BudgetFix: Budget Limited Crowdsourcing for Interdependent Task Allocation with Quality Guarantees

Long Tran-Thanh[1], Trung Dong Huynh[1], Avi Rosenfeld[2],
Sarvapali Ramchurn[1] and Nicholas R. Jennings[1]
[1]Electronics and Computer Science
University of Southampton, UK
{ltt08r,tdh,sdr,nrj}@ecs.soton.ac.uk
[2] Jerusalem College of Technology, Israel
rosenfa@jct.ac.il

ABSTRACT

Crowdsourcing is a multi-agent task allocation paradigm that involves up to millions of workers, of varying reliability and availability, performing large numbers of micro-tasks. A key challenge is to crowdsource, at minimal cost and with predictable accuracy, complex tasks that involve different types of interdependent micro-tasks structured into complex workflows. In this paper, we propose the first crowdsourcing algorithm that solves this problem. Our algorithm, called BudgetFix, determines the number of interdependent micro-tasks and the price to pay for each task given budget constraints. Moreover, BudgetFix provides quality guarantees on the accuracy of the output of each phase of a given workflow. BudgetFix is empirically evaluated on a well-known crowdsourcing-based text correction workflow using Amazon Mechanical Turk, and is shown that BudgetFix can provide similar accuracy, compared to the state-of-the-art algorithm for this workflow, but is on average 32% cheaper.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent Systems

General Terms

Algorithms, Performance, Reliability, Human Factors

Keywords

crowdsourcing, find-fix-verify, budgeted task allocation, performance guarantees

1. INTRODUCTION

Crowdsourcing is a multi-agent task allocation paradigm that is increasingly becoming a major tool for organisations (businesses or government agencies) to manage large data sets and to create new products by tasking hundreds or millions of workers online with micro-tasks (e.g., translation, transcription, and design) [1, 4, 11, 12, 14]. Within these systems, a task manager, or an autonomous agent that acts on the behalf of the task manager, allocates different tasks to a group of workers chosen from a population of crowd.

Appears in: *Alessio Lomuscio, Paul Scerri, Ana Bazzan, and Michael Huhns (eds.), Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2014), May 5-9, 2014, Paris, France.*

Crowdsourcing platforms such as Freelancer.com claim to employ 6.5M workers for routine programming and other design tasks,[1] while Amazon Mechanical Turk (AMT) is estimated to regularly task more than 500K workers[2]. Crucially, these platforms are growing in size and in terms of the variety of tasks they allow the workers to perform.

The ubiquity, size and heterogeneity of crowdsourcing networks raises a number of challenges common to multi-agent task allocation problems [7, 15, 16]. First, given that workers may come from different countries (e.g., China, India, UK, US) and with different skills, the quality of their work may vary significantly. For example, native English speakers will likely perform better at correcting spelling mistakes or graduate graphic designers will do a better job at drawing a logo compared to a non-expert. Second, the fact that these workers are located in different time zones means that the set of available workers will also change over time. Third, different workers will likely react differently to the amount of money paid per micro-task making it difficult to set a budget needed to guarantee good task performance [4]. To address these challenges, a number of approaches have been proposed to deal with inaccuracies in worker outputs using probabilistic techniques [8, 5, 9, 13]. These approaches typically rely on the efficient use of redundancy (i.e., allocating the same task to multiple users from the crowd), and on the assumption that the tasks are independent from each other and can be arbitrarily allocated to the crowd (see Section 2 for more details). Thus, while they provide efficient solutions for crowdsourcing applications with *independent* micro-tasks in many existing applications, there is a need to allocate to the crowd more complex problems that need to be solved in multiple phases [4, 11, 14]. In fact, these phases can be solved using sets of *interdependent* micro-tasks. For example, in [4], a Find-Fix-Verify (FFV) workflow is used to correct and shorten text by passing Finds of mistakes in sentences by a set of workers to another set of workers who Fix these mistakes. In addition, another set of workers verify these mistakes (see more details in Section 2). A similar workflow is used by [11, 14] albeit with more complex tasks.

A key challenge within the above-mentioned applications is to determine *how many* tasks should be allocated to each phase and *how much to pay* for each task in each phase. This is particularly difficult when organisations have a *limited budget* to spend on a crowdsourcing problem. Given these challenges, crowdsourcing

[1] http://techcrunch.com/2012/12/08/asias-secret-crowdsourcing-boom/
[2] http://www.theverge.com/2013/3/7/4075810/amazon-mechanical-turk-users-study-finds-half-have-public-profiles.

endeavours may result in high costs resulting in low quality outputs. In particular, existing approaches typically have to allocate exponentially increasing numbers of tasks in subsequent phases [4, 11, 12]. For example, in the FFV workflow, for each Find, there are multiple possible fixes and each fix requires a number of possible verifications.

Against this background, this paper addresses the challenge of interdependent task allocation under budget constraints in crowdsourcing problems. In particular, we consider the FFV workflow and develop a novel crowdsourcing algorithm, called BudgetFix, that our agent can use to allocate interdependent tasks within crowdsourcing systems. Given a budget B, BudgetFix specifies the number of micro-tasks to be allocated at each phase of a workflow and how much to pay for each micro-task. Crucially, BudgetFix determines *which* outcomes from one phase should be passed to the next. By so doing BudgetFix combats the exponential growth of micro-tasks in each phase yet often spends less than a given budget. Moreover, we prove that BudgetFix is able to constrain the number of tasks performed while guaranteeing an accurate solution with at least $e^{-O(B)}$ probability. This makes BudgetFix the first crowdsourcing algorithm to provide such provable quality guarantees for this problem. In more detail, we advance the state of the art in the following ways:

- We propose BudgetFix, the first task allocation algorithm for crowdsourcing systems with interdependent tasks that *provably* achieves $1 - e^{-O(B)}$ accuracy probability.
- We demonstrate through real experiments on AMT that our method is both more efficient in terms of cost and accuracy, compared to the existing approaches. In particular, BudgetFix is shown to be 32% cheaper than the baseline on average.

The rest of this paper is structured as follows. Section 2 summarises the related work, and Section 3 describes our model. We then introduce BudgetFix in Section 4, along with proofs of theoretical guarantees. Section 5 empirically evaluates Budget fix. Section 6 concludes.

2. RELATED WORK

Here we discuss the literature on crowdsourcing, taking into account aspects of accuracy and budget constraints. Moreover, we present the FFV workflow from [4] on which we demonstrate the performance of BudgetFix.

Crowdsourcing for Accuracy
Previous work on crowdsourcing algorithms have typically considered how to allocate relatively simple micro-tasks to a crowd. For example, they aim to get many workers to classify objects (e.g., to detect inappropriate websites or galaxies). To do so, these approaches use redundancy to minimise the estimation error. For example, Wellinder et al. proposed a multidimensional model of users in order to estimate the accuracy of a particular user's answer to improve the estimation of the ground truth [17]. In a similar vein, [8, 13] use Bayesian learning techniques to predict the users' responses. Moreover Dai *et al*. proposed a PoMDP-based (for partially observable Markov decision process) approach to model the estimation's quality [5]. This work not only demonstrated the effectiveness of the approach within relatively simple classification tasks, but was also used within more complex iterative tasks in which workers iteratively refine each others' answers. However, contrary to BudgetFix, none of these methods are applicable to complex workflows where different costs exist for performing micro-tasks in multiple phases, taking into account that the expected performance of the worker pool is heterogeneous and not known a priori. Moreover, none of these approaches have considered the budget limited case.

Budget-Constrained Crowdsourcing
In terms of budget-aware crowdsourcing, work by [9] presented more advanced techniques to aggregate results from tasks performed by the crowd, and crucially, performance guarantees are provided when the algorithm is given a limited budget. However, in contrast to BudgetFix, they did not consider how to budget for complex workflows that interleave heterogeneous micro-tasks (e.g, Find-Fix-Verify). Instead, they assume that all budgeted tasks have an equal level of difficulty, something which we observe is not the case in many real-world environments including the text correction task we consider in this paper. Other work that addresses the budget allocation between tasks include [7, 15, 16] but they do not take into account the interdependency between tasks, and thus, are not suitable for our settings. Azaria et al. [3] model an environment in which a task allocator must allocate many similar tasks in a crowdsourcing environment with minimal cost. The authors show that this problem in NP-hard and therefore introduce two heuristic based agents shown to outperform humans. However, they assume that the task allocator has many similar tasks. Furthermore, in contrast to our work they do not show any upper-bound limit on the expected cost.

Find-Fix-Verify
Bernstein *et al.* [4] showed how to allocate heterogeneous workers within a multiple-phased crowdsourcing environment. They developed the Soylent system, which breaks a large, complex task into three micro-tasks: Find, Fix and Verify (FFV). Within the Find phase, the workers must identify an error that needs to be corrected. For example, we consider a text correction task where the goal is to find a spelling or grammatical error within the text. After a set of workers have submitted their responses for the Find phase, it then asks a new group of users to suggest corrections. As is common practice, they intentionally use workers who did not participate in the previous phase to ensures a varied worker base, thus yielding better results [4, 9]. Last, in the Verify phase a third, independently selected group of workers performs quality control on the previously submitted work. By either accepting or rejecting the work from the previous phase, this further improves the result. Crowdsourced workers in particular have been shown to be effective in yielding improved task quality by using this model [4, 9].

To address the possibility that multiple phases will generate an exponentially increasing numbers of crowdsourcing tasks, Soylent relies on very simple heuristics (e.g., only 'find's with 20% agreement are passed to the Fix phase and only similar 'fixes' are passed to the Verify phase) that determine the number of outcomes to pass from one phase to another in an ad hoc fashion. While this helps achieve high accuracy of the outcomes in practice, as we show in Section 5 it is by no means budget–efficient nor does it come with any guarantees of accuracy. In contrast, BudgetFix yields results with similar (or better) accuracy while providing provable performance guarantees. In addition, it is able to do so at a lower cost and can do so when given a limited budget.

3. PROBLEM DEFINITION

In this section, we formally define a class of complex tasks that are solved by a crowd in multiple interdependent phases[3]. In particular,

[3]The problem is solved in multiple steps to avoid lazy workers – see [4, 11] for more details.

we aim to crowdsource the solution to the text correction problem as posed by [4] using the FFV workflow. Note that, even though we focus on text correction, BudgetFix is applicable to many complex tasks that can be solved in multiple interdependent phases (as exemplified in this section). Specifically here, the task involves finding a grammatical or spelling mistake within a sentence.[4] Furthermore, we assume that there is only one mistake to fix within a given sentence.[5] In what follows, we first present the FFV workflow and then describe the constraints and assumptions under which it is executed.

3.1 The Find-Fix-Verify Workflow

This workflow allows the agent to separate the tasks of finding a mistake, correcting it, and ensuring that all corrections to fixes received are themselves correct. This workflow was shown to be effective in getting text corrected to a high accuracy (> 90%) without any budget limitations. Here we formalise and then extend it to consider budget constraints. The details of each step are as follows:

Find: the agent asks the workers to identify the position of the error in t. Let X denote the set of possible error candidates (i.e., all positions in the text at which an error can occur). We assume that these responses are drawn from an *unknown* distribution D^X, and that the error is at true position x^t. The output of this phase is a set X of possible errors.

Fix: the agent asks the workers to fix the identified errors at positions $X' \subseteq X$ (i.e., we may not request all the errors found to be fixed). In addition, for each $x \in X'$ possible error position, let $Y(x)$ denote the set of possible fixes that belong to position x. We assume that for each $x \in X'$, $Y(x)$ contains the "No–Fix" response. This response represents the case when x is believed to be non–erroneous. We assume that for each $x \in X'$, the fix responses that belong to x are drawn from an also unknown distribution $D^{Y(x)}$. Hence, for each $x \in X'$ we can define its true fix $y^t(x)$.

Verify: the agent asks the workers to vote on whether a possible error-fix pair (x, y) is correct. Let $Z(x, y)$ denote the set of verification responses of pair (x, y). For each $z \in Z(x, y)$, we have $z = 1$ if the corresponding user thinks the fix y of position x is correct (i.e., $y(x) = y^t(x)$), and $z = 0$ otherwise. Thus, the responses that belong to each pair (x, y) can be regarded as random variables drawn from an unknown Bernoulli distribution $D^{Z(x,y)}$.

3.2 The Budget Limited Case

Let the costs for requesting a task in each of the Find, Fix, and Verify phases be defined as c^X, c^Y, and c^Z, respectively (all real valued) and let N^X, N^Y, and N^Z denote the number of responses we require from the crowd for each of the phases respectively. Now, the task requested cannot exceed a budget limit $B \in \Re^+$, that is, the crowdsourcing of tasks is subject to the following budget constraint:

$$N^X c^X + N^Y c^Y + N^Z c^Z \leq B \quad (1)$$

In addition, we assume that the users are not malicious (i.e., they do not provide wrong answers on purpose). As such, they only provide their best guess based on their knowledge of the task, which, however, might be inaccurate with some uncertainty (hence the distributions D^X, $D^{Y(x)}$, and $D^{Z(x,y)}$). This justifies the following assumptions:

Assumption 1: In the Find phase, the users choose x^t with the highest probability in D^X. That is, x^t is the mode of D^X. This assumption guarantees that on average, the answers will tend towards the true error position x^t

Assumption 2: In the Fix phase, if $x \neq x^t$, the user chooses $y(x) =$ *No–Fix* with at least $\frac{1}{2}$ probability. That is,

$$P(y(x) = \text{No} - \text{Fix} | x \neq x^t) > \frac{1}{2}$$

Vice versa, for the case of $x = x^t$, the user chooses $y(x) \neq$ *No–Fix* with at least $\frac{1}{2}$. More formally, we have:

$$P(y(x) \neq \text{No} - \text{Fix} | x = x^t) > \frac{1}{2}$$

This assumption guarantees that on average, there is higher probability to get the correct answer.

Assumption 3: In the Verify phase, if $y(x) = y^t(x)$, then

$$\mathbb{E}[z(x, y) | y(x) = y^t(x)] > \frac{1}{2}$$

Otherwise, we have:

$$\mathbb{E}[z(x, y) | y(x) \neq y^t(x)] < \frac{1}{2}$$

for the case of $y(x) \neq y^t(x)$. Similarly to Assumption 2, this guarantees that on average, there is higher probability to get the correct verification.

The next section proposes the BudgetFix algorithm to solve the aforementioned problem in an efficient way. Crucially, we prove that the algorithm guarantees a high probability of *correct* corrections.

4. THE BudgetFix ALGORITHM

In this section, we address two key challenges common to many crowdsourcing applications, namely:

1. how to *minimise the overall cost of the process and maximise the accuracy of the results given the maximum numbers of micro-tasks for each phase* (e.g., N^X for Find, N^Y for Fix, and N^Z for Verify).

2. how to *set the number of micro-tasks for each phase (i.e., how to set N^X, N^Y, and N^Z) given a fixed budget and a cost per task for each phase.*

To this end, in Section 4.1 we first describe a quality control process (called $AccurateAlloc$), and in Section 4.2 we define a procedure the agent can use to determine the number of micro-tasks for each phase. The BudgetFix algorithm combines these procedures. In particular, BudgetFix aims to avoid passing *all* the tasks executed in one phase to the next one, while maintaining quality guarantees (i.e., filtering most of the incorrectly completed tasks in earlier phases). As described in Section 1, this is particularly important in multi-phase crowdsourcing processes as each task passed on to the next phase results in multiple tasks in later phases, causing an exponential growth in the number of allocated tasks in those phases. However, as we show in the following sections, BudgetFix filters incorrectly performed tasks (with high probability) in early phases to avoid this exponential growth. Crucially, we provide theoretical results that describe the *performance guarantees* of BudgetFix in terms of the bounds on the error in its final output. Moreover, later in Section 5 we empirically demonstrate the

[4]The same approach works if the mistake is non-textual, such as a logical mistake or a mistake within a picture/map.

[5]In future work, we will extend the model to consider multiple mistakes.

strength of this approach. Note that due to space limitations, we only provide sketches of proofs for the key results.

4.1 The Quality Control Procedure

Within the quality control procedure $AccurateAlloc$, the agent takes as input the maximum number of tasks to be executed by the crowd for each phase of the crowdsourced solution process. In the case of Find, Fix, Verify, these are N^X, N^Y,and N^Z. Then, $AccurateAlloc$ aims to efficiently allocate micro-tasks to the crowd, in order to maximise the probability of correctness (or minimise the probability of incorrectly done tasks) given these upper-bounds on the number of tasks for each phase. In addition to N^X, N^Y, and N^Z, $AccurateAlloc$ also takes as input a tuning parameter $0 < \varepsilon \leq 1$, which we describe later (in Section 4.1.1). In what follows, we describe how $AccurateAlloc$ operates for each phase of the solution process and, more importantly, we show how the error in the output can be bounded.

4.1.1 Quality Control for Find

In this phase, $AccurateAlloc$ requests all N^X micro-tasks to identify candidates for the position of the error in t. In particular, we focus on how to avoid passing on all these candidates to the Fix phase. Here, we assume that all of the N^X received responses are randomly drawn from the unknown distribution V^X. Thus, let $\hat{D}^X$ denote the empirical distribution of X (i.e., the estimate of V^X) and $\hat{x}^t$ denote the mode of $\hat{D}^X$, that is:

$$\hat{x}^t = \arg\max_x P_{\hat{D}^X}(x) \tag{2}$$

where $P_{\hat{D}^X}(x)$ denotes the probability of x in $\hat{D}^X$. Given this, we can define the set S of *selected candidates* x as:

$$S = \{x | P_{\hat{D}^X}(x) \geq P_{\hat{D}^X}(\hat{x}^t) - \varepsilon\}$$

where the probability of candidate x in $\hat{D}^X$ is within the ε distance from the probability of $\hat{x}^t$. Thus, only those candidate errors with the highest probability, or within ε of the error found with the highest probability, will be forwarded to the next phase. Note that this is starkly different from the original Find, Fix and Verify model where all error candidates with over 20% frequency were automatically forwarded to the next phase.

Given these definitions, the following result holds:

THEOREM 1. *Suppose that* Assumption 1 *holds, we have* $x^t \in S$ *with at least a probability of* $\left(1 - 2\exp\left\{\frac{-N^X\varepsilon^2}{2}\right\}\right)$.

PROOF SKETCH. From the Dvoretzky-Kiefer-Wolfowitz inequality we have that the probability that the estimation error of the empirical distribution $\hat{D}^X$ is larger than $\frac{\varepsilon}{2}$ is at most $2\exp\left\{\frac{-N^X\varepsilon^2}{2}\right\}$ after receiving N^X responses. This implies that the mode of D^X (i.e., the position that has the highest probability mass), which is equivalent to the true position x^t, is within S with the requested probability. □

After $AccurateAlloc$ receives N^X submissions, the set S is computed as above and passed on to the Fix phase.

4.1.2 Quality Control for Fix

In this phase, apart from collecting possible corrections of the candidates from S, the algorithm aims to identify the true error position x^t with high accuracy (assuming that $x^t \in S$). To do so, it maintains a "fitness" value $r(x)$ for each $x \in S$ as follows. For each $x \in S$, if we receive a correction $y(x) \neq$ No–Fix, the algorithm allocates a reward value 1 to x, and 0 otherwise. Let $r(x)$ denote the average reward value that x has received so far. It is easy to show that $r(x)$ is an unbiased estimator of the probability $y(x) \neq$ No–Fix, that is,

$$\mathbb{E}[r(x)] = P(y(x) \neq \text{No} - \text{Fix}) \tag{3}$$

Now, if Assumption 2 holds, we have $\mathbb{E}\left[r(x^t)\right] > \frac{1}{2}$ and $\mathbb{E}[r(x)] < \frac{1}{2}$ if $x \neq x^t$. Given this, the true error position x^t has the highest expected fitness value. However, we do not know in advance the probability of $y(x) \neq$ No–Fix, and thus, the expected fitness for each x. Given this, we need to estimate these values from the received submissions within this phase and choose the one with the highest average fitness. To do so, we rely on the Successive Reject (SR) algorithm, proposed by Audibert *et al.* [2]. In particular, this online learning algorithm is proven to provide the best performance in identifying the fittest from a set of options (i.e., Find candidates) with unknown fitness values. This algorithm can be adapted to our settings as follows:

Recall that the maximum number of requests we can have within this phase is N^Y. Let $K = |S|$ denote the size of S. In addition, let $N^X(x)$ denote the number of times $x \in S$ was chosen to be a candidate within the first phase.

We run $K - 1$ episodes as follows. We first define function v:

$$v(N) = \frac{1}{2} + \sum_{k=2}^{N} \frac{1}{k} \tag{4}$$

for any $N > 0$ integer. In addition, let $S_1 = S$, $m_0 = 0$, and for each $k \in \{1, \ldots, K-1\}$, we have:

$$m_k = \left\lceil \frac{1}{v(K)} \frac{N^Y - K}{K + 1 - k} \right\rceil \tag{5}$$

For each episode k, we request $(m_k - m_{k-1})$ submissions for each $x \in S_k$. Note that each $x \in S_k$ receives exactly m_k submission requests up to episode k. It can be shown that the total number of requests within this phase is at most N^Y (for more details, see Audibert *et al.* [2]).

The fitness $r_k(x)$ of $x \in S_k$ can be calculated as:

$$r_k(x) = \frac{N^X(x) + \sum_{i=1}^{m_k} \mathbb{I}(y^i(x) \neq \text{No} - \text{Fix})}{N^X(x) + m_k} \tag{6}$$

where $\mathbb{I}()$ is the indicator function. Note that we also take into account the submissions in the Find phase that indicate x is a possible error position (i.e., by taking $N^X(x)$ into the formula). Given the fitness values, we eliminate the x that has the lowest fitness value from S_k, and update the set of candidates, that is:

$$S_{k+1} = S_k \setminus \{\arg\min_{x \in S_k} r_k(x)\} \tag{7}$$

Since we eliminate one candidate at the end of each episode, after the $K-1$ episodes, S_K only contains one candidate. Let x^* denote this candidate. In addition, let $U = \{y(x^*)\}$ denote the set of possible corrections of x^* within this phase. The output of this phase is then the pair $\langle x^*, U \rangle$.

4.1.3 Quality Control for Verify

Given the pair $\langle x^*, U \rangle$, we now focus on identifying the correct fix of error candidate x^*. To do so, we also measure the fitness of each correction. Recall that due to Assumption 3, for each $y(x)$ correction, we have $\mathbb{E}[z(x,y)] > \frac{1}{2}$ if and only if y is correct (i.e., $y = y^t(x)$). Given this, by considering $\mathbb{E}[z(x,y)]$ as the fitness of $y(x)$, the desired correction has the highest fitness. Thus, we

Algorithm 1 BudgetFix

1: **Inputs**: budget B, costs c^X, c^Y, c^Z, $0 < \varepsilon \leq 1$;
2: Compute N^X, N^Y and N^Z using Theorem 3 and Equation 12;
3: **Run** $AccurateAloc$ **as follows**:
4: *Find phase*: Determine set of possible positions S (see Section 4.1.1);
5: *Fix phase*: Use SR to identify possible corrections U (see Section 4.1.2);
6: *Verify phase*: Use SR to identify the best position – correction pair $\langle x^*, y^* \rangle$ (see Section 4.1.3);

can again use the SR algorithm to identify the correction with the highest fitness level. To do so, let $L = |U|$, $U_1 = U$, $m_0 = 0$, and

$$m_l = \left\lceil \frac{1}{v(L)} \frac{N^Z - L}{L + 1 - l} \right\rceil \tag{8}$$

At each episode l, we allocate $(m_l - m_{l-1})$ request for each $y \in U_l$ and the fitness $r_l(y)$ of $y \in U_l$ can be calculated as:

$$r_l(y) = \frac{N^Y(y) + \sum_{i=1}^{m_l} \mathbb{I}(z^i(x,y) = 1)}{N^Y(y) + m_l} \tag{9}$$

where $N^Y(y)$ is the number of submissions within the Fix phase that proposes y as a correction of x^*. By eliminating the weakest correction candidate at each episode, we can see that U_L only contains one single correction, denoted with y^*. The output of $AccurateAlloc$ is then the pair $\langle x^*, y^* \rangle$.

The performance of $AccurateAlloc$ can be guaranteed in terms of bounds on the error in the final output as follows:

THEOREM 2. *Suppose that* Assumptions 1–3 *hold. Given this, for each $N^X, N^Y, N^Z > 0$ integers and $0 < \varepsilon \leq 1$, the above-mentioned algorithm guarantees that the probability of having the output pair $\langle x^*, y^* \rangle$ incorrect (i.e., either $x^* \neq x^t$ or $y^* \neq y^t(x^t)$) is at most:*

$$2\exp\left\{\frac{-N^X\varepsilon^2}{2}\right\} + \frac{K(K-1)}{2}\exp\left\{-\frac{N^Y - K}{v(K)K}\right\} + \frac{L(L-1)}{2}\exp\left\{-\frac{N^Z - L}{v(L)L}\right\} \tag{10}$$

PROOF SKETCH. Suppose that $AccurateAlloc$ provides an inaccurate solution. This implies that it was inaccurate in at least one of the phases. The probability that it is inaccurate in phase Find is at most $2\exp\left\{\frac{-N^X\varepsilon^2}{2}\right\}$ (from Theorem 1). By using the theoretical results from Audibert *et al.* [2], we obtain that the probability of inaccuracy in the Fix and Verify phases are $\frac{K(K-1)}{2}\exp\left\{-\frac{N^Y-K}{v(K)K}\right\}$ and $\frac{L(L-1)}{2}\exp\left\{-\frac{N^Z-L}{v(L)L}\right\}$, respectively. Using the union rule of probability, we obtain the requested error bound. □

4.2 Selecting the Number of Micro-Tasks per Phase

Here we describe the procedure to set the values of N^X, N^Y, and N^Z, given the budget limit B to guarantee the algorithm's efficiency. To do so, we first refine the performance analysis of $AccurateAloc$ procedure in order to simplify the calculations. In particular, in order to simplify the computation of N^X,N^Y and N^Z, we assume that $AccurateAlloc$ has some upper limit for the value of K and L, respectively. Note that the values K (i.e., the size of the candidate set S in Find) and L (i.e., the size of candidate set U in Fix) are not known in advance, and can only be identified at the end of each corresponding phases. In particular, if $K > K_{\max}$, the core algorithm will only choose the best $K_{\max}$ candidates in the candidate set S.[6] Similarly, in the Fix phase, if $L > L_{\max}$, the core algorithm will only choose the best $L_{\max}$ candidates in the candidate set U. By doing so, we can reformulate Theorem 2 to be as follows.

The probability that the core algorithm provides an incorrect outcome is at most:

$$2\exp\left\{\frac{-N^X\varepsilon^2}{2}\right\} + \frac{K_{\max}(K_{\max}-1)}{2}\exp\left\{-\frac{N^Y - K_{\max}}{v(K_{\max})K_{\max}}\right\} + \frac{L_{\max}(L_{\max}-1)}{2}\exp\left\{-\frac{N^Z - L_{\max}}{v(L_{\max})L_{\max}}\right\} \tag{11}$$

This formula is indeed more convenient, as we now can minimise it before running $AccurateAlloc$. In particular, we have the following result.

THEOREM 3. *We relax N^X, N^Y, N^Z to be fractional values. Let*

$$W^X = \frac{\varepsilon^2}{2},\ V^X = \ln 2$$
$$W^Y = \frac{1}{v(K_{\max})K_{\max}},\ V^Y = \frac{1}{v(K_{\max})} + \ln\frac{K_{\max}(K_{\max}-1)}{2}$$
$$W^Z = \frac{1}{v(L_{\max})L_{\max}},\ V^Z = \frac{1}{v(L_{\max})} + \ln\frac{L_{\max}(L_{\max}-1)}{2}$$

The optimal values of $\tilde{N}^X, \tilde{N}^Y, \tilde{N}^Z$ (that provides minimal error bound) of Equation 11 are as follows:

$$\tilde{N}^X = \frac{1}{W^X}\left[\frac{1}{\sum_j \frac{c^j}{W^j}}\left(B - \sum_j c^j \frac{V^j + \ln\frac{W^j}{c^j}}{W^j}\right) + \left(V^X + \ln\frac{W^X}{c^X}\right)\right]$$
$$\tilde{N}^Y = \frac{1}{W^Y}\left[\frac{1}{\sum_j \frac{c^j}{W^j}}\left(B - \sum_j c^j \frac{V^j + \ln\frac{W^j}{c^j}}{W^j}\right) + \left(V^Y + \ln\frac{W^Y}{c^Y}\right)\right]$$
$$\tilde{N}^Z = \frac{1}{W^Z}\left[\frac{1}{\sum_j \frac{c^j}{W^j}}\left(B - \sum_j c^j \frac{V^j + \ln\frac{W^j}{c^j}}{W^j}\right) + \left(V^Z + \ln\frac{W^Z}{c^Z}\right)\right]$$

where the index $j \in \{X, Y, Z\}$ iterates over letters "X", "Y", and "Z", respectively.

PROOF SKETCH. By using the Lagrangian relaxation technique to solve the convex optimisation problem posed above, we obtain that the optimal solution of the relaxed problem is as given in the theorem. □

Based on this theorem, we set the values of N^X, N^Y and N^Z to be

$$N^X = \lfloor \tilde{N}^X \rfloor,\ N^Y = \lfloor \tilde{N}^Y \rfloor,\ N^Z = \lfloor \tilde{N}^Z \rfloor, \tag{12}$$

where $\lfloor x \rfloor$ denotes the floor integer value of x. The pseudo code of BudgetFix is shown in Algorithm 1. In particular, it calculates the number of task allocations for each phase (line 2) and then uses the $AccurateAlloc$ procedure to control the tasks passed on between phases while ensuring the best candidate solutions are passed on (lines $4-6$). Thus, it first identifies the set of possible error positions (i.e., S) (line 4) and then determines the set of possible corrections (i.e., U) using the SR algorithm (line 5). Finally it calculates the best position-correction pair, also using SR (line 6).

[6]Ties can be broken in any arbitrary way.

We now turn to prove the quality guarantees of BudgetFix. In particular, we state the following:

THEOREM 4. *Suppose that* Assumptions 1–3 *hold. Let* $0 < \varepsilon \leq 1$, $C_1 = \sum_j \frac{c^j}{W^j}$ *and* $C_2 = \sum_j c^j \frac{V^j + \ln \frac{W^j}{c^j}}{W^j}$. *Given this, BudgetFix guarantees that the probability of having the output pair* $\langle x^*, y^* \rangle$ *incorrect (i.e., either* $x^* \neq x^t$ *or* $y^* \neq y^t(x^t)$*) is at most* $\exp\left\{\frac{-(B-C_2)}{C_1} + \ln 3\right\}$.

That is, the probability that BudgetFix will provide an inaccurate solution is at most $e^{-O(B)}$ where B is the budget limit. The proof of this theorem is omitted.

Budget saving procedure: In practice, we can achieve further savings if S or U contains fewer candidates than $K_{\max}$ or $L_{\max}$, respectively. In particular, suppose that $K = |S| < K_{\max}$. In this case, instead of requesting $N^Y = \lfloor \tilde{N}^Y \rfloor$ tasks within the Fix phase as defined in Equation 12, we set $N^Y = \left\lfloor \tilde{N}^Y \frac{K}{K_{\max}} \right\rfloor$. That is, we proportionally decrease the budget spent on fixing if the the number of candidates carried from the Find phase is less than $K_{\max}$. Similarly, we set $N^Z = \left\lfloor \tilde{N}^Z \frac{L}{L_{\max}} \right\rfloor$ if $L < L_{\max}$. By doing so, we can clearly reduce the amount spent in the Fix and Verify phases (i.e., the total cost will be lower, compared to the original budget B). Note that in this case, Theorem 2 still holds (with the modified values of K and L), and thus, this modified version of BudgetFix still provides performance guarantees but with a lower total cost. In addition, if the tuple $\langle x^*, U \rangle$ that BudgetFix propagates to the Verify phase consists of one single pair $\langle x^*, y(x^*) \rangle$ (i.e., there is only one type of fix for x^*), we stop the process, and provide the pair $\langle x^*, y(x^*) \rangle$ as the solution. By doing so, we can refine the probability of inaccuracy given in Theorem 2 by leaving out the third term.

5. EMPIRICAL EVALUATION

While the previous sections developed theoretical upper-bounds for the expected total estimation error of BudgetFix, we also wish to examine its performance in a realistic setting. To this end, we evaluate BudgetFix through replicating an experiment similar to the text correction task originally used to evaluate the Soylent system [4]. We create a dataset of a total of 97 sentences to be corrected by both algorithms.[7] For each sentence we intentionally placed one mistake, facilitating the existence of a known truth so that the accuracy of the worker's performance could be evaluated. We generate three types of mistakes with equal distribution: spelling mistakes based on an added letter, spelling mistakes based on a missing letter, and grammar mistakes. We place these mistakes into sentences that were of three different complexity levels which generally represent sentences of high, medium, and low complexity. The high complexity sentences are taken from AAMAS articles published between 2008 and 2012. The medium complexity sentences are taken from open computer science textbooks[8] and the low complexity sentences are taken from publicly available children's stories.[9] To quantify the complexity of these sentences we measure the Flesch Reading Ease, Fog Scale Level, and Flesch-Kincaid Grade Level measures, which constitute accepted measures of sentence complexity [6, 10]. These measures for the sentences are 91.19, 6.14, and 3.74, respectively for the easy category, 53.23, 14.64, and 10.85 for the medium category, and 37.55, 18.69, and 14.81 for the hard category.

We then proceed to the Find, Fix and Verify phases as per both algorithms. As was done in the original Soylent experiments, we use Amazon Mechanical Turk and workers are paid the same amount – i.e. \$0.06 per Find task, \$0.08 for Fix tasks, and \$0.04 for Verify tasks. Within Soylent, regardless of the sentence difficulty or budget, a minimum of 10 Find, 5 Fix, and 5 Verify tasks are generated per sentence (see Section 2).

In contrast, BudgetFix generates the number of micro-tasks as per its budget and only propagates those subtasks where fix candidates and are within ε of the most popular candidate. As per Equation 12, BudgetFix sets its budget by taking as its input B, $K_{\max}$, $L_{\max}$, and ε, which in turn set the values of N^X, N^Y, and N^Z, thus controlling how many instances are used for each type of task. Therefore, different combinations of B, ε, $K_{\max}$ and $L_{\max}$ determine the maximum budget per task type. For example, if B = \$2.25, ε=0.1, $K_{\max} = 2$, $L_{\max} = 3$, then $N^X = 10$, $N^Y = 9$, and $N^Z = 22$, which yields a maximum total budget of \$2.20 per sentence. Accordingly, a total of 10 Finds will be used, up to 9 Fixes will be divided among any candidates with no less than 10% of the number of Finds received by the leading candidate, and up to 22 Verify workers will be employed. Note that these values are maximum values. Hence, for a sentence, *no more* than 10 Find, 9 Fix, and 22 Verify micro-tasks will be generated within this configuration. In contrast, Soylent could theoretically pass 5 different Finds to the Fix phase, generating 25 Fix micro-tasks. These potentially could produce 25 different Fixes, which would generate 25×5 Verify micro-tasks.

5.1 Cost versus Accuracy

We first study the results from running the Soylent algorithm, which correctly fixed 88.66% of the sentences in the dataset (86 of 97 sentences) for a total cost of \$135.40 (average \$1.40). We find that Soylent's average cost per sentence linearly increases with the sentence difficulty. This is because Soylent uses 5 Fixes for every potentially viable mistake – which they defined as having an occurrence of more than 20% of the time (at least 2 of 10 occurrences). Similarly, each potential fix candidate is given its own set of 5 verification tasks. Thus, the size of the budget needed will grow linearly with the number of Fix candidates found in the first phase, something our results corroborate. For the set of easier sentences Soylent spends on average \$1.32, for the medium sentences they spends \$1.40, and \$1.46 for the hardest sentence set.

We then proceed to study how BudgetFix performs with the same dataset and how its cost and accuracy are both impacted by different parameter settings for B, ε, $K_{\max}$, and $L_{\max}$. In doing so, we run the BudgetFix algorithm with all permutations of $B \in \{\$1.00, \$1.25, \$1.50, \$1.75, \$2.00, \$2.25, \$2.50\}$, $K_{\max} \in \{2, 3, 4\}$, $L_{\max} \in \{2, 3, 4\}$, and $\varepsilon = \{0.l, 0.2, 0.5, 1.0\}$. In theory we would like to test all $7 \times 3 \times 3 \times 4 = 252$ permutations of BudgetFix. However, only 163 permutations are feasible as 89 permutations yield zero or negative values for N^X. Note that in our experiments, we run the variant of BudgetFix with the *budget savings procedure* (see Section 4.2 for more detail). This implies that the above mentioned budget values of B are maximum budget values. The actual spending costs were typically much less since only a fraction of $\tilde{N}^Y$ and $\tilde{N}^Z$ will be used in cases where fewer Find and Fix candidates are identified in practice, which was found to be the case in sentences with more obvious mistakes. In

[7] This dataset is publicly available at `http://users.ecs.soton.ac.uk/ltt08r/BudgetFix/data.csv`. It initially contained 100 sentences, but three were later discovered to be ambiguous after our trial finished. They were, therefore, removed, leaving 97 sentences in the dataset.

[8] `http://freecomputerbooks.com/compscCategory.html`

[9] `http://www.mightybook.com/free_to_read.html`

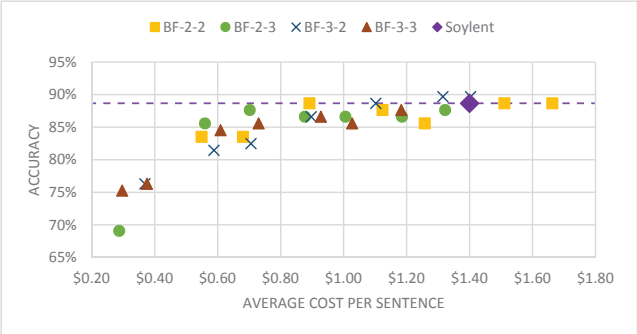

Figure 1: Cost vs Accuracy – comparing the actual cost and accuracy of Soylent and various configurations of BudgetFix (= 1 0).

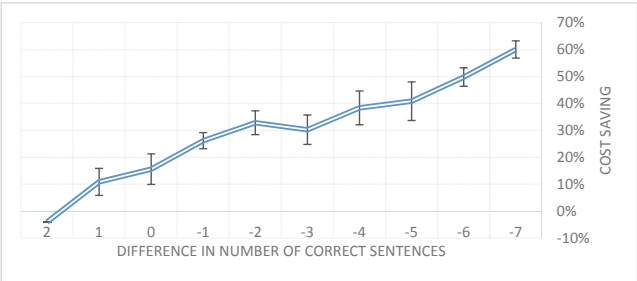

Figure 2: Savings (in % of Soylent's spending, with standard error bars) made by BudgetFix at various levels of accuracy.

fact, over the 163 runs, BudgetFix actually spent on average less than half its allocated budget (44%, standard deviation 15%).

Note that tuning the parameters for $K_{\max}$, $L_{\max}$, and ε for a given budget yields the same or better performance for significantly less cost than the Soylent baseline. For example, in Figure 1 we show a scatter-plot for the cost vs accuracy of various BudgetFix configurations with $K_{\max} \in \{2, 3\}$, $L_{\max} \in \{2, 3\}$, and $\varepsilon = 1.0$ (i.e., all unique Finds to be selected for the Fix phase). The cost and accuracy of Soylent is also shown (i.e., the diamond point) for comparison.

While BudgetFix's main contribution is that it provides performance guarantees while strictly adhering to a budget, and while we found that in all cases BudgetFix spent far less than the budget allows, we also found that not all parameter settings yielded improved performance over the Soylent baseline algorithm. Given the large number of runs and the limited space, we cannot present all these results in this paper, but give a summary of our findings.

We first looked at the accuracy difference between BudgetFix and Soylent. In order to do so, we calculate the Jeffreys binomial proportion confidence interval at 95% confidence level of the Soylent's accuracy: 81.23–93.82%. In particular, provided a configuration of BudgetFix has an accuracy in this range, the null hypothesis (stating that its accuracy is different from that of Soylent) cannot be rejected with at least 95% confidence. Hence, such a BudgetFix configuration is considered to have a similar accuracy level to that of Soylent. In fact, we found 124 such configurations. On average they *spend 32% less* than Soylent (standard deviation: 22%). The remaining 39 configurations are considered significantly less accurate than Soylent because their parameters give either a small N^X or N^Y, i.e. 2, 3, 4. Such low numbers of Find or Fix tasks are shown here to significantly constraint the discovery of the true solution (i.e., x^t and/or $y^t(x^t)$). Therefore, it is not advised to choose a values of B, $K_{\max}$, $L_{\max}$, and ε that result in N^X or N^Y smaller than 4.

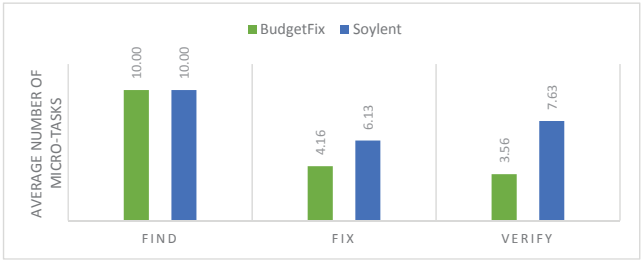

Figure 3: The average numbers of Find, Fix, and Verify tasks allocated by BudgetFix and Soylent.

We now focus on the group of equal accuracy to Soylent's. To further drill down into the performance of this group, we compare the number of correct sentences produced by BudgetFix against that of Soylent (i.e., 86 sentences) and calculate the average spending difference in each case. The result is plotted in Figure 2, which shows the average savings made by BudgetFix at the exact same accuracy (0), at 1 or 2 more correct sentences, and at 1, 2, ..., 7 less correct sentences. In brief, the savings by BudgetFix increase monotonically as the level of accuracy decreases, as expected. However, the magnitude of the savings is notable. For the same level of accuracy, BudgetFix spent on average 16% less than Soylent. More impressively, in cases where its spending is to be reduced by roughly a third (compared to Soylent's), it only sacrifices 2–3% in accuracy (i.e. 2–3 sentences).

The results above suggests that BudgetFix distributes its budget to the three FFV phases in an efficient manner while it guarantees the cost per sentence not to exceed the given maximal budget. In that respect, the next section looks into the difference in the way BudgetFix and Soylent allocate tasks (and money) to the different phases.

5.2 Task Allocation

The key to the success of BudgetFix is the method by which it dynamically allocates its budget, especially in the Fix and Verify phases. To explore this point, we study the relationship between BudgetFix and Soylent's budget allocation, and present the average numbers of Finds, Fixes and Verifies used by both algorithms in Figure 3. These results are from BudgetFix configuration with $B = \$2.25$, $K_{\max} = 2$, and $L_{\max} = 3$, and $\varepsilon = 0.1$. This particular configuration is chosen because it uses the same 10 Find tasks as in the case of Soylent and both have the same accuracy, allowing us to easily compare the number of Fix and Verify tasks between the two algorithms.[10] It should be noted that, at the same level of accuracy, this configuration of BudgetFix spends 23% less per sentence than Soylent, \$1.08 vs \$1.40. This is possible thanks to the Successive Reject approach in the Fix and Verify phases (see Sections 4.1.2 and 4.1.3) and the budget saving procedure (see Section 4.2). In conjunction, they help BudgetFix control the average numbers of Fix and Verify tasks, 4.16 and 3.56 respectively, as shown in Figure 3. In contrast, as Soylent 'branches' tasks on all unique (with 20% agreement) Finds and Fixes, the number of Fix tasks is higher at 6.13 and the number of Verify tasks is significantly higher at 7.63.

We now look into how the Fix and Verify tasks are allocated to sentences of different difficulty levels. Table 1 shows the average number of tasks generated in the Fix and Verify phases by BudgetFix and Soylent for sentences of easy, medium, and hard levels. In the Fix phase the number of tasks requested by both

[10]There is no difference in the number of Find tasks since both algorithms use all their allowance in the Find phase.

	Fix		Verify	
Difficulty	BudgetFix	Soylent	BudgetFix	Soylent
Easy	4.00 (1.41)	5.16 (1.53)	4.69	7.66
Medium	4.27 (1.23)	6.50 (1.40)	3.00	7.00
Hard	4.23 (1.23)	6.71 (1.62)	3.00	8.14

Table 1: The average number of tasks in the Fix and Verify phases generated by BudgetFix(2.25, 2, 3, 0.1) and Soylent with respect to sentence complexity. Values in bracket represent the average number of tasks passed on to the Verify from the Fix phase.

algorithms generally (at least for Soylent) increases with the difficulty of the sentences. Intuitively, one would expect AMT workers would pick out more 'incorrect' mistakes from long and difficult sentences than short and easy one in the Find phase, hence requiring more Fix tasks. However, we note that more difficult sentences do not necessarily generate more tasks in the Verify phase for both BudgetFix and Soylent. In that phase, since the total number of Verifies depends on the number of unique Fixes created by AMT workers, the lower the number of unique Fixes, the lower the number of Verifies. In fact, as can be seen from Table 1, this particular BudgetFix configuration chooses fewer Fixes than Soylent, out of which even fewer unique answers are passed to the Verify phase (1.41, 1.23, 1.23 for BudgetFix compared to 1.53, 1.40, and 1.62 for Soylent). This result suggests that it is not just the complexity of a crowdsourcing task that determines the cost but, crucially, on the performance of the workers (e.g., in creating few unique Fixes) at different phases of the process.

6. CONCLUSIONS

We investigated the interdependent task allocation under budget constraints in crowdsourcing systems. In particular, we consider the FFV workflow, where the goal is to maximise the accuracy level of the outcome with respect to a budget limit of task allocation. To solve this problem, we proposed BudgetFix, a novel crowdsourcing algorithm, that efficiently identifies the total number of tasks for the Find, Fix, and Verify phases. We also proved that the algorithm can guarantee that the probability of receiving an inaccurate outcome is at most $e^{-O(B)}$ where B is the budget limit. We demonstrated through real Amazon Mechanical Turk experiments that with 16% less budget, our method can achieve similar accuracy, compared to that of an existing algorithm used in Soylent. Moreover, we also showed that by reducing the budget of BudgetFix by roughly a fourth, the accuracy would only take a maximum 2-3% hit, and by spending less than half of Soylent's average budget results in a maximum sacrifice of 7% in accuracy. In summary, BudgetFix can achieve similar accuracy, compared to Soylent, but with a significantly lower spending cost. Given this, it is very useful in applications with complex workflows and low budgets.

Note that once a phase is considered as ready, BudgetFix will move to the next phase and does not return to the previous ones. This might cause inefficiency if the output of the previous phase is inaccurate, and will affect the quality of the work done within the subsequent phases. Given this, one possible way to make BudgetFix more efficient is to allow loopy behaviour. That is, we allow BudgetFix to return to previous phases, once it detects inaccuracy of the outcome within a particular phases. However, adding such behaviour changes makes our model significantly more complex, and the techniques we use within this paper are not suitable for analysing such complex models. Therefore, as future work, we aim to extend our analysis to the abovementioned model, making BudgetFix more efficient.

Acknowledgements

This work was carried out as part of the ORCHID project funded by EPSRC (EP/I011587/1).

7. REFERENCES

[1] O. Amir, D.G. Rand, and Y. Gal. Economic games on the internet: The effect of $1 stakes. *PloS one*, 7(2), 2012.

[2] J.-Y. Audibert, S. Bubeck, and R. Munos. Best arm identification in multi-armed bandits. *COLT*, pages 41–53, 2010.

[3] Amos Azaria, Yonatan Aumann, and Sarit Kraus. Automated agents for reward determination for human work in crowdsourcing applications. *Autonomous Agents and Multi-Agent Systems*, pages 1–22, 2013.

[4] M. S. Bernstein, G. Little, R. C. Miller, B. Hartmann, M. S. Ackerman, D. R. Karger, D. Crowell, and K. Panovich. Soylent: a word processor with a crowd inside. In *UIST*, pages 313–322, 2010.

[5] P. Dai, C. H. Lin, Mausam, and D. S. Weld. POMDP-based control of workflows for crowdsourcing. *Artificial Intelligence (AIJ)*, 202:52–85, 2013.

[6] R. Flesch. A new readability yardstick. *Journal of Applied Psychology*, 32(3):221–233, 1948.

[7] C.-J. Ho and J. Wortman-Vaughan. Online task assignment in crowdsourcing markets. In *AAAI*, pages 45–51, 2012.

[8] E. Kamar, S. Hacker, and E. Horvitz. Combining human and machine intelligence in large–scale crowdsourcing. *AAMAS*, pages 467–474, 2012.

[9] D. R. Karger, S. Oh, and D. Shah. Budget-optimal task allocation for reliable crowdsourcing systems. *CoRR*, abs/1110.3564, 2011.

[10] J. P. Kincaid, R. P. Fishburne, R. L. Rogers, and B. S. Chissom. Derivation of New Readability Formulas (Automated Readability Index, Fog Count and Flesch Reading Ease Formula) for Navy Enlisted Personnel. Technical report, February 1975.

[11] C. H. Lin, Mausam, and D. S. Weld. Dynamically switching between synergistic workflows for crowdsourcing. *AAAI*, pages 87–93, 2012.

[12] G. Little, L. B. Chilton, M. Goldman, and R. C Miller. Turkit: tools for iterative tasks on mechanical turk. *KDD-HCOMP*, pages 29–30, 2009.

[13] E. Simpson, S. Roberts, I. Psorakis, A. Smith, and Chris Lintott. Bayesian combination of multiple imperfect classifiers. *NIPS*, 2011.

[14] R. Stranders, S. Ramchurn, B. Shi, and N. R. Jennings. Collabmap: Augmenting maps using the wisdom of crowds. In *HCOMP*, 2011.

[15] L. Tran-Thanh, S. Stein, A. Rogers, and N. R. Jennings. Efficient crowdsourcing of unknown experts using multi-armed bandits. In *ECAI*, pages 768–773, 2012.

[16] L. Tran-Thanh, M. Venanzi, A. Rogers, and N. R. Jennings. Efficient budget allocation with accuracy guarantees for crowdsourcing classification tasks. *AAMAS*, pages 901–908, 2013.

[17] P. Welinder, Branson S., S. Belongie, and P. Perona. The multidimensional wisdom of crowds. *NIPS*, pages 2424–2432, 2010.