arXiv:1603.04549v2 [cs.LG] 18 Jun 2017

Matching while Learning

Ramesh Johari* Vijay Kamble† Yash Kanoria‡

June 20, 2017

Abstract

We consider the problem faced by a service platform that needs to match supply with demand, but also to learn attributes of new arrivals in order to match them better in the future. We introduce a benchmark model with heterogeneous workers and 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 payoff from a match depends on the pair of types and the goal is to maximize the steady-state rate of accumulation of payoff.

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 trade-off for each worker between myopically maximizing payoffs (*exploitation*) and learning the type of the worker (*exploration*). This creates a multitude of multi-armed 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."

Keywords: matching, learning, two-sided platform, multi-armed bandit, capacity constraints.

1 Introduction

This paper considers a central operational challenge faced by platforms that serve as matchmakers between supply and demand. Such platforms face a fundamental *exploration-exploitation* trade-off: 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"). In this paper, we develop a structurally simple and nearly optimal approach to resolving this trade-off.

In the model we consider, there are two groups of participants: *workers* and *jobs*. The terminology is inspired by online labor markets (e.g., Upwork for remote work, Handy for house-cleaning, Thumbtack and Taskrabbit for local tasks, etc.); however, our model can be viewed as a stylized abstraction of many other matching platforms as well. Time is discrete, and new workers and jobs arrive at the beginning of every time period. Workers depart after performing

*Stanford University (rjohari@stanford.edu)
†Stanford University (vjkamble@stanford.edu)
‡Columbia Business School (ykanoria@gsb.columbia.edu)

a specified number of jobs. 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.

As our emphasis is on the interaction between matching and learning, our model has several features that focus our analysis in this paper. 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 goal of maximizing the steady-state rate of payoff generation.[1]

We now describe the learning challenge faced by the platform. In most platforms, more is known about one side of the platform than the other; accordingly, we assume job types are known, while the type of a new worker is unknown. The platform learns about workers' types through the payoffs obtained when they are matched to jobs. However, because the supply of jobs is limited, using jobs to learn can reduce immediate payoffs, as well as 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 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.*

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*, DEEM has a natural decentralization property: it determines the choice of job type for a worker based only on that worker's history. This decentralization is arguably essential in large-scale online platforms, where matching is typically carried out on an individual basis, rather than centrally. In order to accomplish this decentralization, 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 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

[1]This is a reasonable proxy for the goal of a platform that, say, takes a fraction of the total surplus generated through matches. More generally, we believe that this is a benchmark problem whose solution informs algorithmic design for settings with other related objectives, such as revenue maximization.

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 in the exploration phase, and to then design DEEM to meet these learning goals while maximizing payoff generation.

Third, DEEM involves a carefully constructed exploitation phase 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 suitable tie-breaking 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 tie-breaking via simple but dynamically updated shadow prices.

Our main result (Theorem 5.1) shows that DEEM achieves essentially optimal regret as the number of jobs N performed by each worker during their 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. DEEM achieves this level of regret to leading order when $C > 0$, while it achieves a regret of $O(\log \log N/N)$ if $C = 0$.

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 example above, where each worker can be either an expert or novice programmer, as well as an expert or novice graphic designer. Suppose 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.[2] But if we are learning worker types, then expert graphic designers must be matched to approximately $O(\log N)$ programming jobs to learn whether they are novice or expert programmers – and in turn, whether they should be matched to graphic design or programming jobs, respectively. Thus $\mathrm{O}(\log N/N)$ 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, and an implementation and simulation that demonstrates a natural way of translating our work into practice. 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 above 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 6 we present a sketch of the proof of the main result. In Section 7, discuss practical implementation of DEEM and present the heuristic DEEM$^+$. In Section 8, we use simulations to compare the performance of DEEM and DEEM$^+$ with well-known multi-armed bandit algorithms. We conclude in Section 9. All proofs are in the appendices.

[2]This would be the case, e.g., if programming jobs are both in high demand and more valuable (conditional on successful completion) than graphic design jobs.

2 Related literature

A foundational model for investigating the exploration-exploitation tradeoff is the stochastic multi-armed bandit (MAB) problem [9, 18, 23]. The goal 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) [4, 10, 28]. The closest work in this literature to our paper is by Agrawal et al. [1]; in their model, they assume 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.

Our work is also related to recent literature on MAB problems with capacity constraints; we refer to these broadly as *bandits with knapsacks*. The formulation is the same as the classical MAB problem, with the modification that every pull of an arm depletes a vector of resources which are limited in supply [14]. The formulation subsumes several related problems in revenue management under demand uncertainty [11, 16, 17, 34, 37], and budgeted dynamic procurement [13, 36]; there have been a variety of extensions [2, 15], with recently a significant generalization of the problem to a contextual bandit setting, with concave rewards and convex constraints [3, 5]. There is considerable difference between our model and bandits with knapsacks. Bandits with knapsacks 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; these MAB problems are coupled together by the capacity constraints on arriving jobs. Indeed, as noted in the introduction, a significant structural point for us is to solve these problems in a decentralized manner, to ease their implementation in large-scale online platforms.

We conclude by discussing some other directions of work that are related to this paper. There are a number of recent pieces of work that consider efficient matching in dynamic two-sided matching markets [6, 7, 12, 20, 21, 24, 25, 27]; a related class of dynamic resource allocation problems, online bipartite matching, is also well studied in the computer science community (see [32] for a survey). Similar to the current paper, Fershtman and Pavan [22] 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. Finally, a recent work [31] studies a pure learning problem in a setting similar to ours with capacity constraints on each type of server/expert; while 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.

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 givea formal specification of the matching process we study. We conclude by precisely defining the optimization problem of interest that we solve in this paper.

3.1 Preliminaries

Workers and jobs. For convenience we adopt the terminology of *workers* and *jobs* to describe the two sides of the market. We assume a fixed set of job types $\mathcal{J}$, and a fixed set of worker types $\mathcal{I}$.

A key point is that the model we consider is a continuum model, and so the evolution of the system will be described by *masses* of workers and jobs.[3] In particular, at each time step, a mass $\hat{\rho}(i) > 0$ of workers of type i and a mass $\mu(j) > 0$ of jobs of type j arrive. In what follows, we model the scenario where type uncertainty exists only for workers; i.e., the platform will know the types of arriving jobs exactly, but will need to learn the types of arriving workers. We also assume for now that the arrival rates of jobs and workers are known to the platform; later in Section 7, we discuss how the platform might account for the possibility that these parameters are unknown.

Matching and the payoff matrix. If a mass of workers of type i is matched to a mass of jobs of type j, we assume that a fraction $A(i, j)$ of this mass of matches generates a reward of 1 (per unit mass), while a fraction $1 - A(i, j)$ generates a reward of zero (per unit mass). This formal specification is meant to capture a large-system model in a setting where matches between type i workers and type j jobs generate a Bernoulli$(A(i, j))$ payoff. We do not concern ourselves with the division of payoffs between workers and employers in this paper; instead we assume that the platform's goal is to maximize the total rate of payoff generation.[4] We call the matrix A the *payoff matrix*; throughout, we assume that no two rows of A are identical.[5]

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 understand compatibility between different worker and job types; however, for any *given* worker newly arriving to the platform, the platform does not know the worker's type. Thus, from the perspective of the platform, there will be uncertainty in payoffs in each period because although the platform knows that a given mass of workers of type i exist in the platform, the identity of the workers of type i is not known.

We define an "empty" job type κ, such that all worker types matched to κ generate zero reward, i.e., $A(i, \kappa) = 0$ 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, i.e., $\mu(\kappa) = \infty$.

Worker lifetimes. We imagine that each arriving worker lives in the system for N time steps,[6] and has the opportunity to be matched to a job in each time step (so each job takes one unit of time to complete). We assume the platform knows N.

Note that we have $\rho(i) = \hat{\rho}(i)N$ as the total mass of workers of type i in the system at each time step. For our theoretical analysis, we later consider a scaling regime where $N \to \infty$, and $\hat{\rho}(i) \to 0$, while $\rho(i)$ remains fixed. In this regime, worker lifetimes grow to infinity, and arrival rates scale down, but the total mass of workers of each type available in each time period remains fixed.

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

[3]Formally, this can be seen as a continuum scaling of a discrete system; see, e.g., [19, 29, 30]

[4]This would be the case, e.g., in a platform where the operator takes a fixed percentage of the total payoff generated from a match.

[5]This mild requirement simply ensures that it is possible, in principle, to distinguish between each pair of worker types.

[6]Our analysis and results generalize to random worker lifetimes that are i.i.d. across workers of different types, with mean N and any distribution such that the lifetime exceeds $N/\text{polylog}(n)$ with high probability.

types $(\mathcal{I}', \mathcal{J}')$, such that the total worker arrival rate of $\mathcal{I}'$ exactly matches the total job arrival rate of $\mathcal{J}'$. Formally,

$$\sum_{i \in \mathcal{I}'} \rho(i) \neq \sum_{j \in \mathcal{J}'} \mu(j) \quad \forall \mathcal{I}' \subseteq \mathcal{I}, \mathcal{J}' \subseteq \mathcal{J}, \mathcal{I}' \neq \phi\,.$$

The generalized imbalance condition holds generically.[7] Note that this condition does not depend on the matrix A.

Worker history. To define the state of the system and the resulting matching dynamics, we need the notion of a worker history. A *worker history* is a tuple $H_k = ((j_1, x_1), \dots, (j_k, x_k))$, where j_m is the job type this worker was matched to at her m-th time step in the system, for $1 \leq m \leq k$; and $x_m \in \{0, 1\}$ is the corresponding reward obtained. Note that since workers live for N jobs, the histories will have $k = 0, \dots, N-1$. We let ϕ denote the empty history (for $k = 0$).

3.2 System dynamics

Our goal is to model the following process. The operator observes, at any point in time, the distribution of histories of workers in the platform, and also knows the job arrival rate μ. The matching policy of the platform amounts to determining what mass of workers of each type of history will be matched to which type of jobs. Ultimately, for this process to generate high payoffs over time, the platform must choose jobs to learn worker types in order to optimize payoffs.

With this intuition in mind, we now give a formal specification of our system dynamics.

System profile. A *system profile* ν is a joint measure over worker histories and worker types; i.e., $\nu(H_k, i)$ is the mass of workers in the system with history H_k and type i. The evolution of the system is a discrete-time dynamical system $\nu_0, \nu_1, \nu_2, \dots$, where each ν_t is a system profile.[8]

Matching policy. To describe the dynamics we assume that the platform uses a *matching policy* to match the entire mass of workers to jobs in each time step (we think of unmatched workers as being matched to the empty job type κ). We assume that any mass of jobs left unmatched in a given period disappears at the end of that period (our results do not depend on this assumption).

Suppose that the system starts at time $t = 0$ with no workers in the system before this time.[9] A *matching policy* $\pi_0, \pi_1, \dots$ for the system specifies, at each time t, given a system profile ν_t, the mass of workers with each history that is matched to jobs of each type. In particular, let $\pi_t(H_k, j|\nu_t)$ denote the fraction of workers with history H_k matched to jobs of type j at time t, given a system profile ν_t. (Thus $\sum_j \pi_t(H_k, j|\nu_t) = 1$ for all t, H_k, and ν_t.) Note that the matching policy acts on each worker's history, not on the true type of each worker: this is because the platform is assumed to not know worker types, except as learned through the history itself.

Dynamics. These features completely determine the evolution of the system profile $\{\nu_t\}$. Observe that $\nu_t(H_k, i)\pi_t(H_k, j|\nu_t)$ is the total mass of workers of type i with history H_k who are

[7]The set $(\hat{\rho}, \mu)$ for which the condition holds is open and dense in $\mathbb{R}_{++}^{|\mathcal{I}|+|\mathcal{J}|}$, where $\mathbb{R}_{++}$ are the strictly positive real numbers.

[8]The platform cannot directly observe the system profile, but can infer it. The platform observes the mass of workers with each possible history $\left(\sum_{i \in \mathcal{I}} \nu(H_k, i)\right)_{H_k}$. It can then infer $\nu(H_k, i)$'s individually by using knowledge of arrival rates $\hat{\rho}(i)$'s, and the A matrix (which allows it to calculate the likelihood of seeing the sequence of outcomes in H_k under the worker type i), together with Bayes' rule.

[9]In what follows we ultimately consider a steady-state analysis of the dynamical system, and initial conditions will be irrelevant as long as the initial mass of workers is bounded.

matched to jobs of type j at time t, given policy π_t and system profile ν_t. For all i, j, and t, we have

$$\nu_{t+1}(\phi, i) = \hat{\rho}(i); \tag{1}$$
$$\nu_{t+1}((H_k, (j,1)), i) = \nu_t(H_k, i)\pi_t(H_k, j|\nu_t)A(i,j), \qquad k = 0, \dots, N-2; \tag{2}$$
$$\nu_{t+1}((H_k, (j,0)), i) = \nu_t(H_k, i)\pi_t(H_k, j|\nu_t)(1 - A(i,j)), \; k = 0, \dots, N-2. \tag{3}$$

Decentralization through worker-history-only (WHO) policies. Note that, in general, policies may be time-varying, and may have complex dependence on the system profile ν_t. We consider a much simpler class of policies that we call *worker-history-only (WHO) policies.* These are policies where there exists a π such that

$$\pi_t(H_k, j|\nu_t) = \pi(H_k, j).$$

In other words, in a WHO policy, the fraction of workers with history H_k who are matched to jobs of type j does not depend on either time or on the full system profile. Thus WHO policies are *decentralized.*

An obvious concern at this point is that a policy cannot allocate more jobs of type j than there are. We formalize this capacity constraint in (8) below: in particular, a WHO policy does not exceed the capacity of any job type in any period if and only if it satisfies (8).

Let Π^N denote the class of WHO policies, for a given N. In Section D.1 in Appendix D, we establish that it suffices to restrict attention to policies in Π^N that satisfy (8).

Remark 1. *For any feasible policy, there exists a WHO policy satisfying capacity constraints that achieves a payoff accumulation rate arbitrarily close to that of the former policy. In particular, WHO policies satisfying capacity constraints suffice to achieve the highest possible payoff accumulation rate.*

Steady state of a WHO policy π. First, suppose that there are no capacity constraints, and consider the system dynamics (1)–(3), assuming the system initially starts empty. The dynamics (1)–(3) yields a unique steady state that can be inductively computed for $k = 0, 1, \dots$:

$$\nu_\pi(\phi, i) = \hat{\rho}(i); \tag{4}$$
$$\nu_\pi((H_k, (j,1)), i) = \nu_\pi(H_k, i)\pi(H_k, j)A(i,j), \; k = 0, \dots, N-2; \tag{5}$$
$$\nu_\pi((H_k, (j,0)), i) = \nu_\pi(H_k, i)\pi(H_k, j)(1 - A(i,j)), \; k = 0, \dots, N-2. \tag{6}$$

We refer to the measure ν_π as the *steady state* induced by the policy π.

Routing matrix of a WHO policy π. If the system is in steady state, then at any time period, π induces a steady-state fraction $x_\pi(i,j)$ of the mass of workers of type i that are assigned to type j jobs. We have $x_\pi(i,j) = \frac{\sum_H \nu_\pi(H,i)\pi(H,j)}{\sum_H \nu_\pi(H,i)} = \frac{\sum_H \nu_\pi(H,i)\pi(H,j)}{\rho(i)}$. We call $\{x_\pi(i,j)\}_{\mathcal{I}\times\mathcal{J}}$ the *routing matrix* achieved by the policy π. This is a (row) stochastic matrix; i.e., each row sums to 1. Observe that the mass of demand for jobs of type j from workers of type i in any time period is $\rho(i)x_\pi(i,j)$, and the total mass of demand for jobs of type j in any time period is $\sum_{i\in\mathcal{I}} \rho(i)x_\pi(i,j)$.

Let $\mathcal{X}^N = \left\{ x_\pi : \pi \in \Pi^N \right\} \subseteq [0,1]^{|\mathcal{I}|\times|\mathcal{J}|}$ be the set of routing matrices achievable (when each worker does N jobs) by WHO policies. (Again, we note that capacity constraints are ignored in the definition of $\mathcal{X}^N$.) In Appendix D, we show that $\mathcal{X}^N$ is a convex polytope (see Proposition D.4).

3.3 The optimization problem

Our paper focuses on maximization of the *steady-state rate of payoff accumulation*, subject to the capacity constraints. This leads to the following optimization problem:

$$\begin{aligned} \text{maximize} \quad & W^N(\pi) \triangleq \sum_{i \in \mathcal{I}} \rho(i) \sum_{j \in \mathcal{J}} x_\pi(i,j) A(i,j) && (7) \\ \text{subject to} \quad & \sum_{i \in \mathcal{I}} \rho(i) x_\pi(i,j) \le \mu(j) \qquad \forall j \in \mathcal{J}\,; && (8) \\ & x_\pi \in \mathcal{X}^N. && (9) \end{aligned}$$

The objective is the steady-state rate of payoff accumulation per time period, expressed in terms of the routing matrix induced by a (WHO) policy π. The constraint is the capacity constraint: the system will be stable if and only if the total "demand" for jobs of type j is not greater than the arrival rate of jobs of type j.

Since $\mathcal{X}^N$ 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. The remainder of our paper is devoted to solving this problem and characterizing its value, by considering an asymptotic regime where $N \to \infty$.

3.4 The benchmark: Known worker types

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

$$\mathcal{D} = \left\{ x \in \mathbb{R}^{|\mathcal{I}| \times |\mathcal{J}|} : x(i,j) \ge 0;\ \sum_{j \in \mathcal{J}} x(i,j) = 1 \right\}. \tag{10}$$

Note that any routing matrix in $\mathcal{D}$ is implementable by a simple policy under known worker types: given a desired routing matrix x, route 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 accumulation is given by the solution to the following optimization problem:

$$\begin{aligned} \text{maximize} \quad & \sum_{i \in \mathcal{I}} \rho(i) \sum_{j \in \mathcal{J}} x(i,j) A(i,j) && (11) \\ \text{subject to} \quad & \sum_{i \in \mathcal{I}} \rho(i) x(i,j) \le \mu(j) \qquad \forall j \in \mathcal{J}\,; && (12) \\ & x \in \mathcal{D}. && (13) \end{aligned}$$

We let V^* denote the maximal value of the preceding optimization problem, and let x^* denote the solution. This linear program is a special case of the "static planning problem" that arises frequently in the operations literature (see, e.g. [8]). The problem can also be viewed as a version of the assignment problem due to Shapley and Shubik [35], in which the resources are divisible.

3.5 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 (8), 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 allows us to identify structural aspects of policies that perform well. In Appendix D (see Proposition D.3), we show that it is relatively "easy" to design policies that achieve a vanishing regret (and even regret that is within a constant factor of the smallest possible). The idea is straightforward: informally, when N is large, policies that "explore" for a vanishing fraction of worker lifetimes will be able to learn the worker's true type sufficiently well to yield a rate of payoff accumulation such that regret converges to zero in the limit.

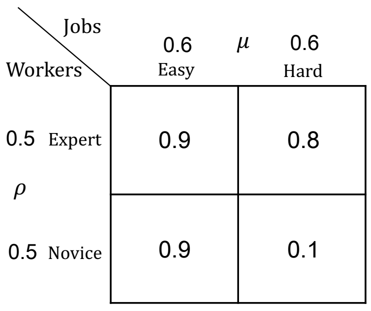
For this reason, our analysis focuses on a more refined notion of asymptotic optimality. In particular, we focus on developing policies that achieve a nearly optimal *rate* at which the regret $V^* - W^N(\pi_N)$ approaches zero. This is formalized in Theorem 5.1 below.

3.6 A note on terminology

Note that, intuitively, WHO policies have the feature that decisions are taken on the basis of the history of a given worker, not on the basis of the system profile as a whole. In the sequel, *we will typically refer to $\pi(H_k, j)$ as "the probability that a worker of history H_k is matched to a job of type j."* We use this terminology to make the presentation more intuitive, since the intention is that our algorithms be implemented at the level of each individual worker's history. However, to formalize all our arguments, we emphasize that our proofs translate $\pi(H_k, j)$ as the fraction of workers of history H_k matched to a job type j; this correspondence applies throughout the technical development.

4 Decentralized Explore-then-Exploit for Matching (DEEM): A payoff-maximizing policy

In this section we present the design of a sequence of policies π^*_N that achieves a nearly optimal rate of convergence of regret $V^* - W^N(\pi^*_N)$. We refer to our policy design as *DEEM: Decentralized Explore-then-Exploit for Matching.* Our main result, stated in the next section, is Theorem 5.1: there we exactly quantify the regret performance of DEEM (an upper bound on its regret), and characterize it as nearly optimal (a lower bound on the regret of any feasible WHO policy).



		0.6	μ	0.6
Workers \ Jobs		Easy		Hard
0.5	Expert	0.9		0.8
ρ				
0.5	Novice	0.9		0.1

Figure 1: An example

To begin to understand the challenges involved, consider the example in Figure 1. In this example, there are two types of workers: "novice" and "expert," with a mass of $\rho = 0.5$ of each present in steady state. There are two types of jobs: "easy" and "hard," each arriving at rate 0.6.

We make several observations regarding this example that inform our subsequent work.

1. *The benchmark.* In this example, the optimal solution to the benchmark problem (11)–

(13) with *known* types routes all novices to easy jobs, a mass 0.1 of experts to easy jobs, and a mass 0.4 of experts to hard jobs. Of course, our problem is that we do not know worker types on arrival.

2. *Capacity constraints affect an optimal WHO policy's need to learn.* If easy and hard jobs are in infinite supply, then the WHO policy π that matches all workers to easy jobs is optimal. However, with the finite supply of available easy jobs, some workers must do hard jobs. But which workers?

Clearly, for payoff optimality, an optimal policy should aim to match *experts* to hard jobs. But this is only possible if it first learns that a worker is an expert. Because of the structure of A, the type of a worker can only be learnt by matching it to hard jobs; those who perform well on these jobs are experts, and those who fail are novices.

3. *Minimizing regret requires learning up front.* Assigning workers of unknown type to hard jobs necessarily incurs regret relative to the benchmark. Indeed, novices unknowingly matched to hard jobs lead to a regret of 0.8 per unit mass of such workers in each period. Minimizing this regret therefore requires that the algorithm not only learn worker types, but also do so relatively early in their lifetime, so that workers identified as experts can be assigned many hard jobs.

In our work, this leads to a structure where we separate our policy into *exploration* and *exploitation* phases: the policy first tries to learn a worker's type, and then "exploits" by assigning this worker to jobs while assuming that the learned type is correct. The exploration phase will be of length $\mathrm{O}(\log N)$, which is short relative to the worker's lifetime.

4. *Some mistakes in the exploration phase are worse than others.* There are two kinds of mistakes that the policy can make while learning: it can mistakenly identify novices as experts, and it can mistakenly identify experts and novices. These mistakes differ in their impact on regret.

Suppose that at the end of the exploration phase, the algorithm misclassifies a novice as an expert. This has a dire impact on regret: the novice is then assigned to hard jobs in the exploitation phase, and as noted above, this incurs a regret of 0.8 per unit mass (of workers misclassified this way) per unit time. Thus we must work hard in the exploration phase to avoid such errors.

On the other hand, suppose that at the end of the exploration phase, the algorithm misclassifies an expert as a novice. This mistake is far less consequential: workers misclassified in this way will be assigned to easy jobs. But a mass 0.1 of experts must be assigned to easy jobs even in the benchmark solution with known types. Therefore, as long as this misclassified mass is not too large, we can adjust for it in the exploitation phase.

This discussion highlights the need to precisely identify the *learning goals* of the algorithm: to minimize regret, how strongly does each worker type need to be distinguished from others? A major contribution of our work is to demonstrate an optimal construction of learning goals for regret minimization. As noted above, the capacity constraints fundamentally influence the learning goals of the algorithm.

In the remainder of the section, we describe key ideas behind the construction of our policy, highlighted by the issues raised in the preceding example. We formally describe DEEM in Section 4.4. We state our main theorem in Section 5.

4.1 Key idea 1: Use shadow prices as an "externality adjustment" to payoffs

We begin by first noticing an immediate difficulty that arises in using WHO policies in the presence of capacity constraints. WHO policies are decentralized, i.e., they act only on the history of the worker; as such, they cannot use *aggregate* state information about the system, that conveys whether capacity constraints are being met or not. In order to solve (7)–(9), therefore, we need to find a way to "adjust" for capacity constraints despite the fact that our policy acts only at the level of worker histories.

Our key insight is to use *shadow prices* for the capacity constraints to adjust payoffs; we then measure regret with respect to these adjusted payoffs. Recall that (7)–(9) is a linear program. Let p^N be the optimal shadow prices (dual variables) for the capacity constraints (8). Then by standard duality results, it follows that the policy that is optimal for (7)–(9) is also optimal for the following unconstrained optimization problem:

$$\text{maximize}_{x_\pi \in \mathcal{X}^N} \sum_{i \in \mathcal{I}} \rho(i) \sum_{j \in \mathcal{J}} x_\pi(i,j)(A(i,j) - p^N(j)). \tag{14}$$

Thus one may attempt to account for capacity constraints using shadow prices[10] $p^N(j)$.

The challenge here is that the set $\mathcal{X}^N$ is quite complex, and thus characterizing the optimal shadow prices of (7)–(9) is not a reasonable path forward. Instead, we use the optimal shadow prices in the benchmark linear program with *known types* (11)–(13) to adjust payoffs; we then measure regret with respect to these adjusted payoffs (the practical heuristic we implement uses a different, instance-independent approach to estimate shadow prices; see Section 7).

We let p^* denote the vector of optimal shadow prices for the capacity constraint (12) in the problem with known types (11)–(13). Using the generalized imbalance condition, we show that these prices are uniquely determined; see Proposition D.2 in Appendix D.

Although $p^*(j) \neq p^N(j)$, for large N, the platform should be able to learn the type of a worker type early in her lifetime, leading to small $|p^*(j) - p^N(j)|$. This motivates an analog of (14):

$$\text{maximize}_{x_\pi \in \mathcal{X}^N} \sum_{i \in \mathcal{I}} \rho(i) \sum_{j \in \mathcal{J}} x_\pi(i,j)(A(i,j) - p^*(j)). \tag{15}$$

We develop a near-optimal algorithm for problem (15), such that (1) constraints on job capacities are not violated, and (2) complementary slackness conditions are satisfied, i.e., if $p^*(j) > 0$, then the job type j is fully utilized. We then show this leads to the upper bound in the main result.

4.2 Key idea 2: Meet required learning goals while minimizing regret

As noted in our discussion of the example in Figure 1, we must carefully define the *learning goals* of the algorithm: which worker types need to be distinguished from which others, and with what level of confidence? A key contribution of our work is to formalize the learning goals of our algorithm. In this section we define the learning goals of the algorithm, and outline the exploration phase that meets these goals.

Let the set of optimal job types for worker type i be defined by $\mathcal{J}(i) = \arg\max_{j \in \mathcal{J}} A(i,j) - p^*(j)$. (A standard duality argument demonstrates that in any optimal solution of the benchmark (11)–(13), a worker type i is assigned only to jobs in $\mathcal{J}(i)$.)

[10]Further effort is needed to ensure the policy does not violate capacity constraints, and that complementary slackness holds.

Recall that in the example in Figure 1, it is far more important not to misclassify a novice as an expert than to misclassify an expert as a novice. We formalize this distinction through the following definition.

Definition 1. *We say that a type i needs to be* strongly distinguished *from a type i' if $\mathcal{J}(i) \setminus \mathcal{J}(i') \neq \emptyset$. For each worker type i, let* $\mathrm{Str}(i)$ *be the set of all types i' from which i needs to be strongly distinguished, i.e.,* $\mathrm{Str}(i) = \{i' : \mathcal{J}(i) \setminus \mathcal{J}(i') \neq \emptyset\}$.

In words, this means that i needs to be strongly distinguished from i' if it has at least one optimal job type that is not optimal for i', whereas it needs to be only *weakly distinguished* from i' if all optimal job types for i are also optimal for i'. This definition is most easily understood through the example in Figure 1 and our subsequent discussion. In particular, note that for that example, the benchmark shadow prices are $p^*(\text{easy}) = 0.1$ and $p^*(\text{hard}) = 0$, and thus $\mathcal{J}(\text{novice}) = \{\text{easy}\}$, while $\mathcal{J}(\text{expert}) = \{\text{easy}, \text{hard}\}$. Thus experts need to be strongly distinguished from novices, since hard jobs are optimal for experts but not for novices; on the other hand, novices need to be only weakly distinguished from experts, since easy jobs are optimal for experts as well.

In the exploration phase of our algorithm, our goal is to classify a worker's type as quickly as possible: the preceding definition is what we use to formalize the learning goals in this phase. In particular, consider making an error where the true type is i', but we misclassify it as i. If i' is not in $\mathrm{Str}(i)$, any probability of an error of $o(1)$ for such misclassification error is tolerable as N grows large (as in the example in Figure 1). We choose $\Theta(1/\log N)$ as the target error probability for this kind of error. On the other hand, for any $i' \in \mathrm{Str}(i)$, the optimal target error probability is much smaller. In particular, the optimal target error probability can be shown to be approximately $1/N$: if we choose a larger target, we will incur a relatively large expected regret during exploitation due to misclassification, if we choose a smaller target, the exploration phase is unnecessarily long, and we thus incur a relatively large regret in the exploration phase.

With the learning goals defined, the exploration phase of DEEM operates in one of two subphases: either "guessing" or "confirmation," as follows. At every job allocation opportunity, we check whether the posterior probability of the *maximum a posteriori* (MAP) estimate of the worker type is sufficiently high. If this probability is low, we say the policy is in the "guessing" subphase of the exploration phase, and a job type is chosen at random for the next match. On the other hand, if it is high (in particular, greater than $\log N$ times the posterior probability of any other worker type), then we say that the policy is in the "confirmation" subphase of the exploration phase: in this regime, the policy works to confirm the MAP estimate. Specifically, in the confirmation subphase, the policy focuses only on *strongly distinguishing* the MAP from all other types in $\mathrm{Str}(i)$; the trade-off is that this must be done with minimum regret. We frame this as an optimization problem (see (16) below): essentially, the goal is to find a distribution over job types that minimizes the expected regret until the confirmation goals are met. In the confirmation subphase, the policy allocates the worker to jobs according to this distribution, until the type is confirmed.

We conclude by briefly explaining the role of the guessing phase in minimizing regret. Informally, guessing is necessary so that confirmation minimizes regret for the correct worker type with high probability. In particular, suppose that there are two worker types i and i' that have the same optimal job types, i.e., $\mathcal{J}(i) = \mathcal{J}(i')$, and with $A(i, j) = A(i', j)$ for all $j \in \mathcal{J}(i)$. In this case, payoff maximization does not require distinguishing i from i'. Nevertheless, it is possible that the *confirmation* policies for i and i' differ, without necessarily distinguishing i from i'. In this case, i first needs to be distinguished from i' with probability of error $o(1)$ to achieve optimal regret to leading order. Concretely, if there is no guessing phase and the MAP is

i early in the worker's lifetime, the policy will never discover its mistake and ultimately confirm using the wrong policy, incurring an additional (leading order) regret of $\Theta(\log N/N)$.

4.3 Key idea 3: Optimally allocate in the exploitation phase while meeting capacity constraints

When the algorithm completes the exploration phase, it enters the exploitation phase; in this phase, the algorithm aims to match a worker to jobs that maximize the rate of payoff generation, given the confirmed type label. A naive approach would match a worker labeled type i to any job type in $\mathcal{J}(i)$, since these are the optimal job types for worker type i after externality adjustment.

This approach turns out to fail spectacularly and generically leads to $\Omega(1)$ regret (this occurs for any set of fixed shadow prices). To see why, we need the following fact.

Fact 1. *Under generalized imbalance, as long as there is at least one capacity constraint that is binding in some optimal solution x^* to the benchmark problem* (11)–(13) *with known types, there is at least one worker i such that $x^*(i,\cdot)$ is supported on multiple job types.*

This fact implies that appropriate *tie-breaking* between multiple optimal job types is *necessary during exploitation* for one or more worker types in order to achieve vanishing regret.

In order to implement appropriate tie-breaking, suppose that we assign jobs during the exploitation phase using the routing matrix x^* that solves the benchmark problem (11)–(13); in this case, each worker with confirmed type i is matched to job type j with probability $x^*(i,j)$. However, this naive approach needs further modification to overcome two issues. First, some capacity is being used in the exploration phase and the effective routing matrix during the exploration phase does not match x^*. Second, the exploration phase can end with an incorrectly classified worker type.

Our policy in the exploitation phase chooses a routing matrix y^* that resembles x^*, but addresses the two concerns raised in the preceding paragraph. Crucially, the chosen y^* should ensure that only job types in $\mathcal{J}(i)$ are assigned with positive probability, and satisfy the complementary slackness conditions. We show (in Proposition A.5, using Fact 1) that such a y^* indeed exists for an N large enough under the generalized imbalance condition, and we show how to compute it. Note that as y^* is a fixed routing matrix, it can be implemented in a decentralized manner.

We comment here that y^* is largely a theoretical device used to obtain the provable regret optimality of our policy. In our implementation of DEEM (see Section 7), we propose a far simpler solution: we use *dynamically updated shadow prices* to automatically achieve appropriate tie-breaking. The shadow prices respond in a "tâtonnement" manner based on the currently available supply of different job types: the price of job type j rises when the available supply falls. In particular, fluctuations in these shadow prices naturally lead to the necessary tie-breaking for efficient exploitation.

4.4 Formal definition of DEEM

In this section we provide a formal definition of the policy π_N^*, based on the discussion above.

First, for each i define the maximal externality-adjusted utility $U(i) = \max_{j\in\mathcal{J}} A(i,j) - p^*(j)$.

Then choose $\alpha(i)$ such that:

$$\alpha(i) \in \mathcal{A}(i) = \arg\min_{\alpha \in \Delta(\mathcal{J})} \frac{\sum_{j\in\mathcal{J}} \alpha_j \big(U(i) - [A(i,j) - p^*(j)]\big)}{\min_{i' \in \mathrm{Str}(i)} \sum_{j\in\mathcal{J}} \alpha_j \mathrm{KL}(i,i'|j)}, \tag{16}$$

where $\mathrm{KL}(i,i'|j)$ is the Kullback–Leibler divergence[11] between the distributions Bernoulli$(A(i,j))$ and Bernoulli$(A(i',j))$, and $\Delta(\mathcal{J})$ is the set of distributions over $\mathcal{J}$. The idea is that sampling job types from $\alpha(i)$ allows the policy to distinguish i simultaneously from all $i' \in \mathrm{Str}(i)$, while incurring the smallest possible externality-adjusted regret. In Appendix B, we show that (16) can be written as a small linear program. If the optimization problem in (16) has multiple solutions, we pick the one that has the largest denominator (and hence the largest numerator as well), thus maximizing learning rate subject to optimality; i.e., we choose

$$\alpha(i) \in \arg\max_{\alpha \in \mathcal{A}(i)} \min_{i' \in \mathrm{Str}(i)} \sum_{j\in\mathcal{J}} \alpha_j \mathrm{KL}(i,i'|j). \tag{17}$$

We discuss details in the Appendix B.

For $m = 1, \cdots, N$, let the job type chosen at opportunity m be j_m and the outcome be X_m. For any $i \in \mathcal{I}$ and $j \in \mathcal{J}$, let $l(X,i,j) = A(i,j)\mathbf{1}_{\{X=1\}} + (1 - A(i,j))\mathbf{1}_{\{X=0\}}$. Define $\lambda_0(i) = 1$, and for $k \geq 1$, let $\lambda_k(i) = \prod_{m=1}^{k} l(X_m, i, j_m)$ denote the likelihood of the observed history until the k-th job under worker type i. Let $\mathrm{MAP}_k = \arg\max_{i\in\mathcal{I}} \hat{\rho}(i)\lambda_k(i)$ be the MAP estimate based on the history, and define $\Lambda_k(i,i') = \hat{\rho}(i)\lambda_k(i)/\hat{\rho}(i)\lambda_k(i')$, i.e., the ratio of the posterior probabilities of type i and i'. For convenience, we refer to $\Lambda_0(i,i')$ as the *prior odds* of i relative to i', and $\Lambda_k(i,i')$ as the *posterior odds* of i relative to i' after k jobs.

DEEM is defined as follows.

1. **Phase 1: Exploration.** Suppose that $i = \mathrm{MAP}_k$.

(a) **Guessing subphase**. If $\min_{i' \neq i} \Lambda_k(i,i') < \log N$, choose the next job type uniformly at random in $\mathcal{J}$.

(b) **Confirmation subphase to strongly distinguish i from types in Str(i)**. If we have
$\min_{i' \neq i} \Lambda_k(i,i') \geq \log N$ but $\min_{i' \in \mathrm{Str}(i)} \Lambda_k(i,i') < N$, draw the next job type i.i.d. from the distribution $\alpha(i)$.

(c) **Exit condition for the exploration phase.** If $\min_{i' \neq i} \Lambda_k(i,i') \geq \log N$ and
$\min_{i' \in \mathrm{Str}(i)} \Lambda_k(i,i') \geq N$, then the worker is labeled as being of type i and the policy moves to the exploitation phase. (The worker is never returned to the exploration phase.)

2. **Phase 2: Exploitation.** For every job opportunity, for a worker confirmed to be of type i, choose a job in $\mathcal{J}(i)$ with probability $y^*(i,j)$, where y^* is a routing matrix (specified in Proposition A.5 in Appendix A) such that system capacity constraints are not violated in steady state.

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 the sequence of policies π^*_N constructed in the preceding section (essentially) achieves this lower bound.

[11] The KL divergence between a Bernoulli(q) and a Bernoulli(q') distribution is defined as $q \log \frac{q}{q'} + (1 - q) \log \frac{1-q}{1-q'}$.

Theorem 5.1. *Fix (ρ, μ, A) such that: (a) no two rows of A are identical; and (b) the generalized imbalance condition holds. Then there is a constant $C = C(\rho, \mu, A) \in [0, \infty)$ such that*

1. *(Lower bound) For any N and any WHO policy π that is feasible for (7)–(9),*

$$V^* - W^N(x_\pi) \geq \frac{C \log N}{N}\big(1 + \mathrm{o}(1)\big) \text{ and} \tag{18}$$

2. *(Upper bound) The sequence of policies π_N^* is feasible for (7)–(9) for each N, with:*

$$V^* - W^N(\pi_N^*) \leq \frac{C \log N}{N}\big(1 + \mathrm{o}(1)\big) + \mathrm{O}\Big(\frac{\log \log N}{N}\Big). \tag{19}$$

The constant C that appears in the theorem depends on the primitives of the problem, i.e., (ρ, μ, A); it is defined as follows:

$$C(i) = \min_{\alpha \in \Delta(\mathcal{J})} \frac{\sum_{j \in \mathcal{J}} \alpha_j \big(U(i) - [A(i,j) - p^*(j)]\big)}{\min_{i' \in \mathrm{Str}(i)} \sum_{j \in \mathcal{J}} \alpha_j \mathrm{KL}(i, i'|j)}\,; \qquad C = \sum_{i \in \mathcal{I}} \rho(i) C(i)\,. \tag{20}$$

(Note that $C(i)$ captures the regret per unit mass of service opportunities from workers of type i.) Informally, instances in which there is a conflict between exploration (i.e., learning worker type) and exploitation (i.e., maximizing short-term payoffs) have larger values of C. The case $C = 0$ corresponds to instances where the goals of learning and regret minimization are aligned; i.e., learning does not require regret of $\Omega(\log N/N)$. In this case, our result establishes that our chosen policies are nearly asymptotically optimal, to within $\mathrm{O}(\log \log N/N)$. On the other hand, the instances with $C > 0$ are those instances with a non-trivial tension between learning and short-term payoffs. For these instances, our result establishes that our chosen policies $(\pi_N^*)_{N \geq 1}$ achieve asymptotically optimal regret upto leading order in N.

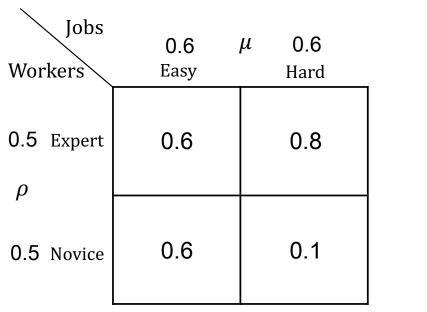


Figure 2: An example where $\Omega(\log N/N)$ regret is unavoidable.

The constant C is best understood in terms of the definition of α in the exploration phase (cf. (16)). Note that for a fixed α, for workers of true type i, the smallest value of the log posterior odds, $\min_{i' \in \mathrm{Str}(i)} \log \Lambda_n(i, i')$, increases at an expected rate of $\min_{i' \in \mathrm{Str}(i)} \sum_{j \in \mathcal{J}} \alpha_j \mathrm{KL}(i, i'|j)$ during confirmation. Thus, when N is large, the time taken to confirm i against worker types in $\mathrm{Str}(i)$ is approximately $\log N/(\min_{i' \in \mathrm{Str}(i)} \sum_{j \in \mathcal{J}} \alpha_j \mathrm{KL}(i, i'|j))$. Hence, the externality-adjusted regret incurred until confirmation is complete, per unit mass of workers of type i, is approximately $\sum_{j \in \mathcal{J}} \alpha_j \big(U(i) - [A(i,j) - p^*(j)]\big) \log N/(\min_{i' \in \mathrm{Str}(i)} \sum_{j \in \mathcal{J}} \alpha_j \mathrm{KL}(i, i'|j))$. Optimizing over α results in an expected regret of nearly $C(i) \log N$ that must be incurred until the strong distinguishing goals are met for a unit mass of workers of type i. This translates to an expected regret of nearly $\hat{\rho}(i) C(i) \log N = \rho(i) C(i) \log N/N$ owing to workers of type i per time unit. This reasoning forms the basis of our lower bound, formalized in Proposition A.1 in Appendix A.

Now, a regret of $\Theta(\log N/N)$ is unavoidable when $C(i) > 0$ for some i. To develop some intuition for this case, consider the same example as before, but with a modified payoff matrix shown in Figure 2. It can be shown that in this case, a regret of $\Omega(\log N/N)$ is unavoidable in the event that the true type of the worker is novice [1]. The problem is the following: to distinguish novices from experts, the policy must allocate workers to hard jobs. But hard jobs are strictly suboptimal for novices, and so if the true type of the worker is novice, some regret is unavoidable.

In particular, to develop intuition for the magnitude of this regret, imagine a policy that assigns workers hard jobs for the first k steps (leading to $\Theta(k)$ absolute regret per unit mass of workers), and based on the realized payoffs, estimates the worker type (with confidence $1-\exp(-\Theta(k))$). If the worker is estimated to be a novice, the policy can choose to assign only easy jobs to the worker. However, this means that there will be no further learning about the worker type, and the expected contribution to absolute regret is about N times the probability that the worker is truly an expert, i.e., $N\exp(-\Theta(k))$, per unit mass of workers. Combining, we see the total absolute regret is at least $\Theta(k)+N\exp(-\Theta(k)) \geq \Theta(\log N)$ over the lifetime of a unit mass of workers. (Here $k=\Theta(\log N)$ is needed to achieve $\Theta(\log N)$ absolute regret.) We then divide by N to obtain the regret per unit mass of service opportunities by workers.

This discussion motivates the following definition.

Definition 2. *Consider a worker type i. Suppose that there exists another type i' such that $A(i,j)=A(i',j)$ for all $j\in\mathcal{J}(i)$ and $\mathcal{J}(i)\cap\mathcal{J}(i')=\phi$. Then we say that the ordered pair (i,i') is a* difficult type pair.

A similar definition also appears in [1]; the modification here is that the sets $\mathcal{J}(i)$ are defined with respect to externality-adjusted payoffs to account for capacity constraints.

The constant $C(i)>0$ if and only if there is some other i' such that (i,i') is a difficult type pair. In general, if: (1) none of the job types in $\mathcal{J}(i)$ allow us to distinguish between i and i'; and (2) all the jobs in $\mathcal{J}(i)$ are strictly suboptimal for i', then any policy that achieves small regret must distinguish between i and i' and must assign the worker to jobs outside $\mathcal{J}(i)$ to make this distinction. This leads to a regret of $\Omega(\log N)$ per unit mass of workers of type i (over the lifetime of the workers).

On the other hand, if there is no difficult type pair, then there is no conflict between learning and regret minimization. Here, one can show that $C(i)=0$ for each i, and this value is attained by some distribution $\alpha(i)$ that is supported on $\mathcal{J}(i)$. To see this note that if α is fully supported on $\mathcal{J}(i)$ (i.e., $\alpha_j>0$ for all $j\in\mathcal{J}(i)$), then the numerator is 0; however if there is no type i' such that (i,i') is a difficult type pair, then the denominator is strictly positive, and thus $C(i)=0$. In this case, $C=0$ and our main result says that our algorithm achieves a regret of $\mathrm{O}(\log\log N/N)$ asymptotically.[12] This regret basically results from the uniform sampling of the job types during the guessing phase, which accounts for $\mathrm{O}(\log\log N/N)$ fraction of the lifetime of the worker.

6 Proof sketch

The proof of Theorem 5.1 can be found in Appendix A. 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 with non-negative prices p^* from the optimization problem (11) with known worker types.

$$W_{p^*}^N = \max_{x\in\mathcal{X}^N}\sum_{i\in\mathcal{I}}\rho(i)\sum_{j\in\mathcal{J}}x(i,j)A(i,j)-\sum_{j\in\mathcal{J}}p^*(j)\Big[\sum_{i\in\mathcal{I}}\rho(i)x(i,j)-\mu(j)\Big]. \tag{21}$$

[12]In fact, our proof demonstrates that this regret can be brought down to any $\mathrm{O}(f_N/N)$ such that $f_N=\mathrm{o}(1)$ by choosing a different threshold in the guessing phase.

Lower bound on regret. If $C > 0$ (i.e., if there is at least one difficult pair of worker types; cf. Section 5), there is an upper bound on the performance of any policy in this problem, expressed relative to V^*. This result follows directly from [1]:

$$W^N_{p^*} \le V^* - \frac{C \log N}{N}(1 + \mathrm{o}(1)),$$

where $C \ge 0$ is precisely the constant appearing in (20). By a standard duality argument, we know that $W^N \le W^N_{p^*}$, and hence this bound holds for W^N as well (see Proposition A.1), yielding the lower bound on regret on our original problem (7).

Upper bound on regret. There are two key steps in proving that π^*_N is feasible for problem (7)–(9), and $W^N(\pi^*_N) \ge V^* - C(\log N/N)(1 + \mathrm{o}(1))$.

1. First, we show that our policy π^*_N, with an arbitrary exploitation-phase routing matrix supported on $\mathcal{J}(i)$ for each $i \in \mathcal{I}$, achieves near optimal performance for the single multi-armed bandit problem (21). Formally, if (with some abuse of notation) we let $W^N_{p^*}(\pi)$ denote the value attained by a policy π in problem (21), i.e.,

$$W^N_{p^*}(\pi) = \sum_{i\in\mathcal{I}} \rho(i) \sum_{j\in\mathcal{J}} x_\pi(i,j) A(i,j) - \sum_{j\in\mathcal{J}} p^*(j) \Big[\sum_{i\in\mathcal{I}} \rho(i) x_\pi(i,j) - \mu(j) \Big],$$

then

$$V^* - W^N_{p^*}(\pi^*_N) = C(\log N/N)(1 + \mathrm{o}(1)) + \mathrm{O}(\log\log N/N).$$

This is shown in[13] Proposition A.2. Thus we have $W^N_{p^*}(\pi^*_N) \ge W^N_{p^*} - \mathrm{o}(\log N/N)$, i.e., π^*_N is near-optimal in problem (21).

2. In the next part of the proof, we show that we can design a routing matrix y^* (that depends on N) for the exploitation phase of the policy π^*_N, such that the following conditions are satisfied:

(a) (Complementary slackness) $\sum_{i\in\mathcal{I}} \rho(i) x_{\pi^*_N}(i,j) - \mu(j) = 0$ for all j such that $p^*(j) > 0$, and,

(b) (Feasibility) $\sum_{i\in\mathcal{I}} \rho(i) x_{\pi^*_N}(i,j) - \mu(j) \le 0$ for all other $j \in \mathcal{J}$.

This is shown in Proposition A.5. We deduce that π^*_N with this choice of y^* in the exploitation phase is feasible for problem (7)–(9) and the complementarity slackness property implies that $W^N(\pi^*_N) = W^N_{p^*}(\pi^*_N)$, yielding our upper bound on regret.

Construction of y^*. At the end of the exploration phase of π^*_N, the correct label of the worker is learned with a confidence of at least $(1 - \mathrm{o}(1))$. This fact, coupled with the generalized imbalance condition (leading to flexibility in modifying x^*; cf. Fact 1), is sufficient to ensure an appropriate and feasible choice of $y^* = x^* + \mathrm{o}(1)$ will correct the deviations from x^* in terms of capacity utilizations of job types with $p^*(j) > 0$ arising 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.

[13][1] proves this result for a similar policy.

7 Practical considerations and a heuristic

Our theoretical analysis of DEEM focused on an asymptotic regime where $N \to \infty$. In this section, we focus on a number of practical considerations that arise when considering implementation of a policy like DEEM. First, we discuss a practical approach to managing capacity constraints via dynamic *queue-based* shadow prices. Second, we discuss two modifications to the algorithm that improve performance when N is finite, and suggest a modified heuristic that we call DEEM$^+$. In the next section, we simulate both DEEM and DEEM$^+$, and evaluate their performance.

7.1 Dynamic queue-based shadow prices

A key step in making DEEM practical is to use *dynamic* shadow prices based on supply-demand imbalances in the market. In our mathematical model, we had assumed that the masses of new workers and jobs arrive instantaneously at the beginning of each period, after which they are instantaneously matched, and further, each job either gets matched immediately on arrival or disappears at the end of the period. However, in real platforms, the arrivals, departures, and matchings of workers and jobs occur sequentially in continuous time. In these settings, it is common for platforms to maintain a "queue" of jobs of each type that grows when new jobs arrive (in continuous time) and shrinks when existing jobs are matched. In this scenario, the queue length at any time can be leveraged to compute an instantaneous shadow price for the job type that can be utilized for externality adjustment of the payoffs.

A reasonable approach is to set the shadow price on each job type via a decreasing function of the corresponding queue length. One natural way to do this is as follows: assume that in practice, the arriving jobs accumulate in queues for the different types, each with a finite capacity B (if the capacity is exceeded, jobs are lost). If the queue length of job type j at any instant is $q(j)$, we set the price of j at that instant to $p_q(j) = (B - q(j))/B$ (thus the price lies in $[0, 1]$). Note that $p_q(j)$ changes every time a job is assigned to a worker, or a new job arrives, because the queue length changes. We implement a discrete-time analog of this approach in our simulated marketplace in the next section.

Computing the prices in this fashion obviates the need to explicitly compute y^* in the exploitation phase of our policy; instead the exploitation phase can be implemented by allocating optimally for each worker given the current queue-based prices (still a fully decentralized solution). The natural fluctuation in these prices ensures appropriate tie-breaking in allocation (cf. Fact 1).

These queue-based prices can be incorporated in the implementation of DEEM in the following way.

1. **Modifying the exploration and exploitation phases.** While computing $\alpha(i)$ for each $i \in \mathcal{I}$ in the confirmation phase of DEEM, replace $p^*(j)$ by the instantaneous queue-based shadow prices $p_q(j)$ in Eq. (16). Similarly, in the exploitation phase, instead of explicitly computing the routing matrix y^*, use these prices to decide assignments in the following manner. Define the sets $\mathcal{J}^*(i)$ as:

$$\mathcal{J}^*(i) = \arg\max_{j \in \mathcal{J}} A(i, j) - p_q(j).$$

If an assignment has to be made in the exploitation phase for some worker who has already been labeled as being of type i, then a job type $j^* \in \mathcal{J}^*(i)$ is chosen (note that typically, $\mathcal{J}^*(i)$ will be a singleton).[14]

2. **Price-smoothing for learning goals.** The platform can also determine the strong distinction requirements (see Definition 1), i.e., the set $\mathrm{Str}(i)$ for each i based on the sets $\mathcal{J}^*(i)$ induced by the instantaneous prices defined above. But this approach suffers from the drawback that random fluctuations of the shadow prices around their mean values can result in changes in the sets $\mathcal{J}^*(i)$, and hence in the learning goals, which could be detrimental to the performance of our policy. On the other hand these fluctuations are essential for appropriate tie-breaking across multiple optimal job types in the exploitation phase.

Thus we propose the following modification: we utilize an *average* of recent prices within a fixed recent window of time, and modify the definition of $\mathcal{J}^*(i)$ to incorporate a small tolerance so that the set $\mathrm{Str}(i)$ remains unaffected by the fluctuations in the prices. To be precise, for a window size W, let $\bar{p}_q(i)$ be the unweighted average of the queue length based prices seen over the past W epochs of changes in the price (again, note that $p_q(i)$ changes every time a job is assigned to a worker, and also when new jobs arrive). Next, for a tolerance $\varepsilon > 0$, we define

$$\mathcal{J}^*_\varepsilon(i) = \{j \in \mathcal{J} : A(i,j) - \bar{p}_q(j) \geq \max_{j'}[A(i,j') - \bar{p}_q(j')] - \varepsilon\}.$$

Then the set $\mathrm{Str}(i)$ for each i (see Definition 1), is defined based on $\mathcal{J}^*_\varepsilon(i)$.

7.2 Improving performance in the finite N regime

We propose two changes that improve performance in the finite N regime. First, recall that if a worker type i has an optimal job type j that is not optimal for some worker type i', then DEEM tries to achieve a probability $1/N$ of misclassifying a worker of type i' as type i. For small N, however, we can do better: the desired probability of error should explicitly depend on how much regret type i' incurs by performing job j – if this regret is very small, then it isn't worth trying to make this distinction with high precision.

In particular, for each $i' \in \mathrm{Str}(i)$, define:

$$R(i,i') \triangleq \max_{j \in \mathcal{J}(i),\, j \notin \mathcal{J}(i')} U(i') - [A(i',j) - p^*(j)].$$

$R(i,i')$ is the highest regret incurred if type i' is matched to a suboptimal job that is optimal for type i. A reasonable approach is to aim for a probability of misclassification of i' as i of $1/NR(i,i')$ instead of $1/N$, thus accounting for the fact that if $R(i,i')$ is small, then we can tolerate a higher probability of error.

The second change we propose is to explicitly incorporate the posterior into the exploration phase. Recall that in DEEM, we guess and then confirm in the exploration phase; and guessing is not optimized, but rather involves exploration uniformly at random. When N is finite, we can gain by instead leveraging the posterior at each round to appropriately allocate confirmation effort across the different types, until the learning goals are met for some type i. In principle, this approach can subsume the guessing and confirmation phases into a uniformly defined exploration phase; the challenge is to precisely describe how the posterior is used to guide the exploration phase.

[14] In practice, we can continue to benefit from learning during the exploitation phase: instead of optimizing the price-adjusted payoff for the confirmed worker label, we can optimize for the current MAP estimate, thus accounting for the possibility that we may have confirmed incorrectly. Clearly, doing so can only improve performance.

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

In this subsection, we incorporate the two suggestions of the preceding subsection into a formal heuristic we refer to as DEEM$^+$.

It will be convenient to define $R(i,i')=1$ for all $i' \notin \mathrm{Str}(i)$, and define $\hat{\Lambda}_k(i,i')$ as follows:

1. If $i' \in \mathrm{Str}(i)$, then $\hat{\Lambda}_k(i,i') \triangleq \Lambda_k(i,i')/R(i,i')$.

2. If $i' \notin \mathrm{Str}(i)$, then $\hat{\Lambda}_k(i,i') \triangleq \Lambda_k(i,i')/R(i,i')$ if $\Lambda_k(i,i')/R(i,i') < \log N$, and $\hat{\Lambda}_k(i,i') \triangleq N$ otherwise.

Next, after k matches, for each type i define:

$$\mathcal{L}_k(i) \triangleq \{i' \neq i : \hat{\Lambda}_k(i,i') < N\}.$$

$\mathcal{L}_k(i)$ is the set of types that i remains to be distinguished from after k opportunities. In case the true worker type is i, effort should ideally be directed towards these distinctions in order to speed up confirmation. Next, define the posterior probability of the worker being of type i after opportunity k as:

$$g_k(i) \triangleq \frac{\hat{\rho}(i)\lambda_k(i)}{\sum_{i' \in \mathcal{I}} \hat{\rho}(i')\lambda_k(i')}.$$

Then DEEM$^+$ is defined as follows:

1. **Phase 1: Exploration.** DEEM$^+$ is in this phase as long as there is no i such that $\min_{i' \neq i} \hat{\Lambda}_k(i,i') \geq N$. After k allocations, choose a job from a distribution α^k that satisfies:

$$\alpha^k \in \mathcal{A} = \underset{\alpha \in \Delta(\mathcal{J})}{\arg\min} \frac{\sum_i g_k(i) \sum_{j \in \mathcal{J}} \alpha_j \big(U(i) - [A(i,j) - p^*(j)]\big)}{\min_{i \in \mathcal{I};\, i' \in \mathcal{L}_k(i)} (1/g_k(i)) \sum_{j \in \mathcal{J}} \alpha_j \mathrm{KL}(i,i'|j) / \left(1 + \log R(i,i')/\log N\right)}, \qquad (22)$$

α^k can be computed as a solution to a linear program as shown in Appendix B. [15]

Exit from exploration. If at some opportunity k, there is a worker type i such that $\min_{i' \notin i} \hat{\Lambda}_k(i,i') \geq N$, then label the worker as being of type i and enter the exploitation phase.

2. **Phase 2: Exploitation.** Exploitation is the same as defined in DEEM.

DEEM$^+$ shares the same explore-then-exploit structure as DEEM, but with changes to the exploration phase to optimize learning at finite N; as we will see in our simulations, these optimizations allow DEEM$^+$ to substantially outperform DEEM at small N.

Although an exact analysis is beyond the scope of this work, we conjecture that DEEM$^+$ inherits the asymptotic performance bounds that hold for DEEM. Informally, consider the periods in DEEM$^+$ where $\hat{\Lambda}_k(i,i') < \log N$ for the MAP estimate i, for some type $i' \neq i$; these periods are analogous to the guessing phase in DEEM. Similar to DEEM, one can argue that this phase accounts for at most $\mathrm{O}(\log\log N/N)$ fraction of the worker lifetime. The stages where $\hat{\Lambda}_k(i,i') > \log N$ for all $i' \neq i$ but $\hat{\Lambda}_k(i,i') < N$ for some i' are similarly analogous to the confirmation subphase of DEEM. Here we can informally argue that on the event that i is the true type, the posterior distribution quickly concentrates sufficiently on i, and the policy defined by

[15] The adjustment in the denominator precisely accounts for differences in regret incurred in making each distinction.

(22) asymptotically achieves the same regret until confirmation (in the leading order term) as the stationary randomized policy $\alpha(i)$ defined in DEEM. To see this, observe that as $g_k(i) \to 1$ for some i, the objective function in (22) converges to the objective function for computing $\alpha(i)$ in (16), modulo the $(1 + \log R(i, i')/\log N)$ factors that simply capture the fact that the learning goals have been adjusted for small N.

Just like DEEM, DEEM$^+$ can be analogously implemented using queue-based shadow prices instead of $p^*(j)$ (with the sets $\mathrm{Str}(i)$ computed using smoothed prices).

8 Simulations

In this section, we simulate DEEM and DEEM$^+$ in a market environment with queue-based shadow prices. We compare performance of these policies against a greedy policy, as well as benchmark MAB approaches. We consider instances with 3 types of workers and 3 types of jobs. We assume that $N = 30$, and $\rho(i) = 1$ for each i (so that $\hat{\rho}(i) = 1/30$). We generated 350 instances, where for each instance, independently: (1) $\mu(j)$ is sampled from a uniform distribution on $[0.5, 1.5]$; and (2) each entry of the expected payoff matrix is sampled from a uniform distribution on $[0, 1]$.

Given an instance $(\hat{\rho}, N, \mu, A)$, our simulated marketplace is described as follows.

Arrival process. Time is discrete, $t = 1, 2, \cdots, \mathbb{T}$, where we assume $\mathbb{T} = 150$. At the beginning of each time period t, $M_t(i)$ number of workers of type i and $L_t(j)$ jobs of type j arrive, such that $M_1(i), M_2(i), \cdots$ and $L_1(i), L_2(i), \cdots$ are i.i.d. sequences. For a scaling constant $\tau = 900$, we assumed that $M_t(i) = \tau\hat{\rho}(i) = 900 \times (1/30) = 30$ for all t (recall that $\hat{\rho}(i) = 1/30$ for all i in all our instances), i.e., $M_t(i)$ is deterministic. We generated $L_t(j)$ from a binomial distribution with mean $\tau\mu(j) = 900\mu(j)$.[16] Each worker stays in the system for N periods and then leaves. Each job requires one period to perform.

Queues. We assume that the arriving jobs accumulate in queues for the different types, each with a finite buffer of capacity B, where we choose $B = 10,000$. If the buffer capacity is exceeded for some job type then the remaining jobs are lost.

Matching process. In the beginning of each period, once all the new workers and jobs have arrived, the platform sequentially considers each worker in the platform,[17] and generates an assignment based on the history of the worker and the chosen policy. 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.

Queue-based prices. The platform maintains queue-based prices for the jobs in the following way: if the queue length of job type j at any instant is $q(j)$, the price of j at that instant is set to be $p_q(j) = (B - q(j))/B$. The prices thus change when either (1) new jobs arrive at the beginning of each period, or (2) a job gets matched to a worker.

A remark on the choice of instances. In all our test instances, all the entries of the expected payoff matrix A are distinct. We conjecture that this would typically be the case in many settings in practice – exact indistinguishability of different worker types using a particular

[16]A binomial distribution has two parameters (n, p), where n is the number of trials, and p is the probability of success at each trial; we chose $n = 2700$ and $p = \mu(j)/3$ for generating each $L_t(j)$. Note that since $\mu(j) \in [0.5, 1.5]$, we have $p < 1$.

[17]This consists of the new workers and all the workers who have arrived in the past $N - 1$ periods, or from the beginning of time if $t < N$.

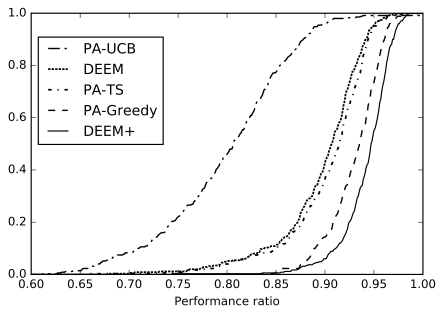


Figure 3: The empirical CDF of the performance ratios of the different policies.

Policy	Avg. Perf. Ratio ($\pm$ std. error)
PA-UCB	0.797 ($\pm$ 0.007)
PA-TS	0.904 ($\pm$ 0.005)
DEEM	0.898 ($\pm$ 0.005)
PA-Greedy	0.931 ($\pm$ 0.003)
DEEM$^+$	0.943 ($\pm$ 0.003)

Table 1: Average performance ratios of different policies across 350 instances, along with standard errors.

job type wouldn't be commonly encountered. For such instances, as we discussed in section 5, there is no conflict between learning and regret minimization in the confirmation subphase of DEEM, and it incurs a regret of $\mathrm{O}(\log\log N/N)$, where the leading order term results entirely from the regret incurred due to uniform sampling of the job types while guessing. In fact, in this case we can show that a "greedy" policy that maximizes the price-adjusted payoff for the MAP estimate throughout the exploration phase and enters exploitation after strong distinction requirements are met for some worker type, incurs a regret of $\mathrm{O}(1/N)$.[18] It would thus appear that this *price-adjusted* greedy policy, which is attractive for its simplicity, would be a reasonable solution in such cases. However our simulations show that when N is small, DEEM$^+$ can lead to significant gains over the price-adjusted greedy approach. We discuss this result further below.

8.1 Results

We implemented five policies: DEEM, DEEM$^+$, and price-adjusted versions of UCB [10], Thompson sampling (TS) [4] and Greedy, and compared their performance. All algorithms measure payoffs adjusted by the queue-based shadow prices; in this way, they all effectively account for capacity constraints. We have already described this implementation for the DEEM variants earlier. Price-adjusted Greedy (PA-Greedy) simply chooses the job type that maximizes the instantaneous shadow price adjusted payoff for the MAP estimate throughout the lifetime of the worker.[19] UCB and TS are well known algorithms for the standard stochastic multi-armed bandit problem and the details of their implementation in the presence of shadow prices, which we will denote PA-UCB and PA-TS, can be found in Appendix C.

Figure 3 shows the cumulative distribution function – over the 350 instances – of the ratio of the payoff generation rate attained by a policy and the optimal payoff generation rate if the worker types are known, for the five candidate policies. The average of these ratios over the sample space for each policy is given in Table 2. As one can observe, DEEM$^+$ significantly outperforms PA-UCB, PA-Greedy, DEEM and PA-TS. PA-Greedy, PA-TS, DEEM and DEEM$^+$ perform considerably better than UCB on average, presumably benefiting from the knowledge of the ex-

[18]Informally, since every job can make every possible distinction between worker types, the probability that the true worker type is not identified as the MAP estimate at opportunity t decays as $\exp(-\eta t)$, where η is an instance dependent constant. Thus the total expected regret over the lifetime of a worker is bounded as $\mathrm{O}(1)$ in expectation.

[19]With queue-based shadow prices, tie-breaking during exploitation is rendered unnecessary. Moreover, we allow the algorithm to continue to benefit from learning during exploitation by optimizing for the current MAP estimate rather than for the confirmed type. Thus the distinction between exploration and exploitation disappears.

pected payoff matrix A. In contrast to PA-Greedy, both DEEM and PA-TS actively experiment in order to learn quickly: DEEM experiments during its guessing phase where it uniformly samples job types, and PA-TS experiments due to sampling from the posterior (see Appendix C for details), especially in the early stages when the posterior is not sufficiently concentrated. While this experimentation is desirable, neither of them efficiently trade off between payoff maximization and learning, resulting in a degraded performance in comparison to PA-Greedy. On the other hand PA-Greedy suffers from excessive exploitation, resulting in performance which, although better than DEEM and PA-TS, is still significantly worse than DEEM$^+$. We now focus on this latter difference.

DEEM$^+$ vs. PA-Greedy. We had discussed in our remark earlier that in instances without exactly indistinguishable type pairs, PA-Greedy is expected to perform reasonably well. However in our simulations we see that on an average across all instances, DEEM$^+$ results in about 15% reduction in regret as compared to PA-Greedy.

In order to gain intuition for this gain, observe that although exactly indistinguishable type pairs are rarely encountered, it could frequently be the case that two type pairs i and i' have their expected payoffs $A(i, j)$ and $A(i', j)$ under some job type j close enough that practically it would take too long to distinguish them with a reasonably small probability of error. This results in *approximately* difficult type pairs – two types i and i' that have different optimal job types where none of the optimal job types for i is able to distinguish between i and i' reasonably quickly. In these situations, if at any point the MAP estimate under the greedy policy is i when the true type is i', then exploiting for i may not allow the policy to recover from its bad estimate within a reasonable number of jobs, thus incurring high regret. There is high probability of encountering such a situation in the early stages of the algorithm when the confidence in the MAP estimate is not sufficiently high.

This is where DEEM$^+$'s approach of appropriately allocating confirmation efforts depending on the posterior results in significant gains in performance over the greedy approach. In particular, there are situations where DEEM$^+$ will appropriately prioritize learning and will actively explore instead of simply choosing the optimal job type for the MAP estimate. For example, suppose there are only two types i and i', where (i, i') is close to being a difficult pair, i.e., the learning rate offered by the optimal job type for i towards the (i, i') distinction is close to 0. In this case, even if the current MAP estimate is i with high confidence (although $\hat{\Lambda}_k(i, i') < N$, so that we are still in the exploration phase), instead of choosing the optimal job type for i, DEEM$^+$ may choose some other job type that will quickly distinguish i from i'.

Thus we expect DEEM$^+$ to outperform PA-Greedy more significantly in situations where there are approximately difficult type pairs. In order to verify that this is indeed the case, first, we formally define a simple notion of approximate indistinguishability and difficulty. We say that the type pair (i, i') is Δ-*indistinguishable* using a job type j if $\mathrm{KL}(i, i'|j) < \Delta$; otherwise we say that it is Δ-distinguishable. We say that type pair (i, i') is Δ-*difficult* if $\mathrm{KL}(i, i'|j) < \Delta$ for all $j \in \mathcal{J}(i)$ and $\mathcal{J}(i) \cap \mathcal{J}(i') = \phi$.

For $\Delta \in \{0.1, 0.4\}$, we picked those instances out of the 350 in which there is at least one pair (i, i') such that (1) (i, i') is Δ-difficult, and (2) there is some j such that $\mathrm{KL}(i, i'|j) > 0.5$; i.e., instances with at least one Δ-difficult type pair, such that there exists a job type under which this pair is 0.5-distinguishable. We will call such instances Δ-DBD (Δ-difficult-but-distinguishable) instances. These are precisely the instances where measured exploration in cases where the MAP estimate is a worker type that forms a Δ-difficult type pair with some other type can lead to significant gains in performance.[20] Note that as Δ increases, the set of instances that satisfy

[20]Note that if $\mathrm{KL}(i, i'|j) > 0.5$, the job j can distinguish i from i' with a misclassification error of 1/30 for

Sample	Number of instances	Avg. regret reduction
A	78	21%
B	82	12%

Table 2: DEEM$^+$'s percentage reduction in regret relative to Greedy on average in sample A, consisting of 0.1-DBD instances, and in sample B consisting of 0.4-DBD instances that aren't 0.1-DBD.

these conditions grows progressively larger (a Δ-DBD instance is also a Δ'-DBD instance for $\Delta' > \Delta$).

We next considered two sets of samples: sample A is the set of 0.1-DBD instances, and sample B is the set of 0.4-DBD instances that are not 0.1-DBD. Based on the discussion above, we should expect a substantial reduction of regret in the 0.1-DBD instances, relative to those 0.4-DBD instances that are not 0.1-DBD. Indeed, consistent with our intuition, a one tailed two sample t-test showed that the mean percentage reduction of regret in sample A is larger than that in sample B with a p-value of 0.009. The sample average percentage reduction of regret in the two samples is given in Table 2.

9 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, while we consider a finite-type model, a richer model of types would admit a wider range of applications; e.g., workers and jobs may be characterized by features in a vector-valued space, with compatibility determined by the inner product between feature vectors. Second, while 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). We expect that a similar approach using externality prices to first set learning objectives, and then achieve them while incurring minimum regret, should be applicable even in these more general settings.

Third, recall that we assumed the expected surplus from a match between a worker type and a job type (i.e., the A matrix) is known to the platform. This reflects a first order concern of many platforms, where aggregate knowledge is available, but learning individual user types quickly is challenging. Nevertheless, it may also be of interest to study how A can be efficiently learned by the platform. This direction may be related to issues addressed by the literature on (a single) multi-armed bandit under capacity constraints [2].

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.

References

[1] Rajeev Agrawal, Demosthenis Teneketzis, and Venkatachalam Anantharam. Asymptotically efficient adaptive allocation schemes for controlled iid processes: finite parameter space.

strong distinction, in at most about 7 jobs on average, which is reasonably quick.

Automatic Control, IEEE Transactions on, 34(3):258–267, 1989.

[2] Shipra Agrawal and Nikhil R Devanur. Bandits with concave rewards and convex knapsacks. In *Proceedings of the fifteenth ACM conference on Economics and computation*, pages 989–1006. ACM, 2014.

[3] Shipra Agrawal and Nikhil R Devanur. Linear contextual bandits with global constraints and objective. *arXiv preprint arXiv:1507.06738*, 2015.

[4] Shipra Agrawal and Navin Goyal. Analysis of thompson sampling for the multi-armed bandit problem. *arXiv preprint arXiv:1111.1797*, 2011.

[5] Shipra Agrawal, Nikhil R Devanur, and Lihong Li. Contextual bandits with global constraints and objective. *arXiv preprint arXiv:1506.03374*, 2015.

[6] Mohammad Akbarpour, Shengwu Li, and Shayan Oveis Gharan. Dynamic matching market design. *Available at SSRN 2394319*, 2014.

[7] Ross Anderson, Itai Ashlagi, David Gamarnik, and Yash Kanoria. A dynamic model of barter exchange. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1925–1933. SIAM, 2015.

[8] Baris Ata and Sunil Kumar. Heavy traffic analysis of open processing networks with complete resource pooling: asymptotic optimality of discrete review policies. *The Annals of Applied Probability*, 15(1A):331–391, 2005.

[9] J.-Y. Audibert and R. Munos. Introduction to bandits: Algorithms and theory. In *ICML*, 2011.

[10] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[11] Moshe Babaioff, Shaddin Dughmi, Robert Kleinberg, and Aleksandrs Slivkins. Dynamic pricing with limited supply. *ACM Transactions on Economics and Computation*, 3(1):4, 2015.

[12] Mariagiovanna Baccara, SangMok Lee, and Leeat Yariv. Optimal dynamic matching. *Available at SSRN 2641670*, 2015.

[13] Ashwinkumar Badanidiyuru, Robert Kleinberg, and Yaron Singer. Learning on a budget: posted price mechanisms for online procurement. In *Proceedings of the 13th ACM Conference on Electronic Commerce*, pages 128–145. ACM, 2012.

[14] Ashwinkumar Badanidiyuru, Robert Kleinberg, and Aleksandrs Slivkins. Bandits with knapsacks. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 207–216. IEEE, 2013.

[15] Ashwinkumar Badanidiyuru, John Langford, and Aleksandrs Slivkins. Resourceful contextual bandits. In *Proceedings of The 27th Conference on Learning Theory*, pages 1109–1134, 2014.

[16] Omar Besbes and Assaf Zeevi. Dynamic pricing without knowing the demand function: Risk bounds and near-optimal algorithms. *Operations Research*, 57(6):1407–1420, 2009.

[17] Omar Besbes and Assaf Zeevi. Blind network revenue management. *Operations research*, 60(6):1537–1550, 2012.

[18] Sébastien Bubeck and Nicolo Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Machine Learning*, 5(1):1–122, 2012.

[19] Jim G Dai. On positive harris recurrence of multiclass queueing networks: a unified approach via fluid limit models. *The Annals of Applied Probability*, pages 49–77, 1995.

[20] Ettore Damiano and Ricky Lam. Stability in dynamic matching markets. *Games and Economic Behavior*, 52(1):34–53, 2005.

[21] Sanmay Das and Emir Kamenica. Two-sided bandits and the dating market. In *Proceedings of the 19th international joint conference on Artificial intelligence*, pages 947–952. Morgan Kaufmann Publishers Inc., 2005.

[22] Daniel Fershtman and Alessandro Pavan. Dynamic matching: experimentation and cross subsidization. Technical report, Citeseer, 2015.

[23] John Gittins, Kevin Glazebrook, and Richard Weber. *Multi-armed bandit allocation indices.* John Wiley & Sons, 2011.

[24] Ming Hu and Yun Zhou. Dynamic matching in a two-sided market. *Available at SSRN*, 2015.

[25] Sangram V Kadam and Maciej H Kotowski. Multi-period matching. Technical report, Harvard University, John F. Kennedy School of Government, 2015.

[26] Emilie Kaufmann, Nathaniel Korda, and Rémi Munos. Thompson sampling: An asymptotically optimal finite-time analysis. In *Algorithmic Learning Theory*, pages 199–213. Springer, 2012.

[27] Morimitsu Kurino. Credibility, efficiency, and stability: A theory of dynamic matching markets. 2005.

[28] Tze Leung Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22, 1985.

[29] Constantinos Maglaras and Assaf Zeevi. Pricing and capacity sizing for systems with shared resources: Approximate solutions and scaling relations. *Management Science*, 49(8):1018–1038, 2003.

[30] Constantinos Maglaras and Assaf Zeevi. Pricing and design of differentiated services: Approximate analysis and structural insights. *Operations Research*, 53(2):242–262, 2005.

[31] Laurent Massoulie and Kuang Xu. On the capacity of information processing systems, 2016. Unpublished.

[32] Aranyak Mehta. Online matching and ad allocation. *Theoretical Computer Science*, 8(4): 265–368, 2012.

[33] Daniel Russo and Benjamin Van Roy. Learning to optimize via posterior sampling. *Mathematics of Operations Research*, 39(4):1221–1243, 2014.

[34] Denis Sauré and Assaf Zeevi. Optimal dynamic assortment planning with demand learning. *Manufacturing & Service Operations Management*, 15(3):387–404, 2013.

[35] Lloyd S Shapley and Martin Shubik. The assignment game i: The core. *International Journal of game theory*, 1(1):111–130, 1971.

[36] Adish Singla and Andreas Krause. Truthful incentives in crowdsourcing tasks using regret minimization mechanisms. In *Proceedings of the 22nd international conference on World Wide Web*, pages 1167–1178. International World Wide Web Conferences Steering Committee, 2013.

[37] Zizhuo Wang, Shiming Deng, and Yinyu Ye. Close the gaps: A learning-while-doing algorithm for single-product revenue management problems. *Operations Research*, 62(2): 318–331, 2014.

Appendices

A Proof of Theorem 5.1

For the rest of this section, let C be the quantity defined in (20). We present it again for the convenience of the reader:

$$C(i) = \min_{\alpha \in \Delta(\mathcal{J})} \frac{\sum_{j\in\mathcal{J}} \alpha_j \big(U(i) - [A(i,j) - p^*(j)]\big)}{\min_{i'\in \mathrm{Str}(i)} \sum_{j\in\mathcal{J}} \alpha_j \mathrm{KL}(i,i'|j)}\,; \qquad C = \sum_{i\in\mathcal{I}} \rho(i) C(i)\,.$$

Recall problem 7. We will first show the following lower bound on the difference between V^* and W^N, which follows directly from Agrawal et al. [1].

Proposition A.1.

$$\limsup_{N\to\infty} \frac{N}{\log N}\big(V^* - W^N\big) \geq C.$$

Proof. Consider the following relaxed problem:

$$W^N_{p^*} \;=\; \max_{x\in\mathcal{X}^N} \sum_{i\in\mathcal{I}} \rho(i) \sum_{j\in\mathcal{J}} x(i,j)A(i,j) - \sum_{j\in\mathcal{J}} p^*(j)[\sum_{i\in\mathcal{I}} \rho(i)x(i,j) - \mu(j)]. \tag{23}$$

By a standard duality argument, we know that $W^N_{p^*} \geq W^N$. The optimal policy in this problem is a solution to

$$\mathrm{maximize}_{\pi\in\Pi^N} \sum_{i\in\mathcal{I},\, j\in\mathcal{J}} \rho(i) x_\pi(i,j)(A(i,j) - p^*(j))\,. \tag{24}$$

Then from Theorem 3.1 in Agrawal et al. [1], we know that

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

The result then follows from the fact that $W^N \leq W^N_{p^*}$. □

Let $W^N_{p^*}(\pi^*)$ be the value attained by DEEM in optimization problem (21) (same as (23)), assuming that the routing matrix y in the exploitation phase is such that $y(i,.)$ is supported on

$\mathcal{J}(i)$. We will prove an upper bound on the difference between V^* and $W^N_{p^*}(\pi^*)$. Note that the difference in these values of the two problems is the same as the difference in

$$\sum_{i\in\mathcal{I}}\rho(i)\sum_{j\in\mathcal{J}}x_{\pi^*}(i,j)(A(i,j)-p^*(j)),$$

and $\sum_{i\in\mathcal{I}}\rho(i)U(i)$. Following is the result.

Proposition A.2. *Consider the sequence of policies $(\pi^*(N))_{N\geq 1}$ such that the routing matrix y used in the exploitation phase satisfies $y(i,.)\in\Delta(\mathcal{J}(i))$. Then,*

$$\limsup_{N\to\infty}\frac{N}{\log N}\big(V^*-W^N_{p^*}(\pi^*)\big)\leq C.$$

Further, suppose that there are no difficult type pairs. Then,

$$\limsup_{N\to\infty}\frac{N}{\log\log N}\big(V^*-W^N_{p^*}(\pi^*)\big)\leq K$$

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

In order to prove this Proposition, we need the following result that follows from Theorem 4.1 in [1].

Lemma A.3. *Let $X_1,X_2,\cdots$ be i.i.d. random variables where X_i is the outcome of choosing a job type $j\in\mathcal{J}$ according to a distribution $\alpha\in\Delta(\mathcal{J})$. Suppose $i\in\mathcal{I}$ and $\mathcal{B}\subseteq\mathcal{I}\setminus\{i\}$ are such that*

$$\sum_{j\in\mathcal{J}}\alpha_j\mathrm{KL}(i,i'|j)>0$$

for each $i'\in\mathcal{B}$. Let $\Lambda^{\mathcal{B}}_k(i)=\min_{i'\in\mathcal{B}}\Lambda_k(i,i')$. Then,

1. $$\limsup_{N\to\infty}\frac{\mathrm{E}_i[\inf\{k\geq 0|\Lambda^{\mathcal{B}}_k(i)\geq f(N)\}]}{\log f(N)}\leq\frac{1}{\min_{i'\in\mathcal{B}}\sum_{j\in\mathcal{J}}\alpha_j\mathrm{KL}(i,i'|j)},$$

2. $$\mathrm{P}_{i'}(\Lambda_k(i,i')\geq a \textit{ for some } k\leq N)\leq\frac{\hat\rho(i)/\hat\rho(i')}{a}.$$

for any $a>0$.

Next, we also need the following result.

Lemma A.4. *Let $X^j_1,X^j_2,\cdots$ be i.i.d. random variables for each $j=1,\cdots,K$, such that $|X^j_i|\leq M$ and $\mathrm{E}(X^j_i)=m^j>0$. Let a, b and k^j be such that $a<k^j<b$ for each j. Let $S^j_n=k^j+\sum_{i=1}^n X^j_i$ and let $\mathcal{B}\subseteq\{1,\cdots,K\}$. Let E be the event:*

$$\{S^{j'}_n<a \textit{ for some } j' \textit{ before } S^j_n>b \textit{ for all } j\in\mathcal{B}\}.$$

Let $T=\inf\{n:S^j_n<a$ for some $j\}$. Then

$$\mathrm{E}[T\mathbf{1}_E]\leq G$$

for some $0<G<\infty$ that does not depend on a, b, or k^j for any j.

Proof. Define $k^j - a \triangleq z^j$. If we define $E_j = \{S_n^{j'} < a \text{ for some } n\}$ and $T_j = \inf\{n : S_n^j < a\}$, then we have $E \subseteq \cup_j E_j$, and thus we have $\mathrm{E}[T\mathbf{1}_E] \le \sum_{j=1}^K \mathrm{E}[T\mathbf{1}_{E_j}] \le \sum_{j=1}^K \mathrm{E}[T_j\mathbf{1}_{E_j}]$. Now we have

$$\begin{aligned}
\mathrm{E}[T_j\mathbf{1}_{E_j}] &= \sum_{n=1}^{\infty} n\mathrm{P}(T=n)\\
&\le \sum_{n=1}^{\infty} n\mathrm{P}(\sum_{i=1}^{n} X_i^j \le -z_j)\\
&\le \sum_{n=1}^{\infty} n\exp(\frac{-(nm_j+z_j)^2}{4nM^2})\\
&= \sum_{n=1}^{\infty} n\exp(-\frac{nm_j^2}{4M^2} - \frac{m_j z_j}{2M^2} - \frac{z_j^2}{4nM^2})\\
&\le \sum_{n=1}^{\infty} n\exp(-\frac{nm_j^2}{4M^2}) = G(m^j, M) < \infty,
\end{aligned}$$

where the second inequality results from the Hoeffding bound. Taking $G = \sum_{j=1}^K G(m^j, M)$ proves the result.

$\square$

Proof of Proposition A.2. Let X denote the type of the worker. Let $R(i)$ denote the expected total regret over the lifetime of a worker on the event $\{X = i\}$, defined as

$$R(i) = N\max_{j\in\mathcal{J}}[A(i,j) - p^*(j)] - N\sum_{j\in\mathcal{J}} x_{\pi^*}(i,j)[A(i,j) - p^*(j))].$$

Here $Nx_{\pi^*}(i,j)$ is the expected total number of times a job of type j is allotted to a worker of type i under the policy $\pi^*(N)$. We will refer to the above quantity as just regret. For the rest of the proof, all the expectations are on the event $\{X = i\}$. The proof will utilize the fact that the log of the ratio of the posteriors, $\log(\Lambda_k(i,i'))$, for any i and i' is a random walk, such that if α^k is the probability distribution over job types chosen at opportunity k, then

$$\log(\Lambda_{k+1}(i,i')) - \log(\Lambda_k(i,i')) = \log(\frac{p(X_k, i, j_k)}{p(X_k, i', j_k)}),$$

where the random variables $\{\log(\frac{p(X_k,i,j_k)}{p(X_k,i',j_k)})\}_k$ are independent random variables with a finite support (since X_k and j_k take finite values), and with mean $\sum_j \alpha_j^k \mathrm{KL}(i,i'|j)$. (Note here that if $\sum_j \alpha_j^k \mathrm{KL}(i,i'|j) = 0$ then since $\mathrm{KL}(i,i'|j) \ge 0$, it must be that $\mathrm{KL}(i,i'|j) = 0$ for all j such that $\alpha_j^k > 0$, and in this case we must have $A(i,j) = A(i,j')$ for all such j. Thus $\log(\frac{p(X_k,i,j_k)}{p(X_k,i',j_k)}) = 0$, i.e., the if the drift of the random walk is 0 at some k then the random walk has stopped.) Recall that $\log(\Lambda_0(i,i')) = \log(\hat{\rho}(i)/\hat{\rho}(i'))$.

Our goal is to compute an upper bound on $R(i)$. To do so we first compute the expected regret incurred till the end of the exploration phase in our algorithm. Denote this by $R_e(i)$. Below, we will find an upper bound on this regret assuming that the worker performs an unbounded number of jobs. Clearly the same bound holds on the expected regret until the end of exploration phase if the worker leaves after N jobs.

Our strategy is as follows: we will decompose the regret till the end of exploration into the regret incurred till the first time one of the following two events occurs:

1. Event A: $\min_{i'\neq i} \log \Lambda_k(i,i') \geq \log\log N$ (or $\min_{i'\neq i} \Lambda_k(i,i') \geq \log N$) and
2. Event B: $\min_{i'\neq i} \log \Lambda_k(i,i') \leq -\log\log N$ (or $\min_{i'\neq i} \Lambda_k(i,i') \leq \frac{1}{\log N}$),

followed by the residual regret, which will depend on which event occurred first. Note that one of these two events will occur with probability 1.

We will compute two different upper bounds, depending on two different regimes of initial posterior distributions of the different types (note that the posterior probabilities of the different types i under the observed history is a sufficient statistic at any opportunity under our policy). First, suppose that $\bar{R}(i)$ is highest expected regret incurred over all possible starting posteriors that a) do not satisfy the conditions of both A and B and b) such that $\min_{i'\neq i} \log \Lambda_0(i,i') \geq \min_{i'\neq i} \log(\hat{\rho}(i)/\hat{\rho}(i'))$. Let L_1 be the set of starting posteriors that satisfy these conditions. Next, suppose that $\tilde{R}(i)$ is the highest expected regret incurred, where the supremum is taken over all possible starting posteriors that a) do not satisfy the conditions of both A and B and b) such that $\min_{i'\neq i} \log \Lambda_n(i,i') < \log(\hat{\rho}(i)/\hat{\rho}(i'))$. Let L_2 be the set of posteriors that satisfy these conditions. Clearly, $R_e(i) \leq \bar{R}(i)$.

Let $G(i)$ denote the maximum expected regret incurred by the algorithm till one of A or B occurs, where the maximum is taken over all possible starting posteriors that do not satisfy the conditions of both A and B, i.e., $L_1 \cup L_2$. For convenience, we denote $A < B$ as the event that A occurs before B and vice versa (similarly for any two events). Thus we have

$$\begin{aligned}\bar{R}(i) \leq G(i) &+ \sup_{l_1\in L_1} \mathrm{P}(A < B|l_1)\mathrm{E}(\text{Residual regret}|A, l_1)\\ &+ \sup_{l_1\in L_1} \mathrm{P}(B < A|l_1)\mathrm{E}(\text{Residual regret}|B, l_1)\end{aligned}$$

and

$$\begin{aligned}\tilde{R}(i) \leq G(i) &+ \sup_{l_2\in L_2} \mathrm{P}(A < B|l_2)\mathrm{E}(\text{Residual regret}|A, l_2)\\ &+ \sup_{l_2\in L_2} \mathrm{P}(B < A|l_2)\mathrm{E}(\text{Residual regret}|B, l_2).\end{aligned}$$

First, let us find a bound on $G(i)$. This is easy, because, $G(i) \leq \mathrm{E}(\inf\{k > 0 : \min_{i\neq i'} \Lambda_k(i,i') \geq \log^2 N\}) = \mathrm{O}(\log\log N)$ from Lemma A.3 (since if neither condition A, nor B is satisfied, then the policy in the guessing phase, and thus all job types are utilized with positive probability, and hence the condition in the Lemma of the requirement of a positive learning rate for each distinction is satisfied). Also, from the second statement in Lemma A.3, since the posteriors in L_1 are such that $\min_{i'\neq i} \Lambda_n(i,i') \geq \min_{i'\neq i} \hat{\rho}(i)/\hat{\rho}(i')$, we have that $\mathrm{P}(B < A|l_1) \leq \mathrm{P}(\text{B ever occurs}) \leq \mathrm{O}(1)/\log N$. Finally we have $\sup_{l_2\in L_2} \mathrm{P}(B < A|l_2) = w < 1$. We thus have

$$\bar{R}(i) \leq \mathrm{O}(\log\log N) + \sup_{l_1\in L_1} \mathrm{E}(\text{Residual regret}|A, l_1) + \frac{1}{\log N} \sup_{l_1\in L_1} \mathrm{E}(\text{Residual regret}|B, l_1) \text{ and} \tag{25}$$

$$\tilde{R}(i) \leq \mathrm{O}(\log\log N) + \sup_{l_2\in L_2} \mathrm{E}(\text{Residual regret}|A, l_2) + w \sup_{l_2\in L_2} \mathrm{E}(\text{Residual regret}|B, l_2). \tag{26}$$

Next, consider $\sup_{l_r\in L_r} \mathrm{E}(\text{Residual regret}|A, l_r)$. This depends on which of the following two events happens next:

1. Event A': $\min_{i'\neq i} \log \Lambda_k(i,i') < \log\log N$ (or $\min_{i'\neq i} \Lambda_k(i,i') < \log N$),

2. Event A'': i gets confirmed, i.e., $\min_{i'\in \mathrm{Str}(i)} \log \Lambda_k(i,i') > \log N$ (or $\min_{i'\in \mathrm{Str}(i)} \Lambda_k(i,i') > N$).

Again conditional on A, one of the two events will occur with probability 1. We have

$$\sup_{l_r\in L_r} \mathrm{E}(\text{Residual regret}|A,l_r) = \sup_{l_r\in L_r} [\mathrm{E}(\text{Residual regret}|A,A'<A'',l_r)\mathrm{P}(A'<A''|A,l_r) \\ + \mathrm{E}(\text{Residual regret}|A,A'>A'',l_r)\mathrm{P}(A'>A''|A,l_r)].$$

Now from Lemma A.4 it follows that

$$\mathrm{E}(\text{Residual regret}|A,A'<A'',l_r)\mathrm{P}(A'<A''|A,l_r) \\ = \mathrm{E}(\text{Residual regret } \mathbb{I}_{\{A'<A''\}}|A,l_r) \le M + \bar{R}(i)\mathrm{P}(A'<A''|A,l_r)$$

for some constant M that does not depend on l_r or N. To see this, note that $A'<A''$ is the event that, starting from some values between $\log\log N$ and $\log N$, some random walk $\Lambda_k(i,i')$ for $i'\neq i$ crosses the lower threshold $\log\log N$ before all the random walks $\Lambda_k(i,i')$ for each $i'\in Str(i)$ cross the upper threshold $\log N$. Now between these two thresholds, the job distribution α_k equals $\alpha(i)$ for all k. Hence the drift for any of the random walks $\Lambda_k(i,i')$ for each $i'\in Str(i)$ is strictly positive and finite. Further, as we argued earlier, if the drift for any of these random walks is 0, then that random walk has stopped, and such random walks can be ignored. Thus the conditions of Lemma A.4 are satisfied, and hence $\mathrm{E}((\text{Time till } A')\ \mathbb{I}_{\{A'<A''\}}|A,l_r) = G < \infty$. Since the regret per unit time is bounded, the deduction follows. Moving on, we have

$$\mathrm{E}(\text{Residual regret}|A,A''<A',l_r) \\ \le \mathrm{E}(\inf\{k>0 : \min_{i'\in Str(i)} \Lambda_k(i,i') \ge N\}) \sum_{j\in\mathcal{J}} \alpha_j\big(U(i) - [A(i,j) - p^*(j)]\big).$$

Thus we have

$$\sup_{l_r\in L_r} \mathrm{E}(\text{Residual regret}|A,l_r) \le \mathrm{O}(1) + \sup_{l_r\in L_r} \big[\mathrm{P}(A'<A''|A,l_r)\bar{R}(i) \tag{27}$$

$$+ (1-\mathrm{P}(A'<A''|A,l_r))\mathrm{E}(\inf\{k>0 : \min_{i'\in \mathrm{Str}(i)} \Lambda_k(i,i') \ge N\}) \sum_{j\in\mathcal{J}} \alpha_j\big(U(i) - [A(i,j)-p^*(j)]\big)\big]. \tag{28}$$

Thus we have

$$\sup_{l_r\in L_r} \mathrm{E}(\text{Residual regret}|A,l_r) \le \mathrm{O}(1) + q_k\bar{R}(i) \tag{29}$$

$$+ (1-q_k)\mathrm{E}(\inf\{k>0 : \min_{i'\in \mathrm{Str}(i)} \Lambda_k(i,i') \ge N\}) \sum_{j\in\mathcal{J}} \alpha_j\big(U(i) - [A(i,j)-p^*(j)]\big), \tag{30}$$

for some q_k where $q_k<1$ since $\sup_{l_r\in L_r} \mathrm{P}(A'<A''|A,l_r) < 1$.

Next, consider $\sup_{l_r\in L_r} \mathrm{E}(\text{Residual regret}|B,l_r)$. This depends on which of the following two events occurs next:

1. Event B': $\min_{i'\neq i} \log \Lambda_k(i,i') \ge -\log\log N$ (or $\min_{i'\neq i} \Lambda_k(i,i') \ge \frac{1}{\log N}$),

2. Event B'': Some $i'\neq i$ gets confirmed, i.e., $\min_{i''\in \mathrm{Str}(i')} \log \Lambda_k(i',i'') > \log N$ (or $\min_{i''\in \mathrm{Str}(i')} \Lambda_k(i',i'') > N$).

Again, conditional on B, one of the two events will occur with probability 1. Let $K(i)$ be the maximum expected regret incurred till either B or B' occurs given that B has occurred and the starting likelihoods were in L_r. Note that if $B'' < B'$ then the exploration phase ends and hence there is no residual regret (Although note that if i' is such that $i \in \mathrm{Str}(i')$, then $\mathrm{P}(B'' < B'|B, l_r) \leq \mathrm{O}(1)/N$ from the second statement in Lemma A.3.). Then we have

$$\sup_{l_r \in L_r} \mathrm{E}(\text{Residual regret}|B, l_r) \leq K(i) + \sup_{l_r \in L_r} \mathrm{P}(B' < B''|B, l_r)\tilde{R}(i).$$

Now we can show that if there is a type i' such that $i \in \mathcal{I} \setminus \mathrm{Str}(i')$, then $K(i) \leq \mathrm{O}(\log N)$, where as if there is no such type, then $K(i) = \mathrm{O}(1)$. We first show this. Let $T(i)$ be the maximum expected time taken till either B or B' occurs given that B has occurred and the starting likelihoods were in L_r. Clearly $K(i) \leq T(i)$ since the price adjusted payoffs lie in $[0, 1]$. Now, let T_1 be the time spent after B has occurred, before B or B' occurs, while either a) algorithm is in the guessing phase or b) the algorithm is in the confirmation phase for some guessed type i' such that $\alpha(i')_j > 0$ for some j such that $\mathrm{KL}(i, i'|j) > 0$. Under this case, we will say that the algorithm is in state 1, and let $\mathbf{1}_k$ be the event that the algorithm is in state 2 at time k. Next let T_2 be the time spent after B has occurred, before B or B' occurs, while the algorithm is in the confirmation phase for some guessed type i' such that $\alpha(i')_j = 0$ for all j such that $\mathrm{KL}(i, i'|j) > 0$ (clearly this can happen only for i' such that $i \in \mathcal{I} \setminus \mathrm{Str}(i')$; thus if such an i' doesn't exist, then $T_2 = 0$). Under this case, we will say that the algorithm is in state 2, and let $\mathbf{2}_k$ be the event that the algorithm is in state 2 at time k. Now we clearly have $T(i) \leq \sup_{r \in L_r} \mathrm{E}(T_1|B, l_r) + \sup_{r \in L_r} \mathrm{E}(T_2|B, l_r)$.

Let $\Gamma_k(i) = \min_{i' \neq i} \log \Lambda_k(i, i')$. Then observe that $\mathrm{E}[(\Gamma_{k+1}(i) - \Gamma_k(i))\mathbb{I}_{\mathbf{1}_k}|B, l_r] > m$ for some $m > 0$ that depends only on the primitives of the problem and $\mathrm{E}[(\Gamma_{k+1}(i) - \Gamma_k(i))\mathbb{I}_{\mathbf{2}_k}|B, l_r] = 0$, i.e, when the algorithm is in state 1, the drift in $\Gamma_k(i)$ is strictly positive where as when the algorithm is in state 2, then $\Gamma_k(i)$ does not change.

Now consider $\mathrm{E}(T_1|B, l_r)$. Let k^* be the opportunity that B occurred for the first time. Then clearly $\Gamma_{k^*}(i) = -\log\log N - \varepsilon$, where $\varepsilon \geq 0$ is bounded by some constant M depending only on the problem instance. Thus $\mathrm{P}(T_1 > t|B, l_r) \leq \mathrm{P}(\Gamma_{k'}(i) - \Gamma_{k^*}(i) \leq \varepsilon|B, l_r)$ for some $k' > k^*$ such that the algorithm has been in state 1 t times at opportunity k'. Thus our observation above implies that $\mathrm{P}(T_1 > t|B, l_r) \leq \exp(-ct)$ for some $c > 0$, by a standard application of a concentration inequality. Thus $\mathrm{E}(T_1|B, l_r) = \mathrm{O}(1)$.

Next consider $\mathrm{E}(T_2|B, l_r)$. Consider the successive returns of the algorithm to state 2. Conditional on the algorithm having entered state 2, the expected time spent in that state is bounded by the expected time till the guessed type i' is confirmed, which is $\mathrm{O}(\log N)$ from Lemma A.3, and the conditional probability that i' gets confirmed is some $q > 0$. Thus the total expected number of returns to state 2 is bounded by $1/q$. Thus $\mathrm{E}(T_2|B, l_r) = \mathrm{O}(\log N)$ as well. Thus $K(i) \leq O(\log N)$ and we have

$$\sup_{l_r \in L_r} \mathrm{E}(\text{Residual regret}|B, l_r) \leq \mathrm{O}(\log N) + \tilde{R}(i).$$

And thus we finally have

$$\bar{R}(i) \leq \mathrm{O}(\log\log N) + q_1\bar{R}(i) + \frac{1}{\log N}\big(\mathrm{O}(\log(N) + \tilde{R}(i)\big)$$
$$+ (1-q_1)\mathrm{E}(\inf\{k>0 : \min_{i'\in\mathrm{Str}(i)} \Lambda_k(i,i') \geq N\}) \sum_{j\in\mathcal{J}} \alpha_j\big(U(i) - [A(i,j) - p^*(j)]\big); \tag{31}$$
$$\tilde{R}(i) \leq \mathrm{O}(\log\log N) + q_2\bar{R}(i) + w\mathrm{O}(\log N) + w\tilde{R}(i)$$
$$+ (1-q_2)\mathrm{E}(\inf\{k>0 : \min_{i'\in\mathrm{Str}(i)} \Lambda_k(i,i') \geq N\}) \sum_{j\in\mathcal{J}} \alpha_j\big(U(i) - [A(i,j) - p^*(j)]\big). \tag{32}$$

Combining the above two equations, we deduce that

$$R_e(i) \leq \bar{R}(i) \leq \frac{1-q_1}{1-q_1-q_2/\log N}\big(\mathrm{O}(\log\log N)$$
$$+ \mathrm{E}[\inf\{k>0 : \min_{i'\in Str(i)} \Lambda_n(i,i') \geq N\}] \sum_{j\in\mathcal{J}} \alpha_j\big(U(i) - [A(i,j) - p^*(j)]\big)\big)$$
$$= \mathrm{O}(\log\log N)$$
$$+ (1+o(1))\mathrm{E}[\inf\{k>0 : \min_{i'\in\mathrm{Str}(i)} \Lambda_n(i,i') \geq N\}] \sum_{j\in\mathcal{J}} \alpha_j\big(U(i) - [A(i,j) - p^*(j)]\big). \tag{33}$$

Now, we observed earlier that $\mathrm{P}(i'$ gets confirmed $\mid \{X=i\}) \leq 1/N$ if $i' \in \mathrm{Str}(i)$. Thus the regret in the exploitation phase is in the worst case of order $\mathrm{O}(N)$ with probability $1/N$ and 0 otherwise. Thus the total expected regret in the exploitation phase is $\mathrm{O}(1)$. Thus

$$R(i) \leq \mathrm{O}(\log\log N) + (1+o(1))\mathrm{E}[\inf\{k>0 : \min_{i'\in\mathrm{Str}(i)} \Lambda_k(i,i') \geq N\}] \sum_{j\in\mathcal{J}} \alpha_j\big(U(i) - [A(i,j) - p^*(j)]\big).$$

Thus Lemma A.3 implies the result (note that if there are no difficult type pairs, then $\sum_{j\in\mathcal{J}} \alpha_j\big(U(i) - [A(i,j) - p^*(j)] = 0\big)$.

□

Next, we prove that for a large enough N, one can choose a routing matrix y^* in the exploitation phase of DEEM that will ensure that matches optimize price-adjusted payoffs, and such that the capacity and complementary slackness conditions are satisfied.

Proposition A.5. *Suppose that the generalized imbalance condition is satisfied. Consider any optimal routing matrix x^* that is an optimal solution to problem (11). Then in the policy $\pi^*(N)$ for the N-job problem, for any N large enough, one can choose a routing matrix y^* such that $y^*(i,.) \in \Delta(\mathcal{J}(i))$ and that satisfies:*

1. *$\sum_{i\in\mathcal{I}} \rho(i)x_{\pi^*}(i,j) = \mu(j)$ for any j such that $\sum_{i\in\mathcal{I}} \rho(i)x^*(i,j) = \mu(j)$, and*
2. *$\sum_{i\in\mathcal{I}} \rho(i)x_{\pi^*}(i,j) < \mu(j)$ for any other j.*

We remark that the y^* we construct satisfies $\|y^* - x^*\| = o(1)$. In order to prove this proposition, we will need the following Lemma.

Lemma A.6. *Suppose that the generalized imbalance condition is satisfied. Consider any feasible routing matrix $[x(i,j)]_{\mathcal{I}\times\mathcal{J}}$. Consider any job j such that $\sum_{i\in\mathcal{I}} \rho(i)x(i,j) = \mu(j)$. Then there is a path on the complete bipartite graph between worker types $\mathcal{I}$ and job types $\mathcal{J}$ with the following properties:*

- *One end point is job j.*
- *The other end point is a job type whose capacity is under-utilized (it is permitted to be κ).*
- *For every job type on the path in between, they are operating at capacity/all jobs are being served. (All worker types are fully utilized by definition, since we formally consider an unassigned worker as being assigned to job type κ.)*
- *For every undirected edge on the path, there is a positive rate of jobs routed on that edge in x.*

Proof. Consider a bi-partite graph with jobs representing nodes on one side and workers on the other. There is an edge between a job j' and a worker i if $x(i, j) > 0$. Consider the connected component of job type j in this graph. Suppose it includes no job type that is underutilized. Then the arrival rate of jobs from the set of workers in the connected component exactly matches the total effective service rate of the sellers in connected component. But this is a contradiction since generalized imbalance holds. Hence there exists an underutilized job type j' that can be reached from j. Take any path from j to j'. Traverse it starting from j and terminate it the first time it hits any underutilized job type. □

Proof of Proposition A.5. Recall that for a given routing matrix $[y(i, j)]_{\mathcal{I}\times\mathcal{J}}$, $x_{\pi^*}(i, j)$, is the resulting fraction of jobs of type j directed to worker type i. In the course of this proof, we will suppress the subscript π^*. Clearly, there exist $\varepsilon_{i'}(i, j)$ for each $i \in \mathcal{I}, i' \in \mathcal{I} \cup \{0\}, j \in \mathcal{J}$ such that we have

$$x(i, j) = \varepsilon_0(i, j) + (1 - \varepsilon_i(i, j))y(i, j) + \sum_{i' \in \mathcal{I}\setminus\{i\}} \varepsilon_{i'}(i, j)y(i', j)\,. \tag{34}$$

The ε's depend on the guessing and confirmation phases but not on y. (In particular, ε_0 arises from the overall routing contribution of the guessing and confirmation phases, and ε_i's arise from the small likelihood that a worker who is confirmed as type i is actually some other type.) A key fact that we will use is that all ε's are uniformly bounded by $o(1)$.

Let $\mathcal{J}_{x^*} = \{s : \sum_{i\in\mathcal{I}} \rho(i)x^*(i, j) = \mu(j)\}$ and $\mathcal{J}_{\pi^*} = \{j : \sum_{i\in\mathcal{I}} \rho(i)x_{\pi^*}(i, j) = \mu(j)\}$. Now we want to find a y such that $y(i, \cdot) \in \Delta(\mathcal{J}(i))$ for all $i \in \mathcal{I}$ (call (i, j) a "permissible edge" in the bipartite graph between workers and jobs if $j \in \mathcal{J}(i)$), and such that:

- For each $j \in \mathcal{J}_{x^*}$ we also have $j \in \mathcal{J}_{\pi^*}$, i.e., $\mathcal{J}_{x^*} \subseteq \mathcal{J}_{\pi^*}$.
- $\|y - x^*\| = o(1)$.

Note that the two bullets together will imply the proposition, since $\|x-x^*\| = o(1)$ from Eq. (34), and this leads to $\sum_{i\in\mathcal{I}} \rho(i)x(i, j) = \sum_{i\in\mathcal{I}} \rho(i)x^*(i, j) + o(1) < \mu(j)$ for all $j \in \mathcal{J}\setminus\mathcal{J}_{x^*}$, for large enough N.

The requirement in the first bullet can be written as a set of linear equations using Eq. (34). Here we write y (and later also x^*) as a column vector with $|\mathcal{I}||\mathcal{J}|$ elements:

$$By + \hat{\varepsilon} = (\mu(j))_{j\in\mathcal{J}_{x^*}}\,.$$

Here we have $\|\hat{\varepsilon}\| = o(1)$ and matrix B can be written as $B = B_0 + B_\varepsilon$, where B_0 has 1's in columns corresponding to dimensions $(\cdot, s)$ and 0's everywhere else, and $\|B_\varepsilon\| = o(1)$. Expressing y as $y = x + z$, we are left with the following equation for z,

$$Bz = -(B_\varepsilon x^* + \hat{\varepsilon}) \tag{35}$$

using the fact that $B_0 x^* = (\mu(j))_{j\in\mathcal{J}_{x^*}}$ by definitions of B_0 and $\mathcal{J}_{x^*}$. We will look for a solution to this underdetermined set of equations with a specific structure: we want z to be a linear combination of flows along $|\mathcal{J}_{x^*}|$ paths coming from Lemma A.6, one path λ_j for each $j \in \mathcal{J}_{x^*}$. Each λ_j can be written as a column vector with $+1$'s on the odd edges (including the edge incident on j) and -1's on the even edges. Let $\Lambda = [\lambda_j]_{j\in\mathcal{J}_{x^*}}$ be the path matrix. Then z with the desired structure can be expressed as $\Lambda\eta$, where η is the vector of flows along each of the paths. Now note that $Bz = (B_0 + B_\varepsilon)\Lambda\eta = (I + B_\varepsilon\Lambda)\eta$. Here we deduced $B_0\Lambda = I$ from the fact that λ_j is a path which has j as one end point, and a worker or else a job not in $\mathcal{J}_{x^*}$ as the other end point. Our system of equations reduces to

$$(I + B_\varepsilon\Lambda)\eta = -(B_\varepsilon x^* + \hat{\varepsilon})\,,$$

Since $\|B_\varepsilon\| = o(1)$, the coefficient matrix is extremely well behaved being $o(1)$ different from the identity, and we deduce that this system of equations has a unique solution η^* that satisfies $\|\eta^*\| = o(1)$. This yields us $z^* = \Lambda\eta^*$ that is also of size $o(1)$, and supported on permissible edges since each of the paths is supported on permissible edges (Lemma A.6). Thus, we finally obtain $y^* = x^* + z^*$ possessing all the desired properties. Notice that the (permissible) edges on which y^* differs from x^* had strictly positive values in x^* by Lemma A.6, and hence this is also the case in y^* for large enough N.

□

Finally, we show that with the choice of y^* constructed in Proposition A.5 in the exploitation phase, the sequence of policies $(\pi^*(N))$ asymptotically achieve the required upper bound on regret.

Proposition A.7. *Suppose that the generalized imbalance condition is satisfied. Consider the sequence of policies $(\pi^*(N))_{N\geq 1}$, with the routing matrix y^* proposed in Proposition A.5. Let $W^N(\pi^*)$ be the value attained by this policy in optimization problem* (7)*. Then*

$$\limsup_{N\to\infty} \frac{N}{\log N}\big(V^* - W^N(\pi^*)\big) \leq C.$$

Further, suppose that there are no difficult type pairs. Then,

$$\limsup_{N\to\infty} \frac{N}{\log\log N}\big(V^* - W^N(\pi^*)\big) \leq K$$

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

Proof. From Proposition A.5 it follows that the policy π^* is feasible in problem 7, and further

$$\begin{aligned} W^N_{p^*}(\pi^*) &= \sum_{i\in\mathcal{I}}\rho(i)\sum_{j\in\mathcal{J}} x_{\pi^*}(i,j)A(i,j) - \sum_{j\in\mathcal{J}} p^*(j)[\sum_{i\in\mathcal{I}}\rho(i)x_{\pi^*}(i,j) - \mu(j)]. && (36)\\ &= \sum_{i\in\mathcal{I}}\rho(i)\sum_{j\in\mathcal{J}} x_{\pi^*}(i,j)A(i,j), && (37)\end{aligned}$$

where the second equality follows from the fact that if $p^*(j) > 0$, then $\sum_{i\in\mathcal{I}}\rho(i)x^*(i,j) - \mu(j) = 0$ by complementary slackness, and hence from Proposition A.5 we obtain that $\sum_{i\in\mathcal{I}}\rho(i)x_{\pi^*}(i,j) - \mu(j) = 0$ as well for these j. Thus we have a policy π^* that is feasible, and that gives a rate $W^N_{p^*}(\pi^*)$ of accumulation of payoff in problem (7). Thus the result follows from Proposition A.2.
□

B Computation of the policy in the confirmation subphase

Denoting $\min_{i' \in \mathrm{Str}(i)} \sum_{j \in \mathcal{J}} \alpha_j \mathrm{KL}(i, i'|j)$ as h (where h is non-negative), the optimization problem is the same as:

$$\begin{aligned}
\min & \sum_{j \in \mathcal{J}} \frac{\alpha_j}{h} \big(U(i) - [A(i,j) - p^*(j)]\big) \\
\text{s.t.} & \sum_{j \in \mathcal{J}} \frac{\alpha_j}{h} \mathrm{KL}(i, i'|j) \geq 1 \text{ for all } i' \in \mathrm{Str}(i), \\
& \sum_{j \in \mathcal{J}} \frac{\alpha_j}{h} = \frac{1}{h}, \ \frac{1}{h} \geq 0 \text{ and } \frac{\alpha_j}{h} \geq 0 \text{ for all } j.
\end{aligned}$$

Now redefine $\frac{\alpha_j}{h} \triangleq \bar{\alpha}_j$ and $\frac{1}{h} \triangleq \bar{h}$ to obtain the linear program:

$$\begin{aligned}
\min & \sum_{j \in \mathcal{J}} \bar{\alpha}_j \big(U(i) - [A(i,j) - p^*(j)]\big) \\
\text{s.t.} & \sum_{j \in \mathcal{J}} \bar{\alpha}_j \mathrm{KL}(i, i'|j) \geq 1 \text{ for all } i' \in \mathrm{Str}(i), \\
& \bar{\alpha}_j \geq 0 \text{ for all } j,
\end{aligned}$$

where at any optimal solution $\bar{\alpha}^*$, we have $\bar{h}^* = \sum_{j \in \mathcal{J}} \bar{\alpha}_j^*$, and thus $\alpha(i)_j = \bar{\alpha}_j^* / \sum_{j \in \mathcal{J}} \bar{\alpha}_j^*$. Note that a feasible solution exists to this linear program as long as $\mathrm{KL}(i, i'|j) > 0$ for some j for each i'. When there are multiple solutions, we choose the solution with the largest learning rate; i.e., we choose a solution with the smallest $\bar{h}^*$, i.e., $\sum_{j \in \mathcal{J}} \bar{\alpha}^*$. One way to accomplish this is to modify the objective to minimize $\sum_{j \in \mathcal{J}} \bar{\alpha}_j \big(U(i) - [A(i,j) - p^*(j)] + \tau \sum_{j \in \mathcal{J}} \bar{\alpha}_j$ for some small $\tau > 0$.

For small problems, we can simply evaluate all the finite extreme points of the constrained set $\{\bar{\alpha} : \sum_{j \in \mathcal{J}} \bar{\alpha}_j \mathrm{KL}(i, i'|j) \geq 1 \text{ for all } i' \in \mathrm{Str}(i); \ \bar{\alpha}_j \geq 0 \text{ for all } j\}$; i.e., all the extreme points such that $\bar{\alpha}_j < \infty$ for all j. This is sufficient because we can show that there always exists a finite solution to the linear program. To see this, note that

$$\bar{\alpha}_j = \bar{\alpha}_j^m \triangleq \sum_{i' \in \mathrm{Str}(i)} \frac{1}{\mathrm{KL}(i, i'|j)} \mathbb{I}_{\{\mathrm{KL}(i,i'|j) > 0\}}$$

is feasible and finite. Further, for any solution such that $\alpha_j^* > \bar{\alpha}_j^m$, α_j^* can be reduced to $\bar{\alpha}_j^m$ without loss in objective while maintaining feasibility.

For the practical heuristic DEEM$^+$, α^k can be computed as a solution to the following optimization problem.

$$\begin{aligned}
\min & \sum_{i} g_k(i) \sum_{j \in \mathcal{J}} \frac{\alpha_j}{h} \big(U(i) - [A(i,j) - p^*(j)]\big) \\
\text{s.t.} & \sum_{j \in \mathcal{J}} \frac{\alpha_j}{h} \mathrm{KL}(i, i'|j) \geq g_k(i) \left(1 + \log R(i, i') / \log N\right) \text{ for all } i \in \mathcal{I}, \ i' \in \mathcal{L}_k(i), \\
& \sum_{j \in \mathcal{J}} \frac{\alpha_j}{h} = \frac{1}{h}, \ \frac{1}{h} \geq 0 \text{ and } \frac{\alpha_j}{h} \geq 0 \text{ for all } j.
\end{aligned}$$

We can again redefine $\frac{\alpha_j}{h} \triangleq \bar{\alpha}_j$ and $\frac{1}{h} \triangleq \bar{h}$ to obtain a linear program.

C Practical implementation of other policies

1. **PA-UCB:** The upper confidence bound (UCB) algorithm is a popular multi-armed bandit algorithm [9, 10, 18], that embodies the well known approach of "optimism in the face of uncertainty" to solving these problems. In its classical implementation, one keeps track of high-probability confidence intervals for the expected payoffs of each arm, and at each step chooses an arm that has the highest upper confidence bound, i.e., the highest upper boundary of the confidence interval. To be precise, if $\overline{r}_j(t)$ is the average reward seen for some arm j, that has been pulled n_j times until time t, then the upper confidence bound for the mean reward for this arm is given by:

$$u_j(t) = \overline{r}_j(t) + \sqrt{2\log t/n_j}.$$

The algorithm chooses arm $j = \arg\max_j \mu_j(t)$. In our context, the arms are the job types, and if k jobs have already been allotted to a worker, and $\overline{r}_j(k)$ is the average payoff obtained from past assignments of job j, and n_j is the number of these assignments, we will define

$$u_j(k) = \overline{r}_j(k) + \sqrt{2\log k/n_j} - p_q(j),$$

where $p_q(j)$ is the current queue length based price for job j. The PA-UCB algorithm then chooses job type $j = \arg\max_{j\in\mathcal{J}} \mu_j$ to be assigned to the worker next. Note that this algorithm does not require the knowledge of the instance primitives $(\hat{\rho}, N, \mu, A)$.

2. **Thompson sampling (PA-TS):** Thompson sampling is another popular multi-armed bandit algorithm [9, 18] employing a Bayesian approach to the problem of arm selection. The description of the algorithm is simple: starting with a prior, at every step select an arm with a probability equal to the posterior probability of that arm being optimal. These posterior probabilities are updated based on observations made at each step. One can incorporate information about correlation between the rewards of the different arms in computing these posteriors, which makes it a versatile algorithm that exploits the reward structure in multiple settings. It is known to give asymptotically tight regret guarantees in many multi-armed bandit problems of interest [4, 26, 33].

In our simulations, a price-adjusted version of TS is implemented as follows. The prior probability of the worker being of type i is $\frac{\hat{\rho}(i)}{\sum_{i'}\hat{\rho}(i')}$. With this as the prior, depending on each worker's history, a posterior distribution of the type of the worker is computed using the knowledge of the expected payoff matrix A. Then a worker type is sampled from this distribution. Suppose this type is i, then a job type $j^* = \arg\max_j A(i,j) - p_q(j)$ is assigned to the worker. In contrast to the PA-UCB algorithm, Thompson sampling does utilize the knowledge of the expected payoff matrix A as well as the arrival rates $\hat{\rho}$ (the latter to construct a starting prior, and the former for the posterior updates).

D Other proofs

D.1 Sufficiency of worker-history-only policies

We show that there is a worker-history-only (WHO) policy that achieves a rate of payoff accumulation that is arbitrarily close to the maximum possible. We will think of N as being fixed throughout this section.

Suppose the system starts at time $t = 1$ with no workers already present before the period.[21] Arrivals thereafter occur as described in Section 3. Consider any (arbitrary, time varying) policy π and let $x_{\pi,t}(i,j)$ denote the derived quantity representing the fraction of workers of type i who are assigned jobs on type j in period t under π. Then the largest possible rate of payoff accumulation under policy π over long horizons is

$$\overline{V}(\pi) = \text{limsup}_{T\to\infty} V_T(\pi) \tag{38}$$

$$\text{where } V_T(\pi) = \frac{1}{T}\sum_{t=1}^{T}\sum_{i\in\mathcal{I}}\rho(i)\sum_{j\in\mathcal{J}} x_{\pi,t}(i,j)A(i,j)\,. \tag{39}$$

Note that we have ignored the effect of less than $\rho(i)$ workers of type i being present for the first N periods, but this does not change the limiting value $\overline{V}$. Also, note that randomization in π cannot increase the achievable value of $\overline{V}$, since one can always do as well by picking the most favorable sample path.

Claim D.1. *Fix any policy π and any $\varepsilon > 0$. Then there is a worker-history-only (WHO) policy that achieves a steady state rate of payoff accumulation exceeding $\overline{V}(\pi) - \varepsilon$.*

Proof. We suppress dependence on π. By definition of $\overline{V}$, we know that there exists an increasing sequence of times $T_1, T_2, \ldots$ such that $V_{T_i} > \overline{V} - \varepsilon/2$ for all $i = 1, 2, \ldots$. We will construct a suitable WHO policy by using a sufficiently large time in this sequence. Let $\nu_t(H_k)$ be the measure of workers in the system with history H_k just before the start of time t, and abusing notation, let $\nu_t(H_k)_j$ be the measure of such workers who are assigned to job type j at time t. Since the policy cannot assign more jobs than have arrived in any period, we have

$$\sum_{k=1}^{N}\sum_{H_k}\nu_t(H_k)_j \le \mu(j) \qquad \text{for all } t \ge 1\,. \tag{40}$$

Fix T, which we think of as a large member of the sequence above. The average measure of workers with history H_k who are present is

$$\bar{\nu}(H_k) = \frac{1}{T}\sum_{t=1}^{T}\nu_t(H_k) \qquad \text{for all } H_k \text{ and } k = 1, 2, \ldots, N\,. \tag{41}$$

The average measure of such workers who are assigned job j is similarly defined and denoted by $\bar{\nu}(H_k, j)$. We immediately have that

$$\sum_{k=1}^{N}\sum_{H_k}\bar{\nu}(H_k)_j \le \mu(j)\,, \tag{42}$$

by averaging Eq. (40) over times until T. Now, consider a worker with history H_k assigned a job of type j. Using the known A matrix and arrival rates ρ, we can infer the posterior distribution of the worker type based on H_k, and hence, the likelihood of the job of type j being successfully completed. Let $p(H_k, j)$ denote the probability of success. Then the distribution of H_{k+1} for the worker is simply given by

$$H_{k+1} = \big(\, H_k, \big(j, \text{Bernoulli}(p(H_k, j))\big)\,\big)\,.$$

[21] The analysis would be very similar and produce the same results if the starting state is an arbitrary one with a bounded mass of workers already present.

Barring the edge effect at time T caused by workers whose history was H_k at time T, this allows us to uniquely determine $\bar{\nu}(H_{k+1})$ based on $\bar{\nu}(H_k, j)$'s. In particular, for any $\delta_1 > 0$, if $T \geq \max_{i \in \mathcal{I}} \rho(i)/(N\delta_1)$ we have that

$$\begin{aligned}\bar{\nu}(H_k, (j,1)) &\overset{\delta_1}{\approx} \bar{\nu}(H_k)_j p(H_k, j) \\ \bar{\nu}(H_k, (j,0)) &\overset{\delta_1}{\approx} \bar{\nu}(H_k)_j \left(1 - p(H_k, j)\right).\end{aligned} \tag{43}$$

Here, $a \overset{\delta}{\approx} b$ represents the bound $|a - b| \leq \delta$. Note that we have

$$V_T = \sum_{k=1}^{N} \sum_{H_k} \bar{\nu}(H_k)_j p(H_k, j). \tag{44}$$

We are now ready to define our WHO policy $\underline{\pi}$. For every H_k such that $\bar{\nu}(H_k) \geq \delta_2$, this policy will attempt to assign a fraction $\bar{\nu}(H_k)_j/\bar{\nu}(H_k)$ of workers with history H_k to jobs of type j. Ignore capacity constraints for the present. (We will find that the capacity constraints will be almost satisfied.) Leave the choice of δ_2 for later; we will choose δ_2 small and then choose $0 < \delta_1 < \delta_2/N$ so as to achieve the desired value of δ_3 below. Workers with *rare* histories, i.e., histories such that $\bar{\nu}(H_k) < \delta_2$, will not be assigned jobs under $\underline{\pi}$. Note that the definition of rare histories refers to frequency of occurrence under π. Then, this uniquely specifies $\underline{\pi}$ as well as the steady state mix of workers before any time t. In particular, the steady state mass $\underline{\nu}(H_k)$ under $\underline{\pi}$ of workers with history H_k that are not rare is bounded as

$$(1 - \delta_1/\delta_2)^{k-1} \bar{\nu}(H_k) \leq \underline{\nu}(H_k) \leq (1 + \delta_1/\delta_2)^{k-1} \bar{\nu}(H_k) \tag{45}$$

using Eq. (43), and the fact that all subhistories of H_k are also not rare. It follows that

$$\underline{\nu}(H_k) \overset{\delta_3}{\approx} \bar{\nu}(H_k) \quad \text{where } \delta_3 = \max(\exp(N\delta_1/\delta_2) - 1, \delta_2), \tag{46}$$

for all histories (including rare histories), using $k \leq N$ and $\nu(H_k) \leq 1$. Violation of the j-capacity constraint under $\underline{\pi}$ is given by

$$\left(\sum_{k=1}^{N} \sum_{H_k} \underline{\nu}(H_k)_j - \mu(j) \right)_+ \leq \left(\sum_{k=1}^{N} \sum_{H_k} \bar{\nu}(H_k)_j - \mu(j) \right)_+ + 2^N |\mathcal{J}|^{N-1} \delta_3 = 2^N |\mathcal{J}|^{N-1} \delta_3$$

using Eq. (46) and Eq. (42), and the fact that there are $\sum_{k \leq N} (2|\mathcal{J}|)^{N-1} \leq 2^N |\mathcal{J}|^{N-1}$ possible histories. It follows that the sum of capacity constraint violations across $j \in \mathcal{J}$ is bounded by $(2|\mathcal{J}|)^N \delta_3$. Pick an arbitrary set of workers to go unmatched to get rid of any capacity violations (this can be done while remaining within the class of WHO policies). In worst case, this will cause payoff loss of 1 for each period remaining in the worker's lifetime. Thus, the loss caused by the need to remedy capacity violations is bounded by $\delta_4 = N(2|\mathcal{J}|)^N \delta_3$ per period.

Ignoring capacity violations, the steady state rate of accumulation of payoff under $\underline{\pi}$ is

$$\sum_{k=1}^{N} \sum_{H_k} \underline{\nu}(H_k)_j p(H_k, j) \overset{\delta_5}{\approx} \sum_{k=1}^{N} \sum_{H_k} \bar{\nu}(H_k)_j p(H_k, j) = V_T(\pi)$$
$$\text{where } \delta_5 = 2^N |\mathcal{J}|^{N-1} \delta_3 < \delta_4. \tag{47}$$

again using Eq. (46) and the fact that there are $\sum_{k\le N}(2|\mathcal{J}|)^{N-1} \le 2^N|\mathcal{J}|^{N-1}$ possible histories.

Let $V(\underline{\pi})$ denote the true steady state rate of accumulation of payoff under $\underline{\pi}$ when capacity constraints are considered. Combining the above, we deduce that $V(\underline{\pi}) \ge V_T(\pi) - 2\delta_4$. The time T will be chosen as a member of the sequence defined at the beginning of the proof, ensuring $V_T(\pi) \ge \overline{V}(\pi) - \varepsilon/2$; hence it will suffice to show $V(\underline{\pi}) \ge V_T(\pi) - \varepsilon/2$. Hence, it suffices to have $\delta_4 = \varepsilon/4$, which can achieved using $\delta_3 = \delta_2 = \varepsilon/(4N(2|\mathcal{J}|)^N)$ and $\delta_1 = \delta_3 \log(1+\delta_3)/N$ and T a member of the sequence satisfying $T \ge \max_{i\in\mathcal{I}} \rho(i)/(N\delta_1)$. This yields the required bound of $V(\underline{\pi}) \ge \overline{V}(\pi) - \varepsilon$. □

D.2 Uniqueness of prices under generalized imbalance

Proposition D.2. *Under the generalized imbalance condition, the job shadow prices p^* are uniquely determined.*

Proof of Proposition D.2. The dual to problem 11 can be written as

$$\begin{aligned} &\text{minimize} \sum_{j\in\mathcal{J}} \mu(j)p(j) + \sum_{i\in\mathcal{I}} \rho(i)v(i) \\ &\text{subject to} \\ &p(j) + v(i) \ge A(i,j) \quad \forall i \in \mathcal{I}, j \in \mathcal{J}\,, \\ &\qquad p(j) \ge 0 \quad \forall j \in \mathcal{J}\,, \\ &\qquad v(i) \ge 0 \quad \forall i \in \mathcal{I}\,. \end{aligned}$$

The dual variables are (P,V) where "job prices" $P = (p(j))_{j\in\mathcal{J}}$ and "worker values" $V = (v(i))_{i\in\mathcal{I}}$. We will prove the result by contradiction. Suppose there are multiple dual optima. Let D be the set of dual optima. Let $\mathcal{J}'$ be the set of jobs such that the prices of those jobs take multiple values in D. Formally,

$$\mathcal{J}' = \{j \in \mathcal{J} : p(j) \text{ takes multiple values in } D\}\,. \tag{48}$$

Similarly, let $\mathcal{I}'$ be the set of workers such that the prices of those workers take multiple values in D. Formally,

$$\mathcal{I}' = \{i \in \mathcal{I} : v(i) \text{ takes multiple values in } D\}\,. \tag{49}$$

For each $j \in \mathcal{J}'$, we immediately deduce that there exists a dual optimum with $p(j) > 0$, and hence the capacity constraint of job type j is tight in all primal optima. Similarly, we deduce that for each $i \in \mathcal{I}'$, worker type i is assigned a job in all periods, i.e., $\sum_{j\in\mathcal{J}} x(i,j) = 1$. By assumption, we have

$$\sum_{i\in\mathcal{I}'} \rho(i) \ne \sum_{j\in\mathcal{J}'} \mu(j)\,.$$

Suppose the left hand side is larger than the right (the complementary case can be dealt with similarly). Take any primal optimum x^*. The jobs in $\mathcal{J}'$ do not have enough capacity to serve all workers in $\mathcal{I}'$, hence there must be some worker $i \in \mathcal{I}'$ and a job $s \notin \mathcal{J}'$ such that $x^*(i,s) > 0$. Since $s \notin \mathcal{J}'$, we must have that $p(j)$ has a unique optimum value in D. Call this value $p^*(j)$. Let the largest and smallest values of $v(i)$ in D be $v^{\max}(i)$ and $v^{\min}(i)$. By complementary slackness, we know that

$$\begin{aligned} v^{\max}(i) + p^*(j) &= A(i,j) = v^{\min}(i) + p^*(j) \\ \Rightarrow v^{\max}(i) &= v^{\min}(i)\,. \end{aligned}$$

But since $i \in \mathcal{I}'$ we must have $v^{\max}(i) > v^{\min}(i)$. Thus we have obtained a contradiction.

□

The proof of the next proposition shows that a very simple "learn then exploit" strategy achieves a regret of $\mathrm{O}(\log N/N)$. This follows from the fact that, under an identifiability condition, the sequence of sets $(\mathcal{X}^N)$ converges to the set $\mathcal{D}$ in an appropriately defined distance.

Proposition D.3. *Suppose that no two rows in A are identical. Then* $\sup_{x\in\mathcal{D}} \inf_{y\in\mathcal{X}^N} \|x-y\| = \mathrm{O}\Big(\frac{\log N}{N}\Big)$

Proof. It is clear that $x^N \subseteq \mathcal{D}$. We will find an inner approximation $\tilde{\mathcal{X}}^N$ to $\mathcal{X}^N$ such that $\tilde{\mathcal{X}}^N \subseteq \mathcal{X}^N$, and $\tilde{\mathcal{X}}^N$ converges to $\mathcal{D}$ in an appropriate sense as N goes to infinity. To define this approximation, suppose that in the learning problem corresponding to a fixed N, one starts off with a exploration phase of a fixed length $\mathrm{O}(\log N)$, where each job j is presented to the worker O_s number of times (where $O_s = \mathrm{O}(\log N)$, fixed a priori), so that after this phase, the type of the worker becomes known with a probability of error at most $\mathrm{O}(1/N)$. This will then allow us to relate the problem to the problem in which the user type is known.
Suppose after this phase, the probability that a worker of type i is correctly identified is $p(i)$ and the probability that she is mis-identified as some other type i' is $p(i,i')$. Note that since no two rows in A are identical, $p(i,i') = \mathrm{O}(1/N)$ for all $i \neq i'$. Let $d(i,j)$ denote the expected number of times a worker that has been identified as being of type i (correctly or incorrectly) is directed towards job j after the exploration phase, i.e., from job $O_s + 1$ till the N^{th} job. Let $\bar{d}(i,j) = d(i,j)/N$. Then we can see that, one can attain all x in the following set:

$$\tilde{\mathcal{X}}^N = \Big\{x \in \mathbb{R}^{|\mathcal{I}|\times|\mathcal{J}|} : \bar{d}(i,j) \geq 0;\ \sum_{j\in\mathcal{J}} x(ib,s) = 1; \tag{50}$$

$$x(i,j) = \frac{O_s}{N} + p(i)\bar{d}(i,j) + \sum_{i'\neq i} p(i,i')\bar{d}(i',s)\Big\} \tag{51}$$

Now since $\bar{d}(i,j) \leq 1$, and since $p(i,i') \leq \mathrm{O}(1/N)$, we can express the above set as:

$$\tilde{\mathcal{X}}^N = \Big\{x \in \mathbb{R}^{|\mathcal{I}|\times|\mathcal{J}|} : \bar{d}(i,j) \geq 0;\ \sum_{j\in\mathcal{J}} x(i,j) = 1;\ x(i,j) = \bar{d}(i,j) + \mathrm{O}\Big(\frac{\log N}{N}\Big)\Big\} \tag{52}$$

This in turn is the same as:

$$\tilde{\mathcal{X}}^N = \Big\{x \in \mathbb{R}^{|\mathcal{I}|\times|\mathcal{J}|} : x(i,j) \geq \mathrm{O}\Big(\frac{\log N}{N}\Big);\ \sum_{j\in\mathcal{J}} x(i,j) = 1\Big\} \tag{53}$$

Note that by construction, $\tilde{\mathcal{X}}^N \subseteq x^N$. But we can now see that $\tilde{\mathcal{X}}^N$ converges to $\mathcal{D}$ in the sense that

$$\sup_{x\in\mathcal{D}} \inf_{y\in\tilde{\mathcal{X}}^N} \|x-y\| = \mathrm{O}\Big(\frac{\log N}{N}\Big),$$

and hence,

$$\sup_{x\in\mathcal{D}} \inf_{y\in\mathcal{X}^N} \|x-y\| = \mathrm{O}\Big(\frac{\log N}{N}\Big)$$

as well. □

Proposition D.4. *The set $\mathcal{X}^N$ is a convex polytope.*

Proof. For the purpose of this proof, let

$$\overline{\mathcal{X}}^N = \{Nx : x \in \mathcal{X}^N\}.$$

We will show that $\overline{\mathcal{X}}^N$ is a polytope, from which the result will follow. We will prove this using an induction argument. We will represent each point in $\overline{\mathcal{X}}^N$ as a $|\mathcal{I}| \times |\mathcal{J}|$ matrix $(x(i,j))_{|\mathcal{I}|\times|\mathcal{J}|}$. Let worker types in $\mathcal{I}$ be labeled as $i_1, \ldots, i_{|\mathcal{I}|}$ and let job types in $\mathcal{J}$ be labeled as $j_1, \ldots, j_{|\mathcal{J}|}$.

Now clearly, $\overline{\mathcal{X}}^0 = \{(0)_{|\mathcal{I}|\times|\mathcal{J}|}\}$ which is a convex polytope. We will show that if $\overline{\mathcal{X}}^N$ is a convex polytope, then $\overline{\mathcal{X}}^{(N+1)}$ is one as well, and hence the result will follow. To do so, we decompose the assignment problem with $(N+1)$ jobs, into the first job and the remaining N jobs.

A policy in the $(N+1)$- jobs problem is a choice of a randomization over the jobs in $\mathcal{J}$ for the first job, and depending on whether a reward was obtained or not with the chosen job, a choice of a point in $\overline{\mathcal{X}}^N$ to be achieved for the remaining N jobs. Each such policy gives a point in the $\overline{\mathcal{X}}^{(N+1)}$. Suppose that $\eta_1 \in \Delta(\mathcal{J})$ is the randomization chosen for job 1, and let $R(j,1) \in \overline{\mathcal{X}}^N$ and $R(j,0) \in \overline{\mathcal{X}}^N$ be the points chosen to be achieved from job 2 onwards depending on the job j that was chosen, and whether a reward was obtained or not, i.e.. $R(.,.)$ is a mapping from $\mathcal{J} \times \{0,1\}$ to the set $\overline{\mathcal{X}}^N$. Then this policy achieves the following point in the $(N+1)$- jobs problem:

$$\begin{bmatrix} \eta_1(j_1) & \eta_1(j_2) & \ldots & \eta_1(j_{|\mathcal{J}|}) \\ \vdots & \vdots & \ddots & \vdots \\ \eta_1(j_1) & \eta_1(j_2) & \ldots & \eta_1(j_{|\mathcal{J}|}) \end{bmatrix} + \sum_{j\in\mathcal{J}} \eta_1(j)\Big(\text{Diag}[A(.,j)]R(j,1) + \text{Diag}[\bar{A}(.,j)]R(j,0)\Big),$$

where

$$\text{Diag}[A(.,j)] = \begin{bmatrix} A(i_1,j) & 0 & \ldots & 0 \\ 0 & A(i_2,j) & \ldots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \ldots & A(i_{|\mathcal{I}|},j) \end{bmatrix}$$

and

$$\text{Diag}[\bar{A}(.,j)] = \begin{bmatrix} 1-A(i_1,j) & 0 & \ldots & 0 \\ 0 & 1-A(i_2,j) & \ldots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \ldots & 1-A(i_{|\mathcal{I}|},j) \end{bmatrix}.$$

And thus we have

$$\overline{\mathcal{X}}^{(N+1)} = \left\{ \begin{bmatrix} \eta_1(j_1) & \eta_1(j_2) & \ldots & \eta_1(j_{|\mathcal{J}|}) \\ \vdots & \vdots & \ddots & \vdots \\ \eta_1(j_1) & \eta_1(j_2) & \ldots & \eta_1(j_{|\mathcal{J}|}) \end{bmatrix} + \sum_{j\in\mathcal{J}} \eta_1(j)\Big(\text{Diag}[A(.,j)]R(j,1) + \text{Diag}[\bar{A}(.,j)]R(j,0)\Big) : \eta_1 \in \Delta(\mathcal{J}),\ R(.,.) \in \overline{\mathcal{X}}^N \right\}.$$

Let $\mathbf{1}_s$ be the $|\mathcal{I}| \times |\mathcal{J}|$ matrix with ones along column corresponding to job type j and all other entries 0. Then the set

$$\mathcal{J}(s) = \left\{ \mathbf{1}_s + \text{Diag}[A(.,j)]R(j,1) + \text{Diag}[\bar{A}(.,j)]R(j,0) : R(s,.) \in \overline{\mathcal{X}}^N \right\},$$

is a convex polytope, being a linear combination of two convex polytopes, followed by an affine shift. It is easy to see that $\overline{\mathcal{X}}^{(N+1)}$ is just a convex combination of the polytopes $\mathcal{J}(s)$ for $j \in \mathcal{J}$, and hence $\overline{\mathcal{X}}^{(N+1)}$ is a convex polytope as well. $\square$