Online Assignment of Heterogeneous Tasks in Crowdsourcing Markets

Sepehr Assadi, Justin Hsu, Shahin Jabbari
Department of Computer and Information Science, University of Pennsylvania
{sassadi, justhsu, jabbari}@cis.upenn.edu

May 13, 2015

Abstract

We investigate the problem of *heterogeneous task assignment* in crowdsourcing markets from the point of view of the requester, who has a collection of tasks. Workers arrive online one by one, and each declare a set of feasible tasks they can solve, and desired payment for each feasible task. The requester must decide on the fly which task (if any) to assign to the worker, while assigning workers only to feasible tasks. The goal is to maximize the number of tasks with a fixed overall budget.

We provide upper and lower bounds on the competitive ratio of any online algorithm for this problem against an arbitrary (possibly worst-case) sequence of workers who want small payments relative to the requester's total budget. We also show algorithms achieving improved competitive ratio in the random permutation model, where the group of workers is fixed but the order of arrival of the workers is random.

1 Introduction

Crowdsourcing markets have seen a remarkable rise in recent years, as more and more markets use the internet to connect people with tasks to solve, to people willing to solve tasks in exchange for payment. While these tasks were originally simple tasks that could be accomplished while sitting at a computer—say, labeling images or cleaning data—recent systems handle complex tasks in the real world. Services like TaskRabbit let users hire workers to run errands, like picking up a package; ride-sharing applications like Lyft or Uber provide drivers on-demand to transport customers; other services offer meal or grocery delivery, house cleaning, and even on-demand massages[1] to customers.

Accordingly, an important challenge for task requestors in crowdsourcing (and more general) platforms is handling more and more *heterogenous* workers and tasks. A worker may not be able to solve all the tasks—say, they may only be able to translate certain languages, or they may only be able to handle time-sensitive tasks a deadline far enough into the future. Among the solvable tasks, a worker may want different payment for different tasks; a short task can require a small payment, while a more complex task may warrant a higher payment.

At the same time, requestors must cope with a highly *dynamic* flow of workers. Since it is common for workers to work for several different platforms simultaneously, the pool of available workers is constantly changing. The requestor may not have the luxury of seeing all the workers, and then selecting the right workers. Instead, requestors may need to select workers in an *online* fashion, where workers arrive one by one and must be hired (or not) as soon as they arrive.

We take aim at both of these challenges in crowdsourcing by considering the following *task assignment problem*: if workers (i) have an arbitrary set of feasible tasks, (ii) demand heterogeneous payments for feasible tasks, and (iii) arrive online, how can a requestor assign workers to tasks in order to maximize the number of completed tasks given a fixed overall budget?

Singer and Mittal [2011] were the first to study a simpler version of this problem. In their setting, each worker has a single value for all the tasks; all the tasks are treated *homogeneously*, and workers are assumed to be able to solve all

[1] www.zeel.com

tasks (for the right price). We generalize their setting in two significant ways: First, we allow workers to specify only a subset of feasible tasks that they can handle. Second, we work in a setting with *heterogeneous* tasks: workers can have different preferences over tasks. As we will show, this heterogeneity significantly complicates the task assignment problem; algorithms for the heterogeneous case may look nothing like their counterparts in the homogeneous case.

We first warm up by providing two algorithms for the offline problem: first, the optimal algorithm via min-cost flows, and second, a fixed-threshold algorithm inspired by the algorithm of Singer and Mittal [2011]. Then, we move to the online setting, where we draw on techniques from the online knapsack literature.

As is typical for online algorithms, we measure the performance of our algorithm via the *competitive ratio*, $i.e.$, the ratio of the number of tasks assigned by the best offline algorithm to the number of tasks assigned by the online algorithm on the same problem. When the bids are bounded in $[1, R]$, the bidders are small compared to the budget, and the order is worst-case, we give an algorithm that achieves an $O(R^{\epsilon} \ln(R))$ competitive ratio when $R \leq \epsilon B$. The central idea is to use a moving threshold, and take all workers with bid below the threshold. As the budget is depleted, the threshold steadily decreases. Our algorithm is inspired by Zhou et al. [2008], who give algorithms for the online knapsack problem. We also show a lower bound: for any (possibly randomized) online algorithm, there exists an input that forces competitive ratio of at least $\Omega(\ln(R))$.

Then, we consider the *random permutation* setting where the bids are adversarial, but where the order of bidders is chosen uniformly at random. Then, we measure the performance of an online algorithm by comparing the number of tasks assigned *averaged* over all permutations to the number of tasks assigned by the offline optimal.

2 Model

Let us begin by modeling our setting. We will use j to index tasks and i to index workers throughout. We model the problem from the perspective of a requester who has a collection of m tasks.

Each worker i picks a subset of tasks $J_i \subseteq [m]$, along with the numeric bid b_{ij} for each task $j \in J_i$. Workers arrive *online*: the sequence of the workers and their bids are initially unknown to the requester but once a worker arrives, her bids for all the tasks are revealed to the requester. We assume that each worker can be assigned to at most one task; we can relax this assumption by having multiple copies of each worker. We denote the total number of workers by n. As customary in the crowdsourcing setting we assume workers are abundant so we assume n is large (see Section 3 and **??** for more details).

We model the sequence of workers (their bids and their order of arrivals) in two different ways. First, we consider the case that we have no assumptions on the sequence of workers. This worst case scenario has been referred to as an *adversarial setting* in the literature. Second, we consider the *random permutation model* in which we still make no assumptions on the sequence of the workers but we assume the sequence is randomly permuted before being presented to the requester. We define these two settings in more detail in Section 3 and **??**.

The requester has a budget of B and has to decide on the fly which task to assign to the worker who arrives; of course, a worker can only be assigned to a task he is willing to do. If task j is assigned to worker i, then the requester pays worker i *at least* the price b_{ij}. The goal of the requester is to maximize the number of tasks he assigns to the workers while spending at most a budget of B and satisfying all the constraints of the workers (on the tasks they are willing to solve).

Similar to the online algorithms literature, we compare the performance of our online algorithms to the performance of the best *offline* algorithm (denoted by OPT), which can see all the bids and the order of arrival of workers before allocating tasks to workers. As is standard, we measure the performance via the *competitive ratio*: the ratio of the number of tasks assigned in the offline optimal solution to the number of tasks assigned by the online algorithm (so competitive ratio is at least 1, and smaller competitive ratio is more desirable).

3 Adversarial Setting

When comparing an online algorithm with an offline algorithm, we first need to precisely specify the inputs on which we make the comparison. Let's first consider the worst case for the requester: *adversarial inputs*. In this setting, there

is a fixed input and order, and we compare an online algorithm in this single input to an offline algorithm on the same input via the competitive ratio. We are interested in bounding this ratio in the worst case, i.e., the max over all inputs.

Of course, if we really do not make any assumption on the input, we cannot hope to compete with an offline algorithm—the competitive ratio may be arbitrarily high [see Zhou et al., 2008, Section 1.2].

So, we restrict the adversary's power by making one main assumption on the relationship between worker bids and the budget: the ratio of the largest bid to the smallest bid should be small compared to the budget. More precisely, we can scale worker bids so that $b_{ij} \in [1, R]$, and we write $R = \epsilon B$. We will frequently consider ϵ to be small—this is appropriate for the crowdsourcing problems we have in mind, where *(i)* the scale of the budget is much larger than the scale of payments to bidders and *(ii)* the tasks are not extremely difficult, so no worker charges an exorbitantly high price. We refer to this assumption as *large market* assumption because when the bids are small compared to the budget, the requester can hire a large number of workers before exhausting his budget. This is also in line with the assumption we made in Section 2 for n to be large.

Before diving into the technical details, let us consider which range of parameters and competitive ratios is interesting. Since the bids are restricted in $[1, R]$, achieving a competitive ratio of R is trivial. So, we will be mainly interested in the following question: Can we design an algorithm that has a competitive ratio much smaller than R for any sequence of workers?

The rest of this section is organized as follows. We first investigate the offline problem, describing how the problem can be solved optimally. Then we propose a simpler algorithm for the offline setting; the performance is a constant factor off from optimal, but the simpler algorithm will be useful in Section 4, when we will work in the *random permutation* setting. We then move to the online setting, giving an algorithm with competitive ratio of $O(R^{\epsilon} \log(R))$ for any sequence of workers. Finally, we give a lower bound showing that any algorithm has competitive ratio of at least $\Omega(\log(R))$ in the worst case.

3.1 The Offline Problem

Let us start by investigating the offline problem where the sequence of workers and their bids are known up front. We first show how the offline optimal assignment can be computed. Then we propose a simpler algorithm that approximates the offline optimal assignment by a factor of 4. We show later in the paper that how this second algorithm, while suboptimal, can be converted to an online algorithm.

The offline problem is a well-known problem in the crowdsourcing literature [Singer, 2010] and can be solved by a reduction to the *min-cost flow* problem defined as follows. Consider a graph with costs and capacities on the edges and two nodes marked as source and target. Given this graph, for a demand value of flow, the goal of the min-cost flow problem is to route this amount of flow from the source to the target while minimizing the total flow cost over the edges, where the flow cost over an edge is equal to the amount of the flow passing the edge multiply by the cost of the edge.

We now briefly describe an algorithm for solving the offline problem optimally which uses min-cost flow algorithms as a subroutine. Note that this algorithm is generally known in the literature and we provide it here for the sake of completeness. In our problem, we want to maximize the number of assigned tasks given a fixed budget. In order to do so, we first construct the following instance of the min-cost flow problem. We start with a bipartite graph with workers on one side and tasks on the other. We then draw an edge from a worker to a task if the worker is willing to solve that task and let the cost of this edge to be equal to the bid of the worker for the task. Additionally, we attach a source node pointing to all the workers and connect all the tasks to an added target node. Finally, we set the capacity of all edges to be 1.

First notice that any feasible flow in this graph corresponds to an assignment of tasks and workers. Moreover, for any integer F, the minimum cost of a flow with value F corresponds to the minimum budget required for assigning F worker-task pairs[2]. Consequently, we can search over all possible $F \in [n]$, solve the min-cost flow problem on the described graph and demand flow of F, and return the maximum value of F where the minimum cost flow is at most the available budget.

[2]Since the amount of flow routed from the source to the target and the capacity of the edges are integral, the min-cost flow problem has an integral solution.

While the above approach achieves the optimal solution, we will see in the random permutation section that using a *fixed threshold* to decide which workers to hire can be very useful. Hence, we next provide a simpler algorithm with this feature; while this simpler algorithm does not guarantee an optimal solution anymore, we show in Theorem 1 that its solution is within a factor of at most 4 from OPT.

The algorithm, which we refer to as *Offline Approximation Algorithm* (OA) (see Algorithm 2) is mainly using a subroutine *Fixed Threshold Policy* (FTP) to compute the final solution. In this subroutine, given a threshold value p, the algorithm goes over workers one by one and assigns a task to an unassigned worker if the bid of the worker for that task is not bigger than p. In case there are more than one unassigned task that the worker bids p or less for, the algorithm break ties arbitrarily. The FTP subroutine is described in Algorithm 1.[3]

Algorithm 1 FTP

Input: Threshold price p, budget B, worker bids $\{b_{ij}\}$, set of tasks J.
  While $B > 0$, on input i:
    Let $C_i := \{j \in J \mid b_{ij} \leq \min(p, B)\}$.
    If $C_i \neq \emptyset$:
      Output: $a(i) \in C_i$.
      Let $B := B - b_{i,a(i)}$.
      Let $J := J \setminus \{a(i)\}$.
    else:
      Output: $a(i) :=\perp$.

OA searches over all possible values of p to find the proper threshold. Although the search space is continuous, it is sufficient for the algorithm to restrict its search to the bids of the workers. We show that the number of assignments of the OA is least a quarter of the OPT for any sequence of worker arrivals.

Algorithm 2 OA

Input: Worker bids $\{b_{ij}\}$, set of tasks J, budget B.
  Let $Q := 0$.
  Foreach b_{ij}:
    Let $q := \text{FTP}(b_{ij}, B, \{b_{i,j}\}, J)$.
    If $q > Q$ **then**: Update $Q := q$.
Output Q (number of assignments) and $p^* := B/Q$ (the threshold price).

Theorem 1. *Let* $\text{ALG}_{\text{OA}}(\sigma)$ *and* $\text{OPT}(\sigma)$ *denote the number of tasks assigned by the* OA *and the offline optimal algorithm for a sequence of workers* σ*, respectively. Then for any* σ *that satisfies the large market assumption,* $\text{OPT}(\sigma) \leq 4 \cdot \text{ALG}_{\text{OA}}(\sigma)$.

Before proving Theorem 1, we state the following useful lemma.

Lemma 1. *Let* $a_1, \ldots a_n, a_{n+1}$ *be a sorted sequence of* $n+1$ *positive numbers in an increasing order such that* $\Sigma_{i=1}^{n} a_i \leq B$ *and* $\Sigma_{i=1}^{n+1} a_i > B$*. Then* $a_{\lfloor n/2 \rfloor} \cdot n/2 \leq B$*. [see Singer and Mittal, 2013, Lemma 3.1]*

The proof of Lemma 1 is the result of the following two simple observations: *(i)* the sum of the second half of the numbers is at most B and *(ii)* each of the numbers in the second half is at least as big as median of the sequence.

Proof of Theorem 1. Consider all the (worker, task) pairs in the offline optimal where a pair simply denotes that the task is assigned to the worker. Sort these pairs in an increasing order of the bids. Let p^* denote the median of the bids.

[3]While we use FTP as an offline algorithm in this section, the sequential nature of the algorithm allows FTP to be used when workers arrive online (one by one). We exploit this feature of FTP in Section 4.

By Lemma 1, we can assign half of these (worker, task) pairs, pay all the workers price p^* (clearly an upper bound on the bid of every considered worker) and be sure not to exceed the budget. However, there are two problems: *(i)* we do not know which (worker, task) pairs are in the optimal solution and hence, *(ii)* we do not know the value p^*.

To deal with problem *(i)*, suppose we somehow knew p^*. We can then assign a worker to any task where the worker has a bid of at most p^* and continue until *(a)* we exhaust the budget or *(b)* there are no more workers or tasks left.

In case *(a)*, the algorithm made at least B/p^* assignments and by Lemma 1, we know this number is at least $\mathrm{OPT}/2$. For case *(b)*, consider the bipartite graph between the tasks and the workers described earlier for the min-cost flow problem construction and remove all the edges with cost bigger than p^*. Since p^* is the median of the bids in the optimal solution, we know that in this graph, there exists a matching M of size at least $\mathrm{OPT}/2$ between the workers and the tasks. On the other hand, since we know OA terminated in this case because there are no tasks or workers left, we know that OA has arrived at a *maximal* matching. Finally, since the size of any maximal matching is at least half of the size of the maximum matching, we know the number of assignments of OA is at least a half of the number of assignments of M. So $\mathrm{ALG}_{\mathrm{OA}}(\sigma) \geq \mathrm{OPT}(\sigma)/4$ if we knew p^*.

To deal with problem *(ii)* (not knowing p^*), we run the FTP for all the values that p^* can take and return the maximum number of assignments as our solution.[4] □

A brief detour: the homogeneous case. As we discussed in the introduction, we make a strong distinction between the heterogeneous tasks and homogeneous tasks. To highlight this difference, we show that in the homogeneous case, the following simple algorithm computes the offline optimal: sort all the workers by their bids and (if possible) assigns a task to the sorted workers until the budget is exhausted or there are no more tasks or workers left.

It is easy to verify that this greedy construction indeed computes the best offline assignments when the workers are homogeneous. However, this method will not result in the optimal offline assignment when the tasks are heterogeneous. For example consider the following toy problem with two workers and two tasks with the single deadline. Let the bids of worker 1 and 2 to be $(0.4, 0.5)$ and $(0.45, 0.7)$ for the two tasks, respectively. With a budget of 1, the optimal offline assignment is to assign task 1 to worker 2 and task 2 to worker 1. However, the homogeneous greedy algorithm will assign task 1 to worker 1 and will not have enough budget left to assign a task to worker 2.

3.2 The Online Problem

Let's now move to the online setting, where workers arrive online and our algorithm must decide which (if any) task to assign the worker before seeing the remaining workers. We propose an online algorithm and prove an upper bound on the competitive ratio of the algorithm which holds against *any* sequence of workers. Our upper bound crucially depends on the large market assumption where we assume that the bids of the workers are small compared to the budget. So throughout this section, we assume the bids are bounded in $[1, R]$ where $R \leq \epsilon B$ for some small constant ϵ.

Our *Online Heterogeneous Algorithm* (OHA) is inspired by an algorithm from the online knapsack literature proposed by Zhou et al. [2008], with an analysis modified for our setting. The idea is to use a potential function $\phi : [0, 1] \to [1, R]$ based on the fraction of budget spent so far as an input. The ϕ function acts as a price threshold: the algorithm assigns a task to a worker only if the bid of the worker for any of the remaining unassigned tasks is below the value of the potential function—intuitively, as the budget shrinks, the algorithm becomes pickier about which workers to hire. Once a worker is selected, the algorithm greedily assigns a task arbitrarily from the remaining set of tasks. See Algorithm 3 for a pseudo-code.[5]

Theorem 2. *Let* $\mathrm{ALG}_{\mathrm{OHA}}(\sigma)$ *and* $\mathrm{OPT}(\sigma)$ *denote the number of tasks assigned by the* OHA *and the offline optimal algorithm for a sequence of workers* σ*, respectively. Then for any* σ *with bids in* $[1, R]$ *such that* $R \leq \epsilon B$*,* $\mathrm{OPT}(\sigma)/\mathrm{ALG}_{\mathrm{OHA}}(\sigma) \leq (Re)^{\epsilon} (\ln(R) + 3)$.

[4] Note that this might result in OA using a threshold p which is different than p^* but since OA picks a p that maximizes the number of assignments we know the number of assignments made by OA using p is at least as big as the number of assignments made by OA using p^*.

[5] In Algorithm 3, e is the base of natural logarithm.

Algorithm 3 OHA

Input: Tasks J, budget B.
Online input: Worker bids $\{b_{ij}\} \in [1, R]$.
  Define $\phi(x) = \min((R \cdot e)^{1-x}, R)$.
  Let $x := 0, f := B$.
  While $B > 0$, on input i:
    Let $C_i := \{j \in J \mid b_{ij} \leq \min(f, \phi(x))\}$.
    If $C_i \neq \emptyset$ **then**
      Output $a(i) \in C_i$.
      Let $x := x + b_{i,a(i)}/B$.
      Let $f := f - b_{i,a(i)}$.
      Let $J := J \setminus \{a(i)\}$.
    else:
      Output: $a(i) :=\perp$.

Proof. Fix a sequence of workers σ. Let $S = \{(i, j)\}$ be the set of (worker, task) pairs assigned by the OHA where i and j index workers and tasks respectively, and let $S^* = \{(i^*, j^*)\}$ be the offline optimal. We want to bound $\mathrm{OPT}(\sigma)/\mathrm{ALG}_{\mathrm{OHA}}(\sigma) = |S^*|/|S|$.

Let x_i and X denote the fraction of the budget used by the OHA when worker i arrives and upon termination, respectively. We will analyze the (worker, task) pairs in three stages. First, consider the common (worker, task) pairs which we denote by $(i, j) \in S \cap S^*$. Let $W = \sum_{(i,j)\in S\cap S^*} b_{ij}$ denote the total bid of such workers. Since each worker i in the common part who is assigned to task j is picked by the OHA, it must be that $b_{ij} \leq \phi(x_i)$. Therefore,

$$|S \cap S^*| \geq \sum_{(i,j)\in S\cap S^*} \frac{b_{ij}}{\phi(x_i)}. \tag{1}$$

Second, consider $(i, j) \in S^* \setminus S$. This can happen either because *(i)* the worker i is assigned to a task different than j by OHA, or *(ii)* the worker i is not assigned to any task by OHA. We know the number of pairs that satisfy the condition *(i)* is at most $|S|$ because all such workers are assigned to a task by both OHA and the offline optimal.

For the pairs that satisfy condition *(ii)*, either *(a)* $b_{ij} \leq \phi(x_i)$ or *(b)* $b_{ij} > \phi(x_i)$. It is easy to see that in case *(a)*, OHA assigned task j to some other worker i' who arrived before i. Again we know this can happen at most $|S|$ times. For case *(b)*, since ϕ is non-increasing then $b_{ij} > \phi(x_i) \geq \phi(X)$ for all such pairs (i, j). Since the offline optimal spent a budget of W for hiring workers in $S^* \cap S$, then it has a budget of at most $B - W$ to hire workers in case *(b)*. Since all the pairs in case *(b)* have $b_{ij} > \phi(x_i) \geq \phi(X)$, the offline optimal can hire at most $(B - W)/b_{ij} \leq (B - W)/\phi(X)$ workers. Adding up cases *(i)* and *(ii)*, we can bound

$$|S^* \setminus S| \leq 2|S| + \frac{(B - W)}{\phi(X)}. \tag{2}$$

Finally, consider $(i, j) \in S \setminus S^*$. Since $b_{ij} \leq \phi(x_i)$ for all such pairs, we know

$$|S \setminus S^*| \geq \sum_{(i,j)\in S\setminus S^*} \frac{b_{ij}}{\phi(x_i)}. \tag{3}$$

Putting all the pieces together, Equations (1) to (3) yield

$$\frac{\mathrm{OPT}(\sigma) - 2 \cdot \mathrm{ALG}_{\mathrm{OHA}}(\sigma)}{\mathrm{ALG}_{\mathrm{OHA}}(\sigma)} = \frac{|S \cap S^*| + |S^* \setminus S| - 2|S|}{|S \cap S^*| + |S \setminus S^*|} \leq \frac{\sum_{(i,j)\in S\cap S^*} b_{ij}/\phi(x_i) + |S^* \setminus S| - 2|S|}{\sum_{(i,j)\in S\cap S^*} b_{ij}/\phi(x_i) + |S \setminus S^*|}. \tag{4}$$

To see why the inequality holds, first note that we know $|S \cap S^*| \geq \sum_{(i,j)\in S\cap S^*} b_{ij}/\phi(x_i)$ by Equation (1). Now if $\mathrm{OPT} = |S^*| < 3|S|$ then we are done. Otherwise,

$$\begin{aligned} |S^*| &\geq 3|S| \\ |S \cap S^*| + |S^* \setminus S| &\geq |S \setminus S^*| + |S \cap S^*| + 2|S| \\ |S^* \setminus S| - 2|S| &\geq |S \setminus S^*|. \end{aligned}$$

Hence, the inequality holds because if $a \geq b$ and $c \geq d$ then $(a+c)/(a+d) \leq (b+c)/(b+d)$.

We bound the right hand side of Equation (4) as follows.

$$\begin{aligned} \frac{\sum_{(i,j)\in S\cap S^*} b_{ij}/\phi(x_i) + |S^* \setminus S| - 2|S|}{\sum_{(i,j)\in S\cap S^*} b_{ij}/\phi(x_i) + |S \setminus S^*|} &\leq \frac{\sum_{(i,j)\in S\cap S^*} b_{ij}/\phi(x_i) + 2|S| + (B-W)/\phi(X) - 2|S|}{\sum_{(i,j)\in S\cap S^*} b_{ij}/\phi(x_i) + |S \setminus S^*|} \\ &= \frac{\sum_{(i,j)\in S\cap S^*} b_{ij}/\phi(x_i) + (B-W)/\phi(X)}{\sum_{(i,j)\in S\cap S^*} b_{ij}/\phi(x_i) + |S \setminus S^*|} \\ &\leq \frac{\sum_{(i,j)\in S\cap S^*} \frac{b_{ij}}{\phi(X)} + (B-W)/\phi(X)}{\sum_{(i,j)\in S\cap S^*} b_{ij}/\phi(x_i) + |S \setminus S^*|} \\ &= \frac{W/\phi(X) + (B-W)/\phi(X)}{\sum_{(i,j)\in S\cap S^*} b_{ij}/\phi(x_i) + |S \setminus S^*|} \\ &= \frac{B/\phi(X)}{\sum_{(i,j)\in S\cap S^*} b_{ij}/\phi(x_i) + |S \setminus S^*|} \\ &\leq \frac{B/\phi(X)}{\sum_{(i,j)\in S\cap S^*} b_{ij}/\phi(x_i) + \sum_{(i,j)\in S\setminus S^*} b_{ij}/\phi(x_i)} \\ &= \frac{B/\phi(X)}{\sum_{(i,j)\in S} b_{ij}/\phi(x_i)} \\ &= \frac{1}{\phi(X)\sum_{(i,j)\in S} \Delta(x_i)/\phi(x_i)}, \end{aligned}$$

where $\Delta(x_i) := x_{i+1} - x_i$. The first inequality is due to Equation (2), the second is due to monotonicity of ϕ and the third inequality is due to Equation (3). Since $(i,j) \in S$, OHA assigns task j to worker i and increases the fraction of the budget consumed by b_{ij}/B, so $b_{ij}/B = \Delta(x_i)$.

We now estimate the sum with an integral. If $\Delta(x_i) \leq \epsilon$,

$$\sum_{(i,j)\in S} \Delta(x_i)\frac{1}{\phi(x_i)} \geq \int_0^{X-\epsilon} \frac{1}{\phi(x_i)}dx.$$

Letting $c = 1/(1+\ln(R))$, we have $\phi(x) = R$ if $x \leq c$. Similar to [Zhou et al., 2008], we bound the integral as follows.

$$\begin{aligned} \int_0^{X-\epsilon} \frac{1}{\phi(x_i)}dx &= \int_0^c \frac{1}{R}dx + \int_c^{X-\epsilon} \frac{1}{\phi(x_i)}dx \\ &= \frac{c}{R} + \frac{1}{Re}\cdot\frac{1}{1+\ln(R)}((Re)^{X-\epsilon} - (Re)^c) \\ &= \left(\frac{c}{R} - \frac{1}{Re}\cdot\frac{1}{1+\ln(R)}(Re)^c\right) + \frac{1}{Re}\cdot\frac{1}{\ln(R)+1}(Re)^{X-\epsilon} \\ &= \frac{1}{Re}\cdot\frac{1}{1+\ln(R)}(Re)^{X-\epsilon} \\ &= \frac{1}{\phi(X)}\cdot\frac{(Re)^{-\epsilon}}{1+\ln(R)}. \end{aligned}$$

It is easy to show by algebraic manipulation that the first term in the 3rd line is equal to 0. Therefore,

$$\frac{\mathrm{OPT}(\sigma) - 2 \cdot \mathrm{ALG}_{\mathrm{OHA}}(\sigma)}{\mathrm{ALG}_{\mathrm{OHA}}(\sigma)} \leq \frac{1}{\phi(X) \sum_{(i,j) \in S} \Delta(x_i)/\phi(x_i)} \leq (Re)^{\epsilon}(\ln(R) + 1).$$

Thus, $\mathrm{OPT}(\sigma) \leq ((Re)^{\epsilon}(\ln(R) + 3) \cdot \mathrm{ALG}_{\mathrm{OHA}}(\sigma)$, as desired. □

3.3 Lower Bound on the Competitive Ratio

To wrap up this section, we show that if the large market assumption is the only assumption on the sequence of workers, then no algorithm (even randomized) can achieve a constant competitive ratio. We prove the result for the special case that all the tasks are homogeneous.

The proof is based on the ideas from a lower bound for the competitive ratio in a variant of the online knapsack problem by Zhou et al. [2008]. In this variant, items arrive one by one online and we have a knapsack with some known capacity. Each item has a weight and a utility parameter and it is assumed that the utility to weight ratio for all the items are within a bounded range. Furthermore, it is assumed that the weight of each item is small compared to the capacity of the knapsack (similar to our large market assumption). The algorithm have to decide whether to pick an item or not upon arrival and the goal is to maximize the utility of the picked items while satisfying the knapsack capacity constraint. Since our setting when the tasks are homogeneous is a special case of the online knapsack variant, the lower bound for that problem might not necessarily provide us with a lower bound. However, we show that with a bit of care, we can achieve the same lower bound using a similar construction.

The main idea of the lower bound is to construct hard sequences of worker arrivals. A hard sequence can be described as follows. The sequence starts with workers with maximum bid, R, and then the following workers progressively have smaller and smaller bids compared to the preceding workers. Then at some random point until the end of the sequence only workers with bid R appear in the sequence. Intuitively, these sequences are hard because no algorithm can foresee whether the bids will decrease (so it should wait for cheaper workers) or increase (so it should spend its budget on the current workers).

Theorem 3. *For any (possibly randomized) online algorithm, there exists a set of sequences of worker arrivals satisfying the large market assumption (all bids in $[1, R]$) such that the competitive ratio of the algorithm on the sequence is at least $\Omega(\log(R))$.*

Proof. We modify the proof of Theorem 2.2 by Zhou et al. [2008] to fit our problem formulation.

We use Yao's *minimax principle*: by constructing a distribution over a set of instances and showing that no deterministic algorithm can achieve a expected competitive ratio which is better than $\ln(R) + 1$ on these instances, Yao's principle implies that there no *randomized* algorithm can beat this competitive ratio on all (deterministic) instances [Yao, 1977].

To construct the distribution, fix $\eta \in (0, 1)$, let k be the smallest integer such that $(1-\eta)^k \leq 1/R$, and define $k+1$ instances indexed by I_0 to I_k as follows. I_0 contains B/R identical workers all with bids equal to R. For all $u > 0$, I_u is I_{u-1} followed by $B/(R(1-\eta)^u)$ workers all with bids equal to $R(1-\eta)^u$. Since these instances have different length we pad all the instances with enough workers with bids R so that all the instances have the same length.

We specify D by $k+1$ values $p_0, \ldots, p_k$ where p_u denotes the probability of occurrence of instance I_u. Let

$$p_0 = p_1 = \ldots = p_{k-1} := \frac{\eta}{(k+1)\eta + 1} \quad \text{and} \quad p_k := \frac{1+\eta}{(k+1)\eta + 1}.$$

On this distribution of inputs, any deterministic algorithm is fully specified by the fraction of budget spent on hiring workers with bid $R(1-\eta)^u$; call each fraction f_u. Since the optimal assignment for instance i is to only hire workers with bids $R(1-\eta)^u$, the *inverse* of the expected competitive ratio can be bounded as follows.

$$\sum_{u=0}^{k} p_u \frac{\sum_{v=0}^{u} f_v B/(R(1-\eta)^v)}{B/(R(1-\eta)^u)} = \sum_{u=0}^{k} p_u \sum_{v=0}^{u} f_v (1-\eta)^{u-v} = \sum_{v=0}^{k} f_v \sum_{u=v}^{k} p_u (1-\eta)^{u-v}, \tag{5}$$

where the last statement is by expanding the sums and reordering the terms.

The second sum in the RHS of Equation (5) is bounded by

$$\sum_{u=v}^{k} p_u(1-\eta)^{u-v} = \frac{2\eta(1-\eta)^{k-v} + (1-\eta)}{(k+1)\eta+1} \leq \frac{2\eta+(1-\eta)}{(k+1)\eta+1} = \frac{1+\eta}{(k+1)\eta+1}, \tag{6}$$

where the first equality is derived exactly similar to Zhou et al. [2008]. Replacing Equation (6) into the RHS of Equation (5),

$$\sum_{v=0}^{k} f_v \sum_{u=v}^{k} p_u(1-\eta)^{u-v} \leq \frac{1+\eta}{(k+1)\eta+1} \sum_{v=0}^{k} f_v \leq \frac{1+\eta}{(k+1)\eta+1},$$

since by definition $\sum_{v=0}^{k} f_v \leq 1$.

Now by definition of k, we know $(1-\eta)^k \leq 1/R$, which implies $k+1 \geq \ln(R)/\ln(1/(1-\eta))$. So,

$$\sum_{v=0}^{k} f_v \sum_{u=v}^{k} p_u(1-\eta)^{u-v} \leq \frac{1+\eta}{(k+1)\eta+1} \leq \frac{1+\eta}{\eta\ln(R)/\ln(1/(1-\eta))+1} = O\left(\frac{1}{\ln(R)}\right)$$

as $\eta \to 0$, because $\lim_{\eta\to 0} \eta/\ln(1/(1-\eta)) = 1$. Since we bound the inverse of the competitive ratio in the above analysis, then the competitive ratio is at least $\Omega(\ln R)$. □

Finally, it is worth mentioning that if we consider the limit where worker's bids are very small compared to the budget ($\epsilon \to 0$), the Theorem 2 shows that Algorithm 3 has a competitive ratio approaching $O(\ln(R))$, the best possible by Theorem 3.

4 Random Permutation Setting

Now that we have considered the worst-case scenario of inputs, let us consider more "well-distributed" inputs. We consider the *random permutation model* [Devanur and Hayes, 2009]. In the random permutation model there is no assumption about the bids of the workers (they can still be chosen by an adversary) but we measure the competitive ratio a bit differently: we take all possible permutations of these workers, and take the average competitive ratio over all permutations. Intuitively, this assumption makes the task assignment easier because the premium workers get distributed evenly in the sequence.

As pointed out by Devanur and Hayes [2009], the random permutation model can be considered as drawing bids from an unknown distribution without replacement. Hence, the random permutation model is very similar to the model that assumes the bids of the workers are drawn $i.i.d.$ from an unknown distribution.

Under a distributional assumption, the online optimal assignment is a random variable while the offline optimal assignment is invariant of the permutation. This makes the analysis of the competitive ratio cleaner in the random permutation model compared to the distributional model.

Throughout this section, we assume that the number of available workers, n, is large, and known to us. Also, we restrict our attention to inputs where the offline optimal algorithm assigns at least a constant fraction of workers, i.e., $\mathrm{OPT}(\sigma) = \Omega(n)$ for all sequences σ.

For the case that the workers are homogeneous, Singer and Mittal [2013] provide an algorithm with competitive ratio of 360. In this section, we extend their result to the case the tasks are heterogeneous. We also improve on their competitive ratio, though the setting is a bit different: as they were concerned with mechanism design properties, it was important for Singer and Mittal [2013] to carefully tune their payments to (i) incentivize workers to bid honestly and (ii) compensate all workers, even workers at the very beginning. Our algorithm does not satisfy these properties.

If we knew the sequence of the workers and their bids, we could run OA to compute a threshold price p and number of assignments Q where we know Q is at least a quarter of the optimal number of assignments by Theorem 1. However, since we are in the random permutation model, we can estimate this threshold with high probability by observing a subset of workers. This idea is summarized as the Random Permutation Algorithm (RPA) in Algorithm 4.

The algorithm takes an input α and return an estimate $\hat{p}$ for the threshold p such that this estimate is larger than p with high probability. While this overestimation might decrease the performance of the algorithm, it makes our analysis easier of competitive ratio of RPA easier.

Algorithm 4 RPA

Input: Parameter $\alpha \in (0,1)$, set of tasks J, budget B.
Online input: Worker bids $\{b_{ij}\}$.
Let C be the first half of workers, don't assign.
Let $\hat{p} := \mathrm{OA}(C, J, B/2)$.
On rest of input, run $\mathrm{FTP}((1+\alpha)\hat{p}, B/2)$.

Theorem 4. *Let $\alpha, \delta \in (0,1)$, and suppose the number of workers is at least*

$$n \geq \Omega\left(\frac{1}{\alpha}\log(\frac{1}{\delta})\right),$$

and $\mathrm{OPT} \geq \Omega(n)$ for every input. For any sequence of workers σ, let $\mathrm{ALG}_{\mathrm{RPA}}(\sigma)$ and $\mathrm{OPT}(\sigma)$ denote the number of tasks assigned by the RPA *and the offline optimal algorithm for a sequence of workers σ, respectively. Then,*

$$\mathrm{OPT}(\sigma) \leq 8(1+\alpha)^2/(1-\alpha) \cdot \mathrm{ALG}_{\mathrm{RPA}}(\sigma),$$

with probability at least $1-\delta$ for all σ.

Proof. If we knew the sequence of the workers and their bids, we could run OA to compute a price p and number of assignments Q. We know $Q \geq \mathrm{OPT}/4$ by Theorem 1. Let us refer to these Q workers hired by OA as *good workers*.

We first claim that if we use a price of $(1+\alpha)\cdot p$ instead of p, the number of assignments will decrease by a factor of $(1+\alpha)$. This is because we can still assign a task to workers who OA assigned a task to with a threshold of p, but now we also have access to workers with bids in $(p, (1+\alpha)\cdot p]$ which may cause us to exhaust the budget faster.

Second, we can estimate p from the first half of the workers. Call this estimate $\hat{p}$. We claim that at least $(1-\alpha)Q/2$ of the good workers are in the second half of the sequence with high probability, and

$$p \leq (1+\alpha)\hat{p} \leq (1+\alpha)p/(1-\alpha). \tag{7}$$

This together with the first claim and the 4-approximation of OA will give us the competitive ratio claimed in the statement of the theorem.

To complete the proof, we first show that there are enough good workers in the second half of the sequence. Since we are in the random permutation model, then with high probability, we know the number of good workers in the first half of the workers is in $[Q(1-\alpha)/2, Q(1+\alpha)/2]$. Chvatal [1979] show that this probability is bounded by

$$1 - 2\sum_{i=Q(1+\alpha)/2}^{Q} \frac{\binom{i}{Q/2}\binom{n/2-i}{n-Q}}{\binom{n}{n/2}} =$$
$$\begin{cases} 1 - O(e^{-\alpha Q^2/n}) & \text{if } Q \geq \frac{n}{2}, \\ 1 - O(e^{-\alpha(n-Q)^2 Q/n^2}) & \text{if } Q < \frac{n}{2} \end{cases} = 1 - O(e^{-\alpha n}).$$

To prove that Equation (7) holds, we consider two cases.

1. The number of good workers in the first half is $[Q(1-\alpha)/2, Q/2)$. In this case $\hat{p} \geq p$ because we can obviously use the budget to assign tasks to all the good workers. And if there is leftover budget, we might be able to assign tasks to workers with bid greater than p. So,

$$\hat{p}\frac{Q}{2}(1-\alpha) \leq \frac{B}{2} = \frac{pQ}{2} \qquad \text{(last line in Algorithm 2)}$$
$$\hat{p} \leq \frac{1}{(1-\alpha)}p,$$

   Combined with $\hat{p} \geq p$, the condition in Equation (7) holds in this case.

2. The number of good workers in the first half is $[Q/2, Q(1+\alpha)/2]$.

   In this case $\hat{p} \leq p$. Suppose by contradiction that $\hat{p} > p$. This means the workers hired by the OA with bids of at most $\hat{p}$ is at least $S = B/(2\hat{p})$. We know $S \geq Q/2$ because all good workers have bids of most p. So,

$$\frac{B}{2\hat{p}} = \frac{pQ}{2\hat{p}} \geq \frac{Q}{2}$$
$$p \geq \hat{p},$$

   which is a contradiction. Thus,

$$\hat{p}\frac{Q}{2}(1+\alpha) \geq \frac{B}{2} = \frac{pQ}{2}$$
$$\hat{p} \geq \frac{1}{1+\alpha}p.$$

   Combined with $\hat{p} \leq p$, the condition in Equation (7) also holds in this case, concluding the proof.

□

5 Experiments

In this section we describe some experiments to compare the performance of our algorithms on synthetic data. In all of our experiments we set the number of tasks and workers to be the same. We set the bids of the workers to be integers in $[1, R]$ (we determine R in the experiments) uniformly at random and set the budget to be equal to the number of tasks and workers. Each worker puts a task in their feasible set with probability 0.1.

In our first experiment, we study the effect of different exploration rates in the performance of RPA. As presented

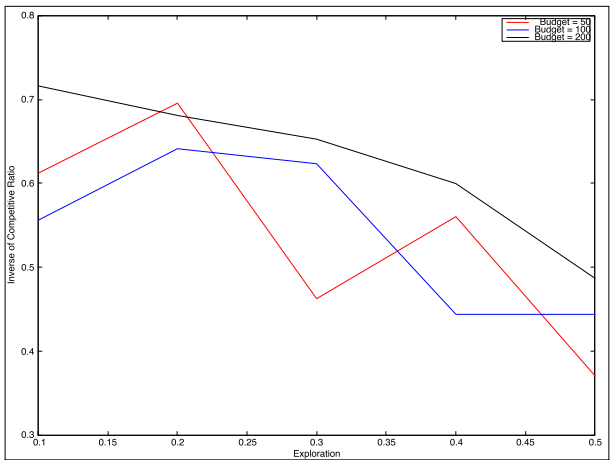

Figure 1: The effect of exploration in RPA.

in Algorithm 4, this algorithm uses the first half of the workers for exploration. Figure 1 shows the performance for three cases when the number of workers are set to 200, 100 and 50 (remembering that the number of tasks and budgets are equal to the number of workers in our experiments). In Figure 1, the x-axis shows the exploration rate—varying from 0.1 to 0.5—and the y-axis shows the *inverse* of the competitive ratio of RPA over 10 trials for $R = 10$. While for our analysis it was convenient to use the exploration rate of 0.5, the experiment shows that the performance of RPA degrades with more explorations. This is because when the bids are chosen from a uniform distribution, a small amount of exploration is sufficient to find the proper threshold in RPA. Also, the experiment shows that the competitive ratio of RPA decreases when the number of workers increases. . Finally, although the competitive ratio proven in Theorem 4

is at least 8, the experiment shows that for almost all of the values of the exploration parameter, RPA has a competitive ratio of 2.5 or smaller (which translates to an inverse competitive ratio of 0.4 or higher in Figure 1).

In our second experiment, we used a similar approach to create data as the previous experiment and compared the performance of RPA and OHA for different values of R. In Figure 2, the x-axis shows the value R which we varied from 5 to 20 and the y-axis shows the inverse of the competitive ratio of RPA and OHA over 10 runs. First, note that while OHA provably performs well against adversarial sequences of workers, Figure 2 shows that it also performs pretty well when the workers are created randomly . This is interesting because unlike our assumptions in the adversarial section, the ratio of budget to R in these experiments are not small.

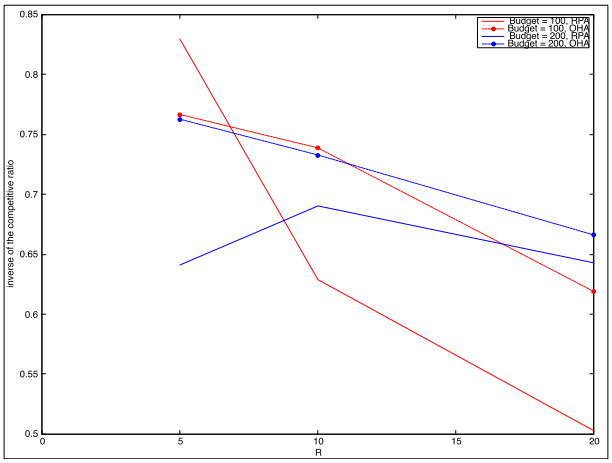

Figure 2: Comparison of RPA and OHA.

Second, OHA outperforms RPA when R and the budget are both small. This is because RPA pays a fixed price to any assigned workers in contrast to OHA which pays the worker no more than their bids. Furthermore, the performance of both algorithms degrades when the bids become bigger. We suspect that when the possible range of bids become bigger (R increases), the ratio of workers with small bid becomes smaller because the bids are chosen uniformly from 1 to R. So it becomes harder for RPA and OHA to select these workers without knowing the sequence of bids ahead of time.

6 Related Work

Pricing and task assignments have been previously studied in the context of *mechanism design* for crowdsourcing markets. Singer and Mittal [2011, 2013] provide mechanisms for task assignment when tasks are homogeneous and the workers are arriving from a random permutation model. Our work in the random permutation section generalizes their work to heterogeneous tasks. Ho and Vaughan [2012] consider task assignment when the tasks are heterogeneous. However, the assume they are task types and the focus of their work is learning qualities of the workers from stochastic observations. Singla and Krause [2013] design price mechanism for crowdsourcing markets using tools from online learning. Goel et al. [2014] consider the heterogeneous task assignment in the *offline* setting and provide near optimal approximation algorithms. However, they focus on the mechanism design aspect of the problem—how to pay workers so that they report their bids truthfully. This line of budget feasible mechanism design is inspired by the work of Singer [2010] and has been followed up in Singer and Mittal [2011, 2013]. Also these pricing mechanisms have close connections to the stochastic online adwords problem [Devanur and Hayes, 2009] and the online primal-dual literature (see [Buchbinder et al., 2007] and references within).

Our work is inspired by variants of the online knapsack problem. In the adversarial setting with homogeneous tasks, our problem is an instance of the online knapsack problem studied by Zhou et al. [2008] (see references within for more information on variants of the online knapsack problem). However, it is not clear how to formulate our problem as a knapsack problem when the tasks are heterogeneous.

Variants of *generalized online matching* and the *adwords* problem are also related to our problem ($e.g.$, see Mehta et al. [2005] and Mehta [2013] for an excellent survey). The adwords problem can be described as follows. There are some bidders and each bidder has a fixed budget. Queries arrive one at the time, bidders bid for the queries and the algorithm has to decide what bidder to assign to the query. If the algorithm assigns a bidder to a query, the bidder pays the amount that is equal to her bid. The goal is to maximize the revenue. While one might attempt to formalize our task assignment problem as an instance of the online adwords problem, it is not hard to see that our constraint on the total budget of the requester cannot be written as an adwords type budget constraint.

References

Niv Buchbinder, Kamal Jain, and Joseph Naor. Online primal-dual algorithms for maximizing ad-auctions revenue. In *Proceedings of the 15th Annual European Symposium on Algorithms*, pages 253–264, 2007.

Vaclav Chvatal. The tail of the hypergeometric distribution. *Discrete Mathematics*, 25(3):285 – 287, 1979.

Nikhil Devanur and Thomas Hayes. The adwords problem: online keyword matching with budgeted bidders under random permutations. In *Proceedings of 10th ACM Conference on Electronic Commerce*, pages 71–78, 2009.

Gagan Goel, Afshin Nikzad, and Adish Singla. Mechanism design for crowdsourcing markets with heterogeneous tasks. In *Proceedings of the Second AAAI Conference on Human Computation and Crowdsourcing*, 2014.

Chien-Ju Ho and Jennifer Vaughan. Online task assignment in crowdsourcing markets. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, 2012.

Aranyak Mehta. Online matching and ad allocation. *Foundations and Trends in Theoretical Computer Science*, 8(4): 265–368, 2013.

Aranyak Mehta, Amin Saberi, Umesh Vazirani, and Vijay Vazirani. Adwords and generalized on-line matching. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, pages 264–273, 2005.

Yaron Singer. Budget feasible mechanisms. In *Proceedings of the 51th Annual IEEE Symposium on Foundations of Computer Science*, pages 765–774, 2010.

Yaron Singer and Manas Mittal. Pricing tasks in online labor markets. In *Proceedings of the 3rd Human Computation Workshop*, 2011.

Yaron Singer and Manas Mittal. Pricing mechanisms for crowdsourcing markets. In *Proceedings of 22nd International World Wide Web Conference*, pages 1157–1166, 2013.

Adish Singla and Andreas Krause. Truthful incentives in crowdsourcing tasks using regret minimization mechanisms. In *Proceedings of the 22nd International World Wide Web Conference*, pages 1167–1178, 2013.

Andrew Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proceedings of the 54th IEEE Annual Symposium on Foundations of Computer Science*, pages 222–227, 1977.

Yunhong Zhou, Deeparnab Chakrabarty, and Rajan Lukose. Budget constrained bidding in keyword auctions and online knapsack problems. In *Proceedings of the 4th International Workshop on Internet and Network Economics*, pages 566–576, 2008.