Efficient Crowdsourcing of Unknown Experts using Multi-Armed Bandits

Long Tran-Thanh and **Sebastian Stein** and **Alex Rogers** and **Nicholas R. Jennings**[1]

Abstract. We address the expert crowdsourcing problem, in which an employer wishes to assign tasks to a set of available workers with heterogeneous working costs. Critically, as workers produce results of varying quality, the utility of each assigned task is unknown and can vary both between workers and individual tasks. Furthermore, in realistic settings, workers are likely to have limits on the number of tasks they can perform and the employer will have a fixed budget to spend on hiring workers. Given these constraints, the objective of the employer is to assign tasks to workers in order to maximise the overall utility achieved. To achieve this, we introduce a novel multi–armed bandit (MAB) model, the bounded MAB, that naturally captures the problem of expert crowdsourcing. We also propose an algorithm to solve it efficiently, called **bounded ε–first**, which uses the first εB of its total budget B to derive estimates of the workers' quality characteristics (exploration), while the remaining $(1-\varepsilon)B$ is used to maximise the total utility based on those estimates (exploitation). We show that using this technique allows us to derive an $O\left(B^{\frac{2}{3}}\right)$ upper bound on our algorithm's performance regret (i.e. the expected difference in utility between the optimal and our algorithm). In addition, we demonstrate that our algorithm outperforms existing crowdsourcing methods by up to 155% in experiments based on real–world data from a prominent crowdsourcing site, while achieving up to 75% of a hypothetical optimal with full information.

1 INTRODUCTION

Recently, businesses and other organisations have started to turn to a new emerging labour market to achieve their operating objectives. Using the internet, they advertise jobs to a global audience and hire workers on a temporary basis to complete tasks in exchange for financial remuneration. This so-called *crowdsourcing* promises considerable flexibility, as it quickly connects workers and contractors across the globe without large recruitment overheads.

Existing research and technologies have so far concentrated largely on facilitating the crowdsourcing of small units of work that can be completed in minutes by untrained labourers, such as participating in surveys, transcribing audio clips or annotating images [5, 6]. However, a growing number of businesses are now beginning to crowdsource work on large-scale projects that require many hours of effort by experts in a particular field. Such *expert crowdsourcing* is used for the development and testing of large software applications, building websites, professionally translating documents or organising marketing campaigns.[2]

In contrast to the crowdsourcing of smaller and simpler units of work, expert crowdsourcing raises a number of challenges that need to be addressed. First, the quality of a completed task can vary greatly both between different workers and even between subsequent tasks completed by the same worker. For example, a highly-skilled software engineer might complete up to ten times as many functions as a novice worker in a single hour, but the same skilled engineer may occasionally struggle with a particular task, perhaps due to adverse personal circumstances [1]. Second, there is typically little or no prior knowledge about the expected quality of a worker, as the online labour market is inherently open and dynamic in nature, with little historical information about worker performance. To illustrate this, more than 85% of workers advertising on `odesk.com` and `vworker.com` have not completed any significant amount of work in the past.[3] Finally, experts often demand widely varying prices for their services. This can be due to differences in skill level, but is similarly influenced by individual expectations, local wages and the cost of living in the worker's country of residence. As an example of this, different workers on `odesk.com` charge from as little as $5 to over $200 for one hour of Web design work.

Taken together, these challenges pose a critical problem to any organisation that wishes to crowdsource a considerable amount of work — how should it allocate tasks to unknown workers in order to achieve the highest possible quality of service while staying within a given budget? For example, a company implementing a large software project may wish to maximise the number of working features that meet at least a certain level of quality; while an organisation crowdsourcing an online marketing campaign might be interested in attracting the highest number of new customers.

Some researchers have explored learning the performance characteristics of workers and using these to improve the outcome of a project [11, 8], but they do not consider the financial cost of hiring the workers. The cost and performance of workers are balanced explicitly in other work on decision-theoretic crowdsourcing [2], and some research has looked into how organisations can set the financial rewards for completing work units, in order to effect participation [7, 3]. However, none of this work applies in the domain of expert crowdsourcing, where individual workers typically advertise their own heterogeneous prices and where the hiring business, or the *employer*, has to select the most suitable workers.

One area of work that is well suited to solving the expert crowdsourcing problem is the field of multi-armed bandits (MABs), a class of problems dealing with decision making under uncertainty. In these optimisation problems, actions (i.e. pulling a single arm) have initially unknown rewards that have to be learnt through noisy observations, and the goal is to maximise the total amount of rewards by

[1] University of Southampton, UK, email: {ltt08r,ss2,acr,nrj}@ecs.soton.ac.uk

[2] For some examples of these, see `vworker.com`, `odesk.com`, `utest.com`, `trada.com` or `freelancer.com`.

[3] In February 2012, only 62,507 out of 443,606 workers had completed at least one hour of work or earned $1 on `odesk.com`, while 43,899 out of 376,966 workers had completed at least one job on `vworker.com`.

sequentially choosing different actions over time. Among existing MABs, one particularly pertinent piece of work is the budget–limited MAB [9], which addresses a similar problem to the one of expert crowdsourcing. In particular, within budget–limited MABs, the actions have different costs (i.e. the price of hiring different experts), and are constrained by a certain total budget (i.e. the crowdsourcing budget of the employer). However, it is not directly applicable to the expert crowdsourcing setting, because it is assumed that individual workers can perform an unlimited amount of tasks and indeed the optimal solution of the budget–limited MAB often assigns most tasks to a single worker. This is not realistic in crowdsourcing, where, due to the workers' individual preferences and other commitments, they cannot be assumed to complete an arbitrary number of tasks.

To address these shortcomings, we first extend the budget–limited MAB model to fit the expert crowdsourcing problem. We denote this new model as the *bounded MAB*. In particular, it describes the expert crowdsourcing problem as a budget-limited MAB with the additional constraint that workers can only complete a limited (bounded) number of tasks. Note that existing MAB algorithms do not address this limitation. Thus, they may fail in tackling the bounded MAB, since they may provide an invalid solution (i.e. a solution that exceeds the task limits). Given this, we develop a novel algorithm, called *bounded ε–first*, that efficiently tackles the bounded MAB as follows: To deal with the unknown performance characteristics of workers, our algorithm divides its budget into two amounts (as dictated by an ε parameter) to be used in two sequential phases — an initial *exploration* phase, during which it uniformly samples the performance of a wide range of workers using the first part of its budget, and an *exploitation* phase, during which it selects only the best workers using its remaining budget. In the latter, the algorithm chooses the best set of workers by solving a *bounded knapsack* problem [4]. The intuition behind the use of the bounded knapsack is that if we knew the real expected value of each worker's performance utility, then the expert crowdsourcing problem could be reduced to a bounded knapsack problem. However, since the bounded knapsack is NP–hard, an exact algorithm (i.e. a method that provides optimal solution) might not be able to guarantee polynomial running time. Thus, we use an efficient approximation approach, *bounded greedy* [4], to estimate the optimal solution of the bounded knapsack. We show that using this algorithm allows us to establish theoretical guarantees for its performance. More specifically, we prove that the *performance regret* (i.e. the difference between the performance of a particular algorithm and that of the theoretical optimal solution) of the bounded ε–first approach is at most $O\left(B^{\frac{2}{3}}\right)$ with a high probability, where B is the total budget. This *sub–linear* theoretical bound implies that our algorithm has the *zero–regret* property; that is, as B is increased, the *average regret* (i.e. the performance regret divided by the total budget) tends to 0. Note that this property is a key measure of efficiency within the bandit literature. Indeed, the zero–regret property guarantees that our algorithm *asymptotically converges* to the optimal solution with probability 1 as B tends to infinity (for more details, see [10]). In addition, to demonstrate the empirical efficiency of the proposed approach, we evaluate its performance by using real data from `odesk.com`, a prominent expert crowdsourcing site. In carrying out this work, we advance the state of the art as follows:

- We propose the first approach that addresses the expert crowdsourcing problem.
- We show that our approach outperforms current crowdsourcing techniques by up to 155%, and achieves 75% of the optimal.

In addition, we make theoretical contributions to MABs as follows:

- We introduce a new version of MABs, called the bounded MAB model, that extends the budget–limited MAB by taking the limits of single arm pulls into account.
- We propose bounded ε–first, the first algorithm that efficiently tackles the bounded MAB model.
- We devise the first theoretically proven upper bound for the performance regret of the bounded ε–first algorithm.

The remainder of this paper is structured as follows. In Section 2, we formally describe the problem we address, in Section 3, we outline our algorithm and then analyse its performance bounds in Section 4. In Section 5, we evaluate it empirically and Section 6 concludes.

2 MODEL DESCRIPTION

We first introduce the bounded MAB model (Section 2.1). Following this, we describe the expert crowdsourcing problem, and show how we can map it to the bounded MAB model (Section 2.2).

2.1 Bounded Multi–Armed Bandits

The budget–limited MAB model consists of a slot machine with N arms, denoted by $1, 2, \ldots, N$. At each time step t, an agent chooses a *non–empty* subset $S(t) \subseteq \{1, \ldots, N\}$ to pull (action). By pulling arm i, the agent has to pay a pulling cost, denoted by c_i, and receives a non–negative reward drawn from a distribution associated with that specific arm. The agent has a cost budget B, which it cannot exceed during its operation time (i.e. the total cost of pulling arms cannot exceed this budget limit). Now, since reward values are typically bounded in real–world applications, we assume that each arm's reward distribution has bounded supports. Let μ_i denote the mean value of the rewards that the agent receives from pulling arm i. Within our model, the agent's goal is to maximise the sum of rewards it earns from pulling the arms of the machine, with respect to the budget B. However, the agent has no initial knowledge of the μ_i of each arm i, so it must learn these values in order to deduce a policy that maximises its sum of rewards. Given this, our objective is to find the optimal pulling algorithm, which maximises the expectation of the total reward that the agent can achieve, without exceeding B.

Formally, let A be an arm–pulling algorithm, giving a finite sequence of pulls. Let $N_i^B(A)$ be the random variable that represents the total number of pulls of arm i by A, with respect to the budget limit B.[4] Thus, we have:

$$N_i^B(A) = \sum_t I\{i \in S^A(t)\},$$

where $S^A(t)$ is the subset that A chooses to pull at time step t and $I\{i \in S^A(t)\}$ denotes the indicator function whether arm i is chosen to be pulled at t. To guarantee that the total cost of the sequence A cannot exceed B, we have:

$$P\left(\sum_i^N N_i^B(A)\, c_i \leq B\right) = 1.$$

In addition, within our model, we assume that the agent cannot pull each arm i more than L_i times in total. That is:

$$\forall i: \quad P\left(N_i^B(A) \leq L_i\right) = 1.$$

Now, let $G^B(A)$ be the total reward earned by using A to pull the arms within budget limit B. The expectation of $G^B(A)$ is:

$$\mathbb{E}\left[G^B(A)\right] = \sum_i^N \mathbb{E}\left[N_i^B(A)\right]\mu_i.$$

[4] Note that $N_i^B(A)$ is a random variable since the behaviour of A depends on the observed rewards.

Then, let A^* denote an optimal solution that maximises the expected total reward, that is:

$$A^* = \arg\max_A \sum_i^N \mathbb{E}\left[N_i^B(A)\right]\mu_i.$$

Note that in order to determine A^*, we have to know the value of μ_i in advance, which does not hold in our case. Thus, A^* represents a theoretical optimum value, which is unachievable in general (but which we will use in Section 5 to benchmark our approach).

Nevertheless, for any algorithm A, we can define the regret for A as the difference between the expected total reward for A and that of the theoretical optimum A^*. More precisely, letting $R(A)$ denote the regret, we have the following:

$$R^B(A) = \mathbb{E}\left[G^B(A^*)\right] - \mathbb{E}\left[G^B(A)\right].$$

The objective here is to derive a method of generating a sequence of arm pulls that minimises this regret for the class of bounded MAB problems defined above.

Note that if we set the limits $L_i = \infty$ for each arm i (i.e. there is no pull limit) and we restrict $|S(t)| = 1$ for each t (i.e. the agent can only pull a single arm at each time step), we get the budget–limited MAB, and in addition, if we set $B = \infty$ (there is no budget limit either), we get the standard MAB model (for more details, see [9, 10]).

2.2 Expert Crowdsourcing

Given the bounded MAB model above, we now show how to map the expert crowdsourcing problem to bounded MABs. In particular, within an expert crowdsourcing system, an employer (agent) can assign tasks to a finite set of workers. This set of workers is usually determined through an open call for participation by the employer, to which qualified and available workers respond.[5] Each worker i corresponds to an arm and assigning a single task to that worker can be regarded as pulling the arm. This incurs a cost c_i that is set by the worker, and the outcome of the assignment is of variable utility with unknown mean μ_i (this corresponds to the rewards in the bounded MAB). As described in Section 1, each worker i has a different maximum number of tasks L_i that can be assigned to it. Finally, the employer has a total budget B to spend on crowdsourcing and it wishes to maximise the overall sum of the achieved utility.

To illustrate this, an employer may wish to carry out a large software development project, where each task represents a single hour of work by one of the workers. The utility generated by such a task is the number of working features that meet certain quality requirements. However, workers charge different prices per hour, c_i, and have different skill levels, represented by their expected number of working features they can implement per hour, μ_i. The employer now has a set budget to spend on developers, e.g., $B = \$5{,}000$, and wishes to maximise the total number of working features.[6] In so doing, it wants to choose the best subset of workers who provide the optimal solution. However, the employer has to take into account the working hour preferences of each worker, that limits the total number of hours a worker can work on the project.

Given the mapping and the illustrative example above, the mapping between expert crowdsourcing and bounded MABs is trivial. With a slight abuse of notation, hereafter we will use both standard terms of MAB (i.e. arms, pulls, and agent) and expert crowdsourcing (i.e. workers, task assignment, employer). In what follows, we propose an efficient algorithm to tackle the bounded MAB. We then continue with its theoretical and empirical performance analysis.

[5] To illustrate this, although there are 100,000s of workers on `oDesk.com`, typically only up to 20 respond to each such job advert (see Figure 1 for an example).

[6] This is a realistic budget — in February 2012, over $19 million were spent on `oDesk.com`, with an average spend per project of over $4,000.

3 THE BOUNDED $\mathcal{E}$–FIRST ALGORITHM

Recall that within our setting, μ_i are unknown *a priori*. Given this, the agent has to *explore* these values by repeatedly pulling a particular arm in order to estimate its expected reward value. However, if it solely focuses on exploration, the agent typically fails to maximise (i.e. *exploit*) the total expected reward. In contrast, if it stops exploring too quickly, it may fail in determining the best arms to pull. Given this, the key challenge of bounded MABs (and of other bandit models in general) is to find an efficient trade–off between exploration and exploitation. Within this section, we propose an efficient algorithm that efficiently trades off exploration with exploitation by splitting exploration from exploitation. In so doing, we first describe the exploration phase of the algorithm in Section 3.1. Following this, we describe the exploitation phase in more detail (Section 3.2).

3.1 Uniform Exploration

Within the exploration (or trial) phase, we dedicate an ε portion of budget B to estimate the expected reward values of the arms. First, we repeatedly pull the arms in the first $\left\lfloor \frac{\varepsilon B}{\sum_{i=1}^N c_i} \right\rfloor$ time steps. That is, $S(t) = \{1, \ldots, N\}$ if $1 \leq t \leq \left\lfloor \frac{\varepsilon B}{\sum_{i=1}^N c_i} \right\rfloor$. Following this, we sort the arms by their cost in an increasing (non–decreasing) order, and we sequentially pull the arms starting from the one with lowest cost, one after the other, until we exceed the remaining budget. We repeat the last step until none of the arms can be further pulled. Given this, if x_i^{explore} denotes the number of times we pull arm i within the exploration phase, we have $\left\lfloor \frac{\epsilon B}{\sum_{i=1}^N c_i} \right\rfloor \leq x_i^{\text{explore}}$. The reason of choosing this method is that, since we do not know which arm will be chosen to pull in the exploitation phase, we need to treat them equally in the exploration phase. Hereafter we refer to the allocation sequence performed by the uniform algorithm as A_{uni}.

3.2 Bounded Knapsack based Exploitation

In this section, we first introduce the foundation of the method used in this phase of our algorithm, the bounded knapsack problem. We then describe an efficient approximation method for solving this knapsack problem, which we subsequently use in the second phase of our bounded ϵ-first algorithm.

The bounded knapsack problem is formulated as follows. Given N types of items, each type i has a corresponding value v_i, and weight w_i. In addition, there is also a knapsack with weight capacity C. The bounded knapsack problem selects integer units of those types that maximise the total value of items in the knapsack, such that the total weight of the items does not exceed the knapsack weight capacity. However, each item i cannot be chosen more than L_i times. That is, the goal is to find the *non–negative integers* $x_1, x_2, \ldots, x_N$ that

$$\max \sum_{i=1}^N x_i v_i \quad \text{s.t.} \quad \sum_{i=1}^N x_i w_i \leq C, \quad \forall i: \quad 0 \leq x_i \leq L_i. \qquad (1)$$

Note that if we set each $L_i = 1$, we get the standard knapsack (or the $0-1$ knapsack) model. Since the bounded knapsack is a well–known *NP*–hard problem, exact algorithms (i.e. methods that achieve optimal solutions) cannot guarantee low computation cost.[7] However,

[7] There are pseudo–polynomial exact algorithms, but as we will show later, we can achieve efficient performance with polynomial approximations.

near–optimal approximation methods have been proposed to solve this problem, such as bounded greedy or greedy (a detailed survey of these algorithms can be found in [4]). In particular, here we make use of a simple, but efficient, approximation method, the *bounded greedy* algorithm, which has $O(N \log N)$ computational complexity, where N is the number of item types [4]. This algorithm works as follows: Let $\frac{v_i}{w_i}$ denote the *density* of type i. At the beginning, we sort the item types in order of the value of their density. This needs $O(N \log N)$ computational complexity. Then in the first round of this algorithm, we identify the item type with the highest density and select as many units of this item as are feasible, without either exceeding the knapsack capacity or its item limit L_i. Then, in the second round, we identify the densest item among the remaining feasible items (i.e. items that still fit into the residual capacity of the knapsack), and again select as many units as are feasible, without exceeding the remaining capacity or the corresponding item limit. We repeat this step in each subsequent round, until there is no feasible item left. Clearly, the maximal number of rounds is N. The reason for choosing this algorithm is that it provides a well behaved sequence of items (i.e. they are ordered by density), that can be efficiently exploited in the theoretical performance analysis.

Now, we reduce the task assignment problem in the exploitation phase to a bounded knapsack problem as follows. Let $\hat{\mu}_i$ denote the estimate of μ_i after the exploration phase. This estimate can be calculated by simply taking the average of received reward samples from arm i. Given this we aim to solve the following integer program:

$$\max \sum_{i=1}^{N} \hat{\mu}_i x_i^{\text{exploit}} \quad \text{s.t.} \quad \sum_{i=1}^{N} c_i x_i^{\text{exploit}} \leq (1-\epsilon)\, B, \qquad (2)$$
$$\forall i: \quad 0 \leq x_i^{\text{exploit}} \leq L_i - x_i^{\text{explore}}.$$

where x_i^{exploit} is the number of times we pull arm i in the exploitation phase. In order to solve this problem, we use the abovementioned bounded greedy algorithm for the bounded knapsack. Having the value of each x_i^{exploit}, we now run the exploitation algorithm as follows: At each subsequent time step t, if the number of times arm i has been pulled does not exceed x_i^{exploit}, then we pull that arm at t. Hereafter we refer to this exploitation approach as A_{greedy}.

4 PERFORMANCE ANALYSIS

In this section, we first derive an upper bound for the bounded ε–first algorithm, for any given ε value. We then show that by efficiently tuning the value of ε, we can refine the upper bound to $O\left(B^{\frac{2}{3}}\right)$.

Recall that both A_{uni} and A_{greedy} together form sequence $A_{\epsilon-\text{first}}$, which is the policy generated by the bounded ϵ–first algorithm. The expected reward for this policy can be expressed as the sum of the expected performance of A_{uni} and A_{greedy}. That is:

$$G^B(A_{\epsilon-\text{first}}) = G^{\varepsilon B}(A_{\text{uni}}) + G^{(1-\varepsilon)B}\left(A_{\text{greedy}}\right), \qquad (3)$$

Now, without loss of generality, we assume that the reward distribution of each arm has support in $[0,1]$, and the pulling cost $c_i > 1$ for each i (our result can be scaled for different size supports and costs as appropriate). In what follows, we first derive lower bounds for $G^{\varepsilon B}(A_{\text{uni}})$ and $G^{(1-\varepsilon)B}\left(A_{\text{greedy}}\right)$ independently. Then, putting these together, we have a lower bound for the expected reward of $A_{\epsilon-\text{first}}$. Following this, we derive an upper bound for the expected reward of the optimal sequence A^*. The difference between the lower bound of $G^B(A_{\epsilon-\text{first}})$ and the upper bound of $G^B(A^*)$ then gives us the upper bound of the performance regret of our proposed algorithm. However, this bound is constructed using Hoeffding's inequality, so it is correct only with a certain probability. Specifically, it is correct with probability $(1-\beta)^N$, where $\beta \in (0,1)$ is a predefined confidence parameter (i.e. the confidence with which we want the upper bound to hold) and N is the number of arms. As a result, we have:

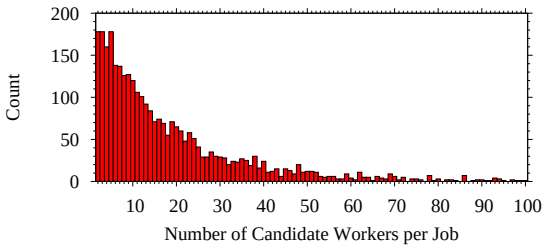

Figure 1. Distribution of applicants for jobs with "Java" keyword on `oDesk.com`.

Theorem 1 *Let $0 < \varepsilon, \beta < 1$. Suppose that $\varepsilon B \geq \sum_{j=1}^{N} c_j$. With probability $(1-\beta)^N$, the performance regret of the bounded ε–first approach is at most*

$$2 + \varepsilon B d_{\max} + 2N\left(\sqrt{\frac{B\left(-\ln\frac{\beta}{2}\right)\sum_{j=1}^{N} c_j}{\varepsilon}}\right), \qquad (4)$$

where $d_{\max} = \max_{i \neq j}\left|\frac{\mu_i}{c_i} - \frac{\mu_j}{c_j}\right|$ (i.e. the largest distance between different density values).

The proof of the theorem can be found in the Appendix. Now, by optimally tuning the value ε so that the upper bound given in Theorem 1 is minimised, we get:

Theorem 2 *Let ε_{opt} denote the value that minimises Equation 4. By setting the exploration budget to be $B\varepsilon_{\text{opt}}$, the regret of the bounded ε–first algorithm is at most*

$$2 + 3B^{\frac{2}{3}}\left(N^2\left(-\ln\frac{\beta}{2}\right)\sum_{j=1}^{N} c_j d_{\max}\right)^{\frac{1}{3}}.$$

That is, the upper bound can be tightened to $O\left(B^{\frac{2}{3}}\right)$. The proof only requires elementary algebra, and is omitted for brevity.

5 EXPERIMENTAL EVALUATION

While we have so far developed theoretical upper bounds for the performance regret of our algorithm, we now turn to practical aspects and examine its performance in a realistic setting. This allows us to investigate whether the algorithm achieves a high utility when applied to practical expert crowdsourcing problems. To this end, we run the algorithm on a range of problems from a large real–world dataset and compare its results with a number of benchmarks. In the following, we first outline the dataset we use to generate our experiments, then describe the benchmarks and finally detail our results.

5.1 Experimental Setup

To test our algorithm on realistic settings, we use real data from the expert crowdsourcing website `oDesk.com`.[8] Specifically, we assume an employer wishes to crowdsource a large-scale software project and is looking to hire Java experts. Since only a small fraction of all registered Java experts will be available at any time, we determine the number of applicants by sampling from the real historical distribution of applicants per Java–related job. This distribution is shown in Figure 1 (we consider only closed jobs and truncate the distribution to the interval $[2, 100]$, as smaller jobs are trivial and as there were a small number of extremely large outliers).

[8] This data is available through their API at `developers.odesk.com`.

	Small	Moderate	Large	Extreme
Bounded ε–first (ε = 0.15)	**59.88(0.35)**	**707.14(3.49)**	**3,833.8(18.61)**	**11,065(54.07)**
Budget–limited ε–first (ε = 0.05)	36.61(0.25)	360.41(1.55)	1,873(7.8)	4,062.8(16.14)
Budget–limited ε–first (ε = 0.10)	48.62(0.27)	382.72(1.56)	1,910.8(7.81)	4,347(16.09)
Budget–limited ε–first (ε = 0.15)	44.03(0.26)	374.15(1.55)	1,951.7(7.82)	4,206.1(16.11)
Trialsourcing	53.29(0.28)	362.80(1.61)	1,804.6(7.86)	3,864.5(16.38)
Random	26.34(0.2)	186.63(0.36)	991.2(6.97)	2,345.6(16.44)
Uniform	24.91(0.08)	135.23(0.55)	723.11(4.25)	2,167.1(13.79)
Optimal	**98.09(0.53)**	**946.66(2.1)**	**4,917.1(20.17)**	**14,102(58.77)**

Table 1. Performance evaluation of the algorithms in different job settings with small (B = 500), moderate (B = 5,000), large (B = 30,000) and extremely large (B = 100,000) budgets. The numbers represent the total collected utility of each algorithm.

To determine the characteristics of those workers, we sample them from the set of more than 30,000 Java experts registered on the website. For each expert i, we use their real advertised hourly costs for c_i, and we randomly determine their task limits L_i by drawing from the discrete uniform distribution on $[1, 5{,}000]$ (since real data on these limits is not available through the API). That is, a worker would spend up to more than a working year on average on a job. Finally, we determine the worker's utility distribution as the sum of two random variables, $0.9 \cdot R_i + 0.1 \cdot U(0, 1)$, where R_i is the empirical distribution of the user's actual ratings obtained on previous jobs[9] and $U(0, 1)$ is the continuous uniform distribution on the interval $[0, 1]$ (to add a small amount of noise). Trivially, μ_i is then $0.9 \cdot \mathbb{E}[R_i] + 0.05$.

5.2 Benchmarks

To demonstrate that our algorithm outperforms the state of the art, we compare its performance to a number of benchmark methods:

1. **Budget–limited ε–first**: a budget–limited MAB algorithm that assigns all tasks to a single agent during the exploitation phase without considering its task limits [9].
2. **Trialsourcing**: an existing approach that is used on the expert crowdsourcing website `vworker.com`. This first assigns a single task to each of the applicants and then chooses the best-performing worker out of these until it reaches its task limit, followed by the second-best, and so on. This algorithm can be regarded as a simpler version of the bounded ε–first with only one round of exploration.
3. **Random**: this algorithm randomly chooses a single worker to whom it assigns all tasks. This represents a typical expert crowdsourcing task allocation, where the employer chooses an applicant from some preferred prior distribution (see, e.g. `freelancer.com` or `utest.com`). Within our experiments, we sample this applicant from a uniform prior distribution (we have also tested with other priors without any significant improvements).
4. **Uniform**: this approach uniformly assigns tasks to all applicants. We include this to test the efficiency of pure exploration (i.e. uniform task assignment).
5. **Optimal**: this is a *hypothetical* optimal algorithm with full knowledge of each worker's mean μ_i. As such, it is an upper bound on our algorithm's performance.

[9] Ratings on `oDesk.com` are 1 – 5 stars, which we map to the interval $[0, 1]$. Note we use this only to generate realistic distributions and assume R_i is unknown to our agent. To avoid bias when only few ratings are available, we pad this empirical distribution with samples from $U(0, 1)$ until it is based on at least five samples.

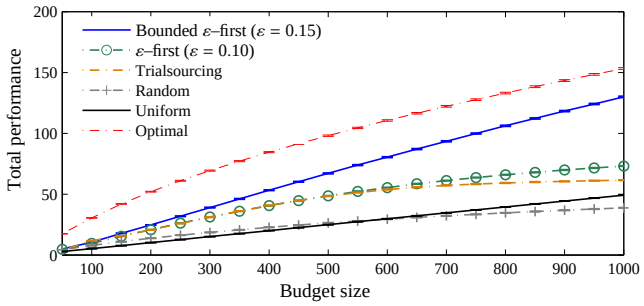

Figure 2. Performance evaluation of the algorithms in case of jobs with small budgets (smaller than $1,000).

5.3 Results

To analyse the behaviour of each algorithm in different job scenarios, we vary the budget B. In particular, we first focus on four different job types: (i) small (B = \$500); (ii) moderate (B = \$5,000); (iii) large(B = \$30,000); and (iv) extremely large (B = \$100,000). These are realistic values based on real jobs that have been advertised on `oDesk.com`. Additionally, for each budget, we re–sample the number and set of experts 10,000 times to achieve statistical significance, and we calculate 95% confidence intervals for all results. These results are depicted in Table 1 (with the 95% confidence intervals shown in brackets). Here, we set the ε value of our algorithm to 0.15, while the ε value of the budget–limited ε–first is set to 0.05, 0.1, and 0, 15, respectively (we have also tested with different ε values, which result in the same broad trends).

As we can see from the results, our algorithm typically outperforms the others by up to 155%. In particular, it outperforms the budget–limited ε–first (which is the best benchmark algorithm) by 23% in the case of a small budget (ε = 0.1 for the budget–limited algorithm). In addition, our method outperforms this benchmark by 85%, 100%, and 155% in the cases of moderate, large, and extremely large budgets, respectively. We also observe that we cannot achieve a high performance without taking task limits into account (as is the case with the budget–limited ε–first), or without having sufficient exploration (as in the case of trialsourcing). Similarly, simple algorithms such as pure exploration or random task allocation do not provide good performance either. Note that our algorithm approaches the theoretical optimum by up to 75% (in the cases of moderate, large and extreme budgets), while it achieves 61% of the optimal solution's performance in the scenario with small budgets.

Note that around 80% of the jobs on `odesk.com` have a budget smaller than \$1,000. Given this, we next further analyse the performance of the algorithms within this budget range. The results are depicted in Figure 2. As we can see, for jobs with very small budgets (i.e. smaller than \$100), the performance of our algorithm is similar to that of the budget–limited ε–first and trialsourcing. This is due to the fact that with a small budget, longer exploration is a luxury, and thus, those approaches perform well with only a small budget for exploration. However, if the budget is higher than \$100, our algorithm clearly outperforms the others by up to 60%. We can also observe that the uniform and random algorithms are clearly worse than our approach for any budget size.

Another interesting set of jobs is those with large budgets, as they present long–term investments that require careful task allocation. Thus, we also vary the budget B from \$5,000 to \$20,000, to analyse the performance of the algorithms. In fact, this range covers 77% of large jobs (i.e. jobs with budget > \$5,000). From Figure 3, we can see that our algorithm typically outperforms the others by up to 200%, and it approaches about 85% of the optimum.

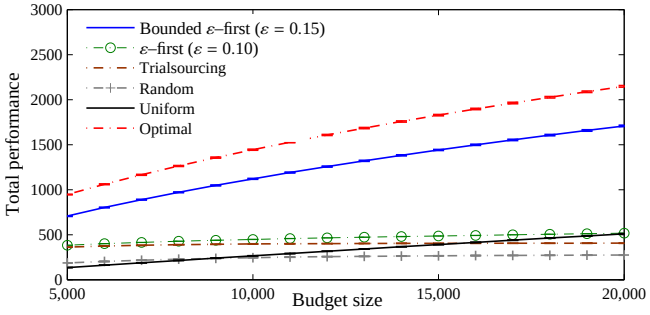

Figure 3. Performance evaluation of the algorithms in case of jobs with large budgets (between \$5,000 and \$20,000).

6 CONCLUSIONS AND FUTURE WORK

In this paper, we introduced the expert crowdsourcing problem with variable worker performance and heterogenous costs. In this problem, an employer wishes to assign tasks within a limited budget to a set of workers such that its total utility is maximised. To solve this problem, we introduced a new MAB model, the bounded MAB with a limited number of pulls per arm. Given this, we proposed a simple, but efficient, bounded ε–first based algorithm that uses a uniform pull strategy for exploration, and a bounded knapsack based approach for exploitation. We proved that this algorithm has a $O\left(B^{\frac{2}{3}}\right)$ theoretical upper bound for its performance regret. Finally, we demonstrated that our algorithm outperforms state-of-the-art crowdsourcing algorithms within this domain by up to 155%, and also consistently achieves up to 75% of the theoretical optimal. As a result, our work could potentially form an efficient basis to crowdsourcing websites which aim to provide efficient teams of experts, so-called *curated crowds*, to meet their customers' business requirements (e.g. `geniusrocket.com` or `blurgroup.com`).

Note that our approach does not exploit the fact that in many real–world applications employers typically have additional information about the applicants, which could be used to find the best workers more quickly (e.g. reputation ratings or lists of qualifications). However, as this information might not be accurate either, it is not trivial how to efficiently handle it in practice. In future work, we intend to extend our analysis to such settings.

REFERENCES

[1] F. P. Brooks, *The mythical man-month : essays on software engineering*, Addison-Wesley Pub. Co, 1995.

[2] P. Dai, Mausam, and D. S. Weld, 'Artificial intelligence for articial artificial intelligence', in *AAAI 2011*, pp. 1153–1159, (2011).

[3] J. J. Horton and L. B. Chilton, 'The labor economics of paid crowdsourcing', in *EC'10*, pp. 209–218, (2010).

[4] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*, Springer, 2004.

[5] A. Kittur, E. H. Chi, and B. Suh, 'Crowdsourcing user studies with mechanical turk', in *CHI'08*, pp. 453–456, (2008).

[6] M. Marge, S. Banerjee, and A.I. Rudnicky, 'Using the amazon mechanical turk for transcription of spoken language', in *IEEE ICASSP'10*, pp. 5270 –5273, (2010).

[7] W. Mason and D. J. Watts, 'Financial incentives and the performance of crowds', in *HCOMP'09*, pp. 77–85, (2009).

[8] E. Simpson, S. J. Roberts, A. Smith, and C. Lintott, 'Bayesian combination of multiple, imperfect classifiers', in *NIPS 2011*, (2011).

[9] L. Tran-Thanh, A. Chapman, J. E. Munoz de Cote, A. Rogers, and N. R. Jennings, 'Epsilon–first policies for budget–limited multi–armed bandits', *AAAI'10*, 1211–1216, (2010).

[10] J. Vermorel and M. Mohri, 'Multi-armed bandit algorithms and empirical evaluation', *ECML'05*, 437–448, (2005).

[11] P. Welinder, S. Branson, S. Belongie, and P. Perona, 'The multidimensional wisdom of crowds', in *NIPS 2010*, 2424–2432, (2010).

APPENDIX: PROOFS

We now sketch the proof of Theorem 1. In so doing, we define some terms. Let $i^{\max} = \arg\max_j \frac{\mu_j}{c_j}$. Similarly, let $i^{\min} = \arg\min_j \frac{\mu_j}{c_j}$. Now, if we relax the bounded knapsack problem defined in Section 3.2 (see Equation 1) such that x_i can be fractional, we get the *fractional* bounded knapsack. It is easy to show that the bounded greedy algorithm provides an optimal solution to the fractional bounded knapsack, and this optimal solution is always at least as high as the optimal solution of the (integer) bounded knapsack (for more details, see [4]). Given this, let $\langle \hat{x}_1, \ldots, \hat{x}_N \rangle$ denote the optimal solution of the knapsack problem defined in Equation 2 (i.e. the problem we have to solve within the exploitation phase), that we can get by using the bounded greedy method. In addition, recall that if we knew the real value of each μ_i, the optimal strategy within the exploitation phase would be the optimal solution of the bounded knapsack problem which we get from the former knapsack by replacing $\hat{\mu_i}$ with μ_i. Thus, let $\langle x_1^+, \ldots, x_N^+ \rangle$ denote the optimal solution of the fractional relaxation of this bounded knapsack, that we also get by using the bounded greedy method. Next, we prove the following auxiliary lemmas.

Lemma 3 $\mathbb{E}\left[G^{\varepsilon B}\left(A_{\text{uni}}\right)\right] \geq \epsilon B\left(\mu_{i^{\min}}/c_{i^{\min}}\right) - 1.$

Lemma 4 $\mathbb{E}\left[G^{(1-\varepsilon)B}\left(A_{\text{greedy}}\right)\right] \geq \sum_{j=1}^{N} \hat{x}_j \mu_j - 1.$

Lemma 5 $\mathbb{E}\left[G^{(1-\varepsilon)B}\left(A^*\right)\right] \leq \sum_{j=1}^{N} x_j^+ \mu_j.$

Proof of Lemma 3. It is easy to show that for any arm j, $\sum_{i=1}^{N} c_i x_i^{\text{explore}} \geq \epsilon B - c_j$, since none of the arms can be pulled after the stop of A_{uni} without exceeding ϵB. Furthermore, $\mu_i = c_i\left(\mu_i/c_i\right) \geq c_i\left(\mu_{i^{\min}}/c_{i^{\min}}\right)$. Recall that $\mu_i \leq 1$. Thus:

$$\sum_{i=1}^{k} x_i^{\text{explore}} \mu_i \geq \left(\sum_{i=1}^{k} x_i^{\text{explore}} c_i\right) \frac{\mu_{i^{\min}}}{c_{i^{\min}}} \geq \left(\epsilon B - c_{i^{\min}}\right) \frac{\mu_{i^{\min}}}{c_{i^{\min}}} \geq \frac{\epsilon B \mu_{i^{\min}}}{c_{i^{\min}}} - 1.$$

Proof of Lemma 4. Without loss of generality, assume that the bounded greedy chooses the arms to pull in the order of $1, 2, \ldots, N$. Let b denote the largest index such that $\hat{x}_b \neq 0$. Since A_{greedy} also uses the bounded greedy, we can easily show that $x_i^{\text{exploit}} = \hat{x}_i$ for $i < b$ and $x_b^{\text{exploit}} = \lfloor \hat{x}_b \rfloor$ (if $i > b$, then $x_i^{\text{exploit}} = \hat{x}_i = 0$). Thus $\mathbb{E}\left[G^{(1-\varepsilon)B}\left(A_{\text{greedy}}\right)\right] = \sum_{j=1}^{b-1} \hat{x}_j \mu_j + \lfloor \hat{x}_b \rfloor \mu_b \geq \sum_{j=1}^{b-1} \hat{x}_j \mu_j + (\hat{x}_b - 1)\mu_b$, which concludes the proof, since $\mu_b \leq 1$.

Proof of Lemma 5. The lemma follows from the fact that the optimal solution of the bounded knapsack cannot exceed that of its fractional counterpart.

Proof sketch of Theorem 1. Using Hoeffding's inequality for each arm i, and for any positive δ_i, we have: $P\left(|\hat{\mu}_i - \mu_i| \geq \delta_i\right) \leq 2\exp\left\{-2\delta_i^2 x_i^{\text{explore}}\right\}$. By setting $\delta_i = \sqrt{\frac{-\ln \frac{\beta}{2}}{2x_i^{\text{explore}}}}$, it is easy to prove that, with probability $(1-\beta)^N$, $|\hat{\mu}_i - \mu_i| \leq \delta_i$ holds for each arm i. Hereafter, we stricly focus on this case. It is easy to show that $\mathbb{E}\left[G^B\left(A^*\right)\right] \leq \varepsilon B \frac{\mu_{i^{\max}}}{c_i^{\max}} + \mathbb{E}\left[G^{(1-\varepsilon)B}\left(A^*\right)\right]$. This implies that

$$R\left(A_{\varepsilon-\text{first}}\right) \leq \left(\varepsilon B \frac{\mu_{i^{\max}}}{c_i^{\max}} - \mathbb{E}\left[G^{\varepsilon B}\left(A_{\text{uni}}\right)\right]\right) + \left(\mathbb{E}\left[G^{(1-\varepsilon)B}\left(A^*\right)\right] - \mathbb{E}\left[G^{(1-\varepsilon)B}\left(B_{\text{greedy}}\right)\right]\right). \quad (5)$$

Using Lemma 3, we can bound the first term on the right hand side as follows:

$$\varepsilon B \frac{\mu_{i^{\max}}}{c_i^{\max}} - \mathbb{E}\left[G^{\varepsilon B}\left(A_{\text{uni}}\right)\right] \leq \varepsilon B\left(\frac{\mu_{i^{\max}}}{c_{i^{\max}}} - \frac{\mu_{i^{\min}}}{c_{i^{\min}}}\right) + 1 = \varepsilon B d_{\max} + 1. \quad (6)$$

We now turn to bound the second term on the right hand side of Equation 5. From Lemmas 4 and 5 we get: $\mathbb{E}\left[G^{(1-\varepsilon)B}\left(A^*\right)\right] - \mathbb{E}\left[G^{(1-\varepsilon)B}\left(B_{\text{greedy}}\right)\right] \leq \sum_{j=1}^{N} x_j^+ \mu_j - \sum_{j=1}^{N} \hat{x}_j \mu_j + 1$. Since $\langle \hat{x}_1, \ldots, \hat{x}_N \rangle$ is the optimal solution of the fractional bounded knapsack that we have to solve at the exploitation phase, we have: $\sum_{j=1}^{N} \hat{x}_j \hat{\mu}_j \geq \sum_{j=1}^{N} x_j^+ \hat{\mu}_j$. Similarly, we have $\sum_{j=1}^{N} x_j^+ \mu_j \geq \sum_{j=1}^{N} \hat{x}_j \mu_j$ (since $\langle x_1^+, \ldots, x_N^+ \rangle$ is the real optimal solution). Recall that $|\hat{\mu}_i - \mu_i| \leq \delta_i$ holds for each arm i. This implies that $\sum_{j=1}^{N} x_j^+ \mu_j - \sum_{j=1}^{N} \hat{x}_j \mu_j \leq \sum_{j=1}^{N} \delta_j \left(x_j^+ + \hat{x}_j\right)$. Note that $\hat{x}_j \leq \frac{(1-\varepsilon)B}{c_j} \leq (1-\varepsilon)B$. Similarly we have: $x_j^+ \leq (1-\varepsilon)B$. This implies that

$$\mathbb{E}\left[G^{(1-\varepsilon)B}\left(A^*\right)\right] - \mathbb{E}\left[G^{(1-\varepsilon)B}\left(B_{\text{greedy}}\right)\right] \leq (1-\varepsilon)B \sum_{j=1}^{N} 2\delta_j \leq B \sum_{j=1}^{N} 2\delta_j. \quad (7)$$

Recall that $\delta_i = \sqrt{\frac{-\ln \frac{\beta}{2}}{2x_i^{\text{explore}}}}$ and $x_i^{\text{explore}} \geq \left\lfloor \frac{\varepsilon B}{\sum_{j=1}^{N} c_j} \right\rfloor \geq \frac{\varepsilon B}{2\sum_{j=1}^{N} c_j}$. The second inequality can be easily proven by using elementary algebra. Substituting these into Equation 7, and combining with Equation 6 we conclude the proof.