dynamics, we need the notion of a worker history; informally, this is the full history of a given worker since her last regeneration. Formally, a *worker history of length* k is a tuple $H_k = ((j_1, r_1), \ldots, (j_k, r_k))$, where $j_{k'}$ is the job type this worker was matched to at her k'th time step in the system since her last regeneration, for $1 \le k' \le k$; and $r_{k'} \in \{0, 1\}$ is the corresponding reward obtained. Because workers persist for N jobs between regenerations, the histories will have lengths $k = 0, \ldots, N-1$. We use H to denote a generic history. We let ϕ denote the empty history (for $k = 0$). We let $\mathcal{H} = \cup_{k=0}^{N-1} (\mathcal{J} \times \{0, 1\})^k$ denote the set of possible histories.

3.2.2. Full System State. The *full system state* (also referred to as the *full state* or simply the *state*) at time t is a mapping from workers to their histories and true types, $\xi_t : [0, 1] \to \mathcal{H} \times \mathcal{I}$.

3.2.3. Observable System State. The platform is not able to observe the true type of a worker; in particular, for any $g \in [0, 1]$, the platform only observes the history of the worker g. Define $\hat{\xi}_t : [0, 1] \to \mathcal{H}$ as the projection of the full state ξ_t onto the set of histories $\mathcal{H}$; this is the *observable state* at time t. Any policy the platform implements must depend on only the observable state.

Recall that the system starts with no workers in the system before time $t = 0$. Our subsequent development will ensure that $\hat{\xi}_t$ is Borel measurable with probability 1 for all times $t = 0, 1, \ldots$.

3.2.4. Matching Policy. The platform uses a *matching policy* to assign each worker to a job type in $\mathcal{J}$ (recall that we think of unmatched workers as being matched to the empty job type κ). As mentioned previously, we assume that any mass of jobs left unmatched in a given period disappears at the end of that period, although our results do not depend on this assumption. Fix N, and recall that the platform is assumed to know N.

Formally, a matching policy is a mapping, for each t, from the observable states $\hat{\xi}_t$ to assignments of workers to job types. We restrict attention to matching policies such that for all $t = 0, 1, \ldots$ and for any measurable $\hat{\xi}_t$, with probability 1, the set of workers with each history in $\mathcal{H}$ assigned to each job type in $\mathcal{J}$ is Borel measurable; we refer to these as *measurable* matching policies.

Furthermore, we restrict attention to matching policies that are *capacity feasible*; a policy is capacity feasible if, for all $t \in \mathbb{N}$ and for any measurable $\hat{\xi}_t$, with probability 1, the set of workers assigned to each job type j has mass (i.e., Lebesgue measure), no more than the capacity μ_j for each $j \in \mathcal{J}$.

The matching policy can choose a randomized assignment; in this case, all relevant randomness used by the policy is encompassed by ω, the state of the world.

The definition of a matching policy and the definitions of measurability and capacity feasibility all

appeal only to the notion of the observable state. We also note in passing that the platform can define a matching policy and check that it is measurable and capacity feasible even without knowing A and ρ.

3.2.5. System Dynamics. Next, we will describe the system dynamics; we subsequently use these to specify the platform objective.

Fix a matching policy. In each period t, for each worker g (with history denoted by H), the matching policy determines the job type j assigned to that worker. If that worker is actually of type i, then the realized payoff is $r \sim \text{Bernoulli}(A(i,j))$ and the new history of g becomes $(H,(j,r))$ (if g does not regenerate); otherwise, the payoff accrues but g regenerates to an empty history with true type resampled (independently) from distribution ρ.

By the same Fubini extension of Sun (2006), for any measurable matching policy π, the set of workers of history H with true type i assigned to job type j will be Borel measurable with probability 1 at all times t; for policy π, call this mass $m_{\pi,t}(H,i,j)$. Then it follows by the ELLN of Sun (2006) that the reward generated from these assignments at time t is $m_{\pi,t}(H,i,j)A(i,j)$. For a general policy π, the mass $m_{\pi,t}(H,i,j)$ is a random variable. Also observe that for any candidate policy π, the platform can compute the distribution of $m_{\pi,t}(H,i,j)$ and hence the reward generated using its knowledge of A and ρ. (The platform can perform this computation offline for any candidate policy, notwithstanding the fact that the true worker types are unobservable.) For brevity we skip the details of the computation for general policies but provide the full calculation for the sufficient subclass of policies that we identify in the next section.

For later reference, we let $x_{\pi,t}(i,j)$ be the derived (random) quantity representing the *fraction* of workers of true type i matched to jobs of type j at time t under policy π; we refer to $x_{\pi,t}$ as the *routing matrix at time t* of policy π. This is a (row) stochastic matrix for each t; that is, each row sums to 1. For times $t \geq N-1$, the mass of workers of true type i in the system is exactly equal to ρ_i. Therefore, for $t \geq N-1$, it follows that $x_{\pi,t}(i,j) = \frac{1}{\rho_i}\sum_{H\in\mathcal{H}} m_{\pi,t}(H,i,j)$.

3.2.6. Platform Objective: Rate of Payoff Generation. Recall that each worker generates a payoff of 1 or 0, in each period. The platform then aims to maximize the long-run average of the mass of workers who generate a payoff of 1 in each period. (This choice of objective is the analog of the total payoff per period objective in a setting with finitely many workers.) As a result of the ELLN of Sun (2006), the long-run average rate of payoff generation is identical to the long-run average of $\sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} A(i,j)\sum_H m_{\pi,t}(H,i,j)$.

The long-run average may not exist for an arbitrary measurable policy, and so formally we define the objective as the limit inferior of the expectation of this quantity:

$$\underline{V}(\pi) = \liminf_{T\to\infty} \mathbb{E}[V_T(\pi)], \tag{2}$$

$$\text{where } V_T(\pi) = \frac{1}{T}\sum_{t=1}^{T}\sum_{i\in\mathcal{I}}\rho_i\sum_{j\in\mathcal{J}} x_{\pi,t}(i,j)A(i,j). \tag{3}$$

In the definition of V_T, we make the substitution that $\rho_i x_{\pi,t}(i,j) = \sum_{H\in\mathcal{H}} m_{\pi,t}(H,i,j)$, because the latter relation holds for all $t \geq N$. The goal is to find policies that maximize this objective. As per our earlier remark, the platform is able to compute offline the objective value $\underline{V}(\pi)$ for any candidate policy π, even though the true worker types are unobservable.

3.3. Worker-History-Only Policies

In general, policies may be time-varying, and may have complex dependence on the observable state $\hat{\xi}_t$. In this section, we introduce a much simpler class of policies that we call *worker-history-only (WHO) policies*. These are policies where, as a function of the history of each individual worker, a job type is drawn independently from a given distribution, which does not depend on time or on the identity of the worker or on the state of the rest of the system.

Formally, a WHO policy is associated with a mapping $\pi : \mathcal{H} \to \Delta_{\mathcal{J}}$, where $\Delta_{\mathcal{J}}$ denotes the probability simplex of distributions on $\mathcal{J}$. (Thus, for WHO policies, we have chosen to identify the notation π with the mapping that defines the policy.) For each worker g with current history H, the job type for g is sampled from the distribution $\pi(H)$, independently of the other workers. We use $\pi(H,j)$ to denote the jth coordinate of $\pi(H)$. WHO policies are *anonymous*; that is, they do not depend on the worker's index. We let Π^N denote the class of WHO policies, for a given N.

The platform operator may choose the mapping π using information available in aggregate, such as the payoff matrix A, the worker type distribution ρ, and the arrival rates of jobs μ. However, the only way that observable state information influences the online matching of a worker to a job in a WHO policy is through the history of the individual worker. For example, suppose that the platform uses a multi-armed bandit algorithm at the level of an individual worker's history to determine the next job they are matched to; in our model this would be a WHO policy. In this sense, WHO policies are *decentralized* in their assignment of intended job types.

In the remainder of the section, we specialize our model to WHO policies; as we show, this yields a substantially more tractable setting. Observe that, a

priori, there is no guarantee that a WHO policy will respect the capacity constraints on jobs.[5] To handle this issue, we begin by ignoring capacity constraints; we define the state dynamics and steady-state of a WHO policy, and use this to identify the steady-state rate of payoff generation for such policies. We then characterize the subclass of WHO policies such that capacity constraints are satisfied. Finally, we make the important observation that we may restrict attention to WHO policies *essentially without loss of optimality* (see Proposition 2). For this reason, in the sequel, we focus on finding approximately optimal WHO policies.

Steady state of a WHO policy π. Assume no capacity constraints, i.e., $\mu_j = \infty$ for all j. Because WHO policies are anonymous, in analyzing WHO policies it is convenient to work instead with a *reduced* state ν_t, called the *system profile*, which only measures the aggregate mass of workers with history H and true type i just before period t, for each pair (H,i). Formally, $\nu_t(H,i) \triangleq |\xi_t^{-1}(H,i)|$, where $|\cdot|$ denotes the Lebesgue measure of the set. As before, we emphasize that this system profile is not observable to the platform, as it does not know the true types of workers. Note that, using the ELLN of Sun (2006), w.p. 1, we have that $m_{\pi,t}(H,i,j) = \nu_t(H,i)\pi(H,j)$ is the total mass of workers of true type i with history H who are assigned to jobs of type j at time t.

The system dynamics are as follows. Because the system starts empty before $t = 0$, we have

$$\nu_0(H,i) = 0 \quad \text{for all non-empty histories } H \in \mathcal{H}\backslash\{\phi\} \text{ and all } i. \tag{4}$$

Worker arrivals and type regenerations lead to

$$\nu_t(\phi,i) = \rho_i/N \qquad \text{for all } t \geq 0 \ldots. \tag{5}$$

For all $i, j, t \geq 1$, and histories $H \in \mathcal{H}$ of length $\leq N-2$, we have

$$\nu_t\big((H,(j,1)),i\big) = \nu_{t-1}(H,i)\pi(H,j)A(i,j); \tag{6}$$
$$\nu_t\big((H,(j,0)),i\big) = \nu_{t-1}(H,i)\pi(H,j)\big(1-A(i,j)\big). \tag{7}$$

Because π and ρ are time independent, the dynamics (4)–(7) yield a unique steady state after $N-1$ time periods; that is, $\nu_s = \nu_t$ for all $s,t \geq N-1$ w.p. 1. Abusing notation, we use ν_π to denote the *steady-state* system profile induced by the WHO policy π. The steady-state can be inductively computed over histories of increasing length: for the empty history ϕ of length zero, we have

$$\nu_\pi(\phi,i) = \rho_i/N. \tag{8}$$

Then for any history H of length $0,\ldots,N-2$, we have

$$\nu_\pi\big((H,(j,1)),i\big) = \nu_\pi(H,i)\pi(H,j)A(i,j); \tag{9}$$
$$\nu_\pi\big((H,(j,0)),i\big) = \nu_\pi(H,i)\pi(H,j)\big(1-A(i,j)\big). \tag{10}$$

Routing matrix of a WHO policy π. In steady state, π induces a time-independent fraction $x_\pi(i,j)$ of the mass of workers of true type i that are assigned to type j jobs in each time step. In particular,

$$x_\pi(i,j) \triangleq \frac{\sum_{H\in\mathcal{H}} \nu_\pi(H,i)\pi(H,j)}{\sum_{H\in\mathcal{H}} \nu_\pi(H,i)} = \frac{\sum_{H\in\mathcal{H}} \nu_\pi(H,i)\pi(H,j)}{\rho_i}. \tag{11}$$

Let

$$\mathcal{X}^N \triangleq \{x_\pi : \pi \in \Pi^N\} \subseteq [0,1]^{|\mathcal{I}|\times|\mathcal{J}|} \tag{12}$$

be the set of (steady-state) routing matrices achievable (when each worker does N jobs) by WHO policies, that is, for $\pi \in \Pi^N$. Again, we emphasize that capacity constraints are ignored in the definition of $\mathcal{X}^N$. In Online Appendix EC.1.4, we show the following.

Proposition 1. *The set $\mathcal{X}^N$ is a convex polytope.*

Steady-state rate of payoff generation of a WHO policy π. Recall the T-period average payoff generation rate defined in (3). Because a WHO policy is in steady state for all $t \geq N-1$, it follows that for such a policy the limit $\lim_{T\to\infty} V_T$ exists and is equal to the following *steady-state rate of payoff generation* $W^N(\pi)$:

$$W^N(\pi) \triangleq \sum_{i\in\mathcal{I}} \rho_i \sum_{j\in\mathcal{J}} x_\pi(i,j)A(i,j). \tag{13}$$

w.p. 1, this is the payoff generated per time step in steady-state by the policy π across the entire population of jobs and workers, because $x_\pi(i,j)$ is the fraction of workers of true type i matched to jobs of type j, and $A(i,j)$ is the fraction of these matches that generate a unit reward. In the sequel, our goal will be to maximize this rate of payoff generation.

3.3.1. Satisfying Capacity Constraints. We now return to enforcing the capacity constraints, i.e., $\mu_j < \infty$ for $j \neq \kappa$. In our analysis, we restrict attention to *WHO policies that satisfy capacity constraints*. Given the above definitions, this is straightforward: we restrict attention to WHO policies $\pi \in \Pi^N$ such that the steady-state routing matrix x_π does not require any more than mass μ_j of jobs of type j:

$$\sum_{i\in\mathcal{I}} \rho_i x_\pi(i,j) = \sum_{i\in\mathcal{I}} \sum_{H\in\mathcal{H}} \nu_\pi(H,i)\pi(H,j) \leq \mu_j \quad \forall j \in \mathcal{J}. \tag{14}$$

Any WHO policy π that satisfies this constraint will ensure that the capacity constraints are satisfied by the implied assignment in steady-state (i.e., for $t \geq N-1$) w.p. 1 by the ELLN of Sun (2006). In fact, because we assume the system starts empty, the following lemma establishes that for any such policy, w.p. 1, capacity constraints are *never* violated. The lemma is proved in Online Appendix EC.1.2.

Lemma 1. *Recall that the system starts empty, i.e., $v_0(H,i) = 0$ for all $H \neq \phi, i \in \mathcal{I}$. Suppose that the WHO policy π satisfies* (14). *Then at all times $t = 0, 1, \ldots$, w.p. 1, the implied assignment satisfies the capacity constraint; i.e., at each time t and for each job type j, the mass of workers matched to jobs of type j does not exceed μ_j:*

$$\sum_{i \in \mathcal{I}} \sum_{H} v_t(H,i)\pi(H,j) \leq \mu_j \qquad \forall j \in \mathcal{J}. \tag{15}$$

Furthermore, the system reaches steady-state at $t = N-1$ and remains in steady-state for all $t \geq N-1$.

3.3.1.1. Optimality of WHO Policies.

We now establish that the restriction to WHO policies is without loss of optimality. Recall that $V_T(\pi)$ as defined in (3) is the T-period average payoff achieved by the (arbitrary, possibly time-varying) measurable and capacity-feasible policy π. Hence, the largest possible asymptotic rate of payoff accumulation under policy π is $\overline{V}(\pi) \triangleq \limsup_{T\to\infty} \mathbb{E}[V_T(\pi)]$. The next proposition establishes that a WHO policy exists that satisfies capacity constraints and yields a steady-state rate of payoff generation arbitrarily close to $\overline{V}(\pi)$. The proof can be found in Online Appendix EC.1.3.

Proposition 2. *Fix A, ρ, μ, and N. Fix any feasible policy π and any $\varepsilon > 0$. Then there is a worker-history-only (WHO) policy satisfying* (14) *that achieves a steady-state rate of payoff accumulation exceeding $\overline{V}(\pi) - \varepsilon$.*

3.4. The Optimization Problem

We are now in position to state our optimization problem of interest. We want to find a WHO policy π that maximizes the steady-state rate of payoff generation $W^N(\pi)$, subject to the capacity constraints (14). Formally, we have the following problem:

$$\text{maximize} \quad W^N(\pi) \triangleq \sum_{i \in \mathcal{I}} \rho_i \sum_{j \in \mathcal{J}} x_\pi(i,j) A(i,j); \tag{16}$$

$$\text{subject to} \quad \sum_{i \in \mathcal{I}} \rho_i x_\pi(i,j) \leq \mu_j \qquad \forall j \in \mathcal{J}; \tag{17}$$

$$x_\pi \in \mathcal{X}^N. \tag{18}$$

Because $\mathcal{X}^N$ (defined in (12)) is a convex polytope, this is a linear program, albeit a complex one. The complexity of this problem is hidden in the complexity of the set $\mathcal{X}^N$, which includes all possible routing matrices that can be obtained using WHO policies $\pi \in \Pi^N$. The remainder of our paper is devoted to solving this problem and characterizing its value by considering an asymptotic regime where $N \to \infty$.

3.5. The Benchmark: Full Information Setting

We evaluate our performance relative to a natural benchmark: the maximal rate of payoff generation possible if worker types are perfectly *known* upon arrival. We will refer to this as the full information setting. In this case, *any* (row) stochastic matrix is feasible as a routing matrix. Let $\mathcal{D}$ denote the set of all row stochastic matrices:

$$\mathcal{D} = \left\{ x \in \mathbb{R}^{|\mathcal{I}|\times|\mathcal{J}|} : x(i,j) \geq 0;\ \sum_{j \in \mathcal{J}} x(i,j) = 1 \right\}. \tag{19}$$

Any routing matrix in $\mathcal{D}$ is implementable by a simple policy if worker types are perfectly known: given a desired routing matrix $x \in \mathcal{D}$, at each time step t we match a fraction $x(i,j)$ of workers of type i to jobs of type j.

Thus, with known worker types, the maximal rate of payoff generation is given by the solution to the following optimization problem:

$$\text{maximize} \quad \sum_{i \in \mathcal{I}} \rho_i \sum_{j \in \mathcal{J}} x(i,j) A(i,j); \tag{20}$$

$$\text{subject to} \quad \sum_{i \in \mathcal{I}} \rho_i x(i,j) \leq \mu_j \qquad \forall j \in \mathcal{J}; \tag{21}$$

$$x \in \mathcal{D}. \tag{22}$$

We let V^* denote the maximal value of the preceding optimization problem and let x^* denote the solution (breaking ties arbitrarily). We further use $\mathcal{J}^*_{\text{full}}$ to denote the set of fully utilized job types

$$\mathcal{J}^*_{\text{full}} \triangleq \left\{ j \in \mathcal{J} : \sum_{i \in \mathcal{I}} \rho_i x^*(i,j) = \mu_j \right\}. \tag{23}$$

This linear program is a special case of the *static planning problem* that arises frequently in the operations literature (Ata and Kumar 2005). The problem can also be viewed as a version of the assignment problem of Shapley and Shubik (1971), in which the resources are divisible. We denote the shadow prices associated with the capacity constraints (21) by[6] $p^* = (p^*_j)_{j \in \mathcal{J}}$. We prove the following fact about these prices in Online Appendix EC.1.5.

Proposition 3. Under the generalized imbalance condition (1), the job shadow prices p^* are uniquely determined.

As we shall see, these uniquely defined prices p^* will be key to our solution to the problem.

3.6. Regret

We evaluate the performance of a given policy in terms of its *regret* relative to V^*. In particular, given N and a WHO policy π satisfying (14), we define the regret of π as $V^* - W^N(\pi)$.

We focus on the asymptotic regime where $N \to \infty$ and try to find policies that have small regret in this regime. This asymptotic regime provides tractability, allowing us to identify structural aspects of policies that perform well. In particular, we focus on developing policies that achieve a nearly optimal *rate* at which the regret $V^* - W^N(\pi_N)$ approaches zero.

3.7. Summary

We summarize our model as follows.

- The platform chooses a matching policy. In particular, without loss of optimality, it chooses a WHO policy π.
- The policy π induces a steady-state system profile ν_π and associated steady-state routing matrix x_π; that is, at all times $t \geq N-1$, the system profile is ν_π and the mass of workers of type i matched to jobs of type j is $x_\pi(i,j)$.
- The steady-state routing matrix x_π induces a steady-state rate of payoff generation $W^N(\pi)$.
- The regret of the policy π is $V^* - W^N(\pi)$. We focus on finding WHO policies that yield low regret.

4. Decentralized Explore-Then-Exploit for Matching (DEEM): A Payoff-Maximizing Policy

In this section, we present our proposed policy *DEEM: Decentralized Explore-then-Exploit for Matching*. Our main result (Theorem 1) will quantify the regret performance of DEEM and characterize it as nearly optimal. DEEM is formally defined in Figure 1 with supporting definitions in Figure 2. To assist the reader, we provide an informal schematic of DEEM in Figure 3.

DEEM operates individually on every arriving worker; in fact DEEM is a WHO policy (WHO policies were defined in Section 3.3). As shown in Figure 3, DEEM is divided into two phases: Explore and Exploit. In the Explore phase the policy efficiently learns the type of the worker with appropriate confidence and generates a type label. In the Exploit phase, the algorithm focuses on payoff maximization for the given type label in a manner that accounts for system-level capacity constraints.

The Explore phase involves two possible modes of operation: Guessing and Confirmation, and starts in Guessing mode. The Guessing mode is in effect when there is not enough confidence in the maximum a posteriori (MAP) estimate of the worker type based on the observations so far (condition (a) in Figure 3). It assigns the worker to job types uniformly at random and aims to build confidence in the MAP estimate. The Confirmation mode is in effect when there is sufficient confidence in the MAP estimate to merit focusing on confirming that it is indeed the true type, but not enough to actually start exploiting (condition (b) in Figure 3). In this mode, DEEM boosts the confidence that the guessed type is correct, trying to rule out types in the carefully defined set $\mathrm{Str}(MAP)$ (defined in Figure 1, Equation (24)) that must be distinguished to facilitate exploitation. It achieves this goal while minimizing the loss in payoff by sampling job types from an appropriate distribution $\alpha(MAP)$, defined in Figure 2. Once an appropriate confidence level is reached in the MAP estimate (condition (c) in Figure 3), the worker is labeled according to this estimate and the algorithm permanently enters the Exploit phase, in which the label is treated as the true type of the worker.

DEEM uses externality adjustments on payoffs to capture the effect of system-wide aggregate capacity constraints. These adjustments are achieved by using *shadow prices* p^* for the capacity constraint (21) in the problem with known worker types. Under these adjusted payoffs, a particular worker type may have multiple optimal job types. Appropriate tie-breaking across these types in the Exploit phase is necessary to satisfy the aggregate capacity constraints. This is achieved by using a specifically designed routing matrix y^*, which is a perturbed version of the solution x^* to the problem with known worker types (20)–(22). The matrix y^* is defined in Figure 2, and the following proposition shows the existence of y^* satisfying the conditions specified in the figure.

Proposition 4. Suppose that the generalized imbalance condition is satisfied. Then, for any N large enough, there exists a feasible routing matrix y^* such that (27)–(31) hold.

The proof is in Online Appendix EC.2.2 and shows, moreover, that as $N \to \infty$, a vanishing fraction of workers are in the Explore phase, that is, $m_{\mathrm{xplr}}(i,j) = o(1)\ \forall i \in \mathcal{I}, j \in \mathcal{J}$, which a vanishing fraction of workers are mislabeled at the end of the Explore phase, that is, $l(i,i') = o(1)\ \forall i \neq i' \in \mathcal{I}$, and that there is a small perturbation of x^* that is a feasible solution to (27)–(31), that is, $y^* = x^* + o(1)$.

In the next section, we use an example to illustrate the operation of DEEM. Before we continue, we make three important remarks.

Remark 1 (DEEM Is a WHO Policy)**.** Observe that DEEM as defined in Figure 1 is a WHO policy; in other words, DEEM defines a mapping from workers' histories to a distribution over job types. The input parameters are the model primitives, which are then used to precompute the derived quantities p^*, $(\mathrm{Str}(i))_{i\in\mathcal{I}}$, $(\alpha(i))_{i\in\mathcal{I}}$, and y^*; these are all functions of model primitives only, with

Figure 1. Definition of DEEM

DEEM: Decentralized Explore-then-Exploit for Matching

Input parameters: $\mathcal{I}$, $\mathcal{J}$, A, ρ, μ, N such that the generalized imbalance condition (1) holds.
Pre-compute:

- The $\mathcal{J}$-vector p^* of shadow prices for the capacity constraint (21) in the problem with known types (20)–(22). (Recall from Proposition 3 that under the generalized imbalance condition (1) the prices p^* are uniquely determined.)
- For each $i \in \mathcal{I}$, the set of worker types

$$\mathrm{Str}(i) \triangleq \{i' : \mathcal{J}(i) \setminus \mathcal{J}(i') \neq \emptyset\} \quad \text{where } \mathcal{J}(i) \triangleq \arg\max_{j \in \mathcal{J}} A(i,j) - p_j^* . \tag{24}$$

- The distribution $\alpha(i) = \alpha(i, \mathcal{I}, \mathcal{J}, A, p^*, \mathrm{Str}(i))$ over $\mathcal{J}$, for all $i \in \mathcal{I}$. Defined in Figure 2.
- $(|\mathcal{I}| \times |\mathcal{J}|)$-right stochastic matrix $y^* = y^*(\mathcal{I}, \mathcal{J}, A, \rho, \mu, N) \in \mathcal{D}$. Defined in Figure 2.

▷ Main Routine
procedure DEEM ▷ Acts independently on each worker, over her lifetime, from arrival to departure
  ▷ Initialization:
  $\lambda(i) \leftarrow \rho_i$ for all $i \in \mathcal{I}$ ▷ The un-normalized posterior probabilities; initialized to the prior
  $\mathrm{MAP} \leftarrow \arg\max_{i \in \mathcal{I}} \lambda(i)$ ▷ Initialization of the MAP estimate
  $Label \leftarrow \phi$ ▷ Worker label; initially unassigned, denoted by ϕ
  $k \leftarrow 0$ ▷ Number of time steps the worker has been in the system = Length of the worker's history

  ▷ Explore phase:
  while $Label = \phi$ and $k < N$ **do**
    $k \leftarrow k + 1$ ▷ At the next time step
    Assign job type $j_k \sim$ Explore(N, λ, MAP, α(MAP))
    Observe reward r_k
    $\lambda(i) \leftarrow \lambda(i) \times (A(i, j_k)\mathbf{1}_{\{r_k = 1\}} + (1 - A(i, j_k))\mathbf{1}_{\{r_k = 0\}})$, for all $i \in \mathcal{I}$
    $\mathrm{MAP} \leftarrow \arg\max_{i \in \mathcal{I}} \lambda(i)$
    if $\min_{i \neq \mathrm{MAP}} \frac{\lambda(\mathrm{MAP})}{\lambda(i)} \geq \log N$ and $\min_{i \in \mathrm{Str}(\mathrm{MAP})} \frac{\lambda(\mathrm{MAP})}{\lambda(i)} \geq N$ **then** ▷ If Confirmation is complete
      $Label \leftarrow \mathrm{MAP}$ ▷ Worker label assigned. Will end Explore phase.
    end if
  end while

  ▷ Exploit phase:
  while $k < N$ **do**
    $k \leftarrow k + 1$ ▷ At the next time step
    Assign job type $j_k \sim$ Exploit($Label$, y^*)
  end while
end procedure

▷ Functions
function Explore(N, λ, MAP, α_{MAP})
  if $\min_{i \neq \mathrm{MAP}} \frac{\lambda(\mathrm{MAP})}{\lambda(i)} < \log N$ **then** ▷ If MAP estimate is noisy
    $dist \leftarrow \mathrm{Uniform}(\mathcal{J})$ ▷ **Guessing**
  else ▷ MAP estimate is somewhat confident
    $dist \leftarrow \alpha_{\mathrm{MAP}}$ ▷ **Confirmation**
  end if
  return $dist$
end function

function Exploit($Label$, y^*)
  $dist \leftarrow y^*(Label, \cdot)$ ▷ Sample from distribution given by the $Label$-th row of y^*
  return $dist$
end function

---

no dependence on the current system state. For any individual worker, the control logic of DEEM relies only on the posterior $\lambda(\cdot)$, the current MAP estimate, and *Label*, all of which are functions of the history of the individual worker. Before *Label* is set, job types are drawn using Explore(), whereas after it is set, job types are drawn using Exploit(). These functions again construct a job type distribution based only on the history of

Figure 2. Definitions of $\alpha(i)$ and y^*

Definitions

Definition of $\alpha(i,\mathcal{I},\mathcal{J},A,p^*,\mathrm{Str}(i)) \in \Delta(\mathcal{J})$.

Let $U(i) \triangleq \max_{j\in\mathcal{J}} A(i,j) - p_j^*$ be the maximal externality-adjusted payoff of worker type i. Define the set

$$\mathcal{A}(i) \triangleq \arg\min_{\alpha\in\Delta(\mathcal{J})} \frac{\sum_{j\in\mathcal{J}} \alpha_j \left(U(i) - [A(i,j) - p_j^*]\right)}{\min_{i'\in\mathrm{Str}(i)} \sum_{j\in\mathcal{J}} \alpha_j \mathrm{KL}(i,i'|j)}, \tag{25}$$

where $\Delta(\mathcal{J})$ is the set of distributions over $\mathcal{J}$ and $\mathrm{KL}(i,i'|j) \triangleq A(i,j)\log\frac{A(i,j)}{A(i',j)} + (1-A(i,j))\log\frac{1-A(i,j)}{1-A(i',j)}$ is the Kullback–Leibler divergence between Bernoulli$(A(i,j))$ and Bernoulli$(A(i',j))$. Then choose $\alpha(i)$ as per

$$\alpha(i) \in \arg\max_{\alpha'\in\mathcal{A}(i)} \min_{i'\in\mathrm{Str}(i)} \sum_{j\in\mathcal{J}} \alpha'_j \mathrm{KL}(i,i'|j), \tag{26}$$

breaking ties arbitrarily.

Definition of $y^*(\mathcal{I},\mathcal{J},A,\rho,\mu,N) \in \mathcal{D}$.

The Explore phase, in particular the function Explore() and the confirmation policy $\alpha(i)$ for all $i\in\mathcal{I}$, defines the following masses, which are time invariant for all $t \geq N$:

- $m_{\mathrm{xplr}}(i,j) \triangleq$ The mass of type i workers who are in the Explore phase and get assigned to type-j jobs in a given time step.
- $l(i,i') \triangleq$ The mass of type i workers in the system who were labeled as being of type i' at the end of the Explore phase, and are now in the Exploit phase.

Then, based on a solution x^* to (20)–(22) and $\mathcal{J}^*_{\mathrm{full}}$ defined in (23), we choose a routing matrix y^* in the Exploit phase that satisfies:

$$y(i,j) = 0 \qquad \forall i\in\mathcal{I}, j\in\mathcal{J} \text{ s.t. } x^*(i,j) = 0\,; \tag{27}$$

$$m(i,j) = m_{\mathrm{xplr}}(i,j) + \sum_{i'\in\mathcal{I}} l(i,i')y(i',j) \qquad \forall i\in\mathcal{I}, j\in\mathcal{J}\,; \tag{28}$$

$$\sum_{i\in\mathcal{I}} m(i,j) = \mu_j \qquad \forall j\in\mathcal{J}^*_{\mathrm{full}}\,; \tag{29}$$

$$\sum_{i\in\mathcal{I}} m(i,j) < \mu_j \qquad \forall j\in\mathcal{J}\backslash\mathcal{J}^*_{\mathrm{full}}\,; \tag{30}$$

$$y \in \mathcal{D}. \tag{31}$$

Since the generalized imbalance condition holds, using Proposition 4, for any N large enough, there exists a feasible routing matrix y^* such that (27)–(31) hold.[7]

Notes. In Online Appendix EC.2.1, we show that (25) can be expressed as a small linear program (with $|\mathcal{I}|$ constraints and $|\mathcal{J}|$ variables). The quantity $m(i,j)$ given by (28) represents the mass of type i workers that is matched to type j jobs in steady state.

the individual worker as captured by $\lambda(\cdot)$, MAP, and *Label* (and model primitives and quantities derived from them).

Remark 2 (DEEM Satisfies Capacity Constraints)**.** Proposition 4 shows that there is a feasible solution to the Constraints (27)–(31). We use this solution y^* as the routing matrix during exploitation. Because the capacity constraints are incorporated in (29) and (30), it follows that DEEM does not run out of jobs of any type, in any time step.

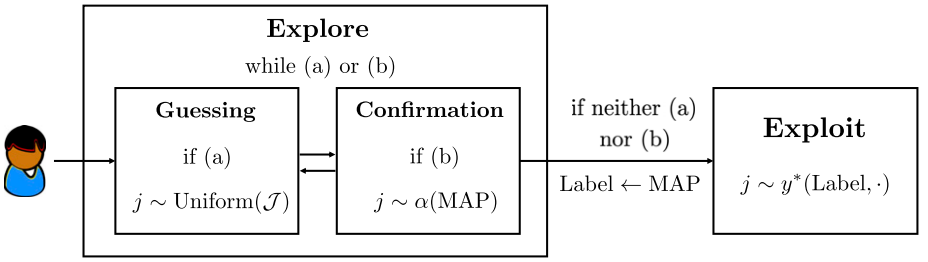
One possible concern could be that the definition of y^* depends on DEEM and DEEM itself depends on y^*. In fact, there is no circularity in the definitions, as explained in the following remark.

Remark 3. Because DEEM is a WHO policy, and under DEEM, each worker completes the Explore phase before entering the Exploit phase, the matrix y^* satisfying (27)–(31) can be computed in an offline fashion using only aggregate statistics $m_{\mathrm{xplr}}(i,j)$ and $l(i,i')$ that depend on the Explore phase alone. This matrix y^* is then only used to define the Exploit phase of DEEM to ensure that capacity constraints are met.

4.1. Key Features of DEEM via an Example

In this section, we discuss the structure and the main features of DEEM by way of an example. Consider a simple setting in the context of a labor platform like Upwork, in which worker skills differ along two dimensions: Programming and Design. Furthermore, suppose the skill level is binary in each dimension: each worker either has that skill or does not. Suppose that the worker population is composed of three worker types (in order): Programmers (who know only Programming), Designers (who know only Design), and All-rounders (who know both Programming and Design). Finally, there are three job types (in order) with the corresponding subsets of relevant skills: Programming (which depend on only the

Figure 3. (Color online) Schematic of DEEM at the Individual Worker Level



Initialization:

$\lambda(i) \leftarrow \rho_i \; \forall \, i \in \mathcal{I}$

$\text{MAP} \leftarrow \arg\max_{i \in \mathcal{I}} \lambda(i)$

Update:

$\lambda(i)$: see line 13 of Figure 1

$\text{MAP} \leftarrow \arg\max_{i \in \mathcal{I}} \lambda(i)$

Conditions:

$$\text{(a)} : \min_{i \neq \text{MAP}} \frac{\lambda(\text{MAP})}{\lambda(i)} < \log N$$

$$\text{(b)} : \min_{i \neq \text{MAP}} \frac{\lambda(\text{MAP})}{\lambda(i)} \geq \log N \text{ and } \min_{i \in \text{Str(MAP)}} \frac{\lambda(\text{MAP})}{\lambda(i)} < N$$

Notes. $\lambda(i)$ for each $i \in \mathcal{I}$ is the unnormalized posterior probability of the type being i before each job assignment. It is initialized with the prior ρ_i, and is updated after each job assignment based on the observed reward (line 13 of Figure 1). α and y^* are defined in Figure 2, whereas (i) is defined in (24).

Programming skill), Design (which depend on only the Design skill), and Mixed (which depend on both skills). Let the payoff matrix (consistent with the subsets of relevant skills) be

$$A = \begin{bmatrix} \text{Programming} & \text{Design} & \text{Mixed} & \\ 0.5 & 0.2 & 0.1 & \text{Programmers} \\ 0.3 & 0.8 & 0.2 & \text{Designers} \\ 0.5 & 0.8 & 0.6 & \text{All-rounders} \end{bmatrix} \tag{32}$$

Let the arrival rates be $\rho = [\,0.4/1.9 \quad 0.6/1.9 \quad 0.9/1.9\,]^T$ and the job type capacities be $\mu = [\,1/1.9 \quad 1/1.9 \quad 1/1.9\,]^T$. In this example, the optimal solution to the benchmark Problem (20)–(22) with *known* types results in the following allocation of the masses of workers to jobs:

$$\rho^T x^* = \left[\rho_i x^*(i,j)\right]_{i \in \mathcal{I}, j \in \mathcal{J}} = \begin{bmatrix} 0.4/1.9 & 0 & 0 \\ 0 & 0.6/1.9 & 0 \\ 0 & 0.4/1.9 & 0.5/1.9 \end{bmatrix}. \tag{33}$$

There are three key features of DEEM, which we now discuss.

1. *Shadow prices to account for capacity constraints.* The intuition behind using p^* for externality adjustment of payoffs under DEEM is that with large N, learning will occur quickly relative to the worker lifetime, and p^* will approximate well the shadow prices even with unknown worker types. At the high level, DEEM is a near-optimal policy for the unconstrained externality-adjusted bandit problem:

$$\text{maximize}_{x_\pi \in \mathcal{X}^N} \sum_{i \in \mathcal{I}} \rho_i \sum_{j \in \mathcal{J}} x_\pi(i,j) \left(A(i,j) - p_j^* \right). \tag{34}$$

In our example, the shadow prices corresponding to the capacity constraints in the benchmark linear program under full information (20)–(22) are[8] $p^* = [\,0 \quad 0.2 \quad 0\,]^T$. DEEM makes job-assignment decisions based on the externality-adjusted payoff matrix

$$\left[A(i,j) - p_j^*\right]_{i \in \mathcal{I}, j \in \mathcal{J}} = \begin{bmatrix} \mathbf{0.5} & 0.0 & 0.1 \\ 0.3 & \mathbf{0.6} & 0.2 \\ 0.5 & \mathbf{0.6} & \mathbf{0.6} \end{bmatrix} \tag{35}$$

instead of the original payoff matrix A. The sets $\mathcal{J}(i)$ defined in Figure 1, Equation (24) capture the job types that maximize the externality-adjusted payoff for worker type i; these maximal externality-adjusted payoffs for each worker type are shown in bold face in the matrix. Hence, we have $\mathcal{J}(\text{Programmer}) = \{\text{Programming}\}$, $\mathcal{J}(\text{Designer}) = \{\text{Design}\}$, and $\mathcal{J}(\text{All-rounder}) = \{\text{Design, Mixed}\}$. The routing matrix y^* in the Exploit phase of DEEM assigns labeled workers exclusively to these jobs during exploitation, just as x^* in the benchmark solution exclusively assigns workers to these jobs (see (33)).

However, DEEM also needs to satisfy capacity constraints to be feasible. As implied by the following fact, in general the $\mathcal{J}(i)$ may not be singleton sets (as is the case for All-rounders) and appropriate *tie-breaking* between multiple optimal job types for one or more worker types is *necessary during exploitation* to avoid capacity violations.

Fact 1. Under the generalized imbalance condition, as long as there is at least one capacity constraint that is binding in some optimal solution x^* to the benchmark problem (20)–(22) with known types, there is at least one worker i such that $x^*(i,\cdot)$ is supported on multiple job types. This implies that $\mathcal{J}(i)$ has more than one element.

Proof of Fact 1. Fix x^* such that at least one capacity constraint binds; that is, $\mathcal{J}^*_{\text{full}}$ defined in (23) is nonempty. Consider worker types $\mathcal{I}^*_{\text{full}} = \{i : \exists j \in \mathcal{J}^*_{\text{full}}$ s.t. $x^*(i,j) > 0\}$. By the generalized imbalance condition and the fact that all worker types are fully matched (recall that the empty job type κ, which serves as a proxy for remaining unmatched, is included in $\mathcal{J}$), there must be some $i \in \mathcal{I}^*_{\text{full}}$ and $j' \notin \mathcal{J}^*_{\text{full}}$ such that $x^*(i,j') > 0$. (If not, $\sum_{i\in\mathcal{I}^*_{\text{full}}} \rho_i = \sum_{j\in\mathcal{J}^*_{\text{full}}} \mu_j$, which contradicts the generalized imbalance condition.)

In DEEM, the exploitation-phase routing matrix y^* is carefully constructed in (27)–(31) to achieve the proper tie-breaking (the feasibility of this construction is shown in Proposition 4). Because of this construction, DEEM simultaneously satisfies (a) aggregate capacity constraints and (b) complementary slackness conditions with respect to the prices p^*. These properties are key to showing the near-optimality of DEEM for the original capacity-constrained optimization problem (16).

2. *Appropriate learning goals for the Explore phase.* DEEM's tolerance for labeling errors in the Explore phase depends on their impact on payoffs during exploitation. For instance, in our example, suppose that at the end of the Explore phase, the algorithm mislabels a Programmer as an All-rounder. This has a dire impact on payoffs: the Programmer is then assigned to Design or Mixed jobs in the Exploit phase (because $\mathcal{J}$(All-rounder) = {Design, Mixed}), neither of which is optimal for Programmers. These errors lead to a constant regret relative to the optimal externality-adjusted payoffs per unit mass of workers per time step.

In fact, in our example, every other kind of mislabeling is also similarly problematic, with one exception: If an All-rounder is labeled as a Designer, this is acceptable, because the worker will then be assigned Design jobs during exploitation, but Design $\in \mathcal{J}$(All-rounder); that is, $\mathcal{J}$(Designer) $\subset \mathcal{J}$(All-rounder). To ensure that capacity constraints do not pose a problem, we nevertheless ensure that even such acceptable mislabeling occurs for only $\Theta(1/\log N)$ fraction of workers.

This motivates our definition of the sets Str(i) for each worker type i: this is the set of types that i must be distinguished from with high confidence or strongly distinguished. In particular, if $\mathcal{J}(i) \setminus \mathcal{J}(i') \neq \phi$, then $i' \in \text{Str}(i)$. The target error probability for types in Str(i) is chosen to be $1/N$ (see the second condition in line 15 of Figure 1): if we choose a much larger target, we will incur a relatively large expected regret during exploitation because of misclassification; if we choose a smaller target, the Explore phase will be unnecessarily long, and we will thus incur a relatively large regret in the Explore phase. In our example Str(Programmer) = {Designer, All-rounder}, Str(Designer) = {Programmer}, and Str(All-rounder) = {Programmer, Designer}. For every other type $i' \notin \text{Str}(i)$, we have $\mathcal{J}(i) \subseteq \mathcal{J}(i')$, and we weakly distinguish i from such i', with a target misclassification probability of $1/\log N$ (see the first condition in line 15 of Figure 1).

3. *Minimizing regret during Confirmation.* After quickly obtaining a fairly confident estimate i for the worker type using the Guessing mode, DEEM attempts to distinguish i from all types in Str(i) with high confidence using the Confirmation mode. Regret relative to the largest possible externality-adjusted payoff $U(i) = \max_j(A(i,j) - p^*_j)$ for worker type i may be inevitable in this process. A key feature underlying the near optimality of DEEM is that it tries to minimize the regret incurred during Confirmation.

For instance, the only job type that is optimal for Programmers; that is, Programming does not allow the policy to distinguish between Programmers and All-rounders. Thus, if the guessed worker type is Programmer, then because All-rounder $\in$ Str(Programmer), during Confirmation, DEEM must assign the worker either Design or Mixed jobs in order to make sure she is not an All-rounder. Thus, confirming the guess necessitates regret in the event that the guess is correct.

To minimize this regret, DEEM samples job types during Confirmation of worker type i from the carefully chosen distribution $\alpha(i)$ defined in Figure 2. For a job type distribution $\alpha \in \Delta(\mathcal{J})$, for workers of true type i, the smallest value of the log posterior odds, $\min_{i'\in\text{Str}(i)} \log \lambda(i)/\lambda(i')$, increases at an expected rate of $\min_{i'\in\text{Str}(i)} \sum_{j\in\mathcal{J}} \alpha_j \text{KL}(i,i'|j)$. In order to confirm i against worker types in Str(i) with a probability of error of $1/N$, the smallest value of log posterior odds needs to cross the threshold of $\log N$ (see second condition in line 15 in Figure 1). The expected number of jobs needed to cross this threshold is $\log N/(\min_{i'\in\text{Str}(i)} \sum_{j\in\mathcal{J}} \alpha_j \text{KL}(i,i'|j))$. Hence, the expected externality-adjusted regret incurred during Confirmation is

$$\frac{\log N}{N} \frac{\sum_{j\in\mathcal{J}} \alpha_j \left(U(i) - \left[A(i,j) - p^*_j\right]\right)}{\min_{i'\in\text{Str}(i)} \sum_{j\in\mathcal{J}} \alpha_j \text{KL}(i,i'|j)},$$

where the factor N in the denominator arises from the worker's lifetime, because regret is defined per period. DEEM choses a policy $\alpha(i)$ that minimizes this quantity, which in effect minimizes the ratio of the rate of regret accumulation and the rate of learning, or, informally, the *regret per unit of learning*. This minimal regret of $C(i)\frac{\log N}{N}$ to leading order is inevitable per unit mass of workers of type i for any optimal policy that solves (34) for a large N, where

$$C(i) \triangleq \min_{\alpha\in\Delta(\mathcal{J})} \frac{\sum_{j\in\mathcal{J}} \alpha_j \left(U(i) - \left[A(i,j) - p^*_j\right]\right)}{\min_{i'\in\text{Str}(i)} \sum_{j\in\mathcal{J}} \alpha_j \text{KL}(i,i'|j)}. \tag{36}$$

In Online Appendix EC.2.1, we show that the above optimization problem reduces to a linear program. In general, this problem can have several optimal solutions, denoted by the set $\mathcal{A}(i)$. Of these, DEEM chooses the one that gives the highest learning rate.[9]

In our example, $U(\text{Programmer}) = 0.5$, and thus

$$\left[U(\text{Programmer}) - \left(A(\text{Programmer}, \text{j}) - p_j^*\right)\right]_{j\in\mathcal{J}} = \begin{bmatrix} 0 & 0.5 & 0.4 \end{bmatrix}^T; \tag{37}$$

$$\left[\text{KL}(\text{Programmer}, \text{Designer}|\text{j})\right]_{j\in\mathcal{J}} = \begin{bmatrix} D_{\text{KL}}(0.5\|0.3) & D_{\text{KL}}(0.2\|0.8) & D_{\text{KL}}(0.1\|0.2) \end{bmatrix}^T; \tag{38}$$

$$\left[\text{KL}(\text{Programmer}, \text{All-rounder}|\text{j})\right]_{j\in\mathcal{J}} = \begin{bmatrix} 0 & D_{\text{KL}}(0.2\|0.8) & D_{\text{KL}}(0.1\|0.6) \end{bmatrix}^T. \tag{39}$$

In this case, one can show that $\mathcal{A}(\text{Programmer}) = \{(1-\epsilon, \epsilon, 0) : \epsilon \in (0,1]\}$ and $C(\text{Programmer}) = 0.5/D_{\text{KL}}(0.2\|0.8) \approx 0.6011$. Of these solutions, DEEM picks $\alpha(\text{Programmer}) = (0,1,0)$ as per (26), because this distribution is the quickest in confirming a Programmer while possessing the optimal regret per unit of learning (this α achieves the log posterior targets for $i' =$ Designer and for $i' =$ All-rounder in the same expected time).

On the other hand, Mixed jobs are optimal for All-rounders and further allow All-rounders to be distinguished from both Designers and Programmers (both these types are in Str(All-rounder)). Thus, no regret needs to be incurred while confirming an All-rounder. Recall that Design jobs are also optimal for All-rounders. It is straightforward to verify that $\mathcal{A}(\text{All-rounder}) = \{(0, 1-\epsilon, \epsilon) : \epsilon \in (0,1]\}$, and $C(\text{All-rounder}) = 0$. The choice of $\alpha(\text{All-rounder}) = (0,0,1)$ results in the highest learning rate as per (26). (One can similarly verify that $C(\text{Designer}) = 0$ and $\alpha(\text{Designer}) = (0,1,0)$.)

The distinction in $C(i)$ for $i =$ Programmer and $i' =$ All-rounder is fundamental, and motivates the following definition.

Definition 1. Consider a worker type i. Suppose that there exists another type $i' \in \mathcal{I}\backslash\{i\}$ such that $A(i,j) = A(i',j)$ for all $j \in \mathcal{J}(i)$, and $i' \in \text{Str}(i) \Leftrightarrow \mathcal{J}(i) \nsubseteq \mathcal{J}(i')$, where $\mathcal{J}(i)$ and $\text{Str}(i)$ are as defined in (24). Then we say that the ordered pair (i,i') is a *difficult type pair*.[10]

Note that $C(i) > 0$ if and only if there is some other i' such that (i,i') is a difficult type pair. In this case, there is a nontrivial (asymptotic) trade-off between myopic payoffs and learning, and a regret of $\Omega(\log N)$ is necessary per unit mass of workers. In the example, (Programmer, All-rounder) is a difficult type pair.

5. Main Result

Our main result is the following theorem. In particular, we prove a lower bound on the regret of any policy and show that DEEM (essentially) achieves this lower bound. For the result and discussion, we denote DEEM_N to be the instantiation of DEEM for a given N.

Theorem 1. *Fix (ρ, μ, A) such that (a) no two rows of A are identical and (b) the generalized imbalance condition holds. Let $C \triangleq \sum_{i\in\mathcal{I}} \rho_i C(i)$ for $C(i)$ as defined in (36). If $C > 0$, we have the following:*

1. *(Lower bound) For any sequence of WHO policies $(\pi_N)_{N\in\mathbb{N}}$, indexed by worker lifetime N, that are feasible for (16)–(18), we have*

$$\liminf_{N\to\infty} \frac{N}{\log N}\left(V^* - W^N(\pi_N)\right) \geq C. \tag{40}$$

2. *(Upper bound) There exists $N_0 < \infty$ such that for all $N \geq N_0$, DEEM_N is feasible for (16)–(18) with*

$$\limsup_{N\to\infty} \frac{N}{\log N}\left(V^* - W^N(DEEM_N)\right) \leq C. \tag{41}$$

If instead $C = 0$, then there exists $N_0 < \infty$ such that for all $N \geq N_0$, DEEM_N is feasible for (16)–(18) with

$$\limsup_{N\to\infty} \frac{N}{\log\log N}\left(V^* - W^N(\text{DEEM}_N)\right) \leq K, \tag{42}$$

where $K = K(\rho, \mu, A) \in [0, \infty)$ is some constant.

Recall that $C(i)$ and hence C depend on the primitives of the problem; that is, (ρ, μ, A) and that $C(i) > 0$ if and only if there exists $i' \neq i$ such that (i,i') is a difficult type pair. We immediately deduce that $C > 0$ if and only if there is a difficult pair of worker types (i,i').

Remark 4. The constant C in Theorem 1 is strictly positive if and only if there exists at least one difficult pair of worker types (i,i'), that is, a pair of distinct worker types (i,i') such that $A(i,j) = A(i',j)\ \forall j \in \mathcal{J}(i)$ and $\mathcal{J}(i) \nsubseteq \mathcal{J}(i')$, where

$$\mathcal{J}(\tilde{i}) \triangleq \arg\max_{j\in\mathcal{J}} A(\tilde{i}, j) - p_j^* \qquad \forall \tilde{i} \in \mathcal{I} \tag{43}$$

and p^* are the shadow prices for the capacity constraints (21) in the problem with known worker types (20)–(22). Theorem 1 implies that the smallest achievable regret is $\Theta(\frac{\log N}{N})$ (large) if there is a difficult type pair, whereas one can achieve a regret of $O(\frac{\log\log N}{N})$ (small) if there is no difficult type pair.

In Section 5.2, we show that in a setting where workers are heterogeneous along more than one skill dimension, and some job type allows one to distinguish only a subset of skills, many problem instances

contain such difficult type pairs; that is, there is often a nontrivial (asymptotic) tradeoff between myopic payoffs and learning.

5.1. Proof Sketch

The proof of Theorem 1 can be found in Online Appendix EC.3.1. Here we present a sketch. The critical ingredient in the proof is the following relaxed optimization problem in which there are no capacity constraints, but capacity violations are charged nonnegative prices p^* from the optimization problem (20)–(22) with known worker types.

$$W^N_{p^*} = \max_{x \in \mathcal{X}^N} \sum_{i \in \mathcal{I}} \rho_i \sum_{j \in \mathcal{J}} x(i,j) A(i,j) - \sum_{j \in \mathcal{J}} p^*_j \left[\sum_{i \in \mathcal{I}} \rho_i x(i,j) - \mu_j \right]. \tag{44}$$

5.1.1. Lower Bound on Regret. If $C > 0$ (i.e., if there is at least one difficult pair of worker types; see Section 5), there is a lower bound on the regret relative to V^* under any policy in this problem. This result follows directly from theorem 3.1 in Agrawal et al. (1989):

$$\liminf_{N \to \infty} \frac{N}{\log N} \left(V^* - W^N_{p^*} \right) \geq C.$$

By a standard duality argument, we know that $W^N \leq W^N_{p^*}$, and hence this bound holds for W^N as well, yielding the lower bound on regret in our original problem (16). This is shown in Proposition EC.1 in the Online Appendix EC.3.1.

5.1.2. Upper Bound on Regret. There are two key steps in proving that DEEM_N is feasible for Problem (16)–(18) and that

$$\limsup_{N \to \infty} \frac{N}{\log N} \left(V^* - W^N(\text{DEEM}_N) \right) \leq C.$$

1. First, in Proposition EC.2, we show that DEEM, with an arbitrary exploitation-phase routing matrix y^* supported on $\mathcal{J}(i)$ for each $i \in \mathcal{I}$, achieves near-optimal performance for the vanilla multiarmed bandit problem (44). Formally, if (with some abuse of notation) we let $W^N_{p^*}(\pi)$ denote the value attained by a policy π in problem (44), that is,

$$W^N_{p^*}(\pi) = \sum_{i \in \mathcal{I}} \rho_i \sum_{j \in \mathcal{J}} x_\pi(i,j) A(i,j) - \sum_{j \in \mathcal{J}} p^*_j \left[\sum_{i \in \mathcal{I}} \rho_i x_\pi(i,j) - \mu_j \right],$$

then we can show that[11]

$$\limsup_{N \to \infty} \frac{N}{\log N} \left(V^* - W^N_{p^*}(\text{DEEM}_N) \right) \leq C \qquad \text{if } C > 0$$

and

$$\limsup_{N \to \infty} \frac{N}{\log \log N} \left(V^* - W^N_{p^*}(\text{DEEM}_N) \right) \leq K \quad \text{if } C = 0,$$

for some constant $K = K(\rho, \mu, A) \in [0, \infty)$. Thus, we have

$$\lim_{N \to \infty} \frac{N}{\log N} \left(W^N_{p^*} - W^N_{p^*}(\text{DEEM}_N) \right) = 0,$$

and hence, DEEM is near-optimal in problem (44). The proof of Proposition EC.2 uses two technical results presented as Lemma EC.3 and Lemma EC.4 in the online appendix.

2. Finally, in Proposition EC.3, we prove that if DEEM_N uses an Exploit-phase routing matrix y^* (that depends on N) that satisfies Conditions (27)–(31) (recall that the existence of such a matrix for a large enough N was shown in Proposition 4), then $W^N(\text{DEEM}_N) = W^N_{p^*}(\text{DEEM}_N)$. In conjunction with Proposition EC.2, this yields our upper bound on regret. The proof of Proposition C.5 crucially uses the fact that our specific choice of the exploitation-phase routing matrix y^* ensures that the routing matrix x_{DEEM_N}, as defined in (11), satisfies the following conditions:

(a) (Complementary slackness) $\sum_{i \in \mathcal{I}} \rho_i x_{\text{DEEM}_N}(i,j) - \mu_j = 0$ for all j such that $p^*_j > 0$, and

(b) (Feasibility) $\sum_{i \in \mathcal{I}} \rho_i x_{\text{DEEM}_N}(i,j) - \mu_j \leq 0$ for all other $j \in \mathcal{J}$.

This shows that DEEM_N with this choice of y^* in the exploitation phase is feasible for problem (16)–(18) and the complementarity slackness property implies that $W^N(\text{DEEM}_N) = W^N_{p^*}(\text{DEEM}_N)$.

This result crucially relies on Proposition 4, which shows the existence of the routing matrix y^* with the required properties. The basic idea behind this construction is as follows. At the end of the exploration phase of DEEM, the correct label of the worker is learned with a confidence of at least $(1 - o(1))$. This fact, coupled with the generalized imbalance condition (leading to flexibility in modifying x^*), is sufficient to ensure that an appropriate and feasible choice of $y^* = x^* + o(1)$ will correct the deviations from x^* in terms of the capacity utilizations of job types that were fully used under x^*, that is, with $p^*_j > 0$ (these deviations arise because of the short exploration phase and because of the infrequent cases in which exploitation is based on an incorrect worker label coming out of the exploration phase).

5.2. Difficult Type Pairs Occur Frequently with Multiple Skill Dimensions

In Section 4.1, we saw a simple and natural example with two skill dimensions that included a difficult type pair. In this section, we show that difficult type pairs occur frequently when there is more than one skill dimension.

Again taking Upwork as an example, one of the categories on this platform is Web Development, and there are a number of relevant Skills, for example, HTML development, WordPress, Python, JavaScript, Payment Gateway Integration, and Web Design. A typical web developer has a subset of these skills. For each of these skill *dimensions*, there may be distinct levels, for example, missing/inexpert/expert. The platform may have a prior but learns about a developer's skill along a certain dimension chiefly by observing outcomes. Moreover, job listings are heterogeneous in terms of the relevant skills. For example, a project requiring the skills Web Design and WordPress would reveal some information about these skill dimensions but not whether the developer is an expert in HTML development. In this section, we formalize a special case of our model with multiple skill dimensions and show that difficult type pairs occur frequently (i.e., in many instances) in such a setup.

Suppose that there is a set $\mathcal{S}$ of skill dimensions, and a worker type i is a tuple $i = (i_1, i_2, \ldots, i_{|\mathcal{S}|})$, where each $i_s \in \mathcal{I}_s$ captures the skill level of the worker in dimension s. Here $\mathcal{I}_s$ is a finite set for each s, and the set of worker types $\mathcal{I} \subseteq \mathcal{I}_1 \times \mathcal{I}_2 \times \cdots \times \mathcal{I}_{|\mathcal{S}|}$. For each job type $j \in \mathcal{J}$, there is a nonempty subset of relevant skill dimensions $\mathcal{S}_j \subseteq \mathcal{S}$. We then require the expected payoff $A(i,j)$ to be *consistent* with $\mathcal{S}_j$; that is, there must be a function $f_j : \prod_{s\in\mathcal{S}_j} \mathcal{I}_s \to [0,1]$ such that $A(i,j) = f((i_s)_{s\in\mathcal{S}_j})$.

Without loss of generality, assume that $\cup_{j\in\mathcal{J}} \mathcal{S}_j = \mathcal{S}$, because if this was not true, we could safely ignore the (irrelevant) skills in $\mathcal{S}\backslash(\cup_{j\in\mathcal{J}}\mathcal{S}_j)$, given that they do not matter for any job type (multiple worker types that differ only in irrelevant skills can be collapsed into a single worker type). Then, the following assumption on $(\mathcal{I}, \mathcal{J}, \mathcal{S}, (\mathcal{S}_j)_{j\in\mathcal{J}})$ is very plausible if there are at least two skill dimensions $|\mathcal{S}| \geq 2$.

Assumption 1. *There is a job type $j \in \mathcal{J}$ and a pair of distinct worker types $i \in \mathcal{I}$ and $i' \in \mathcal{I}\backslash\{i\}$ such that* (i) *a strict subset $\mathcal{S}_j \subset \mathcal{S}$ of skills is relevant to job type j and* (ii) *worker types i and i' differ only in skill dimensions in $\mathcal{S}\backslash\mathcal{S}_j$.*

The assumption implies that job type j cannot distinguish between worker types i and i'.

We establish the following result, showing that under Assumption 1, many instances have difficult type pairs. In particular, difficult type pairs *do not* occur only on a knife edge.

Proposition 5. *Fix $\mathcal{I}$, $\mathcal{J}$, $\mathcal{S}$, and $(\mathcal{S}_j)_{j\in\mathcal{J}}$ such that Assumption 1 holds and $|\mathcal{I}| \geq 3$. Also fix capacity constraints $\mu = (\mu_j)_{j\in\mathcal{J}}$ such that*[12] *$\sum_{j\in\mathcal{J}} \mu_j \neq 1$. Consider the set of possible instances*

$$\mathcal{P} = \left\{(\rho, A) \in \Delta_{|\mathcal{I}|} \times [0,1]^{|\mathcal{I}||\mathcal{J}|} : A(\cdot, j) \text{ is consistent with } \mathcal{S}_j \text{ for all } j \in \mathcal{J}\right\}. \tag{45}$$

The distribution over worker types ρ has $|\mathcal{I}| - 1$ degrees of freedom, and for each $j \in \mathcal{J}$, the jth column of the payoff matrix $A(\cdot, j)$ has $|\mathcal{I}_{\mathcal{S}_j}|$ degrees of freedom, where

$$\mathcal{I}_{\mathcal{S}_j} = \left\{(i_s)_{s\in\mathcal{S}_j} : \exists i' \in \mathcal{I} \text{ s.t. } i'_s = i_s \forall s \in \mathcal{S}_j\right\} \tag{46}$$

is the set of distinct worker type classes that arise when worker types are projected onto $\mathcal{S}_j$. Then the subset of instances

$$\mathcal{P}_{\text{diff}} = \{(\rho, A) \in \mathcal{P} : \text{there is a difficult type pair}\} \tag{47}$$

has full dimension $|\mathcal{I}| - 1 + \sum_{j\in\mathcal{J}} |\mathcal{I}_{\mathcal{S}_j}|$, equal to the dimension of $\mathcal{P}$.

Proposition 5 is proved in Online Appendix EC.3.2. Starting with Assumption 1 and the corresponding $i \in \mathcal{I}$, $i' \in \mathcal{I}$, and $j \in \mathcal{J}$, the main idea is to construct an instance such that in any solution in the full information setting, worker type i will be matched exclusively to job type j, whereas type i' will be matched exclusively to jobs types other than j. We then show that this remains true in a neighborhood of the given set of parameters.

6. Practical Considerations and a Heuristic

Although DEEM minimizes the leading-order term of regret as $N \to \infty$ in our continuum model, it is unclear how to obtain from it a policy that performs well in practice. The first concern is that it is defined in the context of the continuum model, whereas, in reality, we have finite arrivals of workers and jobs over time. The second concern is that, although DEEM is expected to perform well when N is large, it may not perform well when N is relatively small, which is the case in many practical settings. In Online Appendix EC.4.1, we present a fairly straightforward translation of DEEM to a discrete setting (with finite arrivals of workers and jobs at each time), which is a close analogue to our continuum setting.

Although this policy is simple and effectively addresses the first concern, it falls short of addressing the second concern: in fact, we find that it performs well *only* when N is large. A main cause of this deficiency is that DEEM-discrete does not learn the right shadow prices for the constraints. In fact, simulations tell us that for practical values of N, if we use the shadow prices p^* for the capacity constraints from the static planning problem (20)–(22) to define the Explore phase for each worker as in DEEM, then, in many instances, a subset of job types are fully (or substantially) used by workers in the Explore phase itself and are thus unavailable for workers in the Exploit phase (see Remark EC.1 in Online Appendix EC.4.1). In particular, this suggests that in many instances, p^* constitutes a poor estimate of the shadow

prices for finite N. Moreover, this is the case even for a large system with many workers and jobs. Thus, an important practical issue is to learn the right shadow prices for the capacity constraints.

This is where we can leverage a key practical feature of most real platforms: jobs typically queue up rather than get assigned instantaneously. In such systems, the *inventory* of jobs is dynamically evolving, and therefore the shadow prices should be responsive to the inventory of the system. *Backpressure* Tassiulas and Ephremides (1990) is the method of choice for learning shadow prices in a dynamic queueing context by estimating shadow prices based on queue lengths. This prompts us to adopt it for the DEEM-inspired practical heuristic we present in this section. Besides learning the right shadow prices, backpressure has the added advantage of seamlessly handling stochasticity in the (finite) arrivals of workers and jobs.

In the remainder of this section, we present our heuristic derived from DEEM called DEEM$^+$, which is (1) practically implementable in a market environment with discrete arrivals and (2) is designed for good small N performance. DEEM$^+$ is defined in Figures 4 and 5. Similar to DEEM, DEEM$^+$ is a decentralized algorithm that acts individually on each worker and hence is defined at the worker level. DEEM$^+$ makes use of dynamic (queue-based) shadow prices p^q as a key input at the worker level. In our numerical simulations in Section 7, we will define an environment where pending jobs can queue up and specify a backpressure-like approach to computing queue-based prices. Notably, besides its use of queue-based shadow prices, DEEM$^+$ incorporates a few key modifications to the Explore and Exploit phases of DEEM that improve performance when N is small. Below, we discuss the important features of DEEM$^+$ in detail.

6.1. Dynamic Queue-Based Shadow Prices

DEEM$^+$ uses dynamic shadow prices computed using job queue length information. Each new worker corresponds to a multiarmed bandit problem that gets *adjusted* under DEEM$^+$ by the instantaneous prices in the market when the worker arrives, to account for the externalities because of the presence of the capacity constraints. These prices remain fixed throughout the lifetime of the worker; this is analogous to having p^* be the fixed prices under DEEM. We refer to these instantaneous queue-based prices as p^q in the definition of DEEM$^+$ in Figures 4 and 5.

Suitably defined prices as functions of job queue lengths can be used as a feedback control mechanism to stabilize the queue lengths and hence stabilize the prices themselves. (Such *backpressure* pricing based on queue lengths is commonly used in designing distributed algorithms for resource allocation, for example, for congestion control in communication networks (Shakkottai et al. 2008, Srikant 2012).) For our simulations in Section 7, we use a proportional-derivative (PD) control-based definition of these prices as an illustrative example (the technical details of the controller design are presented in Online Appendix EC.5.2). The instantaneous shadow price for each job type is the sum of a *proportional* term and a *derivative* term. The proportional term is an affine function of the queue length with a negative coefficient for the queue length. The idea is that if the queue length is small then this indicates that the job type is in high demand and hence the price for this type should be high. The derivative term is proportional to the negative of the recent rate of change of the queue length to counteract rapid changes in queue length and prevent oscillatory behavior.

Our definition of these prices does not depend on the vector of job arrival rates μ. It also obviates the need to explicitly compute y^* in the exploitation phase of DEEM; instead the Exploit phase can be implemented by allocating optimally for each worker given the queue-based prices that were supplied to the worker when the worker arrived. Natural (small) fluctuations in prices that arise in the process of stabilizing queue lengths result in appropriate tie-breaking in allocation (tie-breaking is typically needed; see Fact 1), with randomization occurring *across workers*. The resulting stability of the queue lengths implies that the capacity constraints are satisfied.

6.2. Optimizing the Explore Phase for Finite N

DEEM$^+$ shares the same explore-then-exploit structure as DEEM, but with a few changes to the exploration phase to optimize learning for finite N. These modifications lead to a significant improvement in performance. In Online Appendix EC.5.4, we show that the regret estimate that the Explore phase of DEEM minimizes is a poor estimate of the true regret for small N. By contrast, the regret estimate that the Explore phase of DEEM$^+$ minimizes captures the true regret reasonably well even for small N, hence explaining the improvement in performance produced by these modifications.

The changes to the Explore phase in DEEM$^+$ are discussed in detail below.

1. **Refining learning goals.** Recall that in DEEM, we achieve a $1/\log N$ probability of error of misclassifying i' as i even if $\mathcal{J}(i) \subseteq \mathcal{J}(i')$. In DEEM$^+$, because we use queue-based prices, the counterparts of the sets $\mathcal{J}(i)$ are the sets

$$\mathcal{J}^q(i) \triangleq \arg\max_{j \in \mathcal{J}} A(i,j) - p_j^q \tag{48}$$

for each $i \in \mathcal{I}$. These are sets of job types that are optimal for the different worker types under the

Figure 4. Definition of DEEM$^+$

DEEM$^+$: A practical WHO heuristic for finite N

Input parameters: $\mathcal{I}$, $\mathcal{J}$, A, ρ, N, queue-based prices p^q.
Pre-compute:
- The sets $\mathcal{J}^q(i)$ defined in (48) for each worker type $i \in \mathcal{I}$.
- The set of worker types $\text{Str}^q(i)$ and $\text{Weak}^q(i)$ defined in Definition 2 (Section 6.2) for all $i \in \mathcal{I}$.
- The maximal externality-adjusted payoffs $U^q(i)$ for all $i \in \mathcal{I}$ defined in (49) and the maximal per-job mislabeling regrets $R(i,i')$ for all $i, i' \in \mathcal{I}$ defined in (50)–(51).

1: ▷ Main Routine
2: **procedure** DEEM$^+$ ▷ Acts independently on each worker, over her lifetime, from arrival to departure

3: ▷ Initialization:
4: $\lambda(i) \leftarrow \rho_i$ for all $i \in \mathcal{I}$ ▷ The un-normalized posterior probabilities; initialized to the prior
5: $\text{MAP} \leftarrow \arg\max_{i \in \mathcal{I}} \lambda(i)$ ▷ Initialization of the MAP estimate
6: $\mathit{Label} \leftarrow \emptyset$ ▷ Worker label; initially unassigned, denoted by $\emptyset$
7: $k \leftarrow 0$ ▷ Number of jobs the worker has performed in the system

8: ▷ Explore phase:
9: **while** $\mathit{Label} = \emptyset$ and $k < N$ **do**
10: Assign job type $j_k \sim$ EXPLORE(N, λ, MAP) ▷ At the next job opportunity
11: Observe payoff r_k
12: $\lambda(i) \leftarrow \lambda(i) \times (A(i,j_k)\mathbf{1}_{\{r_k=1\}} + (1 - A(i,j_k))\mathbf{1}_{\{r_k=0\}})$, for all $i \in \mathcal{I}$
13: $\text{MAP} \leftarrow \arg\max_{i \in \mathcal{I}} \lambda(i)$
14:
15: **if** $\min_{i \in \text{Str}^q(\text{MAP})} \frac{\lambda(\text{MAP})}{\lambda(i)R(\text{MAP},i)} \geq N$ **then** ▷ if Confirmation is complete
16: $\mathit{Label} \leftarrow \text{MAP}$ ▷ Worker label assigned. Will cause while loop to exit.
17: **end if**
18: $k \leftarrow k+1$
19: **end while**

20: ▷ Exploit phase:
21: **while** $k < N$ **do**
22: Assign job type $j_k = $ EXPLOIT(λ) ▷ At the next job opportunity
23: $k \leftarrow k+1$
24: **end while**
25: **end procedure**

26: ▷ Functions
27: **function** EXPLORE(N, λ, MAP)
28: **if** $\min_{i \in \text{Str}^q(\text{MAP}) \cup \text{Weak}^q(\text{MAP})} \frac{\lambda(\text{MAP})}{\lambda(i)R(\text{MAP},i)} < \log N$ **then** ▷ if MAP estimate is noisy
29: $\mathit{dist} \leftarrow \alpha_{\text{guess}}(\lambda)$ ▷ Guessing
30: **else** ▷ MAP estimate is somewhat confident
31: $\mathit{dist} \leftarrow \alpha_{\text{conf}}(\text{MAP}, \lambda)$ ▷ Confirmation
32: **end if**
33: **return** dist
34: **end function**

35: **function** EXPLOIT(λ)
36: $j^* = \arg\max_{j \in \mathcal{J}} \sum_{i \in \mathcal{I}} \lambda(i)\left[A(i,j) - p_j^q\right]$ ▷ Greedy
37: **return** j^*
38: **end function**

Note. The prices p^q depend on the queue lengths of the different job types, as discussed in Section 6.1, and formally defined in Online Appendix EC.5.2.

queue-based prices p^q. Because of natural fluctuations in the prices across jobs, these sets are typically singletons and hence for any two types i and i', either $\mathcal{J}^q(i) \cap \mathcal{J}^q(i') = \emptyset$ or $\mathcal{J}^q(i) = \mathcal{J}^q(i')$. In the latter case, it would appear wasteful to achieve a $1/\log N$ probability of error of misclassifying i' as i, except in the case where the optimal confirmation policies for the two types are different, since confirming using a suboptimal job sampling policy leads to an increase in leading-order regret.[13] Thus, in *DEEM*$^+$, for each worker type i, we define the sets of types it needs to be distinguished from as follows (the definition does not assume that the sets $\mathcal{J}^q(i)$ are singletons).

Definition 2. For each $i, i' \in \mathcal{I}$,

(a) $\text{Str}^q(i) \triangleq \{i' : \mathcal{J}^q(i) \setminus \mathcal{J}^q(i') \neq \emptyset\}$; and

(b) $\text{Weak}^q(i) \triangleq \{i' : \mathcal{J}^q(i) \subseteq \mathcal{J}^q(i')$ and $\alpha(\text{i}, \mathcal{I}, \mathcal{J}, \text{A}, \text{p}^q, \text{Str}^q(i)) \neq \alpha(i', \mathcal{I}, \mathcal{J}, A, p^q, \text{Str}^q(i'))\}$, where the function α is defined as in Figure 2.

In the Guessing mode, *DEEM*$^+$ explicitly distinguishes i only from types in $\text{Str}^q(i) \cup \text{Weak}^q(i)$ (see line 28 and the first condition in line 15 in Figure 4). If, at some point in the Explore phase, type i has been confirmed against all types in $\text{Str}^q(i)$, then the algorithm can safely proceed to the Exploit phase even if i has not been weakly distinguished from some $i' \in \text{Weak}^q(i)$; the latter distinction is rendered unnecessary

Figure 5. Definition of DEEM$^+$ (Continued)

39: **function** $\alpha_{\text{guess}}(\lambda)$
40: For each $j \in \mathcal{J}$, define ▷ Thompson sampling

$$\alpha_j = \left(\sum_{i\in\mathcal{I}} \lambda(i) \frac{\mathbb{I}_{j\in\mathcal{J}^q(i)}}{|\mathcal{J}^q(i)|}\right) \Big/ \sum_{i\in\mathcal{I}} \lambda(i)$$

41: **return** $\alpha = (\alpha_j)_{j\in\mathcal{J}}$
42: **end function**

43: **function** $\alpha_{\text{conf}}(\text{MAP}, \lambda)$
44: Define the set

$$\mathcal{A} = \arg\min_{\alpha\in\Delta(\mathcal{J})} \left\{ \left[\sum_{i\in\mathcal{I}} \lambda(i) \sum_{j\in\mathcal{J}} \alpha_j \left(U^q(i) - [A(i,j) - p_j^q]\right) \right] \left[\max_{i'\in \text{Str}^q(\text{MAP})} \frac{\log N + \log R(\text{MAP}, i')}{\sum_{j\in\mathcal{J}} \alpha_j \text{KL}(\text{MAP}, i'|j)} \right] \right\}$$

45: Choose any

$$\alpha \in \arg\min_{\alpha'\in\mathcal{A}} \max_{i\in\text{Str}^q(\text{MAP})} \frac{\log N + \log R(\text{MAP}, i)}{\sum_{j\in\mathcal{J}} \alpha'_j \text{KL}(\text{MAP}, i|j)}$$

46: **return** α
47: **end function**

in this case. This change accounts for the difference in the conditions for transitioning from Explore to Exploit in line 15 of DEEM vs. line 15 of *DEEM*$^+$.

Next, recall that DEEM tries to achieve a probability $1/N$ of misclassifying a worker of type i' as type i for any $i \in \text{Str}(i)$. For small N, however, we can do better: the desired probability of error should depend on the largest regret that type i' could incur by performing a job that is optimal for i but not for i': if this regret is very small, then it isn't worth trying to make this distinction with high precision. For each $i \in \mathcal{I}$, define the maximal externality-adjusted payoffs

$$U^q(i) \triangleq \max_{j\in\mathcal{J}} A(i,j) - p_j^q. \tag{49}$$

Then we define

$$R(i,i') \triangleq \max_{j\in\mathcal{J}^q(i),\, j\notin\mathcal{J}^q(i')} U^q(i') - \left[A(i',j) - p_j^q\right] \quad \forall\, i' \in \text{Str}^q(i). \tag{50}$$

We informally refer to $R(i,i')$ as the maximal per-step regret of mislabeling i' as i. It will further be convenient to (arbitrarily) define[14]

$$R(i,i') \triangleq 1 \qquad \forall i' \in \text{Weak}^q(i). \tag{51}$$

DEEM$^+$ aims for a probability of mislabeling i' as i of $1/(R(i,i')\log N)$ instead of $1/\log N$ at the end of Guessing for all $i' \in \text{Str}^q(i) \cup \text{Weak}^q(i)$, and of $1/(NR(i,i'))$ instead of $1/N$ at the end of Explore for all $i' \in \text{Str}^q(i)$. These changes account for the fact that if $R(i,i')$ is small, then we can tolerate a higher probability of error (see line 15 in Figure 4). This modification is especially important in preventing wasteful learning in cases where price fluctuations (necessary for tie-breaking; see Section 6.1) result in $\mathcal{J}^q(i) \subsetneq \mathcal{J}^q(i')$ even though $\mathcal{J}(i) \subseteq \mathcal{J}(i')$.

2. **Incorporating the posterior during Explore.** The final change we propose is to explicitly incorporate the posterior into the Explore phase. First, recall that in DEEM, Guessing involves the naive approach of exploring uniformly at random. When N is finite, we can reduce regret by instead leveraging the posterior at each round to appropriately allocate learning effort across the different types until we form a good guess of the worker type. In *DEEM*$^+$, we do so by using a price-adjusted version of Thompson sampling (TS), which is a popular Bayesian multiarmed bandit algorithm, in the Guessing mode. Under this algorithm, a worker type is sampled from the posterior distribution and the job type that maximizes the price-adjusted payoff for this sampled worker type is assigned to the worker; if there are multiple such types, one type is chosen uniformly at random. This is defined in the function α_{guess} in Figure 5.

Similarly, the posterior can also be incorporated in minimizing expected regret in the Confirmation mode. This is captured by the function α_{conf} defined in Figure 5. This function returns a sampling distribution that maximizes the product of the expected per-time-step regret relative to the maximal externality-adjusted payoffs (the term in the first parenthesis in the objective on line 44) and the estimated time until the confirmation of the MAP estimate assuming it is correct (the term in the second parenthesis in the objective in line 44). If there are multiple solutions, α_{conf} is chosen to be the one with the smallest time until confirmation (line 45). Observe that as $\lambda(\text{MAP})/(\sum_{i\in\mathcal{I}} \lambda(i)) \to 1$ for the MAP estimate, the objective function in line 44 in the definition of α_{conf} converges to the objective function used while computing $\alpha(\text{MAP})$ in (25), except for the $(\log N + \log R(i,i'))$ factors that simply capture the fact that the learning goals have been adjusted for small N.

6.3. Optimizing the Exploit Phase for Finite N

In *DEEM*$^+$, we tweak the approach in the Exploit phase of DEEM to improve performance: instead of optimizing the price-adjusted payoff for the confirmed worker label, we can maximize the expected payoff with respect to the posterior distribution, thus accounting for the possibility that we may have

confirmed incorrectly. This is reflected in the new definition of the EXPLOIT() function in line 35. We find that this change significantly improves performance.

7. Simulations

In this section, we simulate $DEEM^+$ in a market environment with queue-based shadow prices and compare its performance against other policies. We consider 350 instances with $|\mathcal{I}| = 4$ worker types and $|\mathcal{J}| = 3$ job types. In each instance, the arrival rates of all the worker types are identical, that is, $\rho_i = 0.25$ for all $i \in \mathcal{I}$, whereas the arrival rates of the job types are randomly chosen. The choice of these instances is discussed in detail in Online Appendix EC.5.1. For our simulations, we consider $N \in \{10, 20, 30, 40\}$. Given an instance (ρ, N, μ, A), our simulated marketplace is described as follows.

7.1. Arrival Process

Time is discrete, $t = 1, 2, \cdots, T$, where we assume that $T \in (200, 400, 600, 800)$ for $N \in (10, 20, 30, 40)$, respectively. At the beginning of each time period t, a fixed $M_N(i)$ number of workers of type i arrive and they stay for N periods. We choose $M_N(i) = 600/N$ for the different values of N for each i, so that, irrespective of N, the total number of workers of any type i present in the market at any time t is $M_N(i) \times N = 600$ (making a total of $600 \times |\mathcal{I}| = 600 \times 4 = 2{,}400$ workers present in the market at any time). Relating back to the continuum model, here we implicitly assume that a mass of 0.25 corresponds to 600 workers or jobs. Observe that the fact that $M_N(i)$ is the same for all $i \in \mathcal{I}$ reflects the fact that $\rho_i = 0.25$ for all i.

Job-matching decisions are sequentially made for each of the 2,400 workers in each time step in an arbitrary order. Between successive allocations, a job of type j arrives with probability $\mu_j / \sum_{i \in \mathcal{I}} \rho_i$ for each type j. Thus, the relative proportions of the number of workers of different types that are present in the market at any time, and the expected number of jobs that arrive in the market in any time step, are proportional to ρ_i and μ_j for $i \in \mathcal{I}$ and $j \in \mathcal{J}$, where a mass of 1 corresponds to 1,200 workers or jobs. We assume that each job takes one period to complete.

7.2. Queues and Queue-Based Prices

As described in Section 6.1, we use a PD-control mechanism to set prices with the goal of stabilizing queue lengths; the details are presented in Online Appendix EC.5.2. Queue lengths change at epochs where either a job arrives or it is assigned to a worker. The arriving jobs accumulate in queues for the different types, each with a finite buffer of capacity B, where we choose $B = 50{,}000$. If the buffer capacity is exceeded for some job type, then the remaining jobs are lost. (We use a large buffer in our simulations to keep the price fluctuations small to ensure that differences in the observed performance of DEEM$^+$ and other benchmark policies are not caused by large price fluctuations.)

7.3. Matching Process

In the period when a worker arrives, when a job-matching decision is made for the worker for the very first time, the instantaneous price at that time is assigned to the worker. This price is used by the policy in determining the job allocations for the worker throughout her lifetime.

At every job-matching opportunity, an assignment is generated by the chosen policy based on the assigned price and the history of assignments and outcomes for the worker. If a job of the required type is unavailable, then the worker remains unmatched. For each worker-job match, a random payoff is realized, drawn from the distribution specified by A, and the assignment-payoff tuple is added to the history of the worker.

7.4. Simulation Output

We look at two main outputs: the average performance and the prices assigned to the workers.

1. **Performance ratio (PR):** For the performance measure, we compute the average reward per worker over the last $T/4$ periods of the simulation horizon (so that the mean queue lengths have had a sufficient time to stabilize) and divide it by the optimal per-worker reward in the full information setting (i.e., the optimal value of (20) divided by $\sum_{i \in \mathcal{I}} \rho_i$). We call this quantity the PR of the instance. The average PR over the 350 instances is a proxy for the performance of a fixed policy for a given N.

2. **Average queue-based prices $\bar{p}_j^q$:** We also examine the prices for the different job types seen by the workers that arrived in the last $T/4$ periods. We look at the average price over these periods for each job type j, denoted by $\bar{p}_j^q$, and also the magnitude of typical fluctuations in prices around their average values. The main goal here is to investigate how these prices compare with p^*.

7.5. Benchmark Policies

Along with $DEEM^+$, we implement two other policies. These policies are formally defined in Online Appendix EC.5.3.

1. **PA-TS:** We consider an extension of TS that Agrawal and Goyal (2011) adapted to our setting, in which the payoffs are adjusted by queue-based prices to account for capacity constraints. We refer to this

policy as PA-TS (for price-adjusted Thompson sampling). PA-TS is formally defined in Figure EC.1 in the online appendix. TS is a popular Bayesian multi-armed bandit algorithm and is a natural benchmark for our setting (with the incorporation of queue-based prices, as we propose). TS explores due to sampling from the posterior distribution of the true (worker) type; the exploration is more aggressive in the early stages when the posterior is not sufficiently concentrated. Because of sufficient exploration, it is known to attain the optimal leading-order of regret in several settings (Agrawal and Goyal 2012, Russo and Van Roy 2016) but not necessarily the optimal constant factor for the leading term.

2. **TS-DEEM$^+$:** We also consider a policy that is a hybrid of DEEM$^+$ and TS that also uses queue-based prices for payoff adjustment to account for capacity constraints. We will refer to this policy as TS-DEEM$^+$. TS-DEEM$^+$ is formally defined in Figure EC.2 in the online appendix. This policy operates individually on each worker and has the same two-phase structure of DEEM$^+$, consisting of an Explore phase and an Exploit phase. The Exploit phase is identical to that in DEEM$^+$, in which the algorithm simply assigns a job that maximizes the expected price-adjusted payoff at each opportunity. The learning goals of the Explore phase are also identical to those of DEEM$^+$: in order to label a worker as being of type i, the probability of misclassifying i' as i for any $i' \in \mathrm{Str}^q(i)$ must be less than $1/N$. The only difference is in the way the policy explores. TS-DEEM$^+$ uses Thompson sampling throughout the Explore phase until the learning goals are met, unlike DEEM$^+$, which uses Thompson sampling only in the Guessing mode and a different sampling policy that optimizes the trade-off between learning and payoff maximization in the Confirmation mode.

Comparing the performance of DEEM$^+$ with PA-TS and TS-DEEM$^+$ allows us to investigate the relative importance of the two main features of DEEM$^+$ that it inherits from DEEM (the other important feature being the use of shadow prices for externality adjustment): (1) setting appropriate learning goals and (2) achieving these goals while minimizing regret in the Confirmation mode. As we have seen earlier, both these features are critical to the leading-order regret optimality of DEEM.

We conjecture that both PA-TS and TS-DEEM$^+$ explore sufficiently and are asymptotically optimal; that is, the average regret per unit mass of workers converges to 0 as $N \to \infty$. However, unlike the Confirmation mode in DEEM or DEEM$^+$, these algorithms don't optimize the exploration versus exploitation tradeoff in a way that is tailored to the instance. Hence, we conjecture that they do not achieve the optimal leading term of regret.

7.6. Results

7.6.1. Performance Comparison. The four panels of Figure 6 show the empirical cumulative distribution function (CDF) over the 350 instances of the per-instance PRs (defined in the discussion on simulation output) for the three candidate policies for different values of N. The average of these ratios over the set of instances for each policy, along with the standard error, is presented in the legends. As one can observe, both DEEM$^+$ and TS-DEEM$^+$ significantly outperform PA-TS by a wide margin in all four settings. This suggests the importance of setting appropriate learning goals and of focusing only on payoff maximization after the learning goals have been met. By contrast, PA-TS explores excessively: it keeps exploring beyond the point where such experimentation has any significant benefit for future payoffs. DEEM$^+$ outperforms TS-DEEM$^+$ in all four settings as well. This points to the added benefit from having an optimized Confirmation mode, which is central to the leading-order regret optimality of DEEM.

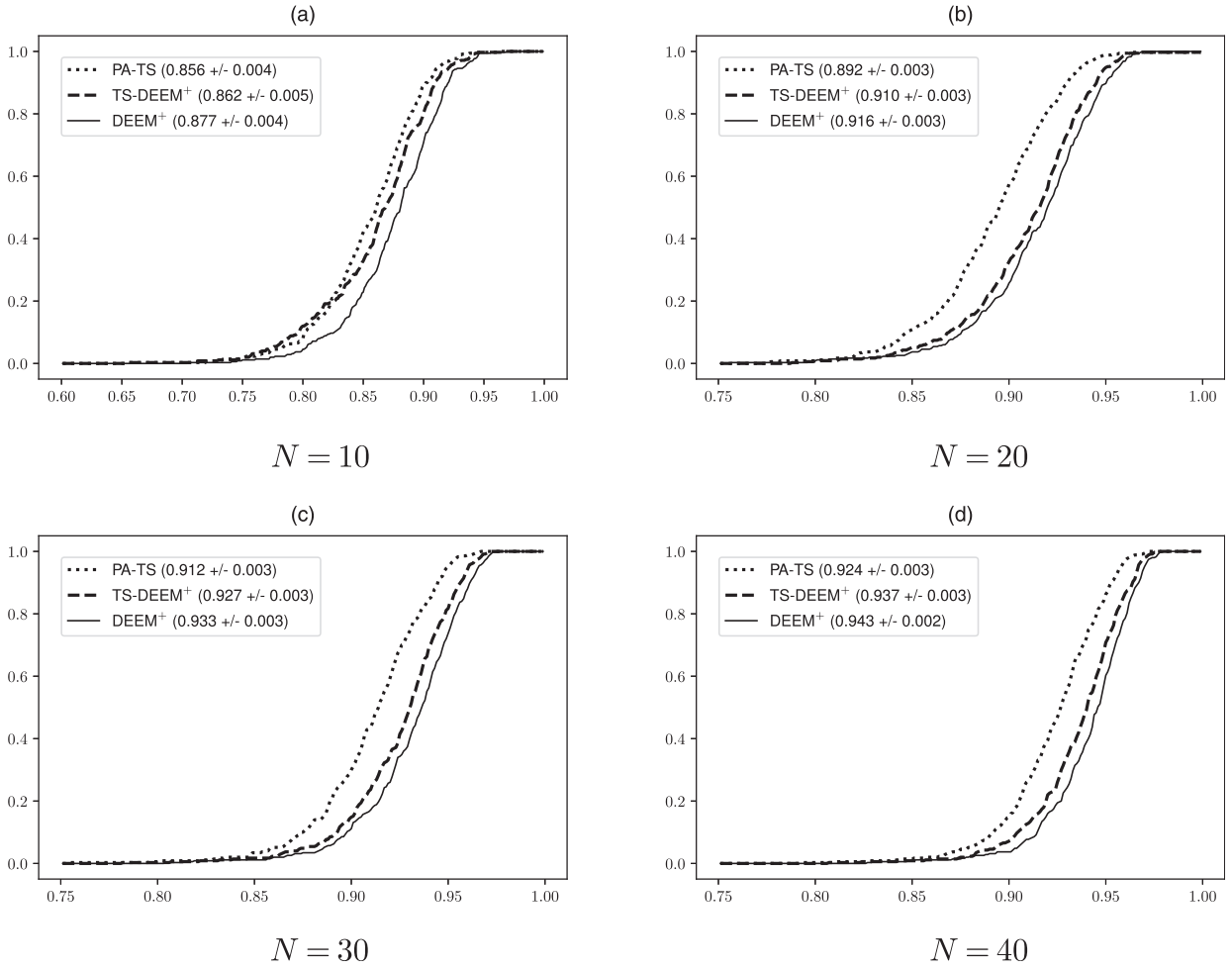
Figure 7 shows the performance improvement of DEEM$^+$ with growing N.

7.6.2. Quality of Our Regret Estimate. In Online Appendix EC.5.4, we numerically investigate the quality of our finite N regret estimate. Because DEEM$^+$ is a modified version of DEEM, this regret estimate is a corresponding modified version of the asymptotic regret estimate $C \log N/N$ in Theorem 1. We find that our regret estimate captures the true regret reasonably well, which suggests that our specific design of the Confirmation mode under DEEM$^+$ is approximately minimizing regret and explains why DEEM$^+$ outperforms the TS-DEEM$^+$ policy.

7.6.3. Prices. Under generalized imbalance (recall Condition (1)), the potential capacity violations because of deviations from the optimal routing during the Explore phase can be corrected in the Exploit phase by designing a routing matrix y^* that perturbs x^* appropriately, but without changing the shadow prices for large enough N (Proposition 4). For finite N, such a correction that leaves the shadow prices unaffected may not be possible if GI is violated or near violated. In these cases, even if p^* is unique, the shadow prices under unknown worker types may a priori be quite different from p^*, and using p^* for externality adjustment in these cases could prove to be detrimental to performance. On the other hand, our queue length–based implementation organically discovers the appropriate shadow prices. It is nevertheless interesting to investigate how different these prices are from p^*.

The first column of Table 1 shows the average and median value of $\|p^* - \bar{p}^q\|_\infty$ under DEEM$^+$ for different values of N. As expected, the average queue-length-based

Figure 6. Empirical Cumulative Distributions of the PRs under the Different Policies for 350 Instances for Different Values of N

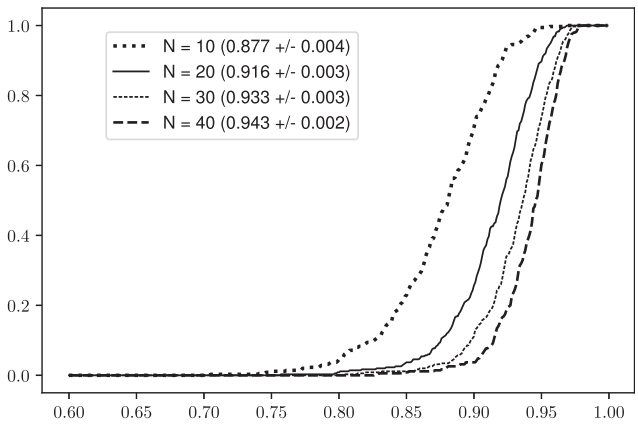


Note. The average PR along with the standard error under a policy is given in the corresponding parentheses.

prices $\bar{p}^q$ are close to p^*, and they get closer for larger N. The difference in these prices is small as compared with the typical range of variation of these prices: the standard deviation of p_j^* for $j \in \mathcal{J}$ (rounded to three decimal places) is $(0.512, 0.530, 0.484)$. The closeness of p^* and $\bar{p}^q$ supports our theoretical device of using p^* to approximately capture the externalities because of capacity constraints in the large N regime. Moreover, we find that the price fluctuations around the mean values $\bar{p}^q$ are relatively small. This can be observed in the second column of Table 1, which shows the median (rounded to three decimal places) across the 350 instances of the standard deviation of the prices seen by workers in the last $T/4$ periods, for different values of N.

Finally, because the fluctuations of p^q around $\bar{p}^q$ are very small and $\bar{p}^q$ itself is typically close to p^*, we expect to frequently encounter difficult type pairs in the price-adjusted multiarmed bandit problems corresponding to the different workers (recall that all 350 instances have a difficult type pair assuming that the shadow prices are p^*). Indeed, we observe that under prices $\bar{p}^q$, each of the 350 instances possesses difficult type pairs.

Figure 7. Empirical Cumulative Distribution of the PRs under DEEM$^+$ for Different Values of N



Note. The average PRs along with standard errors are given in the parentheses.

Table 1. The Queue-Length-Based Prices p^q Are Close to p^*

N	Med. $\|p^* - \bar{p}^q\|_\infty$	(Med. of stdev$(p_j^q))_{j\in\mathcal{J}}$
10	0.119	(0.011, 0.010, 0.011)
20	0.075	(0.006, 0.006, 0.006)
30	0.057	(0.006, 0.006, 0.006)
40	0.053	(0.007, 0.006, 0.007)

Notes. First column: median over 350 instances of difference between the queue-based prices $\bar{p}^q$ under DEEM$^+$ and p^* for different values of N under the $\mathcal{L}^\infty$ norm. Second column: median over 350 instances of the standard deviation of the prices seen by workers in the last $T/4$ periods, under DEEM$^+$, for the different values of N. All values are rounded to three decimal places.

8. Conclusion

This work suggests a novel and practical algorithm for learning while matching, applicable across a range of online matching platforms. Several directions of generalization remain open for future work. First, although we consider a finite-type model, a richer model of types would admit a wider range of applications; for example, workers and jobs may be characterized by features in a vector-valued space, with compatibility determined by the inner product between feature vectors. Second, although our model includes only one-sided uncertainty, in general, a market will include two-sided uncertainty (i.e., both supply and demand will exhibit type uncertainty). Finally, although our results extend immediately to settings where all jobs have a fixed duration larger than one time period, it would be interesting to consider settings where job durations are random and possibly depend on the job type and/or the worker type. We expect that a similar approach using shadow prices for capacity constraints, first to set learning objectives and then to achieve them while incurring minimum regret, should be applicable even in these more general settings.

We conclude by noting that our model ignores strategic behavior by participants. A simple extension might be to presume that workers are less likely to return after several bad experiences; this would dramatically alter the model, forcing the policy to become more conservative. The modeling and analysis of these and other strategic behaviors remain important challenges.

Acknowledgments

An extended abstract for an early version of the paper appeared in ACM EC 2017. We thank the anonymous referees and associate editor, Dan Russo, Kuang Xu, Bert Zwart, and numerous seminar participants for their invaluable suggestions.

Endnotes

[1] We in fact consider a regime where the job capacities are held constant and we reduce the worker arrival rates as their lifetime increases. However, it is straightforward to see that we can equivalently keep the worker arrival rates fixed and increase the job capacities as the worker lifetimes increase, without impacting any of our results or insights.

[2] Our analysis and results generalize to random (exogenous) worker lifetimes that are i.i.d. across workers of different types, with mean N and any distribution such that the lifetime exceeds $N/polylog(N)$ with high probability. In particular, the definition of our DEEM policy in Figures 1 and 2 remains unchanged except that the condition $k < N$ in the while commands in lines 9 and 21 of Figure 1 is replaced by the condition that the worker has not yet left the system. Theorem 1 remains unchanged as well. The platform only needs to know the mean lifetime N beforehand to implement DEEM; it suffices for the platform to find out about the departure (as per the realized lifetime) of a worker only when it occurs.

[3] Here and throughout, independence of a continuum of random variables means that any finite subcollection is mutually independent.

[4] The set (ρ, μ) for which the condition holds is open and dense in $\text{Relint}(\Delta_{|\mathcal{I}|}) \times \mathbb{R}_{++}^{|\mathcal{J}|}$, where $\Delta_{|\mathcal{I}|}$ is the probability simplex in $|\mathcal{I}|$ dimensions, *Relint*$(\cdot)$ denotes the relative interior, and $\mathbb{R}_{++}$ are the strictly positive real numbers.

[5] In other words, it is possible that a WHO policy may lead to a mass of workers matched to jobs of type j in some period that exceeds μ_j. Formally, to ensure that WHO policies satisfy the capacity-feasibility requirement defined in Section 3.2, we define that, if the implied assignment under a WHO policy violates a capacity constraint, the WHO policy does not assign any jobs to workers in that period or in any subsequent period. This definition is merely for concreteness; it does not affect our results because we will ensure that capacity violations occur with probability 0 (Lemma 1).

[6] Formally, the shadow prices p^* are the values of the corresponding dual variables at an optimum of dual linear program (EC.12)–(EC.13) stated in the online appendix.

[7] We ensure that our practical policies derived from DEEM (DEEM-discrete discussed in Section EC.4.1, and DEEM$^+$, specified in Figure 4) are well defined for any value of N.

[8] These shadow prices are consistent with the fact that all Design jobs are assigned (but this is not the case for other job types), and marginal All-rounders could generate $0.8 - 0.6 = 0.2$ more in payoffs per unit if there were more Design jobs available.

[9] In Online Appendix EC.2.1, we show that this choice is well defined, despite the fact that $\mathcal{A}(i)$ could be an open set.

[10] A similar definition also appears in Agrawal et al. (1989); the modification here is that the sets $\mathcal{J}(i)$ are defined with respect to externality-adjusted payoffs to account for capacity constraints.

[11] Agrawal et al. (1989) prove a similar performance guarantee for a slightly different policy.

[12] This technical requirement says that the total arrival rate of jobs should not be exactly equal to the total mass of workers present. We also assume that $|\mathcal{I}| \geq 3$. These assumptions help us get unique shadow prices p^* in our construction, simplifying the analysis, although we expect this proposition can be stated and proved without these assumptions.

[13] In DEEM, we made this distinction even when the confirmation policies for the two types are the same since leading-order regret is unaffected and because it allowed us to leverage the solution to the problem with known worker types to obtain a policy with provable guarantees that satisfies capacity constraints (see the construction of y^* in the proof of Proposition 4 in Online Appendix EC.3.1). The resulting key property is that shadow prices remain close to those under known worker types, and, happily, we observe that this latter property holds under DEEM$^+$ in our numerical experiments (Table 1).

[14] The definition in (51) plays a role only in line 28 of DEEM$^+$ (Figure 4), which contains the criterion for being in Guessing mode.

References

Agrawal R, Teneketzis D, Anantharam V (1989) Asymptotically efficient adaptive allocation schemes for controlled i.i.d. processes: Finite parameter space. *IEEE Trans. Automatic Control* 34(3): 258–267.

Agrawal S, Devanur NR (2014) Bandits with concave rewards and convex knapsacks. *Proc. 15th ACM Conf. Econom. Comput.* (ACM, New York), 989–1006.

Agrawal S, Devanur NR (2019) Bandits with global convex constraints and objective. *Oper. Res.* 67(5):1486–1502.

Agrawal S, Devanur NR, Li L (2016) An efficient algorithm for contextual bandits with knapsacks, and an extension to concave objectives. *J. Machine Learn. Res.* 49:4–18.

Agrawal S, Goyal N (2012) Analysis of thompson sampling for the multi-armed bandit problem. Mannor S, Srebro N, Williamson RC, eds. *Proc. 25th Annual Conf. Learn. Theory (Edinburgh, Scotland)*, vol. 23, 39.1–39.26.

Akbarpour M, Li S, Oveis Gharan S (2014) Dynamic matching market design. Preprint, submitted February 15, https://arxiv.org/abs/1402.3643.

Anderson R, Ashlagi I, Gamarnik D, Kanoria Y (2015) A dynamic model of barter exchange. *Proc. 26th Annual ACM-SIAM Sympos. Discrete Algorithms* (SIAM, Philadelphia), 1925–1933.

Ata B, Kumar S (2005) Heavy traffic analysis of open processing networks with complete resource pooling: Asymptotic optimality of discrete review policies. *Ann. Appl. Probabilities* 15(1A):331–391.

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

Babaioff M, Dughmi S, Kleinberg R, Slivkins A (2015) Dynamic pricing with limited supply. *ACM Trans. Econom. Comput.* 3(1):4.

Baccara M, Lee S, Yariv L (2020) Optimal dynamic matching. *Theoret. Econom.* 15(3):1221–1278.

Badanidiyuru A, Kleinberg R, Slivkins A (2013) Bandits with knapsacks. *Proc. 2013 IEEE 54th Annual Sympos.* (IEEE, New York), 207–216.

Badanidiyuru A, Langford J, Slivkins A (2014) Resourceful contextual bandits. Balcan MF, Feldman V, Szepesvari C, eds. *Proc. 27th Conf. Learn. Theory (Barcelona, Spain)*, vol. 35, 1109–1134.

Besbes O, Zeevi A (2009) Dynamic pricing without knowing the demand function: Risk bounds and near-optimal algorithms. *Oper. Res.* 57(6):1407–1420.

Besbes O, Zeevi A (2012) Blind network revenue management. *Oper. Res.* 60(6):1537–1550.

Blischke W (1964) Estimating the parameters of mixtures of binomial distributions. *J. Amer. Statist. Assoc.* 59(306):510–528.

Bubeck S, Cesa-Bianchi N (2012) Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Machine Learn.* 5(1):1–122.

Chakrabarti D, Kumar R, Radlinski F, Upfal E (2009) Mortal multi-armed bandits. Koller D, Schuurmans D, Bengio Y, Bottou L, eds. *Adv. Neural Inform. Processing Systems*, vol. 21, 273–280.

Chen W, Wang Y, Yuan Y (2013) Combinatorial multi-armed bandit: General framework and applications. *Internat. Conf. Machine Learn. (Atlanta)*, vol. 28, 151–159.

Damiano E, Lam R (2005) Stability in dynamic matching markets. *Games Econom. Behav.* 52(1):34–53.

Das S, Kamenica E (2005) Two-sided bandits and the dating market. *Proc. 19th Internat. Joint Conf. Artificial Intelligence* (Morgan Kaufmann Publishers Inc., Burlington, MA), 947–952.

den Boer AV (2015) Dynamic pricing and learning: historical origins, current research, and new directions. *Survey Oper. Res. Management Sci.* 20(1):1–18.

den Boer AV, Zwart B (2014) Simultaneously learning and optimizing using controlled variance pricing. *Management Sci.* 60(3):770–783.

den Boer AV, Zwart B (2015) Dynamic pricing and learning with finite inventories. *Oper. Res.* 63(4):965–978.

Feldman J, O'Donnell R, Servedio RA (2008) Learning mixtures of product distributions over discrete domains. *SIAM J. Comput.* 37(5):1536–1564.

Ferreira KJ, Simchi-Levi D, Wang H (2018) Online network revenue management using Thompson sampling. *Oper. Res.* 66(6): 1586–1602.

Fershtman D, Pavan A (2017) Matching auctions. Technical Report 0144, Department of Economics, Center for the Study of Industrial Organization, Northwestern University, Evanston, IL.

Gai Y, Krishnamachari B, Jain R (2010) Learning multiuser channel allocations in cognitive radio networks: A combinatorial multi-armed bandit formulation. *IEEE Sympos. New Frontiers Dynamic Spectrum* (IEEE, New York), 1–9.

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

Gittins J, Glazebrook K, Weber R (2011) *Multi-Armed Bandit Allocation Indices* (John Wiley & Sons, New York).

Hsu WK, Xu J, Lin X, Bell MR (2018) Integrating online learning and adaptive control in queueing systems with uncertain payoffs. *2018 Inform. Theory Appl. Workshop (ITA)* (IEEE, New York), 1–9.

Hu M, Zhou Y (2018) Dynamic type matching. Preprint, submitted November 16, https://arxiv.org/abs/1811.07048.

Kadam SV, Kotowski MH (2015) *Multi-period matching*. Technical report, John F. Kennedy School of Government, Harvard University, Cambridge, MA.

Kurino M (2020) Credibility, efficiency, and stability: A theory of dynamic matching markets. *Japanese Econom. Rev.* 71(1): 135–165.

Kveton B, Wen Z, Ashkan A, Szepesvari C (2015) Tight regret bounds for stochastic combinatorial semi-bandits. Lebanon G, Vishwanathan SVN, eds. *Proc. 18th Conf. Artificial Intelligence Statist. (San Diego, CA)*, vol. 38, 535–543.

Lai TL, Robbins H (1985) Asymptotically efficient adaptive allocation rules. *Adv. Appl. Math.* 6(1):4–22.

Lattimore T, Szepesvári C (2020) *Bandit Algorithms* (Cambridge University Press, Cambridge, UK).

Liu K, Zhao Q (2012) Adaptive shortest-path routing under unknown and stochastically varying link states. *Proc. 10th Internat. Sympos. Model. Optim. Mobile Ad Hoc Wireless Networks (WiOpt)* (IEEE, New York), 232–237.

Massoulié L, Xu K (2018) On the capacity of information processing systems. *Oper. Res.* 66(2):568–586.

Mehta A (2012) Online matching and ad allocation. *Theoret. Comput. Sci.* 8(4):265–368.

Modaresi S, Saure D, Vielma JP (2020) Learning in combinatorial optimization: What and how to explore. *Oper. Res.* 68(5):1585–1604.

Özkan E, Ward AR (2020) Dynamic matching for real-time ride sharing. *Stochastic Systems* 10(1):29–70.

Ross S (2008) *Stochastic Processes*, 2nd ed. (Wiley, New York).

Russo D, Van Roy B (2016) An information-theoretic analysis of thompson sampling. *J. Machine Learn. Res.* 17(1):2442–2471.

Sauré D, Zeevi A (2013) Optimal dynamic assortment planning with demand learning. *Manufacturing Service Oper. Management* 15(3): 387–404.

Shakkottai S, Srikant R, et al (2008) Network optimization and control. *Foundations Trends Networking* 2(3):271–379.

Shapley LS, Shubik M (1971) The assignment game I: The core. *Internat. J. Game Theory* 1(1):111–130.

Srikant R (2012) *The Mathematics of Internet Congestion Control* (Springer Science & Business Media, New York).

Sun Y (2006) The exact law of large numbers via Fubini extension and characterization of insurable risks. *J. Econom. Theory* 126(1): 31–69.

Tassiulas L, Ephremides A (1990) Stability properties of constrained queueing systems and scheduling policies for maximum

throughput in multihop radio networks. *29th IEEE Conf. Decision Control* (IEEE, New York), 2130–2132.

Wang Z, Deng S, Ye Y (2014) Close the gaps: A learning-while-doing algorithm for single-product revenue management problems. *Oper. Res.* 62(2):318–331.

Ramesh Johari is a professor of management science and engineering and (by courtesy) computer science and electrical engineering at Stanford University. His research broadly focuses on the design and operation of online platforms and marketplaces, as well as experimentation and online learning in these platforms.

Vijay Kamble is an assistant professor of information and decision sciences in College of Business Administration, University of Illinois at Chicago. His research is in the areas of market design, applied probability, and online learning, with applications to operationalizing modern economic and policy paradigms.

Yash Kanoria is the Sidney Taurel Associate Professor of Business in the Decision, Risk, and Operations division at Columbia Business School, working on the design and optimization of marketplaces, especially matching markets. He received a U.S. National Science Foundation CAREER Award in 2017.