Efficient Crowdsourcing of Unknown Experts using Bounded Multi–Armed Bandits

Long Tran-Thanh, Sebastian Stein, Alex Rogers and Nicholas R. Jennings

University of Southampton, Southampton, SO17 1BJ, UK
{ltt08r,ss2,acr,nrj}@ecs.soton.ac.uk

Abstract

Increasingly, organisations flexibly outsource work on a temporary basis to a global audience of workers. This so-called *crowdsourcing* has been applied successfully to a range of tasks, from translating text and annotating images, to collecting information during crisis situations and hiring skilled workers to build complex software. While traditionally these tasks have been small and could be completed by non-professionals, organisations are now starting to crowdsource larger, more complex tasks to experts in their respective fields. These tasks include, for example, software development and testing, web design and product marketing. While this emerging *expert crowdsourcing* offers flexibility and potentially lower costs, it also raises new challenges, as workers can be highly heterogeneous, both in their costs and in the quality of the work they produce. Specifically, the utility of each outsourced task is uncertain and can vary significantly between distinct workers and even between subsequent tasks assigned to the same worker. Furthermore, in realistic settings, workers have limits on the amount of work they can perform and the employer will have a fixed budget for paying workers. Given this uncertainty and the relevant constraints, the objective of the employer is to assign tasks to workers in order to maximise the overall utility achieved. To formalise this expert crowdsourcing problem, we introduce a novel multi-armed bandit (MAB) model, the bounded MAB. Furthermore, we develop an algorithm to solve it efficiently, called bounded ε-first, which proceeds in two stages: exploration and exploitation. During exploration, it first uses εB of its total budget B to learn estimates of the workers' quality characteristics. Then, during exploitation, it uses the remaining $(1-\varepsilon)B$ to maximise the total utility based on those estimates. Using this technique allows us to derive an $O\left(B^{\frac{2}{3}}\right)$ upper bound on its performance regret (i.e., the expected difference in utility between our algorithm and the optimum), which means that as the budget B increases, the regret tends to 0. In addition to this theoretical advance, we apply our algorithm to real-world data from oDesk, a prominent expert crowdsourcing site. Using data from real projects, including historic project budgets, expert costs and quality ratings, we show that our algorithm outperforms existing crowdsourcing methods by up to 300%, while achieving up to 95% of a hypothetical optimum with full information.

Keywords:
Crowdsourcing, machine learning, multi-armed bandits, budget limitation

1. Introduction

In recent years, a wide range of organisations, including enterprises, governments, academic institutions and charities, have turned 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, often in exchange for financial remuneration. This so-called *crowdsourcing* promises considerable flexibility, as it quickly connects employers and workers across the globe without large recruitment overheads [40, 11].

A significant amount of existing research and technologies have so far concentrated on facilitating the crowdsourcing of small units of work (so-called "micro-tasks") that can be completed in minutes by non-professional labourers, including survey participation, audio clip transcription or image annotation [20, 24]. Here, workers are typically paid small, fixed amounts of money for each successfully completed work unit, or even perform the work for free in the presence of other non-monetary incentives [31]. Prominent examples of mature offerings in this space include Amazon's Mechanical Turk, Galaxy Zoo and Microtask.[1]

However, in contrast to this crowdsourcing of non-professionals, a growing number of businesses are 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] The rising popularity of this approach is evident in the scale of emerging intermediaries that connect employers and expert workers. As of August 2013, oDesk has 2.5m registered workers, while Freelancer has 6.7m, with both having witnessed an approximately two-fold increase in members within 2012.

Unlike the crowdsourcing of smaller and simpler units of work, expert crowdsourcing raises new challenges. First, the quality of a completed task can vary greatly, both between different workers and even between several tasks completed

[1] See `mturk.com`, `galaxyzoo.org` and `microtask.com`, respectively.
[2] For some examples of these, see `odesk.com`, `utest.com`, `trada.com` or `freelancer.com`.

by the same worker. For example, a highly-skilled software engineer might complete several times as many functions as an inexperienced worker in a single hour, but the same skilled engineer may occasionally struggle with a particular task, perhaps due to adverse personal circumstances [7]. This means that an employer needs to select workers carefully, in order to consistently achieve a high quality.

Second, the online labour market is inherently open and dynamic in nature, with a constant influx of new workers. Thus, there is typically little or no prior knowledge about the expected quality of a particular worker. To illustrate this, more than 96% of workers advertising on oDesk have not completed any significant amount of work in the past.[3] As a result, an employer will often need to recruit workers it has not previously dealt with and will only gain information about their performance during the course of a project.

Third, 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 charge from as little as $5 to over $200 for one hour of Web design work. Clearly, an employer here needs to balance the cost of workers with the quality of their work — while some workers may be cheaper than others, their quality could be considerably lower.

Finally, an employer in an expert crowdsourcing setting also has to take into account several real-world constraints. Typically, a project will have a fixed monetary budget that cannot be exceeded. Furthermore, workers cannot complete an arbitrary amount of work within the time scope of the project. In practice, each worker has a limit on the number of hours they can dedicate to a given project.

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.

To address these challenges, we turn to the field of multi-armed bandits (MABs), a class of problems dealing with decision-making under uncertainty [1]. These optimisation problems consider settings where actions (i.e., the pulling of a particular arm) have initially unknown rewards that have to be learnt through noisy obser-

[3] In August 2013, only 85,329 out of the 2.5m registered workers on oDesk had completed at least one hour of work or earned $1.

vations, and the goal is to maximise the total amount of rewards by sequentially choosing different actions over time. This corresponds exactly to choosing initially unknown workers in an expert crowdsourcing setting. However, as we discuss in Section 2, no existing MAB model considers the specific constraints of the expert crowdsourcing setting. While some work considers MAB problems with a fixed budget, termed budget-limited MABs [33], and proposes a budget-limited ε-first algorithm for this, their model does not consider task limits per worker.

Addressing this shortcoming, we propose the *bounded MAB*, a novel MAB model that builds on and extends the budget-limited MAB model to fit the expert crowdsourcing problem. Given this, we develop a new algorithm, called *bounded ε-first*, that efficiently tackles the bounded MAB. Unlike the budget-limited ε-first algorithm it is based on, our algorithm explicitly models and takes into account the task limits per worker. More specifically, it operates 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 [19].

The intuition behind the use of the bounded knapsack is that if we knew the real expected value of each worker's expected 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 the optimal solution) might not be able to guarantee a polynomial running time. Thus, we use an efficient approximation approach, *bounded greedy* [19], to estimate the optimal solution of the bounded knapsack.

Furthermore, 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 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 necessarily implies that our algorithm has the *zero-regret* property, a key measure of efficiency within the MAB literature. That is, as B increases, the *average regret* (i.e., the performance regret divided by the total budget) tends to 0. This property guarantees that our algorithm *asymptotically converges* to the optimal solution with probability 1 as B tends to infinity (for more details, see [36]). As this desirable theoretical property holds only in the limit, we also conduct extensive empirical experiments, in order to ascertain the efficiency of our proposed approach for realistic budgets. To this end, we use real historical data from projects carried out on oDesk, a prominent expert crowdsourcing website.

In carrying out this work, we advance the state of the art as follows:

- We propose the first principled approach that specifically addresses the expert crowdsourcing problem.
- We show that our approach outperforms current crowdsourcing techniques by up to 300% on a real-world dataset, and typically achieves around 90% 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 imposing a limit on the number of times a particular arm may be pulled.
- 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 article is structured as follows. In Section 2, we discuss related work. Then, in Section 3, we formally describe the expert crowdsourcing problem. In Section 4, we outline our algorithm and then analyse its performance bounds in Section 5. In Section 6, we evaluate the algorithm empirically and Section 7 concludes.

2. Related Work

A significant amount of research has been carried out in the general field of crowdsourcing and specifically how to deal with workers of varying quality and how the payments to workers influence the quality of their work. We discuss this work in Section 2.1. Then, in Section 2.2 we turn to the general field of multi-armed bandits, which are a natural model for the expert crowdsourcing setting we consider here.

2.1. Crowdsourcing

Crowdsourcing has received considerable attention in recent years, and there have been many successful applications. These include rapidly collecting information during a disaster [12], completing tasks that are difficult to automate and need to be solved by human workers [5, 39], running large-scale user studies (i.e., surveys)

[20] or contributing to scientific endeavours [9]. To support such applications, several mature platforms have emerged. Amazon's Mechanical Turk, for example, supports the large-scale distribution of micro-tasks to human workers, Ushahidi provides software for collecting information from the public, in particular during crisis situations, and Zooniverse hosts a range of large citizen science projects.[4] To exemplify the scale of these platforms, Amazon's Mechanical Turk lists between 100,000 and 200,000 available micro-tasks at any point in time, Ushahidi received approximately 40,000 reports during the 2010 earthquake in Haiti and Zooniverse currently has more than 700,000 volunteers.

In the context of these applications, some existing work has considered specifically how to deal with the highly heterogeneous performance quality of workers — one of the key challenges for expert crowdsourcing we identified in Section 1. In the crowdsourcing of micro-tasks, many approaches rely on redundantly allocating the same task to multiple workers and then selecting the best result or a consensus opinion, or on iteratively improving on the work of others [22]. In this context, Dai *et al.* [10] describe a decision-theoretic control mechanism that explicitly balances the benefit of further iterations of improvements with the cost this entails. Zaidan and Callison-Burch [39] apply both redundancy and iterative improvements to the problem of crowdsourcing translations, and they show how a classifier can accurately identify the best solutions based on a number of domain-specific features. Other work demonstrates how machine learning and statistical inference techniques can be used to build performance profiles of workers and combine their outputs in classification tasks to achieve a high overall accuracy [38], or to discard inaccurate workers entirely [37].

However, while these techniques deal with the heterogeneous quality of workers in settings with micro-tasks, they are less suitable for the expert crowdsourcing setting we consider. First, they assume that tasks are priced uniformly (or even carried out for free) and that the employer has little influence on selecting particular workers. Thus, the objective is typically to achieve the best possible performance given a fixed set of workers. In our setting, the employer has considerably more control over selecting individual workers, but also needs to take into account potentially highly heterogeneous worker costs. Furthermore, costs are generally higher in expert crowdsourcing, where experts often demand $10–50 per hour of work, compared to the few cents that are normally paid per micro-task. This makes it infeasible to allocate the same tasks redundantly to a large number of workers.

To address the specific challenges of expert crowdsourcing, a number of ad hoc approaches have appeared that are in use on existing crowdsourcing sites. For

[4] See `mturk.com`, `ushahidi.com` and `zooniverse.org`, respectively.

example, the expert crowdsourcing site vWorker has used an approach called *trialsourcing*.[5] Here, a subset of tasks of a larger project is sent to a large number of workers. Based on the quality of their output, the employer then picks the best worker and assigns all remaining tasks to him or her. Another approach that has appeared is the notion of a *curated crowd*, where the expert crowdsourcing site carefully selects and filters its workers based on the quality of their work. Examples of sites using this approach include Genius Rocket and Thinkspeed.[6] However, while these sites consider the heterogeneous quality of workers, they do not deal with task limits and require a labour-intensive manual selection process.

Another strand of work has looked at how to build systems that induce work of a higher quality. Morris *et al.* [28] show how *priming*, i.e., providing implicit cues to effect subconscious changes in behaviour, can be used to achieve higher performance in crowdsourcing tasks. Specifically, they demonstrate that showing positive images or playing positive music while collecting input for micro-tasks increases the productivity of workers. Similarly, Huang *et al.* [17] propose a system that automatically optimises the design of crowdsourcing tasks (including the provided incentives and the size, complexity and number of tasks) to maximise particular performance metrics. To exemplify this, they consider an image annotation task and show that up to 60–71% more unique high-quality tags can be obtained by carefully optimising the size and complexity of individual micro-tasks compared to a simple unoptimised baseline with the same budget and payment per tag. Other work has examined in detail how financial incentives affect the quality of work and the level of participation in a crowdsourcing settings [26, 16]. While the financial incentives are typically set by the workers, and therefore not directly controllable, in the expert crowdsourcing settings we consider, work on inducing higher a quality of work through priming or optimal task design is largely complementary to the work presented in this paper. Specifically, these techniques could be used to optimise how the requested work is presented to selected experts, in order to further increase productivity.

2.2. Multi-Armed Bandits

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 sequentially

[5] Note that vWorker (available at `vworker.com`) has been merged with Freelancer since the time of writing of this paper.

[6] See `www.geniusrocket.com` and `www.thinkspeed.com`.

choosing different actions over time [29, 1, 4]. In particular, a MAB model consists of a machine with K arms, each of which delivers rewards that are independently drawn from an unknown distribution when the machine's arm is pulled. Our goal is to choose which of these arms to play. At each time step, we pull one of the machine's arms and receive a reward (or payoff). The objective is to maximise the return; that is, to maximise the sum of the rewards received over a sequence of pulls. As the reward distributions differ from arm to arm, the goal is to find the arm with the highest expected payoff as early as possible, and then to keep gambling using that best arm [29, 4].

However, this MAB model gives an incomplete description of the sequential decision-making problem facing an agent in many real-world scenarios. To this end, a variety of other related models have been studied recently [2, 8, 13, 6]. Among existing MABs, one particularly pertinent piece of work is the budget-limited MAB [33, 35], 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). To tackle this problem, Tran-Thanh *et al*. proposed a number of efficient algorithms, such as the unbounded ε-first and KUBE [33, 35]. However, the budget-limited MAB model 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. Nevertheless, budget-limited MAB algorithms can form a good basis for benchmarks against our proposed method within the bounded settings, as they provide efficient solutions for related problems (see Section 6.2 for more details).

Another notable piece of related work is from Ho *et al.* [14], who also investigate a multi-armed bandit model in the crowdsourcing domain. In particular, they consider a problem where the system designer has to assign a task from a set of task types to an incoming worker (here, the set of task types represent the arms to be pulled). In this model, each type of task has a finite number of tasks, limiting the number of times they can be allocated to workers. The authors describe an algorithm that achieves near-optimal performance and they provide a competitive ratio. However, since their model does not include a total budget limit (only a limitation in the number of pulls per arm), it requires a different underlying solution technique (i.e., not the bounded knapsack model), and thus, it is not feasible for our setting.

Other work has considered the problem of pure exploration, or arm ranking, in bandit settings [25, 27, 3]. In particular, this problem focusses on identifying the ranking of the arms, given a threshold for the number of total pulls (budget). As we will explain later in Section 4.2, within the exploration phase, our bounded ε-first approach relies on an approximation method that aims to choose arms with highest reward-cost density values. Thus, the pure exploration problem can be regarded as a sub-problem within the exploration phase, where we aim to achieve efficient exploration (i.e., quickly identify the highest ranking arms). A number of algorithms have been proposed to tackle this problem, such as Hoeffding Races [25], Bernstein Races [27], and Successive Rejects (SR) [3]. However, as we will show both in theory (see Section 5.2) and in practice (see Section 6.4), replacing the uniform exploration phase of our algorithm with the above-mentioned techniques does not improve the performance of ε-first. Thus, these approaches do not outperform uniform exploration within our settings.

3. Model Description

We first introduce the bounded MAB model (Section 3.1). Following this, we describe the expert crowdsourcing problem, and show how we can map it to the bounded MAB model (Section 3.2).

3.1. 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 (its action). When 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). Since reward values are typically bounded in real-world applications, we assume that the reward distribution of each arm has a bounded support. 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 choose 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. Note that $N_i^B(A)$ is a random variable since the behaviour of A depends on the observed rewards. Thus, we have:

$$N_i^B(A) = \sum_t I\{i \in S^A(t)\}, \tag{1}$$

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:

$$P\left(\sum_{i=1}^{N} N_i^B(A)\, c_i \leq B\right) = 1, \tag{2}$$

where $P(\cdot)$ denotes the probability of an event. In addition, within our model, we assume that the agent cannot pull each arm i more than L_i times in total. That is:

$$\forall i: \quad P\left(N_i^B(A) \leq L_i\right) = 1. \tag{3}$$

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:

$$\mathbb{E}\left[G^B(A)\right] = \sum_{i=1}^{N} \mathbb{E}\left[N_i^B(A)\right]\mu_i. \tag{4}$$

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

$$A^* = \arg\max_A \sum_{i=1}^{N} \mathbb{E}\left[N_i^B(A)\right]\mu_i. \tag{5}$$

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 optimal algorithm, which is unachievable in general (but which we will use in Section 6 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 $R^B(A)$ denote the regret, we have the following:

$$R^B(A) = \mathbb{E}\left[G^B(A^*)\right] - \mathbb{E}\left[G^B(A)\right]. \tag{6}$$

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 [33, 36]).

3.2. 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.[7] 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 has a set budget to spend on developers, e.g., $B = \$5,000$, and wishes to maximise the total number of working features.[8] 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, which limits the total number of hours a worker can spend 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, and employer). In what follows, we propose an efficient algorithm to tackle the bounded MAB. We then continue with its theoretical and empirical performance analysis.

---

[7]To illustrate this, although there are 100,000s of workers on oDesk, typically only up to 20 respond to each such job advert (see Figure 1 on page 24 for the distribution of responses to adverts).

[8]This is a realistic budget — in August 2013, over $19 million were spent on oDesk, with an average spend per project of over $4,000.

4. The Bounded ε-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 the total expected reward (i.e., *exploit*). In contrast, if it stops exploring too quickly, it may fail to determine 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 a novel algorithm that efficiently trades off exploration with exploitation by splitting exploration from exploitation. The intuition behind this explicit distinction is that by doing so, we can control the degree of exploration by setting the value of ε, which becomes very useful for the theoretical analysis (see Section 5 for more details). Besides, this approach was shown to be efficient in many real-world applications, compared to other bandit based methods such as UCB or ε-greedy [30, 32, 33, 36]. In what follows, we first describe the exploration phase of the algorithm (Section 4.1), followed by its exploitation phase (Section 4.2).

4.1. 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 all 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 the lowest cost, one after the other, until the next pull would exceed the remaining budget. We repeat the last step until none of the arms can be further pulled with the remaining budget. Given this, if x_i^{explore} denotes the number of times we pull arm i within the exploration phase, we have $\left\lfloor \frac{\epsilon B}{\sum_{i=1}^{N} c_i} \right\rfloor \leq x_i^{\text{explore}}$. For the sake of simplicity, we assume that $L_i \geq x_i^{\text{explore}}$. Otherwise, we stop pulling arm i once L_i is reached. The reason for choosing this method is that, since we do not know which arms will be chosen 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}.

4.2. Bounded Knapsack-Based Exploitation

In order to describe the exploitation phase of the bounded ε-first algorithm, we start with the introduction of the bounded knapsack problem, which forms the foundation of the method used in this phase. We then describe an efficient approximation method for solving this knapsack problem, which we subsequently use in the exploitation phase.

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

$$\max \sum_{i=1}^{N} x_i v_i \quad \text{s.t.} \quad \sum_{i=1}^{N} x_i w_i \leq C, \quad \forall i: \quad 0 \leq x_i \leq L_i. \tag{7}$$

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 [19, 23], exact algorithms (i.e., methods that achieve optimal solutions) cannot guarantee a low computation cost.[9] However, 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 [19]). 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 [19]. The reason for this choice is that besides its efficiency, it provides a solution with specific properties that can be used for theoretical analysis (see Section 5 for more details).

The bounded greedy algorithm works as follows: Let $\frac{v_i}{w_i}$ denote the *density* of type i. At the beginning, we sort the item types by decreasing density. This has $O(N \log N)$ 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. Following this, in the second round, we identify the item with the highest density 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

[9] There are pseudo-polynomial exact algorithms such as dynamic programming or dominance relationship based approaches [23], but as we will show later, we can achieve efficient performance with polynomial approximations.

Algorithm 1 Bounded ε-First Algorithm

Exploration phase:
$t = 1$; $B_t^{\text{expl}} = \varepsilon B$;
while pulling is feasible **do**
    pull all the arms;
    $B_{t+1}^{\text{expl}} = B_t^{\text{expl}} - \sum_{k=1}^{N} c_k$; $t = t + 1$;
end while
while pulling is feasible **do**
    if $B_t^{\text{explore}} < \min_i c_i$ **then**
        STOP! {pulling is not feasible}
    end if
    pull arm $i(t)$, where $i(t) = t \mod N$ {choose the subsequent arm to pull};
    $B_{t+1}^{\text{expl}} = B_t^{\text{expl}} - c_{i(t)}$; $t = t + 1$;
end while
Exploitation phase:
use bounded greedy that solves Equation 8 to pull the arms;

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 the received reward samples from arm i. Given this, we aim to solve the following integer program:

$$\max \sum_{i=1}^{N} \hat{\mu}_i x_i^{\text{exploit}} \quad \text{s.t.} \quad \sum_{i=1}^{N} c_i x_i^{\text{exploit}} \leq (1-\epsilon) B, \qquad \forall i: \quad 0 \leq x_i^{\text{exploit}} \leq L_i - x_i^{\text{explore}}, \tag{8}$$

where x_i^{exploit} are the decision variables, representing the number of times we pull arm i in the exploitation phase. In order to solve this problem, we use the above-mentioned 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}. When used together with the uniform exploration technique described above, we refer to this algorithm as *bounded ε-first*, or $A_{\epsilon-\text{first}}$.

The pseudo code of the algorithm is depicted in Algorithm 1. In what follows, we formally examine the performance of this algorithm.

5. 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 $O\left(B^{\frac{2}{3}}\right)$ (Section 5.1). In addition, we also investigate the performance of the modified version of the ε-first, where the uniform exploration phase is replaced with Successive Rejects (SR), a state-of-the-art pure exploration algorithm [3]. In particular, we also provide a $O\left(B^{\frac{2}{3}}\right)$ bound for this modified version, however, with larger coefficient constants (Section 5.2). This implies that even with this more sophisticated exploration method, we cannot achieve a better performance, compared to that of uniform exploration.

5.1. Regret Bounds of ε-First with Uniform Exploration

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:

$$G^B\left(A_{\epsilon-\text{first}}\right) = G^{\varepsilon B}\left(A_{\text{uni}}\right) + G^{(1-\varepsilon)B}\left(A_{\text{greedy}}\right). \tag{9}$$

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). Let $i^{\max} = \arg\max_j \frac{\mu_j}{c_j}$. Similarly, let $i^{\min} = \arg\min_j \frac{\mu_j}{c_j}$. In addition, let $c_{\max} = \max_j \frac{\mu_j}{c_j}$, and $c_{\min} = \min_j \frac{\mu_j}{c_j}$, respectively. We state the following:

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

$$2 + \frac{c_{\min}\mu_{i^{\max}}}{c_i^{\max}} + \varepsilon B d_{\max} + 2N\left(\sqrt{\frac{B\left(-\ln\frac{1-\sqrt[N]{\beta}}{2}\right)\sum_{j=1}^{N} c_j}{\varepsilon}}\right), \tag{10}$$

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).

To prove this theorem, we will make use of the following version of Hoeffding's concentration inequality for bounded random variables:

Theorem 2 (Hoeffding's inequality [15]). *Let $X_1, X_2, \ldots, X_n$ denote the sequence of random variables with common range $[0, 1]$, such that for any $1 \le t \le n$, we have $\mathbb{E}[X_t | X_1, \ldots, X_{t-1}] = \mu$. Let $S_n = \frac{1}{n}\sum_{t=1}^{n} X_t$. Given this, for any $\delta \ge 0$, we have:*

$$P(S_n \ge \mu + \delta) \le e^{-2n\delta^2}, \tag{11}$$

$$P(S_n \le \mu - \delta) \le e^{-2n\delta^2}. \tag{12}$$

The proof can be found, for example, in [15].

Now, if we relax the bounded knapsack problem defined in Section 4.2 (see Equation 7) such that x_i can be fractional, we get the *fractional* bounded knapsack [19, 23]. Marcello and Toth (1990) proved 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 [19]).

Given this, let $\langle \hat{x}_1, \ldots, \hat{x}_N \rangle$ denote the optimal solution to the fractional relaxation of the knapsack problem given in Equation 8 (i.e., the problem we have to solve within the exploitation phase and that uses the estimated $\hat{\mu}_i$ values). In addition, let $\langle x_1^+, \ldots, x_N^+ \rangle$ denote the corresponding optimal solution to this problem when the true μ_i values are known. Recall that both of these solutions can be obtained using the bounded greedy algorithm. Next, we prove the following auxiliary lemmas:

Lemma 3. $\mathbb{E}\left[G^{(1-\varepsilon)B}(A^*)\right] \le \sum_{j=1}^{N} x_j^+ \mu_j$.

Lemma 4. $\mathbb{E}\left[G^{\varepsilon B}(A_{\text{uni}})\right] \ge \epsilon B \left(\mu_{i^{\min}} / c_{i^{\min}}\right) - 1$.

Lemma 5. $\mathbb{E}\left[G^{(1-\varepsilon)B}\left(A_{\text{greedy}}\right)\right] \ge \sum_{j=1}^{N} \hat{x}_j \mu_j - 1$.

Proof of Lemma 3. Note that the right hand side of the inequality is the optimal solution of the fractional bounded knapsack. In addition, the left hand side is the optimal solution of the integer bounded knapsack problem. Moreover, it is well established that the optimal solution of the fractional problem is always higher than that of the integer knapsack [23, 19]. This concludes the proof. □

Proof of Lemma 4. Note that for any arm j, $\sum_{i=1}^{N} c_i x_i^{\text{explore}} \ge \epsilon B - c_j$, since none of the arms can be pulled after the stop of A_{uni} without exceeding ϵB. Furthermore,

$$\mu_i = c_i \left(\frac{\mu_i}{c_i}\right) \ge c_i \left(\frac{\mu_{i^{\min}}}{c_{i^{\min}}}\right).$$

Recall that $\mu_i \le 1$. Thus:

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

□

Proof of Lemma 5. 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 for $i < b$:

$$x_i^{\text{exploit}} = \hat{x}_i,$$

and

$$x_b^{\text{exploit}} = \lfloor \hat{x}_b \rfloor .$$

Note that if $i > b$, then $x_i^{\text{exploit}} \geq 0$. Thus

$$\mathbb{E}\left[G^{(1-\varepsilon)B}\left(A_{\text{greedy}}\right)\right] \geq \sum_{j=1}^{b-1} \hat{x}_j\mu_j + \lfloor \hat{x}_b \rfloor \mu_b \geq \sum_{j=1}^{b-1} \hat{x}_j\mu_j + (\hat{x}_b - 1)\,\mu_b, \tag{13}$$

which concludes the proof, since $\mu_b \leq 1$. □

Proof of Theorem 1. Using Hoeffding's inequality for each arm i, and for any positive δ_i, we have:

$$P\left(|\hat{\mu}_i - \mu_i| \geq \delta_i\right) \leq 2e^{-2\delta_i^2 x_i^{\text{explore}}}.$$

By setting $\delta_i = \sqrt{\frac{-\ln \frac{1-\sqrt[N]{\beta}}{2}}{2x_i^{\text{explore}}}}$, we can prove that, with at least probability β,

$$|\hat{\mu}_i - \mu_i| \leq \delta_i$$

holds for each arm i. Hereafter, we strictly focus on this case. We first show that

$$\mathbb{E}\left[G^B\left(A^*\right)\right] \leq \varepsilon B \frac{\mu_{i^{\max}}}{c_i^{\max}} + \mathbb{E}\left[G^{(1-\varepsilon)B}\left(A^*\right)\right] + \frac{c_{\min}\mu_{i^{\max}}}{c_i^{\max}}. \tag{14}$$

In particular, let σ_i be the difference between the number of pulls of arm i within the optimal solution of $G^B(A^*)$ and that of $G^{(1-\varepsilon)B}(A^*)$. Note that σ_i can be negative. We know that:

$$\mathbb{E}\left[G^B\left(A^*\right)\right] = \sum_{i=1}^{N} \sigma_i\mu_i + \mathbb{E}\left[G^{(1-\varepsilon)B}\left(A^*\right)\right].$$

In addition, from [19, 23], we have:

$$\sum_{i=1}^{N} \sigma_i c_i \leq \varepsilon B + c_{\min},$$

where $c_{\min} = \min_i c_i$. By solving the relaxed unbounded knapsack (and allowing negative σ_i values as well), we have that

$$\sum_{i=1}^{N} \sigma_i \mu_i \leq (\varepsilon B + c_{\min}) \frac{\mu_{i^{\max}}}{c_i^{\max}} = \varepsilon B \frac{\mu_{i^{\max}}}{c_i^{\max}} + \frac{c_{\min} \mu_{i^{\max}}}{c_i^{\max}}.$$

Putting the previous inequalities together, we get Equation 14. This implies that

$$\begin{aligned} R^B(A_{\varepsilon-\text{first}}) \quad &\leq \quad \left(\varepsilon B \frac{\mu_{i^{\max}}}{c_i^{\max}} - \mathbb{E}\left[G^{\varepsilon B}(A_{\text{uni}}) \right] \right) \\ &+ \quad \left(\mathbb{E}\left[G^{(1-\varepsilon)B}(A^*) \right] - \mathbb{E}\left[G^{(1-\varepsilon)B}\left(A_{\text{greedy}}\right) \right] \right). \end{aligned} \tag{15}$$

Using Lemma 4, we can bound the first term on the right-hand side as follows:

$$\varepsilon B \frac{\mu_{i^{\max}}}{c_i^{\max}} - \mathbb{E}\left[G^{\varepsilon B}(A_{\text{uni}}) \right] \leq \varepsilon B \left(\frac{\mu_{i^{\max}}}{c_{i^{\max}}} - \frac{\mu_{i^{\min}}}{c_{i^{\min}}} \right) + 1 = \varepsilon B d_{\max} + 1. \tag{16}$$

We now turn to bound the second term on the right-hand side of Equation 15. From Lemmas 5 and 3 we get:

$$\mathbb{E}\left[G^{(1-\varepsilon)B}(A^*) \right] - \mathbb{E}\left[G^{(1-\varepsilon)B}\left(A_{\text{greedy}}\right) \right] \leq \sum_{j=1}^{N} x_j^+ \mu_j - \sum_{j=1}^{N} \hat{x}_j \mu_j + 1.$$

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

$$\sum_{j=1}^{N} x_j^+ \mu_j \geq \sum_{j=1}^{N} \hat{x}_j \mu_j.$$

This is due to $\langle x_1^+, \ldots, x_N^+ \rangle$ being 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

$$\mathbb{E}\left[G^{(1-\varepsilon)B}(A^*) \right] - \mathbb{E}\left[G^{(1-\varepsilon)B}\left(A_{\text{greedy}} \right) \right] \leq (1-\varepsilon) B \sum_{j=1}^{N} 2\delta_j \leq B \sum_{j=1}^{N} 2\delta_j. \quad (17)$$

Recall that $\delta_i = \sqrt{\frac{-\ln \frac{1-\sqrt[N]{\beta}}{2}}{2 x_i^{\text{explore}}}}$ 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 17, and combining with Equation 16 we conclude the proof. □

Now, by using elementary algebra, we can show that by setting

$$\varepsilon = \left(\frac{N^2}{d_{\max}^2 B} \left(-\ln \frac{1-\sqrt[N]{\beta}}{2} \right) \sum_{j=1}^{N} c_j \right)^{\frac{1}{3}}, \quad (18)$$

the upper bound given in Theorem 1 is minimised. Thus, we get:

Theorem 6. *Let ε_{opt} denote the abovementioned value that minimises Equation 10 and $0 < \beta < 1$. By setting the exploration budget to be $B\varepsilon_{\text{opt}}$, with at least probability β, the regret of the bounded ε-first algorithm is at most*

$$2 + \frac{c_{\min} \mu_{i^{\max}}}{c_i^{\max}} + 3B^{\frac{2}{3}} \left(N^2 \left(-\ln \frac{1-\sqrt[N]{\beta}}{2} \right) \sum_{j=1}^{N} c_j d_{\max} \right)^{\frac{1}{3}}. \quad (19)$$

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. This result implies that the regret bound is guaranteed to be sub–linear (i.e., less than $O(B)$), and thus, our algorithm converges to the optimal solution in an asymptotic manner. In particular, for any $0 < \alpha < 1$, there is a sufficiently large B_0 such that for any budget size $B > B_0$, the performance of our algorithm for that budget size is guaranteed to be better than an α-ratio of the optimal solution.

Algorithm 2 Exploration with Successive Rejects

Initialisation phase:
$A_1 = \{1, 2, \ldots, N\}$, set n_k as given in Equation 20, $i = 1$;
$B^{\text{res}} = \varepsilon B - \sum_{k=1}^{N} n_k c_k$;
while $B^{\text{res}} > 0$ **do**
  pull arm i, $B^{\text{res}} = B^{\text{res}} - c_i$;
  $i = (i + 1) \mod N$;
end while
Exploration phase:
$t = 1$;
while $t < K$ **do**
  pull each arm in A_t with $(n_t - n_{t-1})$ times;
  eliminate the arm with lowest estimated mean reward from A_t and denote the new set with A_{t+1};
  $t = t + 1$;
end while

5.2. Regret Bounds of ε-First with Successive Rejects Exploration

Recall the performance of the exploitation phase mainly relies on how accurately we can estimate the correct ranking (in decreasing order) of the density of the arms. This motivates the usage of the uniform distribution, which explores all arms equally, and thus, the ranking of the arms can be efficiently identified. However, due to the nature of the bounded greedy algorithm, the performance of the exploitation phase in fact typically relies only on the highest-ranking arms, and not the full ordering, as we may run out of budget before reaching the lower-ranking arms. Thus, it is not obvious whether we should focus only on high-ranking arms, instead of aiming to identify the full ordering (as we do with the uniform exploration). Given this, we now analyse the performance of a modified version of the ε-first algorithm, where the uniform exploration approach is replaced with other exploration methods that do not aim to estimate the correct full ordering. As mentioned in Section 2, there are a number of algorithms designed for this problem. Among them, Successive Rejects (SR) proposed by Audibert *et al.* (2010), provably outperforms the other methods (see [3] for more details). Given this, we replace the uniform exploration approach with SR, in order to study whether we can improve the performance of bounded ε-first. In what follows, we first describe how SR can be adapted to our setting and then we provide theoretical regret bounds.

The pseudo code of the SR-based exploration can be found in Algorithm 2. Let $l(N) = \frac{1}{2} + \sum_{j=2}^{N-1} \frac{1}{j}$ and $n_0 = 0$. For each $k \in \{1, 2, \ldots, N-1\}$, we set the value of

n_k as follows:

$$n_k = \left\lfloor \frac{1}{l(N)} \frac{\varepsilon B}{(N+1-k)c_{\max}} \right\rfloor, \tag{20}$$

where $c_{\max} = \max_j c_j$. Within the initialisation phase, we set $B^{\text{res}} = \varepsilon B - \sum_{k=1}^{N} n_k c_k$ and allocate the residual budget B^{res} among the arms (lines $3-7$). Within the exploration phase, at each time step t, we pull all the arms within the set of arms A_t exactly $(n_t - n_{t-1})$ times. We then eliminate the arm with the lowest estimated mean reward from the set of arms and continue with the next time step (lines $10-14$).

Following Audibert *et al.* (2010), we can show that in SR, there is exactly one arm which is pulled n_1 times, one n_2 times, ..., and two that are pulled n_{N-1} times. Furthermore, the total consumed budget does not exceed εB. In particular, without loss of generality, we assume that the order of arm elimination is $1, 2, \ldots, N-1$. We have:

$$\sum_{k=1}^{N} n_k c_k \leq \sum_{k=1}^{N} n_k c_{\max} \leq \sum_{k=1}^{N-1} \frac{1}{l(N)} \frac{\varepsilon B}{(N+1-k)} + \frac{1}{l(N)} \frac{\varepsilon B}{2} \leq \varepsilon B \frac{l(N)}{l(N)} = \varepsilon B.$$

Given this, the regret of this approach can be bounded as follows.

Theorem 7. *Let* $0 < \varepsilon, \beta < 1$. *Suppose that* $\varepsilon B \geq \sum_{j=1}^{N} c_j$. *With at least probability* β*, the performance regret of the bounded* ε*-first with SR exploration approach is at most*

$$2 + \frac{c_{\min}\mu_{i^{\max}}}{c_i^{\max}} + \varepsilon B d_{\max} + 2N \sqrt{\frac{(N+3)\ln N}{2}} \sqrt{\frac{B\left(-\ln \frac{1-\sqrt[N]{\beta}}{2}\right)c_{\max}}{\varepsilon}}. \tag{21}$$

In addition, by optimally tuning ε*, we can show that the regret is at most*

$$2 + \frac{c_{\min}\mu_{i^{\max}}}{c_i^{\max}} + 3B^{\frac{2}{3}} \left(N^2 \frac{(N+3)\ln N}{2} c_{\max} \left(-\ln \frac{1-\sqrt[N]{\beta}}{2}\right) d_{\max}\right)^{\frac{1}{3}}. \tag{22}$$

Note that for $N \geq 9$, this regret bound is clearly worse than that of the ε-first approach with uniform exploration (see Equation 19), as $\frac{(N+3)\ln N}{2} c_{\max} > \sum_{j=1}^{N} c_j$ holds for this case. In particular, for $N \geq 9$, we have

$$\frac{(N+3)\ln N}{2} > (N+3),$$

and thus,

$$\frac{(N+3)\ln N}{2} c_{\max} > (N+3)c_{\max} > \sum_{j=1}^{N} c_j.$$

This implies that for $N \geq 9$, by using uniform exploration, we can achieve a better regret bound, compared to exploration with SR.[10]

Proof of Theorem 7. Similar to the proof of Theorem 1, we can show that with at least β probability, the regret is at most

$$2 + \frac{c_{\min}\mu_{i^{\max}}}{c_i^{\max}} + \varepsilon B d_{\max} + 2B \sum_{j=1}^{N} \delta_j, \tag{23}$$

where $\delta_i = \sqrt{\frac{-\ln \frac{1-\sqrt[N]{\beta}}{2}}{2x_i^{\text{explore}}}}$. Without loss of generality, we assume that within the SR exploration, the order of arm elimination is $1, 2, \ldots, N-1$. From the definition of SR, we have that for each $k \in \{1, 2, \ldots, N-1\}$:

$$x_k^{\text{explore}} \geq \left\lfloor \frac{\varepsilon B}{l(N)(N+1-k)c_{\max}} \right\rfloor \geq \frac{\varepsilon B}{2l(N)(N+1-k)c_{\max}},$$

and

$$x_N^{\text{explore}} \geq \frac{\varepsilon B}{4l(N)c_{\max}}.$$

That is, we get

$$\begin{aligned}
\sum_{j=1}^{N} \delta_j &\leq \sum_{j=1}^{N-1} \sqrt{\frac{-l(N)(N+1-k)c_{\max} \ln \frac{1-\sqrt[N]{\beta}}{2}}{\varepsilon B}} + \sqrt{\frac{-2l(N)c_{\max} \ln \frac{1-\sqrt[N]{\beta}}{2}}{\varepsilon B}} \\
&\leq \sqrt{\frac{-l(N)c_{\max} \ln \frac{1-\sqrt[N]{\beta}}{2}}{\varepsilon B}} \left(\sqrt{2} + \sum_{j=2}^{N} \sqrt{j} \right).
\end{aligned} \tag{24}$$

We now rely on the following fact:

$$l(N) = \frac{1}{2} + \sum_{j=2}^{N} \frac{1}{j} \leq \ln N.$$

In addition, we can use induction to show that

$$\sqrt{2} + \sum_{j=2}^{N} \sqrt{j} \leq N \sqrt{\frac{N(N+1)+1}{2N}} \leq N \sqrt{\frac{N+3}{2}}.$$

[10] For the case of $N < 9$, it is not always guaranteed that the coefficient constant of SR is worse than that of uniform exploration, as it also depends on the values of c_j.

These imply that

$$\sum_{j=1}^{N} \delta_j \leq \sqrt{\frac{-l(N)c_{\max} \ln \frac{1-\sqrt[N]{\beta}}{2}}{\varepsilon B}} N \sqrt{\frac{N+3}{2}}, \tag{25}$$

which concludes the proof. In addition, by optimally tuning the value of ε, we achieve the regret bound given in Equation 22. □

6. 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 realistic settings. This is necessary and complements our theoretical analysis, because the latter concentrates on asymptotic performance bounds as the budget tends to infinity and for arbitrary performance distributions. In this section, we are now interested in how the algorithm performs for realistic budget sizes and performance distributions that occur in real expert crowdsourcing settings. 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 (Section 6.1), then describe the benchmarks (Section 6.2) and detail our results (Section 6.3). In addition, we also compare the performance of our uniform exploration approach with other exploration methods in Section 6.4.

6.1. Experimental Setup

To test our algorithm on realistic settings, we use real data from the expert crowdsourcing website oDesk.[11] 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 was a small number of extremely large outliers).

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

[11]This data is available through their API at `developers.odesk.com` and was downloaded in February 2012.

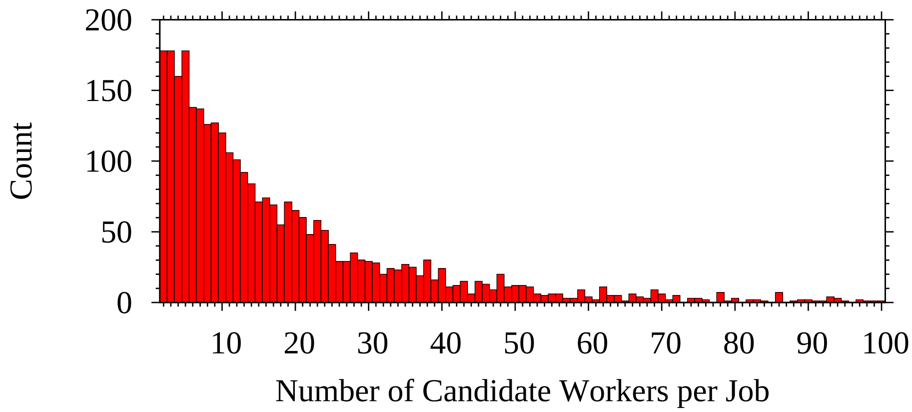

Figure 1: Distribution of applicants for jobs with “Java” keyword on oDesk.

limits L_i by drawing from the discrete uniform distribution on $[1, 5000]$ (since real data on these limits is not available through the API).[12] That is, a worker would spend between a single hour up to approximately two and a half working years on a project.

Finally, to establish the worker’s utility distribution, we use real feedback ratings received from employers for previously completed projects (indicating the quality of their work), as well as some additional noise to account for variability in the work they perform. Specifically, the quality distribution is 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[13] and $U(0, 1)$ is the continuous uniform distribution on the interval $[0, 1]$ (to add a small amount of noise). Thus, a sample from this distribution represents the quality of the work achieved in one hour and ranges from 0 to 1, where 0 is the worst, making no contribution to the employer’s overall utility and 1 is the highest quality achievable. Trivially, the expected quality, μ_i, is then $0.9 \cdot \mathbb{E}[R_i] + 0.05$.

[12]Note that task limits are measured in hours, and 5000 working hours limit is approximately 2 years. This value is reasonable as some workers on oDesk are willing to work on large projects for more than a year.

[13]Ratings on oDesk 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.

6.2. 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 practically efficient budget-limited MAB algorithm that assigns all tasks to a single expert, that can provide the highest total quality with respect to his task limit, during the exploitation phase [33]. This algorithm has been demonstrated to be the most efficient among budget-limited MAB algorithms in practice (see [32] for more details).
2. **Trialsourcing**: an existing approach that is used on the expert crowdsourcing website vWorker (see Section 2.1). This first assigns a single task to each of the applicants and then chooses the worker with the highest estimated quality-cost density out of these until that worker reaches its task limit. This algorithm can be regarded as a simpler version of the budget-limited 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. **Bounded KUBE**: this is a modified version of KUBE, a budget-limited MAB algorithm with optimal theoretical performance regret bounds (see [32, 35] for more details), that is adapted to our bounded knapsack setting. In particular, at each time step, bounded KUBE solves a corresponding bounded knapsack problem and uses the frequency of occurrence of the arms within the optimal solution of the knapsack problem as the distribution from which it randomly chooses an arm to pull. In contrast to our approach, bounded KUBE does not have theoretical performance guarantees, and it is also computationally more expensive (see Section 6.3 for more details). By comparing against this benchmark algorithm, we aim to demonstrate that the ϵ-first approach is typically more efficient than other, more sophisticated, approaches in practice, especially in the budget-limited settings (for similar discussions, see, e.g., [32, 36, 21]).
6. **Simplified bounded KUBE**: this is a simplified version the the bounded KUBE. In particular, in order to improve the computational efficiency of bounded KUBE, it does not solve the corresponding bounded knapsack problem as the bounded KUBE algorithm does (note that bounded knapsack

problems are NP-hard). Instead, the simplified bounded KUBE approach approximates the optimal solution by using the bounded greedy method (see [32, 35] for more details).

7. **Optimal**: this is a *hypothetical* optimal algorithm with full knowledge of each worker's mean quality μ_i. We approximate its performance in this section using the solution to the corresponding fractional bounded knapsack problem. Hence, any results we present are an upper bound on the performance of any algorithm.

6.3. Results

Throughout this section, we adopt the basic setup described in Section 6.1, but vary a number of controlled parameters to evaluate how our algorithm performs in a variety of settings. Specifically, we first consider settings with varying budgets, to represent smaller or larger project sizes (Section 6.3.1). Then, we examine how the algorithm performs when the number of candidates is varied (Section 6.3.2), and then we investigate how varying correlations between the quality and cost of a worker affect the performance of the algorithm (Section 6.3.3).

6.3.1. Performance with Variable Budgets

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). Throughout our experiments, we also restrict the set of candidates for a particular budget, as highly-paid workers are unlikely to apply for a low-budget project. Thus, for the four settings used here, we restrict the candidates to those that charge at most \$30, \$50, \$100 and \$200, respectively. These are realistic values based on real jobs that have been advertised on oDesk. 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 existing algorithms by up to 78%. In particular, it outperforms the budget-limited ε-first 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. This significant improvement over the benchmarks is due to several reasons. First, allocating a

	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)
Bounded KUBE	46.9(0.33)	397.14(3.06)	2,721.04(18.19)	–
Simplified bounded KUBE	28.24(0.31)	277.42(3.25)	2,176.46(20.36)	6,307.07(49.88)
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.

part of the budget to exploration ensures that our algorithm identifies the best-performing workers, which are then exploited with the remaining budget. Second, unlike most of the other benchmarks, it also takes into account task limits in an intelligent way and therefore hires several high-quality workers in parallel while satisfying their respective task constraints. Other benchmarks, such as the budget-limited ε-first algorithm, due to their non-efficient way of handling task limits, here often fail to achieve high performance. As the budget rises, it becomes increasingly likely that this limit is met, explaining the relatively higher performance of our approach compared to the benchmarks in settings with larger budgets. Compared to the budget-limited ε-first algorithm, the other benchmarks perform even worse — trialsourcing lacks the necessary exploration to identify the best-performing workers, while the uniform and random approaches do not take into account the workers' performance distributions at all.

We can also observe that our algorithm outperforms the modified versions of KUBE, a theoretically efficient budget-limited MAB algorithm, by up to 78%. In particular, bounded KUBE always outperforms its simplified counterpart. However, it also incurs a significantly higher computational cost, and thus, it is not possible to use bounded KUBE to calculate the solution for the case of an extremely large budget within reasonable time.[14] More specifically, apart from the modified versions of KUBE, all the algorithms achieve less than 1 second running time for the small, moderate and large cases, and they still need less than 2 seconds for the extremely large case. On the other hand, the simplified bounded KUBE approach needs approximately 7 seconds for the large case, and 17 seconds for the extremely large case. In addition, the running time of the bounded KUBE method is around 1 hour for the large case, and it cannot achieve any results for the extremely large case. Nevertheless, both bounded KUBE and its simplified version are outperformed by our approach. One possible reason is that KUBE needs more exploration to find efficient solutions, and thus, typically provides less efficiency in cases with lower budgets (for more discussions, see [32, 36]).

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. This confirms the theoretical regret bounds that show that our solution quality approaches the optimum with a growing budget.

While these results cover a wide range of possible budget levels, around 80% of

[14] All the numerical tests appearing in this paper are performed on a personal computer, Intel® Xeon® CPU W3520 @2.67GHz with 12GB RAM running the Fedora 18 operation system.

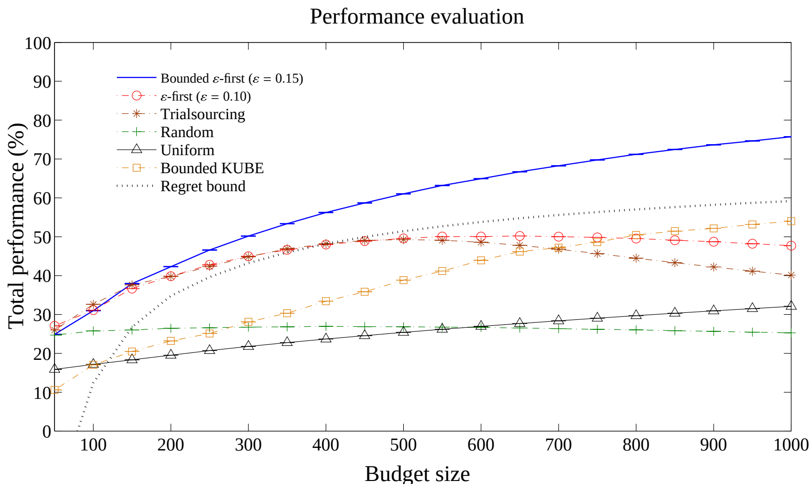

Figure 2: Performance ratio of the algorithms (compared to the optimal solution) in case of jobs with small budgets (smaller than $1,000).

the jobs on oDesk have a budget smaller than $1,000. Given this, we next further analyse the performance of the algorithms within this budget range (restricting the set of candidates to those that charge at most $30 per hour). The results are depicted in Figure 2 (for ease of comparison, the performance is now expressed as a percentage of the optimal). We also depict the regret bound calculated from Theorem 1 as well, to demonstrate that our algorithm indeed can guarantee the regret bound. Note that hereafter we only show the results of the bounded KUBE (as it has been shown in Table 1 that it outperforms its simplified counterpart).

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 67%. As before, this is because our approach identifies the best-performing workers and deals with the task limits of workers (which start to become an issue with a rising budget). We can also observe that the uniform and random algorithms are clearly worse than our approach for any budget size, as they do not take into account the workers' performance characteristics at all. In addition, it can clearly be seen that our algorithm is the only one that can guarantee the regret bound (as the others all perform worse than the regret bound as the budget rises above $150).

Interestingly, the budget-limited ε-first and trialsourcing algorithms first per-

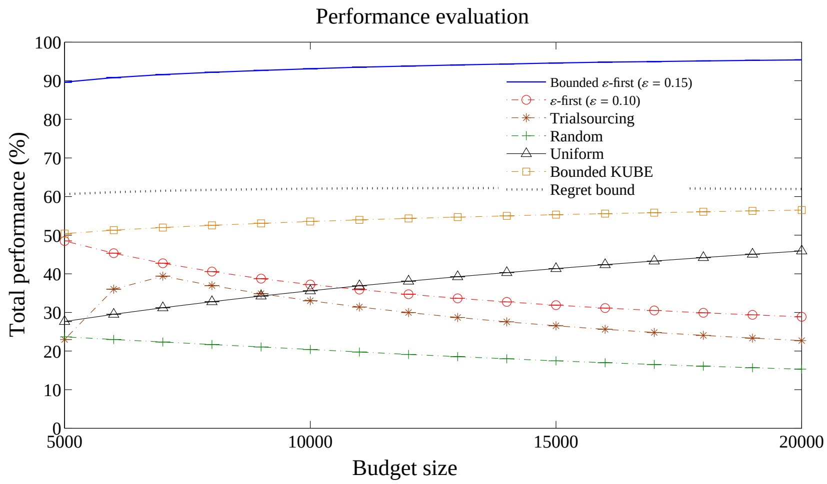

Figure 3: Performance ratio of the algorithms (compared to the optimal solution) in case of jobs with large budgets (between \$5,000 and \$20,000).

form better with an increasing budget (compared to the optimal), but their performance eventually starts to decrease. This is due to two opposing factors — initially, an increasing budget means the approaches can spend more of their budget on exploiting the best workers; however, eventually the task limits become an issue, resulting in workers hitting their limits more frequently. This trend is not displayed by the uniform approach, which consistently performs better with an increasing budget. This is because it is not affected by task limits and because the relative advantage of the optimal solution decreases as more workers are included due to the larger budget. We can also observe that when the budget is small, the performance of bounded KUBE is not efficient, compared to the others, as it needs more time to converge.

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 (for consistency fixing the set of candidates to those that charge at most \$50 per hour). In fact, this range covers 77% of large jobs on oDesk (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 achieves around 95% of the optimum. Here, the significantly higher performance compared to the benchmarks is due to the ability of our algorithm to take into account the workers' task limits and divide the high budget between several workers. In addition, our algorithm outperforms the others by up to 162% (for the case of budget B = \$10,000). We can also see that when the

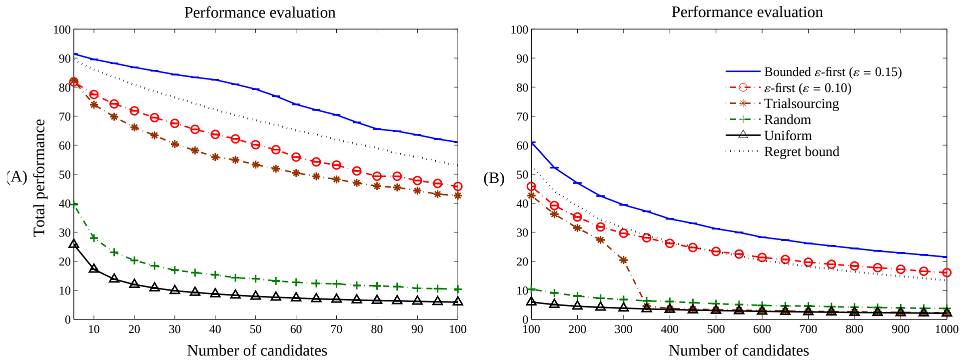

Figure 4: Performance ratio of the algorithms (compared to the optimal solution) with budget B = \$5,000 and: (A) small number of candidates (varied between 5 and 100); (B) large number of candidates (varied between 100 and 1000).

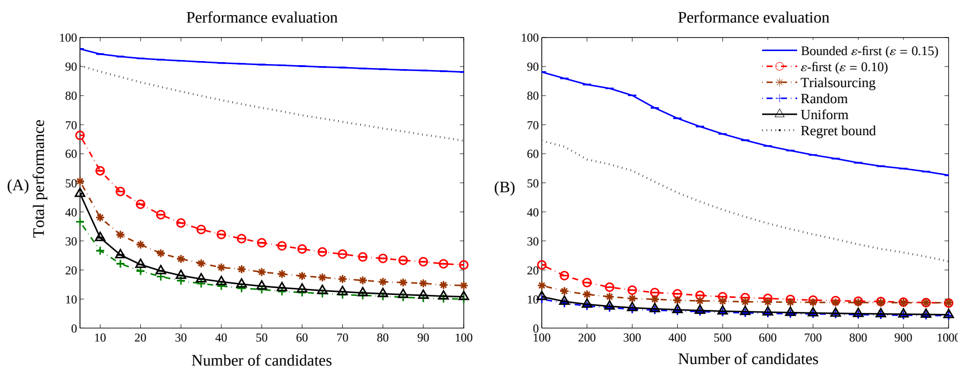

Figure 5: Performance ratio of the algorithms (compared to the optimal solution) with budget B = \$30,000 and: (A) small number of candidates (varied between 5 and 100); (B) large number of candidates (varied between 100 and 1000).

budget is sufficiently large, bounded KUBE achieves a higher performance, compared to other benchmarks. However, it can still only achieve less than 60% of the bounded ε-first.

To conclude this section, we note that the bounded ε-first algorithm performs well in most cases, achieving up to 95% of the optimal solution. This proportion is largest for projects with a high budget, which is not surprising given the performance bounds discussed in Section 5. It also achieves the highest performance gains compared to the benchmarks in those settings, as it reasons about task limits, and so our approach is particularly beneficial for large-scale projects.

6.3.2. Performance with Variable Numbers of Candidates

In this section, we investigate the performance of all algorithms when we increase the number of candidates available for a crowdsourcing project. Settings with a large number of candidates are likely to create new challenges for the learning approaches (bounded ε-first, budget-limited ε-first and trialsourcing), because these rely on exploring *all* candidates first prior to exploitation. To this end, Figures 4 and 5 show the performance results (as a percentage of the optimal) of all algorithms for settings with moderate and extremely large budgets, respectively, as we vary the number of candidates from 5 to 1000 (again, for consistency, including only candidates that charge at most $100 per hour). Note that due to computational issues, we do not show the results of the bounded KUBE algorithms within this section (recall that in general, they are outperformed by our proposed method).

In Figure 4, we note that all learning approaches perform well when there are few candidates, as they can explore all available candidates and are likely to select a good worker during the exploitation phase. However, as the number of candidates is increased, the performance decreases. This is due to several factors. First, as more candidates are available, the quality of the optimal solution increases. Second, both ε-first approaches sample each worker fewer times, leading to less accurate quality estimates. Similarly, trialsourcing has an increasingly smaller budget left for exploitation, which also explains the significant drop in quality when the number of candidates reaches 250. Here, most of the budget is spent purely on exploration, and so the performance of trialsourcing approaches that of the uniform algorithm.

In Figure 5, similar trends can be observed for larger budgets. As in Section 6.3.1, our approach, bounded ε-first, performs significantly better than all other benchmarks when the budget is high. Here, the higher budget also allows it to sustain a high quality of around 80–90% of the optimal even when there are a few hundreds of candidates. This is because it has a sufficient budget to explore even the larger number of candidates. In addition, we can see that our method outperforms the best benchmark by up to 300% (in the case of budget B = 30,000 and when the number of candidates is between 100 and 300). This significant increase in relative performance to the other benchmarks is again due to the ability of our algorithm to rely on several high-quality workers within their respective task limits, while most of the other benchmarks rely on a single worker that eventually hits its task limit.

6.3.3. Performance with Variable Correlation between Cost and Quality

Bounded ε-first, and the other algorithms evaluated here, depend on comparing workers based on their quality-cost density (i.e., their estimated quality divided by their cost). However, when there is a strong correlation between cost and quality, as

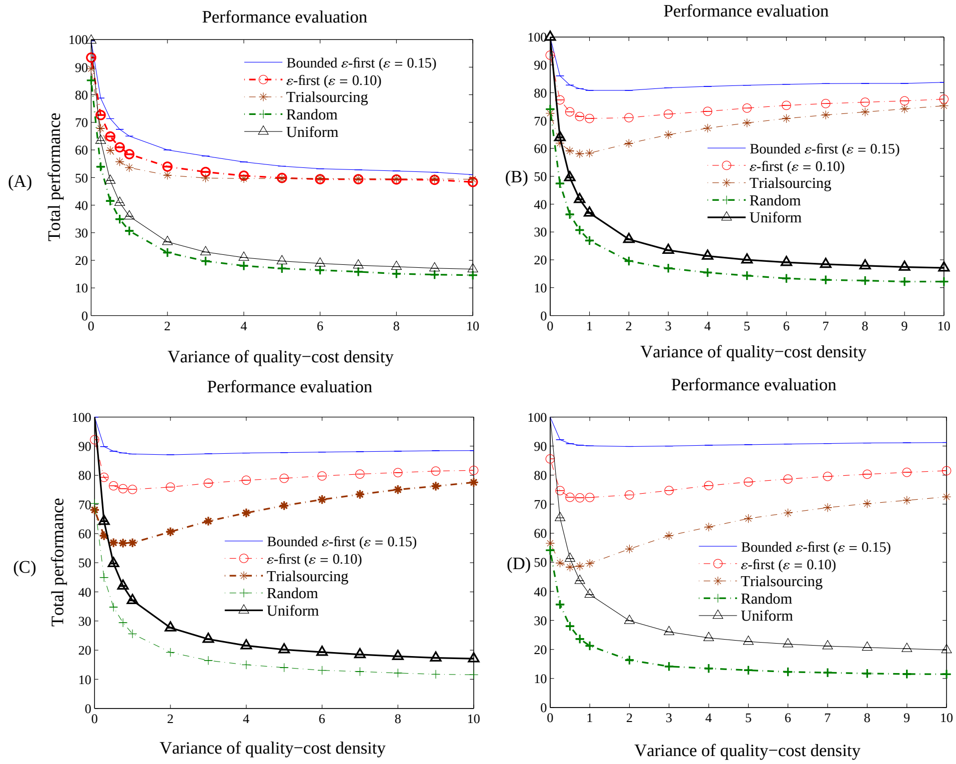

Figure 6: Performance ratio of the algorithms (compared to the optimal solution) with different quality-cost density and with (A) small budget (B = \$500); (B) moderate budget (B = \$5,000); (C) large budget (B = \$30,000); and (D) extremely large budget (B = \$100,000). The noise variance is 1.0 in all the cases.

is often the case in traditional labour markets, where more highly-skilled workers can demand higher wages [18], this may not be an informative feature to distinguish workers. Thus, in this section, we do not use the implicit correlations from the oDesk data set, as we did in previous section, but rather alter this artificially, to test our approach in settings with a range of such correlations.

To achieve this, we use the advertised cost of a worker, c_i, and determine its mean quality as $\mu_i = D \cdot c_i$, where D is a random variable representing the worker's quality-cost density. Here, we sample a value for D for each worker from a distribution with mean $\mathbb{E}[D] = 1$ and variance $\mathrm{Var}[D] = v$, and we vary v to explore different levels of correlation. Thus, when $v = 0$, the quality depends completely on the cost, but as v is increased, the correlation drops. To achieve this, we use a

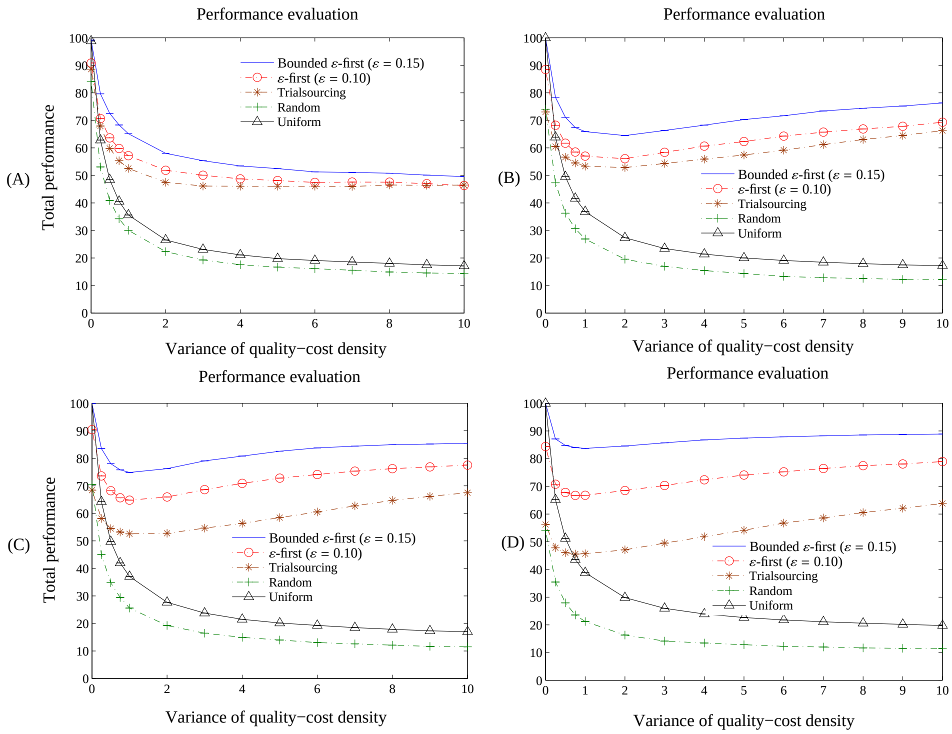

Figure 7: Performance ratio of the algorithms (compared to the optimal solution) with different quality-cost density and with (A) small budget (B = \$500); (B) moderate budget (B = \$5,000); (C) large budget (B = \$30,000); and (D) extremely large budget (B = \$100,000). The noise variance is 10.0 in all the cases.

mixture of uniform distributions for sampling D.[15] Given a mean μ_i, we then produce noisy samples for each worker by multiplying the mean by another random variable N with mean $\mathbb{E}[N] = 1$ and a variance that we set to either $\mathrm{Var}[N] = 1$ (low noise) or $\mathrm{Var}[N] = 10$ (high noise), using the same type of mixture distribution as for D. We vary $\mathrm{Var}[N]$ here to determine how the algorithms respond to different levels of noise.

[15]Specifically, we assume that it has the cumulative probability distribution $F_D(x) = \alpha \cdot x + (1 - \alpha) \cdot \frac{x-1}{k-1}$ for $0 \le x \le k$, where $k = 3 \cdot v + 1$ and $\alpha = 1 - k^{-1}$, while $F_D(x < 0) = 0$ and $F_D(x > k) = 1$. In the special case where $v = 0$, we assume $F_D(x < 1) = 0$ and $F_D(x \ge 1) = 1$. Thus, this distribution is a mixture of two uniform distributions — with probability α, the sample is drawn from a uniform distribution with support $[0, 1]$ and with probability $(1 - \alpha)$, it is drawn from one with support $[1, k]$. We choose this formulation as it is simple and allows us to arbitrarily control the variance while still ensuring a non-negative support.

Figure 6 shows the results in settings with low noise as we increase the variance of the quality-cost density, v, with low (B = \$500), moderate (B = \$5,000), large (B = \$30,000), and extremely large (B = \$100,000) budgets (we choose these as representative results — higher budgets follow similar trends). For the sake of better visibility, the regret bound is left out from the figures (however, they show similar trends to previous figures). Several interesting trends emerge here. When the variance is extremely low (around $v = 0$), all approaches perform well. This is because workers here are completely homogeneous, achieving the same level of quality for each currency unit spent. However, as the variance is increased slightly, performance drops quickly for all approaches, as they are now less likely to choose the best workers.

Interestingly, in the setting with larger budgets (Figures 6 (B), 6 (C), and 6 (D)), the performance of the learning approaches eventually starts rising again. This is because these settings can produce experts with a high quality but low cost that are likely to be identified during the exploration phase and then exploited. This effect does not occur in the setting with a low budget (Figure 6 (A)), because here the exploration budget is low and outliers are less likely to be identified (for the ε-first algorithms) or the exploitation budget is too low (for the trialsourcing algorithm). We can also see that the larger the budget is, the better our algorithm performs compared to the benchmark approaches, for the same reasons as described previously.

Finally, Figure 7 shows the results when individual quality samples of a particular worker have a high variance ($\mathrm{Var}[N] = 10$). Note that we have also left the regret bound out from the figure in order to achieve better visibility. This is a more challenging setting for all of the learning algorithms because it reduces the accuracy of the quality estimates. Here, we first note that in the low budget setting (Figure 7 (A)), there is only a small drop in performance compared to the previous settings with low noise. This is because estimating the quality of workers with such a limited budget is already challenging. A larger drop in quality is apparent for the moderate budget (Figure 7 (B)), where the high noise reduces the accuracy of the quality estimates (as the noise variance now typically exceeds the variance of the quality-cost density). However, despite the significant 10-fold increase in the noise variance, the performance of the learning algorithms is still reasonable, with only an approximately 10% decrease in the total utility achieved. On the other hand, we can see that as the budget is further increased (Figures 7 (C) and 7 (D)), the performance of our algorithm improves, compared to the small and moderate budget cases. This is due to the fact that with a sufficiently large budget size, our algorithm can efficiently explore the quality of each worker, and thus, it can achieve a high performance within the exploitation phase.

To conclude the experimental section, we note that our proposed algorithm,

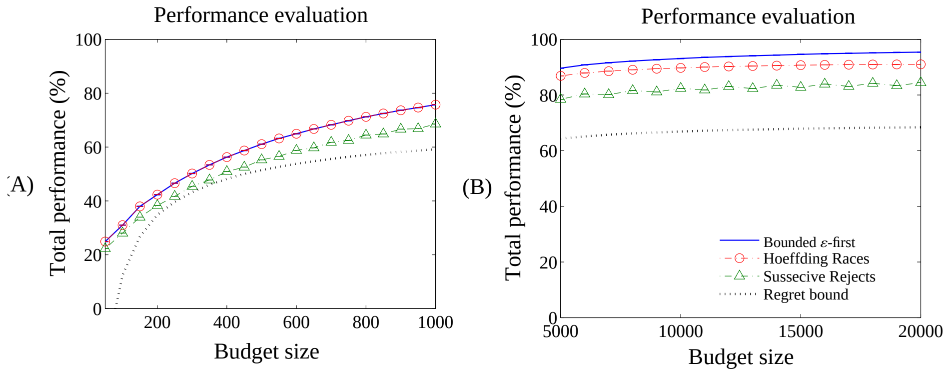

Figure 8: Performance ratio of the algorithms (compared to the optimal solution) in case of jobs with (A) small budgets (smaller than \$1,000); and (B) large budgets (between \$5000 – \$10,000). $\varepsilon = 0.15$ for all the algorithms.

bounded ε-first, consistently outperforms all of the existing benchmark approaches over a range of realistic settings. Sometimes, this results in a many-fold improvement over the best existing approach, and it typically achieves 70-90% of the hypothetical optimal with full information. Performance is particularly good when the overall budget is high (allowing ample exploration) and when the variance of the quality-cost density is high (allowing the algorithm to focus on the most cost-effective workers). On the other hand, when there are many available workers in the system, performance degrades, but our approach still significantly outperforms existing benchmarks.

6.4. Comparison with Other Exploration Policies

We now turn to the investigation of whether we can improve the performance of the bounded ε-first algorithm by replacing the uniform exploration approach with other policies. Recall that in Section 5, we have proved that by replacing the uniform approach with Successive Rejects (SR), the theoretical regret bound, that the bounded ε-first approach can achieve, is increased. Hence, it is less efficient. In this section, we further demonstrate that by using Hoeffding Races for exploration, the performance cannot be improved either. To do so, we compare our algorithm with Hoeffding Races and SR, using the above-mentioned parameter settings. In what follows, we first briefly describe the Hoeffding Races exploration algorithm, and then discuss the numerical results.

The Hoeffding Races algorithm relies on Theorem 2 as follows. Suppose that the number of pulls of arm i is x_i, and let $0 < \beta < 1$. From Theorem 2, we can guarantee that with at least $(1-\beta)$ probability, we have:

$$|\hat{\mu}_i - \mu_i| \leq \sqrt{\frac{-\ln\frac{\beta}{2}}{2x_i}},$$

where $\hat{\mu}_i$ is the current estimate of arm i's expected reward value μ_i. Given this, at each time step t, Hoeffding Races maintains an upper confidence (UC) and lower confidence (LC) value for each arm i, such that

$$UC_i(t) = \hat{\mu}_i(t) + \sqrt{\frac{-\ln\frac{\beta}{2}}{2x_i(t)}}, \quad (26)$$

$$LC_i(t) = \hat{\mu}_i(t) - \sqrt{\frac{-\ln\frac{\beta}{2}}{2x_i(t)}}, \quad (27)$$

where $\hat{\mu}_i(t)$ is the estimate of μ_i at time step t, and $x_i(t)$ is the number of pulls of arm i up to time step t. Hoeffding Races initially uniformly pulls the arms. However, if for a certain t there exist arms $i \neq j$ such that $UC_i(t) < LC_j(t)$, the algorithm eliminates arm i from the set of arms (i.e., it does not pull arm i anymore). The algorithm stops when there is only one arm left. Note that in practice, β is typically set to be 0.05 (see [25] for more details).

To compare the performance of the algorithms, we focus on two scenarios: (i) small budget; and (ii) large budget cases. In particular, due to its nature, Hoeffding Races only displays a different behaviour when the budget is sufficiently large (otherwise it will behave exactly as the uniform exploration). The results are depicted in Figure 8. We can clearly observe that in case the budget is small, both Hoeffding Races and uniform exploration provide the same performance. This is due to the fact that the Hoeffding Races method does not have a sufficient budget to eliminate the arms, and thus, it continues with the initial uniform pull behaviour (Figure 8(A)). On the other hand, as the budget becomes larger, Hoeffding Races can start eliminating the arms within the exploration phase. This, however, results in a decreased performance efficiency. A possible reason is that by eliminating the arms, Hoeffding Races only focuses on the best arms (it pulls them the most). This, however, may lead to poor performance within the exploitation phase, as we might need an accurate estimation of the ranking of all the arms in order to efficiently solve the corresponding bounded knapsack problem. This is also the reason why SR performs poorly, compared to the uniform pull approach. This is in line with our theoretical analysis in Section 5.2.

It is worth noting that we also achieve broadly similar results when we modify Hoeffding Races and SR to find the arm with the highest density, instead of the arm with the highest expected reward. A possible reason behind this is that it is not sufficient either to solely focus on arms with the highest density, as those might have low pulling limits and this will lead to a poor performance in the exploitation phase.

7. Conclusions and Future Work

In this paper, we introduced the expert crowdsourcing problem with variable worker performance, heterogeneous costs and task limits per worker. 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 to represent task limits. 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. This result means that our algorithm has the desirable zero-regret property, implying that the algorithm asymptotically converges to the optimal solution as the budget tends to infinity.

To establish the performance of our algorithm in realistic expert crowdsourcing settings, we also applied it to real data from oDesk, a prominent expert crowdsourcing website. We showed that the algorithm consistently outperforms state-of-the-art crowdsourcing algorithms within this domain by up to 300%, also achieving up to 95% of a hypothetical optimal benchmark that has full information about the workers' performance distributions. Furthermore, the empirical results confirmed our theoretical bounds, indicating that the algorithm works best for projects with large budgets.

As a result, our work could potentially form a promising basis to crowdsourcing websites which aim to provide efficient teams of experts. We envisage that it could be used to automate the formation of curated crowds, which are currently mostly formed on an ad hoc basis (see Section 2.1). In particular, our algorithm could be employed to implement a crowdsourcing intermediary, which, given a customer's budget for a project, automatically explores a potential crowd of workers and then assembles a promising team of the best performers.

In addition to this, our work also constitutes a general contribution to the field of MABs and is applicable to a wide range of decision-making problems under uncertainty beyond the domain of expert crowdsourcing. In more traditional labour markets, our approach could be used to hire unknown contractors to work on a large project, or it could be used to allocate existing workers within a company to a new project (where costs are incurred by removing workers from their day jobs and performance may be unknown if no similar projects have been carried out in the past). Another potential application of our work is cloud computing, where services are potentially unreliable or vary in their quality, and where the maximum number of jobs on one service is restricted either by a fixed deadline or by user quotas. Finally, our work applies generally to resource allocation problems with costly but limited resources of an unknown quality. For example, a government may need to procure medicines to fight a new epidemic, but it is uncertain what medicines work best and it is restricted by budget constraints and stock levels of the medicines.

Currently, our work also has a number of limitations that we will explore further in future work. First, 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. One possible way is to maintain belief-based models for each user's profile, which measures the uncertainty of our knowledge about the user, based on current observations. These belief models are then iteratively updated as we observe the utility values from the users while assigning tasks to them. Our model, however, does not currently handle such belief updates. Thus, as possible future work, we intend to extend our analysis to these settings.

Our current work also assumes that a particular worker's performance is static, that is, it is drawn from a stationary distribution. However, it may be the case that due to external reasons (e.g., health and weather conditions, or other duties), the performance distribution may vary over time. The bounded ε-first algorithm might fail to tackle these settings, as it is not capable of handling dynamic environments. In particular, due to the explicit split of exploration from exploitation, our algorithm might not be able to detect future changes once the exploration phase is completed. One possible way to extend our model is to use bandit algorithms that do not split exploration from exploitation, such as UCB or ε-greedy (for more details, see [30, 32]). However, these algorithms are not designed for the bounded multi-armed bandit model, and thus, it is not trivial how to extend them to our settings. Given this, we also aim to extend our proposed algorithm to systems with dynamic behaviour.

Furthermore, our model considers independent tasks, where the total utility of the tasks is the sum of each individual task's utility. However, tasks may affect each other's value, and thus, the total utility of these tasks may not be equal to their sum of utility. For example, two tasks may contain overlapping parts. This implies that their total utility is less than their sum. In contrast, two other tasks might complement each other, boosting each other's value if both are completed (i.e., their total utility is higher than their sum). As our algorithm is currently not designed to address this setting, we intend to extend our model to this scenario as well.

Acknowledgements

This is a significantly extended version of a prior conference publication [34]. The work was carried out as part of the ORCHID project (`www.orchid.ac.uk`), which is funded by EPSRC, the UK Engineering and Physical Sciences Research Council (EP/I011587/1).

References

[1] Agrawal, R. (1995). Sample mean based index policies with O(log n) regret for the multi-armed bandit problem. *Adv. in Appl. Prob.*, **27**, 1054–1078.

[2] Anantharam, V., Varaiya, P., and Walrand, J. (1987). Asymptotically efficient allocation rules for the multiarmed bandit problem with multiple plays — part I: I.i.d. rewards. *IEEE Transactions on Aumatic Control*, **32(11)**, 977–982.

[3] Audibert, J.-Y., Bubeck, S., and Munos, R. (2010). Best arm identication in multi-armed bandits. *Proceedings of the Twenty-Third Annual Conference on Learning Theory*, pages 41–53.

[4] Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite–time analysis of the multiarmed bandit problem. *Machine Learning*, **47**, 235–256.

[5] Bernstein, M. S., Little, G., Miller, R. C., Hartmann, B., Ackerman, M. S., Karger, D. R., Crowell, D., and Panovich, K. (2010). Soylent: a word processor with a crowd inside. In *Proceedings of the 23rd Annual ACM Symposium on User Interface Software and Technology*, pages 313–322.

[6] Beygelzimer, A., Langford, J., Li, L., Reyzin, L., and Schapire, R. (2011). Contextual bandit algorithms with supervised learning guarantees. In *Proceeding of the Forteenth International Conference on Artificial Intelligence and Statistics*, pages 19–26.

[7] Brooks, F. P. (1995). *The Mythical Man-Month : Essays on Software Engineering*. Addison-Wesley Pub. Co.

[8] Bubeck, S., Munos, R., and Stoltz, G. (2009). Pure exploration for multi-armed bandit problems. In *Proceedings of the Twentieth international conference on Algorithmic Learning Theory*, pages 23–37.

[9] Clery, D. (2011). Galaxy zoo volunteers share pain and glory of research. *Science*, **333**(6039), 173–175.

[10] Dai, P., Mausam, and Weld, D. S. (2011). Artificial intelligence for artificial artificial intelligence. In *Proceedings of the 25th Conference on Artificial Intelligence (AAAI 2011)*, pages 1153–1159.

[11] Doan, A., Ramakrishnan, R., and Halevy, A. Y. (2011). Crowdsourcing systems on the world-wide web. *Communications of the ACM*, **54**(4), 86–96.

[12] Gao, H., Barbier, G., and Goolsby, R. (2011). Harnessing the crowdsourcing power of social media for disaster relief. *IEEE Intelligent Systems*, **26**(3), 10–14.

[13] Guha, S. and Munagala, K. (2009). Multi-armed bandits with metric switching costs. In S. Albers, A. Marchetti-Spaccamela, Y. Matias, S. Nikoletseas, and W. Thomas, editors, *Automata, Languages and Programming*, volume 5556 of *Lecture Notes in Computer Science*, pages 496–507. Springer Berlin / Heidelberg.

[14] Ho, C.-J. and Vaughan, J. W. (2012). Online task assignment in crowdsourcing markets. In *Proceedings of the 26th Conference on Artificial Intelligence (AAAI 2012)*, pages 45–51.

[15] Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, **58**, 13–30.

[16] Horton, J. J. and Chilton, L. B. (2010). The labor economics of paid crowdsourcing. In *Proceedings of the 11th ACM Conference on Electronic Commerce (EC'10)*, pages 209–218.

[17] Huang, E., Zhang, H., Parkes, D. C., Gajos, K. Z., and Chen, Y. (2010). Toward automatic task design: a progress report. In *Proceedings of the ACM SIGKDD Workshop on Human Computation (HCOMP '10)*, pages 77–85.

[18] Juhn, C., Murphy, K. M., and Pierce, B. (1993). Wage inequality and the rise in returns to skill. *Journal of Political Economy*, **101**(3), 410–442.

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

[20] Kittur, A., Chi, E. H., and Suh, B. (2008). Crowdsourcing user studies with mechanical turk. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*, pages 453–456.

[21] Kuleshov, V. and Precup, D. (2010). Algorithms for the multi-armed bandit problem. *Unpublished*.

[22] Little, G., Chilton, L. B., Goldman, M., and Miller, R. C. (2010). Exploring iterative and parallel human computation processes. In *Proceedings of the ACM SIGKDD Workshop on Human Computation (HCOMP '10)*, pages 68–76.

[23] Marcello, S. and Toth, M. (1990). *Knapsack Problems: Algorithms and Computer Implementations*. Wiley.

[24] Marge, M., Banerjee, S., and Rudnicky, A. (2010). Using the amazon mechanical turk for transcription of spoken language. In *Proceedings of the 35th International IEEE Conference on Acoustics, Speech, and Signal Processing (ICASSP'10)*, pages 5270–5273.

[25] Maron, O. and Moore, A. W. (1993). Hoeffding races: Accelerating model selection search for classification and function approximation. *Proceedings of the Seventh Annual Conference on Neural Information Processing Systems*, pages 59–66.

[26] Mason, W. and Watts, D. J. (2009). Financial incentives and the performance of crowds. In *Proceedings of the ACM SIGKDD Workshop on Human Computation (HCOMP '09)*, pages 77–85.

[27] Mnih, V., Szepesvari, C., and Audibert, J. (2008). Empirical bernstein stopping. *Proceedings of the 25th International Conference on Machine Learning*, pages 672–679.

[28] Morris, R. R., Dontcheva, M., and Gerber, E. M. (2012). Priming for better performance in microtask crowdsourcing environments. *IEEE Internet Computing*, **16**, 13–19.

[29] Robbins, H. (1952). Some aspects of the sequential design of experiments. *Bull. of the AMS*, **55**, 527–535.

[30] Sutton, R. S. and Barto, A. G., editors (1998). *Reinforcement Learning: An Introduction*. MIT Press.

[31] Tokarchuk, O., Cuel, R., and Zamarian, M. (2012). Analyzing crowd labor and designing incentives for humans in the loop. *IEEE Internet Computing*, **16**(5), 45–51.

[32] Tran-Thanh, L. (2012). *Budget–Limited Multi–Armed Bandits*. Ph.D. thesis, University of Southampton, School of Electronics and Computer Science, Southampton UK.

[33] Tran-Thanh, L., Chapman, A., de Cote, J. E. M., Rogers, A., and Jennings, N. R. (2010). Epsilon–first policies for budget–limited multi–armed bandits. In *Proceedings of the 24th Conference on Artificial Intelligence (AAAI 2010)*, pages 1211–1216.

[34] Tran-Thanh, L., Stein, S., Rogers, A., and Jennings, N. R. (2012a). Efficient crowdsourcing of unknown experts using multi-armed bandits. In *20th European Conference on Artificial Intelligence (ECAI 2012)*, pages 768–773.

[35] Tran-Thanh, L., Chapman, A., Rogers, A., and Jennings, N. R. (2012b). Knapsack based optimal policies for budget-limited multi-armed bandits. In *Proceedings of the 26th Conference on Artificial Intelligence (AAAI 2012)*, pages 1134–1140.

[36] Vermorel, J. and Mohri, M. (2005). Multi-armed bandit algorithms and empirical evaluation. In *Proceedings of the 16th European Conference on Machine Learning (ECML'05)*, pages 437–448.

[37] Vuurens, J. and de Vries, A. (2012). Obtaining high-quality relevance judgments using crowdsourcing. *IEEE Internet Computing*, **16**(5), 20–27.

[38] Welinder, P., Branson, S., Belongie, S., and Perona, P. (2010). The multidimensional wisdom of crowds. In *Advances in Neural Information Processing Systems 24 (NIPS 2010)*, pages 2424–2432.

[39] Zaidan, O. F. and Callison-Burch, C. (2011). Crowdsourcing translation: professional quality from non-professionals. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1 (HLT '11)*, pages 1220–1229.

[40] Zook, M., Graham, M., Shelton, T., and Gorman, S. (2010). Volunteered geographic information and crowdsourcing disaster relief: A case study of the haitian earthquake. *World Medical & Health Policy*, **2**(2), 7–33.