HAL open science

Reducing the Cost of Aggregation in Crowdsourcing

Rituraj Singh, Loïc Hélouët, Zoltan Miklos

To cite this version:

Rituraj Singh, Loïc Hélouët, Zoltan Miklos. Reducing the Cost of Aggregation in Crowdsourcing. Transactions on Large-Scale Data- and Knowledge-Centered Systems, In press, pp.1-38. hal-03482460

HAL Id: hal-03482460

https://hal.science/hal-03482460v1

Submitted on 15 Dec 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Reducing the Cost of Aggregation in Crowdsourcing

Rituraj Singh, Loïc Hélouët, and Zoltan Miklos

Univ. Rennes/INRIA/CNRS/IRISA
rituraj.singh@irisa.fr,loic.helouet@inria.fr, zotlan.miklos@irisa.fr

Abstract. Crowdsourcing is a way to solve problems that need human contribution. Crowdsourcing platforms distribute replicated tasks to workers, pay them for their contribution, and aggregate answers to produce a reliable conclusion. A fundamental problem is to infer a consensual answer from the set of returned results. Another problem is to obtain this answer at a reasonable cost: unlimited budget allows hiring experts or large pools of workers for each task but a limited budget forces to use resources at best. Last, crowdsourcing platforms have to detect and ban malevolent users (also known as "spammers") to achieve good accuracy of their answers.
This paper considers crowdsourcing of simple Boolean tasks. We first define a probabilistic inference technique, that considers difficulty of tasks and expertise of workers when aggregating answers. We then propose CrowdInc, a greedy algorithm that reduces the cost needed to reach a consensual answer. CrowdInc distributes resources dynamically to tasks according to their difficulty. The algorithm solves batches of simple tasks in rounds that estimate workers expertize, tasks difficulty, and synthesizes a plausible aggregated conclusion and a confidence score using Expectation Maximization. The synthesized values are used to decide whether more workers should be hired to increase confidence in synthesized answers. We show on several benchmarks that CrowdInc achieves good accuracy, reduces costs and we compare its performance to existing solutions. We then use the estimation of CrowdInc to detect spammers and study the impact of spammers on costs and accuracy.

1 Introduction

Crowdsourcing is a way to solve tasks that need human contribution. These tasks include image annotation or classification, polling, etc. Employers publish tasks on an Internet platform, and these tasks are realized by workers in exchange for a small incentive [2]. Workers are very heterogeneous: they have different origins, domains of expertise, and expertise levels. Some of the workers might even behave maliciously and try to receive payment without working (they simply return random answers), or return wrong answers on purpose. These workers are often referred to as *spammers*. To deal with this heterogeneity, tasks are usually replicated: each task is assigned to *several workers*. Redundancy is also essential to collect workers opinion: in this setting, work units are the basic elements of a

larger task that can be seen as a poll. One can safely consider that each worker executes his assigned task independently, and hence returns his own belief about the answer. As workers can disagree, the role of a platform is then to build a consensual final answer out of the values returned.

A fundamental problem in crowdsourcing is then to infer a *correct answer* from the set of returned results. Another challenge is to obtain a reliable answer at a *reasonable cost*: unlimited budget allows hiring experts or large pools of workers for each task but a limited budget forces to use resources at best. Last, crowdsourcing platforms have to detect and ban malevolent users to achieve a good accuracy and avoid paying for random guesses or wrong answers of spammers.

A natural way to derive a final answer is **Majority Voting** (MV), i.e. choose as conclusion the most represented answer. A limitation of MV is that all answers have equal weight, regardless of expertise of workers. If a crowd is composed of only few experts, and of a large majority of novices, MV favors answers from novices. However, in some domains, an expert worker may give better answer than a novice and his answer should be given more weight. One can easily replace MV by a weighted vote. However, this raises the question of measuring workers expertise, especially when workers competences are not known a priori.

Crowdsourcing platforms such as Amazon Mechanical Turk (AMT) do not have prior knowledge about the expertise of their workers. A way to obtain initial measure of workers expertise is to use **Golden Questions** [11]. Several tasks with known *ground truth* are used explicitly or hidden to evaluate workers expertise. As already mentioned, a single answer for a particular task is often not sufficient to obtain a reliable answer, and one has to rely on redundancy, i.e. distribute the same task to several workers and aggregate results to build a final answer. Standard *static* approaches on crowdsourcing platforms fix a prior number of k workers per task. Each task is published on the platform and waits for bids by k workers. There is no guideline to set the value for k, but two standard situations where k is fixed are frequently met. The first case is when a client has n tasks to complete with a total budget of B_0 incentive units. Each task can be realized by $k = B_0/n$ workers. The second case is when an initial budget is not known, and the platform fixes an arbitrary redundancy level. In this case, the number of workers allocated to each task is usually between 3 and 10 [7]. It is assumed that the distribution of work is uniform, i.e. that each task is assigned the same number of workers. An obvious drawback of static allocation of workers is that all tasks benefit from the same work power, regardless of their difficulty. Even a simple question where the variance of answers is high calls for sampling of larger size. So, one could expect each task t to be realized by k_t workers, where k_t is a number that guarantee that the likelihood to change the final answer with an answer returned by one additional worker is low. However, without prior knowledge on task's difficulty and on variance in answers, this number k_t cannot be fixed.

This paper proposes a new algorithm called CrowdInc to address the questions of answers aggregation, task allocation, and cost of crowdsourcing. For

simplicity, we consider Boolean filtering tasks, i.e. tasks with answers in $\{0, 1\}$, but the setting can be easily extended to tasks with any finite set of answers. These tasks are frequent, for instance to decide whether a particular image belongs or not to a given category of pictures. We consider that each binary task has a *truth label*, i.e. there exists a ground truth for each task. Each worker is asked to answer 0 or 1 to such a task and returns a so-called *observed label*, which may differ from the ground truth. The *difficulty* of a task is a real value in $[0, 1]$. A task with difficulty 0 is a very easy task and a task with difficulty 1 a very complex one. The *expertise* of a worker is modeled in terms of *recall* and *specificity*. **Recall** (also called true positive rate) measures the proportion of correct observed labels given by a worker when the ground truth is 1. On contrary, **specificity** (also called true negative rate) measures the proportion of correct observed labels given by a worker when the ground truth is 0. We propose a generating function to measure the probability of accuracy for each of the truth label (0/1) based on the *observed label, task difficulty, and worker expertise*. We rely on an Expectation Maximization (EM) based algorithm to maximize the probability of accuracy of aggregated final answers for each task and jointly estimate the difficulty of each task as well as expertise of the workers. The algorithm provides a greater weight to expert workers. In addition, if a worker with high *recall* makes a mistake in the *observed label*, then it increases the difficulty of the task (correspondingly for specificity). Along with, if expert workers fail to return a correct answer, then the task is considered difficult. The EM algorithm converges with a very low error rate and at the end returns the task *difficulty*, worker *expertise* and the *final estimated label* for each task based on *observed labels*. Additionally, we propose a dynamic worker allocation algorithm that handles at the same time aggregation of answers, and optimal allocation of a budget to reach a consensus among workers. The algorithm works in two phases. It starts with an initial *Estimation* phase. As we do not have any prior information about the tasks difficulties and workers expertise, we allocate one third of total budget to evaluate these parameters. Based on the answers provided by the human workers for each task, we first derive the difficulty of tasks, final aggregated answers, along with the worker expertise using an EM algorithm. For each task, we estimate the likelihood that the aggregated answer is the ground truth. We terminate tasks which are above the derived threshold at that particular instance. The second phase is an *Exploration* phase. Based on each of the estimated task difficulty, we start to allocate workers for each of the remaining tasks. The process continues until all tasks are terminated or the whole budget is consumed.

A second contribution of the paper is an experimental evaluation of the CrowdInc algorithm. We show on several popular benchmarks that CrowdInc achieves a good accuracy (that is as good or even better than existing approaches), for a reduced cost, and with a reasonable time overhead. This overhead is mainly due to the use of Expectation Maximization, which is an iterative process that converges towards a local optimum. An advantage in using EM is that values of several variables such as workers recall and specificity is computed

in addition to synthesis of final aggregated answers. In the last section, we define a simple behavioral model for spammers, and show that the estimation of recall and specificity of workers can be used to detect spammers. We repeat our experimental evaluation, show the effect of spammers on costs and accuracy of answers synthesized by CrowdInc, and show that a simple policy imposing some thresholds on recall and specificity allows to detect most of spammers.

Related work: Several papers have considered tools such as EM to aggregate answers or allocate tasks. We only highlight a few works that are close to our approach, and refer interested readers to [26] for a more complete survey of the domain. This work compares 17 truth inference algorithms, evaluated on 5 real datasets. The study compares techniques, but also evaluates the effect of qualification tests on aggregation mechanisms. One important conclusion of the survey is that there is no ideal algorithm, and that the most appropriate inference algorithm differs depending on the input data. Another interesting conclusion of the survey is that qualification tests do not necessarily improve accuracy when the performance of an algorithm is already good on a particular dataset.

Zencrowd [4] considers workers competences in terms of accuracy (ratio of correct answers) and aggregates answers using EM. PM [12] considers an optimization scheme based on Lagrange multipliers. Workers accuracy and ground truth are the hidden variables that must be discovered in order to minimize the deviations between workers answers and aggregated conclusions. D&S [3] uses EM to synthesize answers that minimize error rates from a set of patient records. It considers recall and specificity, but not difficulty of tasks. The approach of [10] proposes an algorithm to assign tasks to workers, synthesize answers, and reduce the cost of crowdsourcing. It assumes that all tasks have the same difficulty, and that reliability of a worker is a consistent value in $[0, 1]$ (hence considering accuracy as a representation of competences). CrowdBudget [19] is an approach that divides a budget B among K existing tasks to achieve a low error rate, and then uses MV to aggregate answers. Workers answers follow an unknown Bernoulli distribution. The objective is to affect the most appropriate number of workers to each task in order to reduce the estimation error. Aggregation is done using Bayesian classifiers combination (BCC). The approach in [20] extends BCC with communities and is called CBCC. Each worker is supposed to belong to a particular (unknown) community, and to share characteristics of this community (same recall and specificity). This assumption helps improving accuracy of classification. Expectation maximization is used by [17] to improve supervised learning when the ground truth in unknown. This work considers recall and specificity of workers and proposes a maximum-likelihood estimator that jointly learns a classifier, discovers the best experts, and estimates ground truth. Most of the works cited above consider expertise of workers but do not address tasks difficulty. An exception is GLAD (Generative model of Labels, Abilities, and Difficulties) [24] that proposes to estimate tasks difficulty as well as workers accuracy to aggregate final answers. The authors recall that EM is an iterative process that stops only after converging, but demonstrate that the EM approach needs only a few minutes to tag a database with 1 million images. The authors

in [1] consider difficulty and error parameter of the worker. Notice that in most of the works cited above, tasks difficulty is not considered and expertise is modeled in terms of accuracy rather than recall and specificity. Generally the database and Machine Learning communities focus on data aggregation techniques and leave budget optimization apart. Raykar *et al.* [16] introduce sequential crowdsourced labelling: instead of asking for all the labels in one shot, one decides at each step whether evaluation of a task shall be stopped, and which worker should be hired. The model incorporates a Bayesian model for workers (workers are only characterized by their accuracy), and cost. Then, sequential crowdsourced labelling amounts to exploring a (very large) Markov decision process (states contain all pairs of task/label collected at a given instant) with a greedy strategy.

It is usually admitted [26] that recall and specificity give a finer picture of worker's competence than accuracy. Our work aggregates workers answers using expectation maximization with three parameters : task difficulty, recall and specificity of workers. The CrowdInc algorithm uses this EM aggregation to estimate error and difficulty of tasks. This error allows to compute dynamically a threshold to stop tasks which aggregated answers have reached a reasonable reliability and to allocate more workers to the most difficult tasks, hence saving costs. As the difficulty of tasks is initially unknown, we assign the same rewards to every task realization, and accordingly define the cost of dataset tagging as the number of tagging micro-tasks realized.

Spammer detection has been addressed in several works. We mention below some of works that are related to the spammer detection scheme proposed in this paper, without claiming exhaustiveness. [18] explains that without a priori knowledge on accuracy of workers, one runs the risk of hiring spammers, and even a majority of malevolent workers. The authors define spammers as workers that assign labels randomly without looking at tasks, and propose a Bayesian algorithm called SpEM. SpEM computes a spammer score, iteratively eliminates the spammers and estimates the consensus labels based only on answers of faithful workers. Experiments on simulated and real data show that the proposed approach is better than former approaches in terms of accuracy and number of workers hired. The setting proposed hereafter is simpler: it uses thresholds on recall and specificity to characterize spammers, and does not eliminate spammers during the dataset tagging process. Xu *et al.* [25] analyze users behaviors during crowdsourcing campaigns. They search for different types of suspicious behaviors to identify spammer accounts. They were able to reveal a complete ecosystem of colluding spammers with this approach. In our work, we consider spammer types that resemble these suspicious spammer behaviors, but we do not consider collusions of workers. Two types of spammers are considered in [9]: bad faith workers and workers with poor competence. While the former are malevolent workers, [9] advocates that the latter are not, and propose to use Machine Learning to detect real spammers based on side information such as the time spent to answer, the number of tasks performed... The proposed classifier showed good detection scores. In this work, we consider more types of spammers. The

bad faith workers are called Type 1 spammers, and the second type of spammers of [9] are simply faithful workers with poor accuracy in our setting.

[22] considers a particular type of attack called *sybil attack* where an intruder coordinates corrupted workers to influence agregation answers and earn money. It proposes a technique to evaluate workers reliability using golden tasks, a probabilistic task assignment for golden tasks to hide qualification tests from Sybil attacker, and an online golden task creation from answers of trusted workers to avoid shortage of qualification tests. The framework considered is a static allocation of workers to tasks. EM is used to jointly estimate aggregated labels and workers accuracy. The proposed framework computes a reliability score and a sybil score for workers, that measures if a worker contributed to false answers to golden questions. It assign tasks based on reliability and Sybil score and lowers the weight of workers suspected to be corrupted in aggregation. The spammer detection proposed in this paper assumes independent spammers, and does not use golden questions to qualify workers. Hence, spammers that can only be detected through their unusual accuracy scores.

In [13] an attacker hires a set of malicious workers and can manipulate their label to influence the final labels synthesized, while remaining undetected. The setting proposed is a Dawid-Skene model [3], where the probability to return a particular answer m' when the ground truth is m is given by a confusion matrix. The goal of an attacker is to manipulate labels in an optimal way, for instance by getting high values on the malicious workers' ability parameters to favor their answer. This objective is formalized as a cost function to maximize. The function considers both the success of the attack on final answers and the reliability scores of malicious workers. The optimal value for this function and the associated answering strategy are found with an iterative algorithm. A limitation of the approach is that attackers know the answers returned by normal workers, which is not the case in general. In our setting, we assume that a part of the workers is malicious, but they do not know other workers label, which does not allow fine-tuning of an attack on labels. We hence focus on attackers profiles with simple deterministic behaviors that depend only on ground truth.

The crowdsourcing framework of [8] allows workers to reject a task if they do not feel competent for it, and implements a payment scheme that encourages rejection to favor accurate answers. The framework can contain spammers that answer with random guesses or simply reject every task. Aggregation is a weighted majority voting, and perception of probable reward by rational workers is modeled with prospect theory, using cost functions. Spammers and honest workers exhibit different and distinguishable behaviors. The objective is to find the appropriate weight for every worker in the crowd to obtain accurate aggregated answers and rule out spammers. To encourage honest workers to skip questions, the policy is to pay the minimal reward to workers with poor answers to golden questions. Now, honest workers have a distorted vision of their probability to return a correct answer. In the proposed setting aggregation is majority voting, and the difficulty of tasks is assumed identical for each task. In this framework, the goal is then to assign thresholds for acceptable success rate to

encourage skipping tasks, but still allow for detection of spammers and maximization of accuracy of the system. In our setting, we do not consider the possibility to skip questions (i.e. all workers return an answer). We use an aggregation mechanism that accounts for returned answers but estimates tasks difficulty to avoid penalizing honest workers that return wrong answers to difficult questions.

The rest of the paper is organized as follows. In Section 2, we introduce our notations, the factors that influence results during aggregation of answers, and the EM algorithm. In Section 3, we present a model for workers and our EM-based aggregation technique. We detail the CrowdInc algorithm to optimize the cost of crowdsourcing in Section 4. We then give results of experiments with our aggregation technique and with CrowdInc in Section 5. Section 6 addresses the problem of spammer detection. It shows the results of an experiment to study the impact of spammers and proposes a simple spam detection technique based on comparison of workers recall and specificity with fixed thresholds. Finally we conclude and give future research directions in Section 7.

2 Preliminaries

In the rest of the paper, we will work with variables and probabilities. A *random variable* is a variable whose value depends on random phenomenon. For a given variable x, we denote by $Dom(x)$ its domain (Boolean, integer, real, string,...). For a particular value $v \in Dom(x)$ we denote by $x = v$ the event "x has value v". A probability measure $Pr()$ is a function from a domain to interval $[0, 1]$. We denote by $Pr(x = v)$ the probability that event $x = v$ occurs. In the rest of the paper, we mainly consider Boolean events, i.e. variables with domain $\{0, 1\}$. A probability of the form $Pr(x = v)$ only considers occurrence of a single event. When considering several events, we define the *joint probability* $Pr(x = v, y = v')$ the probability that the two events occur simultaneously. The notation extends to an arbitrary number of variables. If x and y are independent variables, then $Pr(x = v, y = v') = Pr(x = v) \cdot Pr(y = v')$. Last, we will use conditional probabilities of the form $Pr(x = v \mid y = v')$, that defines the probability for an event $x = v$ when it is known that $y = v'$. We recall that, when $P(y = v') > 0$ $Pr(x = v \mid y = v') = \frac{Pr(x=v,y=v')}{Pr(y=v')}$.

2.1 Factors influencing efficiency of crowdsourcing

During task labeling, several factors can influence the efficiency of crowdsourcing, and the accuracy of aggregated answers. The first one is **Task difficulty**. Tasks submitted to a crowdsourcing platform by a client are simple questions, but may nevertheless require some expertise. Even within a single application type, the difficulty for the realization of a particular task may vary from one experiment to another: tagging an image can be pretty simple if the worker only has to decide whether the picture contains an animal or an object, or conversely be very difficult if the Boolean question asks whether a particular insect picture shows an hymenopteran (an order of insects). Similarly, **Expertise of workers** plays a major role in accuracy of aggregated answers. In general, an expert worker performs better on a specialized task than a randomly chosen worker

without particular competence in the domain. For example, an entomologist can annotate an insect image more precisely than any random worker.

The technique used for **Amalgamation** also plays a major role. Given a set of answers returned for a task t, one can aggregate the results using *majority voting* (MV), or more interesting, as a weighted average answer where individual answers are weighted by workers expertise. However, it is difficult to get a prior measure of workers expertise and of the difficulty of tasks. Many crowdsourcing platforms use MV and ignore difficulty of tasks and expertise of workers to aggregate answers or assign tasks to workers. We show in Section 5 that MV has a low accuracy. In our approach, expertise and difficulty are hidden parameters evaluated from the sets of answers returned. This allows considering new workers with a priori unknown expertise. One can also start with an a priori measure of tasks difficulty and of workers expertise. Workers expertise can be known from former interactions. It is more difficult to have an initial knowledge of tasks difficulties, but one can start with an a priori estimation. However, these measures need to be re-evaluated on the fly when new answers are provided by the crowd. Starting with a priori measures does not change the algorithms proposed hereafter, but may affect the final aggregated results.

In Section 3, we propose a technique to estimate the expertise of workers and the difficulty of tasks on the fly. Intuitively, one wants to consider that a task is difficult if even experts fail to provide a correct answer for this task, and consider it as easy if even workers with low competence level answer correctly. Similarly, a worker is competent if he answers correctly to difficult tasks. Notice however that to measure difficulty of tasks and expertise of workers, one needs to have the final answer for each task. Conversely, to precisely estimate the final answer one needs to have the worker expertise and task difficulty. This is a chicken and egg situation, but we show in Section 3 how to get plausible values for both using EM.

The next issue to consider is the **cost** of crowdsourcing. Workers receive incentives for their work, but usually clients have limited budgets. Some task may require a lot of answers to reach a consensus, while some may require only a few answers. Therefore, a challenge is to spend efficiently the budget to get the most accurate answers. In Section 4, we discuss some of the key factors in budget allocation. Many crowdsourcing platforms do not considers *difficulty*, and allocate the same number of workers to each task. The allocation of many workers to simple tasks is usually not justified and is a waste of budget that would be useful for difficult tasks. Now, tasks difficulty is not a priori known. This advocates for on the fly worker allocation once the difficulty of a task can be estimated. Last, one can stop collecting answers for a task when there is an evidence that enough answers have been collected to reach a consensus on a final answer. An immediate solution is to measure the confidence of final aggregated answer and take as **Stopping Criterion** for a task the fact that this confidence exceeds a chosen threshold. However, this criterion does not works well in practice as clients usually want high thresholds for all their tasks. This may lead to consuming all available budget without reaching an optimal accuracy. Ideally, we would

like to have a stopping criterion that balances confidence in the final answers and budget, and optimizes the overall accuracy of answers for all the tasks.

2.2 Expectation Maximization

Expectation Maximization [5] is an iterative technique to obtain maximum likelihood estimation of parameter of a statistical model when some parameters are unobserved and *latent*, i.e. they are not directly observed but rather inferred from observed variables. In some sense, the EM algorithm is a way to find the best fit between data samples and parameters. It has many applications in Machine Learning, data mining and Bayesian statistics.

Let $\mathcal{M}$ be a model which generates a set $\mathcal{X}$ of observed data, a set of missing latent data $\mathcal{Y}$, and a vector of unknown parameters θ, along with a likelihood function $L(\theta \mid \mathcal{X}, \mathcal{Y}) = p(\mathcal{X}, \mathcal{Y} \mid \theta)$. In this paper, observed data $\mathcal{X}$ represents the answers provided by the crowd, $\mathcal{Y}$ depicts the *final answers* which need to be estimated and are hidden, and parameters in θ are the *difficulty* of tasks and the *expertise* of workers. The *maximum likelihood estimate* (MLE) of the unknown parameters is determined by maximizing the marginal likelihood of the observed data. We have $L(\theta \mid \mathcal{X}) = p(\mathcal{X} \mid \theta) = \int p(\mathcal{X}, \mathcal{Y} \mid \theta) d\mathcal{Y}$. The EM algorithm computes iteratively MLE, and proceeds in two steps. At the kth iteration of the algorithm, we let θ^k denote the estimate of parameters θ. At the first iteration of the algorithm, θ^0 is randomly chosen.

E-Step: In the E step, the missing data are estimated given observed data and current estimate of parameters. The E-step computes the expected value of $L(\theta \mid \mathcal{X}, \mathcal{Y})$ given the observed data $\mathcal{X}$ and the current parameter θ^k. We define

$$Q(\theta \mid \theta^k) = \mathbb{E}_{\mathcal{Y} \mid \mathcal{X}, \theta^k}[L(\theta \mid \mathcal{X}, \mathcal{Y})] \tag{1}$$

In the crowdsourcing context, we use the E-Step to compute the probability of occurrence of $\mathcal{Y}$ that is the *final answer* for each task, given the observed data $\mathcal{X}$ and parameters θ^k obtained at k^{th} iteration.

M-Step: The M-step finds parameters θ that maximize the expectation computed in Equation. 1.

$$\theta^{k+1} = \arg\max_{\theta} Q(\theta \mid \theta^k) \tag{2}$$

Here, with respect to estimated probability for *final answers* in $\mathcal{Y}$ from the last E-Step, we maximize the joint log likelihood of the observed data $\mathcal{X}$ (answer provided by the crowd), hidden data $\mathcal{Y}$ (final answers), to estimate the new value of θ^{k+1} i.e. the *difficulty* of tasks and the *expertise* of workers. The E and M steps are repeated until the value of θ^k converges. A more general version of the algorithm is presented in Algorithm 1.

3 The Aggregation model

We address the problem of evaluation of binary properties of samples in a dataset by aggregation of answers returned by participants in a crowdsourcing system. This type of application is frequently met: one can consider for instance a database of n images, for which workers have to decide whether each image is clear or blur, whether a cat appears on the image, etc. The evaluated property is binary, i.e. workers answers can be represented as a label in $\{0, 1\}$. From now, we will consider that tasks are elementary work units which objective is to associate a binary label to a particular input object. For each task, an actual

Algorithm 1: General EM Algorithm

Data: Observed Data $\mathcal{X}$
Result: Parameter values θ, Hidden data $\mathcal{Y}$
Initialize parameters in θ^0 to some random values.
while $||\theta^k - \theta^{k-1}|| > \epsilon$ **do**
  Compute the expected possible value of $\mathcal{Y}$, given θ^k and observed data $\mathcal{X}$
  Use $\mathcal{Y}$ to compute the values of θ that maximize $Q(\theta \mid \theta^k)$.
end
return parameter θ^k, Hidden data $\mathcal{Y}$

ground truth exists, but it is not known by the system. We assume a set of k independent workers, whose role is to realize a task, i.e. return an *observed label* in $\{0, 1\}$ according to their perception of a particular sample. We consider a set of tasks $T = \{t_1, \dots t_n\}$ for which a label must be evaluated. For a task $t_j \in T$ the observed label given by worker $1 \leq i \leq k$ is denoted by l_{ij}. We let y_j denote the *final label* of a task t_j obtained by aggregating the answers of all workers. $L_j = \bigcup_{i \in 1..k} l_{ij}$ denotes the set of all labels returned by workers for task t_j, L denotes the set of all observed labels, $L = \bigcup_{j \in 1..n} L_j$. The goal is to estimate the ground truth by synthesizing a set of *final labels* $Y = \{y_j, 1 \leq j \leq n\}$ from the set of *observed labels* $L = \{L_j\}$ for all tasks.

Despite the apparent simplicity of the problem, crowdsourcing binary tagging tasks hides several difficulties, originating from unknown parameters. These parameters are the difficulty of each task, and the expertise of each worker. The difficulty of task t_j is modeled by a parameter $d_j \in (0, 1)$. Here value 0 means that the task is very easy, and can be performed successfully by any worker. On the other hand, $d_j = 1$ means that task t_j is very difficult. A standard way to measure expertise is to define workers accuracy as a pair $\xi_i = \{\alpha_i, \beta_i\}$, where α_i is called the *recall* of worker i and β_i the *specificity* of worker i. The **recall** is the probability that worker i annotates an image j with label 1 when the ground truth is 1, i.e. $\alpha_i = Pr(l_{ij} = 1 | y_j = 1)$. The **specificity** of worker i is the probability that worker i annotates an image j with 0 when the ground truth is 0, i.e. $\beta_i = Pr(l_{ij} = 0 | y_j = 0)$.

In literature,[26] the expertise of workers is often quantified in terms of *accuracy*, i.e. $Pr(l_{ij} = y_j)$. However, if the data samples are unbalanced, i.e. the number of samples with actual ground truth 1 (respectively 0) is much larger than the number of samples with ground truth 0 (respectively 1), defining competences in terms of *accuracy* leads to bias. Indeed, a worker who is good in classifying images with ground truth 1 can obtain bad scores when classifying image with ground truth 0, and yet get a good accuracy (this can be the case of a worker that always answers 1 when choosing a label for a task). *Recall* and *Specificity* overcomes the problem of bias and separates the worker expertise, considering their ability to answer correctly when the ground truth is 0 and

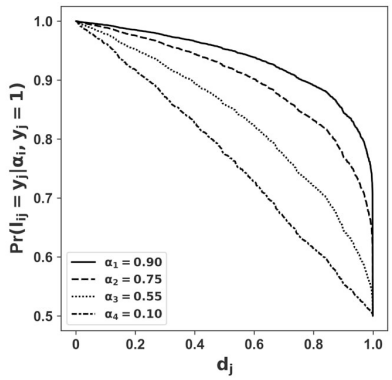

Fig. 1: Generative function for the probability to get $l_{ij} = 1$, given $y_j = 1$, for growing values of task difficulty. The curves represent different recall values for the considered workers.

when it is 1, and hence give a more precise representation of workers competences.

Recall and specificity allow us to build a probabilistic model (a generative model) for workers answers. We assume that workers have constant behaviors and are faithful, i.e. do not return wrong answers intentionally. We also assume that workers do not collaborate (their answers are independent variables). Under these assumptions, knowing the recall α_i and specificity β_i of a worker i, we build a model that generates the probability that he returns an *observed label* l_{ij} for a task j with difficulty d_j:

$$Pr(l_{ij} = y_j | d_j, \alpha_i, y_j = 1) = \frac{1 + (1 - d_j)^{(1-\alpha_i)}}{2} \tag{3}$$

$$Pr(l_{ij} = y_j | d_j, \beta_i, y_j = 0) = \frac{1 + (1 - d_j)^{(1-\beta_i)}}{2} \tag{4}$$

In the rest of the paper, we use pools of synthetic users with different recalls and specificities following equations 3 and 4. Though experiments with synthetic workers does not replace real field experiments on real platforms, this allowed us to test many variants of the dynamic worker allocation scheme proposed hereafter. The functions of equations 3 and 4 obviously share characteristics of faithful workers: the probability of a correct Boolean answer decreases with the difficulty of tasks and with recall (resp. with specificity), and remains 1/2 even for completely incompetent workers, who cannot do better than a random guess. Figure 1 shows the probability of associating label 1 to a task for which the ground truth is 1 when the difficulty of the tagging task varies, and for different values of recall. The range of task difficulty is $[0, 1]$. The vertical axis is the probability of getting $l_{ij} = 1$. One can notice that this probability takes values between 0.5 and 1. Indeed, if a task is too difficult, then returning a value is close to making a random guess of a binary value. Unsurprisingly, as the difficulty of a task increases, the probability of correctly labeling it decreases.

This generative function applies for every worker, but with individual values for recalls and specificities. For a fixed difficulty of task, workers with higher recalls have a higher probability to correctly label a task. Also, note that when the difficulty of a task approaches 1, the probability of answering with label $l_{ij} = 1$ decreases for every value of α_j. However, for workers with high recall, the probability of a correct annotation is always greater than with a smaller recall. Hence, the probability of correct answer depends both on the difficulty of task and on expertise of the worker realizing the task.

3.1 Aggregating Answers

For a given task j, with unknown difficulty d_j, the answers returned by k workers (observed data) is a set $L_j = \{l_{1j}, \dots, l_{kj}\}$, where l_{ij} is the answer of worker i to task j. In addition, workers expertise are vectors of parameters $\alpha = \{\alpha_1, \dots \alpha_k\}$ and $\beta = \{\beta_1, \dots \beta_k\}$ and are also unknown. The goal is to infer the final label y_j, and to derive the most probable values for d_j, α_i, β_i, given the observed answers of workers. We use a standard EM approach to infer the most probable actual answer $Y = \{y_1, \dots y_n\}$ along with the hidden parameters $\Theta = \{d_j, \alpha_i, \beta_i\}$. Let us consider the E and M phases of the algorithm.

E Step: We assume that all answers in $L = \bigcup_{1 \le j \le k} L_j$ are independently given by the workers as there is no collaboration between them. So, in every $L_j = \{l_{1j}, \dots, l_{kj}\}$, l_{ij}'s are independently sampled variables. We compute the posterior probability of $y_j \in \{0, 1\}$ for a given task j given the difficulty of task d_j, worker expertise $\alpha_i, \beta_i, i \le k$ and the worker answers $L_j = \{l_{ij} \mid i \in 1..k\}$. Using Bayes' theorem, considering a particular value $\lambda \in \{0, 1\}$ we have:

$$Pr[y_j = \lambda | L_j, \alpha, \beta, d_j] = \frac{Pr(L_j | y_j = \lambda, \alpha, \beta, d_j) \cdot Pr(y_j = \lambda | \alpha, \beta, d_j)}{Pr(L_j | \alpha, \beta, d_j)} \quad (5)$$

One can remark that y_j and α, β, d_j are independent variables. We assume that both values of y_j are equiprobable, i.e. $Pr(y_j = 0) = Pr(y_j = 1) = \frac{1}{2}$. We hence get:

$$Pr[y_j = \lambda | L_j, \alpha, \beta, d_j] = \frac{Pr(L_j | y_j = \lambda, \alpha, \beta, d_j) \cdot Pr(y_j = \lambda)}{Pr(L_j | \alpha, \beta, d_j)} = \frac{Pr(L_j | y_j = \lambda, \alpha, \beta, d_j) \cdot \frac{1}{2}}{Pr(L_j | \alpha, \beta, d_j)} \quad (6)$$

Similarly, the probability to obtain a particular set of labels is given by:

$$Pr(L_j \mid \alpha, \beta, d_j) = \frac{1}{2} \cdot Pr(L_j \mid y_j = 0, \alpha, \beta, d_j) + \frac{1}{2} \cdot Pr(L_j \mid y_j = 1, \alpha, \beta, d_j) \quad (7)$$

Overall we obtain:

$$Pr[y_j = \lambda | L_j, \alpha, \beta, d_j] = \frac{Pr(L_j | y_j = \lambda, \alpha, \beta, d_j)}{Pr(L_j | y_j = 0, \alpha, \beta, d_j) + Pr(L_j | y_j = 1 \alpha, \beta, d_j)} \quad (8)$$

Let us consider one of these terms, and let us assume that every l_{ij} in L_j takes a value λ_i. We have

$$Pr(L_j \mid y_j = \lambda, \alpha, \beta, d_j) = \prod_{i=1}^{k} Pr(l_{ij} = \lambda_i \mid \alpha_i, \beta_i, d_j, y_j = \lambda) \quad (9)$$

If $\lambda_i = 0$ then $Pr(l_{ij} = \lambda_i \mid \alpha_i, \beta_i, d_j, y_j = 0)$ is the probability to classify correctly a 0 as 0, as defined in Equation 4 denoted by $\delta_{ij} = \frac{1 + (1 - d_j)^{(1 - \beta_i)}}{2}$.

Similarly, if $\lambda_i = 1$ then $Pr(l_{ij} = \lambda_i \mid \alpha_i, \beta_i, d_j, y_j = 1)$ is the probability to classify correctly a 1 as 1, expressed in Equation 3 and denoted by $\gamma_{ij} = \frac{1+(1-d_j)^{(1-\alpha_i)}}{2}$. Then the probability to classify $y_j = 1$ as $\lambda_i = 0$ is $(1 - \gamma_{ij})$ and the probability to classify $y_j = 1$ as $\lambda_i = 0$ is $(1 - \delta_{ij})$. We hence have $Pr(l_{ij} = \lambda_i \mid \alpha_i, \beta_i, d_j, y_j = 0) = (1 - \lambda_i) \cdot \delta_{ij} + \lambda_i \cdot (1 - \gamma_{ij})$. Similarly, we can write $Pr(l_{ij} = \lambda_i \mid \alpha_i, \beta_i, d_j, y_j = 1) = \lambda_i \cdot \gamma_{ij} + (1 - \lambda_i) \cdot (1 - \delta_{ij})$. So Equation 8 rewrites as :

$$\begin{aligned} Pr[y_j = \lambda | L_j, \alpha, \beta, d_j] &= \frac{\prod_{i=1}^{k} Pr(l_{ij} = \lambda_i \mid y_j = \lambda, \alpha_i, \beta_i, d_j)}{Pr(L_j \mid y_j = 0, \alpha, \beta, d_j) + Pr(L_j \mid y_j = 1, \alpha, \beta, d_j)} \\ &= \frac{\prod_{i=1}^{k}(1-\lambda).[(1-\lambda_i)\delta_{ij} + \lambda_i(1-\gamma_{ij})] + \lambda.[\lambda_i.\gamma_{ij} + (1-\lambda_i)(1-\delta_{ij})]}{Pr(L_j \mid y_j = 0, \alpha, \beta, d_j) + Pr(L_j \mid y_j = 1, \alpha, \beta, d_j)} \\ &= \frac{\prod_{i=1}^{k}(1-\lambda).[(1-\lambda_p)\delta_{ij} + \lambda_p(1-\gamma_{ij})] + \lambda.[\lambda_p.\gamma_{ij} + (1-\lambda_p)(1-\delta_{ij})]}{\prod_{i=1}^{k}(1-\lambda_i)\delta_{ij} + \lambda_i \cdot (1-\gamma_{ij}) + \prod_{i=1}^{k}\lambda_i.\gamma_{ij} + (1-\lambda_i)(1-\delta_{ij})} \end{aligned} \tag{10}$$

In the E step, as every α_i, β_i, d_j is fixed, one can compute $\mathbb{E}[y_j | L_j, \alpha_i, \beta_i, d_j]$ and also choose as final value for y_j the value $\lambda \in \{0, 1\}$ such that $Pr[y_j = \lambda | L_j, \alpha_i, \beta_i, d_j] > Pr[y_j = (1 - \lambda) | L_j, \alpha_i, \beta_i, d_j]$. We can also estimate the likelihood for the values of variables $P(L \cup Y \mid \theta)$ for parameters $\theta = \{\alpha, \beta, d\}$, as $Pr(y_j = \lambda, L \mid \theta) = Pr(y_j = \lambda, L).Pr(L_j \mid y_j = \lambda, \theta) = Pr(y_j = \lambda).Pr(L_j \mid y_j = \lambda, \theta)$

M Step: With respect to the estimated posterior probabilities of Y computed during the E phase of the algorithm, we compute the parameters θ that maximize $Q(\theta, \theta^t)$. Let θ^t be the value of parameters computed at step t of the algorithm. We use the observed values of L, and the previous expectation for Y. We maximize $Q'(\theta, \theta^t) = \mathbb{E}[logPr(L, Y \mid \theta) \mid L, \theta^t]$ (we refer interested readers to [6]-Chap. 9 and [5] for explanations showing why this is equivalent to maximizing $Q(\theta, \theta^t)$). We can hence compute the next value as: $\theta^{t+1} = \arg\max_{\theta} Q'(\theta, \theta^t)$. Here in our context the values of θ are α_i, β_i, d_j. We maximize $Q'(\theta, \theta^t)$ using a bounded optimization techniques, namely the truncated Newton algorithm [14] provided by the standard SciPy[1] implementation. We iterate E and M steps, computing at each iteration t the posterior probability and the parameters θ^t that maximize $Q'(\theta, \theta^t)$. The algorithm converges, and stops when the improvement (difference between two successive joint log-likelihood values) is below a threshold, fixed in our case to $1e^{-7}$.

4 Cost Model

A drawback of many crowdsourcing approaches is that task distribution is static, i.e. tasks are distributed to a fixed number of workers, without considering their difficulty, nor checking if a consensus can be reached with fewer workers. Consider again the simple Boolean tagging setting, but where every task realization is paid, and with a fixed total budget B_0 provided by the client. For simplicity, we assume that all workers receive 1 unit of credit for each realized task. Hence, to

[1] docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html

solve n Boolean tagging tasks, one can hire only B_0/n workers per task. In this section, we show a worker allocation algorithm that builds on collected answers and estimated difficulty to distribute tasks to worker at runtime, and show its efficiency with respect to other approaches.

Our algorithm works in rounds. At each round, only a subset $T_{avl} \subseteq T$ of the initial tasks remains to be evaluated. We remember labels produced by workers at preceding rounds, and collect new labels produced by new workers hired for this round to realize these tasks. We aggregate answers using the EM approach described in Section 3. We denote by y_j^q as the final aggregated answer for task j at round q, d_j^q is the current *difficulty* of task and α_i^q, β_i^q denotes the estimated *expertise* of a worker i at round q. We let $D^q = \{d_1^q \dots d_j^q\}$ denote the set of all difficulties estimated as round q. We fix a maximal step size $\tau \geq 1$, that is the maximal number of workers that can be hired during a round for a particular task. For every task $t_j \in T_{avl}$ with difficulty d_j^q at round q, we allocate $\mathbf{a}_j^q = \lceil (d_j^q \times \tau)/\max(D^q) \rceil$ workers for the next round. Once all answers for a task have been received, the EM aggregation can compute a final label $y_j^q \in \{0,1\}$ and difficulty d_j^q for every task t_j, and the expertise of all workers $\alpha_1^q, \dots, \alpha_k^q, \beta_1^q, \dots, \beta_k^q$. Now, it remains to decide whether the confidence in answer y_j^q obtained at round q is sufficient (in which case, we do not allocate workers to this task in the next rounds). Let k_j^q be the number of answers obtained for task j at round q. The *confidence* $\hat{c}_j^q$ in a final label y_j^q is defined as follows:

$$\hat{c}_j^q(y_j^q = 1) = \frac{1}{k_j^q} \cdot \sum_{i=1}^{k_j^q} \left\{ l_{ij} \times \left(\frac{1+(1-d_j^q)^{(1-\alpha_i^q)}}{2}\right) + (1 - l_{ij}) \times \left(1 - \frac{1+(1-d_j^q)^{(1-\alpha_i^q)}}{2}\right) \right\} \quad (11)$$

$$\hat{c}_j^q(y_j^q = 0) = \frac{1}{k_j^q} \cdot \sum_{i=1}^{k_j^q} \left\{ (1 - l_{ij}) \times \left(\frac{1+(1-d_j^q)^{(1-\beta_i^q)}}{2}\right) + (l_{ij}) \times \left(1 - \frac{1+(1-d_j^q)^{(1-\beta_i^q)}}{2}\right) \right\} \quad (12)$$

Intuitively, each worker adds its probability of doing an error, which depends on the final label y_j^q estimated at round q and on his competences, i.e. on the probability to choose $l_{ij} = y_j^q$. Let us now show when to stop the rounds of our evaluation algorithm. We start with n tasks, and let T_{avl} denote the set of remaining tasks at round q. We define $r^q \in [0,1]$ as the ratio of task that are still considered at round q compared to the initial number of task, i.e. $r^q = \frac{|T_{avl}|}{n}$. We start with an initial budget B_0, and denote by B_c^q the total budget consumed at round q. We denote by $\mathcal{B}^q$ the fraction of budget consumed at that current instance, $\mathcal{B}^q = \frac{B_c^q}{B_0}$. We define the stopping threshold $Th^q \in [0.5, 1.0]$ as $Th^q = \frac{1+(1-\mathcal{B}^q)^{r^q}}{2}$.

The intuition behind this function is simple: when the number of remaining tasks decreases, one can afford a higher confidence threshold, because the maximal budget needed to solve all tasks decreases too. Similarly, as the budget decreases, one shall derive a final answer for tasks faster, possibly with a poor confidence, as the remaining budget does not allow hiring many workers. Figure 2-left shows the evolution of threshold when the ration of solved tasks and the ration of consumed budget evolves. Figure 2-right shows the evolution of threshold when the fraction of budget consumed increased, for several fixed

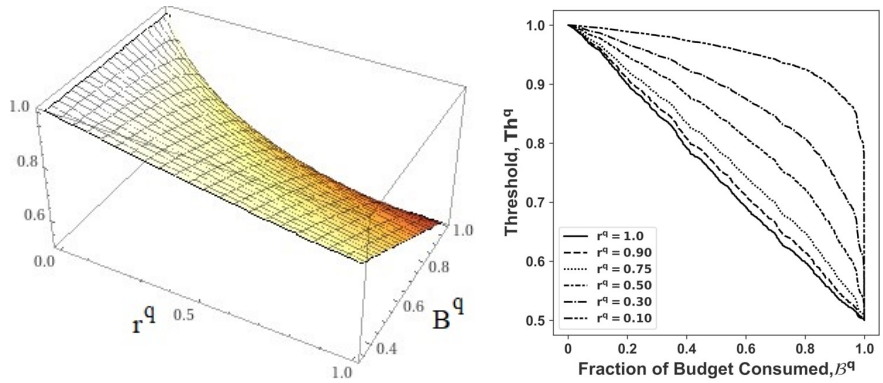

Fig. 2: Evolution of threshold for fixed fraction of consumed budget and fraction of task remaining at the beginning of a round.

values for the ratio of solved tasks r^q. Each curve in the Figure represents this evolution for a fixed value of r^q. Though function Th^q was chosen arbitrarily, one can notice that its value always lays between 0.5 and 1, decreases with available budget, and increases when the number of remaining tasks diminishes. Observe that when r^q is close to 1, the threshold falls rapidly, as a large number of tasks still has to be evaluated within the remaining budget. On the other hand, when there are less unsolved tasks (e.g. when $r^q = 0.10$), the threshold Th^q decreases slowly.

We can now define a crowdsourcing algorithm (CrowdInc) with a dynamic worker allocation strategy to optimize cost and accuracy. This strategy allocates workers depending on current confidence on final answers, and available resources. CrowdInc is decomposed in two phases, *Estimation* and *Convergence*.

Estimation: As *difficulty* of tasks is not known a priori, the first challenge is to estimate it. To get an initial measure of difficulties, each task needs to be answered by a set of workers. Now, as each worker receives an incentive for a task, this preliminary evaluation has a cost, and finding an optimal number of workers for *difficulty* estimation is a fundamental issue. The initial budget gives some flexibility in the choice of an appropriate number of workers for preliminary evaluation of difficulty. Choosing a random number of workers per task does not seem a wise choice. We choose to devote a fraction of the initial budget to this estimation phase. We devote one third of the total budget ($B_0/3$) to the estimation phase. It leaves a sufficient budget ($2{\cdot}B_0/3$) for the convergence phase. Experiments in the next section show that this seems a sensible choice. After collection of answers for each task, we apply the EM based aggregation technique of Section 3 to estimate the *difficulty* of each task as well as the *expertise* of each worker. Considering this as an initial round $q = 0$, we let d_j^0 denote the initially estimated difficulty of each task j, α_i^0, β_i^0 denote the expertise of each worker and y_j^0 denote the aggregated answer for task t_j after the estimation phase. Note that if the difficulty of some tasks is available a priori and is provided by the client, we may skip the estimation step. However, in general clients do not possess such

Algorithm 2: CrowdInc

Data: A set of tasks $T = \{t_1, \ldots, t_n\}$, a budget $= B_0$
Result: Final Answer: $Y = y_1, \ldots, y_n$, Difficulty: d_j,Expertise: α_i, β_i
Initialization : Set every d_j, α_i, β_i to a random value in $[0, 1]$.
$T_{avl} = T$; $q = 0$; $B = B - (B_0/3)$; $B_c = B_0/3$; $r = (B_0/3)/n$
`//Initial Estimation:`
Allocate r workers to each task in T_{avl} and get their answers
Estimate $d_j^q, \alpha_i^q, \beta_i^q, \hat{c}_j^q, 1 \leq j \leq n, 1 \leq i \leq B_0/3$ using EM aggregation
Compute the stopping threshold Th^q.
for $j = 1, \ldots, n$ **do**
  if $\hat{c}_j^q > Th^q$ **then** $T_{avl} = T \setminus \{j\}$;
end
`//Convergence:`
while $(B > 0)$ && $(T_{avl} \neq \emptyset)$ **do**
  $q = q + 1$; $l = |T_{avl}|$
  Allocate $\mathbf{a}_1^q, \ldots, \mathbf{a}_l^q$ workers to tasks $t_1, \ldots t_l$ based on difficulty.
  Get the corresponding answers by all the newly allocated workers.
  Estimate $d_j^q, \alpha_i^q, \beta_i^q, \hat{c}_j^q$ using aggregation model.
  $B = B - \sum_{i \in 1..|T_{avl}|} \mathbf{a}_i^q$
  Compute the stopping threshold Th^q
  for $j = 1, \ldots, n$ **do**
    if $\hat{c}_j^q > Th^q$ **then** $T_{avl} = T_{avl} \setminus \{j\}$;
  end
end

information and this initial step is crucial in estimation of parameters. After this initial estimation, one can already compute Th^0 and decide to stop evaluation of tasks with a sufficient confidence level.

Convergence: The *difficulty* of task d_j^q and the set of remaining tasks T_{avl} are used at each iteration of the convergence phase. Now as the difficulty of each task is estimated, we can use the estimated difficulty d_j^q to allocate the workers dynamically. The number of workers allocated at round $q > 0$ follows a difficulty aware *worker allocation* policy. At each round, we allocate $\mathbf{a}_j^q$ workers to remaining task t_j. This allocation policy guarantees that each remaining task is allocated at least one worker, at most τ workers, and that the more difficult tasks (i.e. tasks that have the more disagreement) are allocated more workers than easier tasks.

Algorithm 2 gives a full description of CrowdInc. We also show the information memorized at each step of the algorithm in Figure 3. Consider a set of n tasks that have to be annotated with a Boolean tag in $\{0, 1\}$. CrowdInc starts with the *Estimation* phase and allocates k workers for an initial evaluation round ($q = 0$). After collection of answers, and then at each round $q > 0$, we first apply EM based aggregation to estimate the difficulty d_j^q of each of task $t_j \in T_{avl}$,

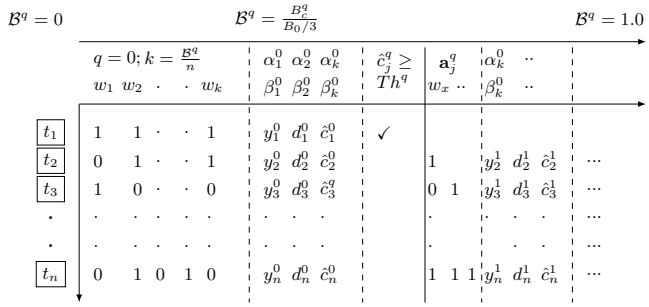

Fig. 3: A possible state for Algorithm 2

the confidence $\hat{c}_j^q$ in final aggregated answer y_j^q, and the expertise α_i^q, β_i^q of the workers. Then, we use the stopping threshold to decide whether we need more answers for each task. If $\hat{c}_j^q$ is greater than Th^q, the task t_j is removed from T_{avl}. This stopping criterion hence takes a decision based on the confidence in the final answers for a task and on the remaining budget. Consider, in the example of Figure 3 that the aggregated answer for task t_1 has high confidence, and that $\hat{c}_1^q \geq Th^q$. Then, t_1 does not need further evaluation, and is removed from T_{avl}. Once all tasks solved at round q have been removed, we allocate $\mathbf{a}_j^q$ workers to each remaining task t_j in T_{avl} following our difficulty aware policy. Note that, each task gets a different number of workers based on task difficulty. The algorithm stops when either the whole budget B_0 is exhausted or there is no unsolved task left. It then returns the set of all aggregated answers $Y = \{y_j^q \mid 1 \leq j \leq n\}$.

Let us stress an important point of the algorithm. The initial estimation phase (lines 1 to 9) is a standard static allocation, but uses only a limited fraction of the allowed budget. Dynamic allocation depending on confidence in aggregated results only starts from line 10. This initialization phase guarantees that all records are evaluated by at least $r = (B_0/3)/n$ workers, and hence that the algorithm terminates with an answer for each record. Termination of the loop on lines 11 to 21 is guaranteed because the available budget decreases at each round. Consistently, when the whole budget is consumed, the threshold computed at the end of a round is 1/2. As every aggregated answer achieves a confidence score higher that random guesses, threshold also helps termination. The algorithm is hence guaranteed to terminate in $O((B_0/3)/n + (2.B_0/3))$ rounds. Hence, each record receives between r and $2 \cdot B_0/3 + r$ answers, depending on the difficulty of tagging this record, but also on the frequency of difficult tasks. Experiments were carried out with different values for the initial number r of workers allocated to a task, but gave less interesting results in terms of cost or in terms of accuracy. Setting $r = (B_0/3)/n$ appears as a good tradeoff, but additional experiments should be carried out to study optimal values for r. However, as very often in crowdsourcing, the optimal value for r might depend on the input dataset, and on the characteristics of workers which are not known a priori.

5 Experiments

We evaluate the algorithm on three public available datasets, namely the Product Identification [21], Duck Identification [23] and Sentiment Analysis [15] benchmarks. We briefly detail each dataset and the corresponding tagging tasks. All tags appearing in the benchmarks were collected via Amazon Mechanical Turk. In the **Product Identification** use case, workers were asked to decide whether a *product-name* and a *description* refer to the same product. The answer returned is *True* or *False*. There are 8315 samples and each of them was evaluated by 3 workers. The total number of unique workers is 176 and the total number of answers available is 24945. In the **Duck Identification** use case, workers had to decide if sample images contain a duck. The total number of tasks is 108 and each task was allocated to 39 workers. The total number of unique workers is 39 and the total number of answers is 4212. In the **Sentiment Analysis** use case, workers had to annotate movie reviews as Positive or Negative opinions. The total number of tasks was 500. Each task was given to 20 unique workers and a total number of 143 workers were involved, resulting in a total number of 10000 answers. All these information are synthesized in Table 1.

Dataset	Number of Tasks	Number of tasks with ground truth	Total Number of answers provided by the crowd	Average number of answers for each task	Number of unique crowd workers
Product Identification	8315	8315	24945	3	176
Duck Identification	108	108	4212	39	39
Sentiment Analysis	500	500	10000	20	143

Table 1: Datasets description.

Evaluation of aggregation: We first compared our aggregation technique to several methods: MV, D&S [3], GLAD [24], PMCRH [12], LFC [17], and ZenCrowd [4]. We ran the experiment 30 times with different initial values for tasks difficulty and workers expertise. The standard deviation over all the iteration was less than 0.05%. Hence our aggregation is insensitive to initial prior values. We now compare *Recall*, *Specificity* and *Balanced Accuracy* of all methods. The results are shown in Table 2. The recall and specificity measures presented in the table characterize the success rate of algorithms on tasks with ground truth 1 and 0 respectively. Balanced Accuracy is the average of recall and specificity (we choose this average to get unbiased estimates on unbalanced dataset). We can observe in Table 2 that our method outperforms other techniques in Duck Identification, Product Identification, and is comparable for Sentiment Analysis.
Evaluation of CrowdInc: The goal of the next experiment was to verify that the cost model proposed in CrowdInc achieves at least the same accuracy but with a smaller budget. We have used Duck identification and Sentiment Analysis

Methods	Recall	Specificity	Balanced Accuracy
MV	0.56	0.91	0.73
D&S [3]	0.81	0.93	0.87
GLAD [24]	0.47	0.98	0.73
PMCRH [12]	0.58	0.95	0.76
LFC [17]	0.87	0.91	0.89
ZenCrowd [4]	0.39	0.98	0.68
EM + recall, specificity & difficulty	**0.89**	**0.91**	**0.90**

(a) Duck Identification

Methods	Recall	Specificity	Balanced Accuracy
MV	0.61	0.93	0.77
D&S [3]	0.65	0.97	0.81
GLAD [24]	0.48	0.98	0.73
PMCRH [12]	0.61	0.93	0.77
LFC [17]	0.64	0.97	0.81
ZenCrowd [4]	0.51	0.98	0.75
EM + recall, specificity & difficulty	**0.77**	**0.90**	**0.83**

(b) Product Identification

Methods	Recall	Specificity	Balanced Accuracy
MV	0.93	0.94	0.4
D&S [3]	0.94	0.94	0.94
GLAD [24]	0.94	0.94	0.94
PMCRH [12]	0.93	0.95	0.94
LFC [17]	0.94	0.94	0.94
ZenCrowd [4]	0.94	0.94	0.94
EM + recall, specificity & difficulty	**0.94**	**0.95**	**0.94**

(c) Sentiment Analysis

Table 2: Comparison of EM + aggregation (with Recall, specificity & task difficulty) with MV, D&S, GLAD, PMCRH, LFC, ZenCrowd.

for this test. We did not consider the Product Identification benchmark: indeed, as shown in Table 1, the Product Identification associates only 3 answers to each task. This does not allow for a significant experiment with CrowdInc. We compared the performance (cost and accuracy) of CrowdInc to other approaches. The results are given in Figure 4. Static(MV) denotes the traditional static allocation used in crowdsourcing platforms with majority voting as aggregation technique and Static(EM) denotes a static allocation combined with a more advanced EM based aggregation technique. Both algorithms allocate all the workers (and hence use all their budget) at the beginning of the crowdsourcing process. Considering these two algorithms allows to highlight the impact of EM and of dynamic allocation on cost and accuracy.

The following observation can be made from Figure 4. First, CrowdInc achieves better accuracy than a static(MV) approach. This is not a real surprise, as MV already showed bad accuracy in Table 2. Then, CrowdInc achieves almost the same accuracy as a Static(EM) based approach in Duck identification, and the same accuracy in Sentiment Analysis. Last, CrowdInc uses a smaller budget than static approaches in all cases.

Table 3 shows the time (in seconds) needed by each algorithm to aggregate answers. Static(MV) is the fastest solution: it is not surprising, as the complexity is linear in the number of answers. We recall however that MV has the worst accuracy of all tested aggregation techniques. We have tested aggregation with EM when the number of workers is fixed a priori and is the same for all tasks (Static(EM)). CrowdInc uses EM, but on a dynamic sets of workers and tasks, stopping easiest tasks first. This results in a longer calculus, as EM is used several times on sets of answers of growing sizes. The accuracy of *static(EM)* and

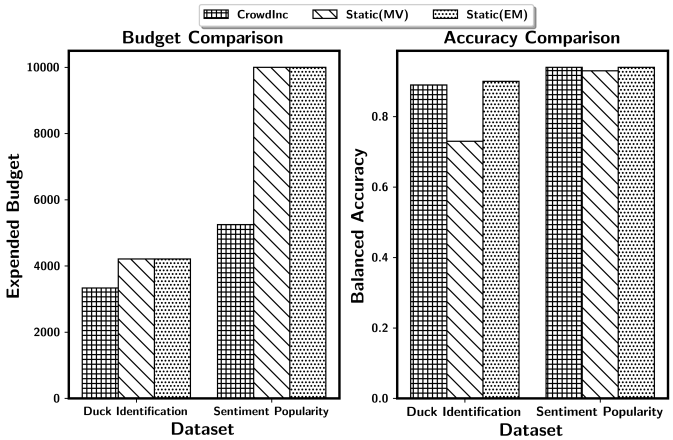

Fig. 4: Comparison of cost vs. Accuracy.

CrowdInc are almost the same. Aggregation with CrowdInc takes approximately 11% longer than *static(EM)* but for a smaller budget, as shown in the Figure 4. To summarize the CrowdInc aggregation needs more time and a smaller budget to aggregate answers with a comparable accuracy. In general, clients using crowdsourcing services can wait several days to see their task completed. Hence, when time is not a major concern. CrowdInc is hence a sensible solution to reduce the cost of crowdsourcing.

Dataset/Methods	**CrowdInc**	**Static(EM)**	**Static(MV)**
Duck Identification	843.26	106.81	0.073
Sentiment Analysis	1323.35	137.79	0.102

Table 3: Running time(in seconds) of CrowdInc, static MV and Static EM.

6 Spammer detection

In the preceding sections, we have considered faithful workers, i.e., workers that do their best to return an answer that is, up to their knowledge, the right answer. Some workers may have low competences, but our experimentation showed that despite errors, Crowdink achieves good accuracy. The reason is that workers are usually competent, and that a limited number of errors per question can be compensated by correct answers. Indeed, in a context where workers are faithful, the largest probability for an individual wrong answer is 0.5, as for incompetent workers, answers are almost a random guess. Now, the probability for k wrong answers for a task, and the probability of k consecutive wrong answers by the same worker are very small (0.5^k). Hence, the high probability that individual errors are corrected by other answers allows to achieve good recall and specificity. This setting is completely changed if a worker returns wrong answer with

a higher probability, either because he is only interested in incentives, and does not really perform the task he was hired for, but rather returns fast thoughtless answers, or because he is trying to influence the results or accuracy of the crowdsourcing platform. In the rest of this section we consider these two types of spammers, and study the impact of growing proportions of these malevolent workers on the overall accuracy of our algorithm. One can expect crowdsourcing to be robust to a single spammer, but a major danger for a platform is to hire too many spammers. An important parameter to know is hence the maximal percentage of spammers that a platform can accept. A second important parameter is how malevolent workers affect costs of our aggregation algorithm. As shown in Section 5, our algorithm allows to save costs when confidence in aggregated answers is sufficient. Now, as malevolence affects answers, it can also reduce confidence in aggregated answers, and subsequently increase the budget spent to reach a consensus. In the rest of this section, we propose a model for several types of spammers, and present the results of experiments showing the maximal percentage of spammers that CrowdInc can accept, and the impact of these spammers on costs.

6.1 Spammers models

We distinguish several types of spammers, with different motivations and hence different behaviors. For some of them, the objective is to earn fast money by performing a maximal number of tasks within the shortest possible time. For others, the objective is to perturb the system, and reduce its overall efficiency and quality. Last, some spammers want to influence the results returned by the crowdsourcing platform. We distinguish these three types of spammer, and for each type define a particular generating function (i.e., a probability law) for returned answers. Our spammer are represented as follows:

- **Type 1 spammers**: The objective of these spammers is to earn money easily through obfuscated use of crowdsourcing platforms. They do not want to spend time thinking on problems posted on the crowdsourcing platform. They favor easy tasks with a small and finite number of answers and answer as fast as possible to gain the incentives given for task completion. These greedy spammers can be seen as returning a random answer. To fight this type of spam, platforms separate rewards in two parts: the first one for accepting to realize a task, and the second for a correct answer (or at least an answer that conforms with the aggregated result obtained from the set of answers of all workers contributing to the task). Another way to avoid greedy spammers is to select workers only after unpaid qualification tests before allowing them to contribute to paid tasks. Greedy spammers are often reluctant to pass these tests that result in a loss of time, and systematically disqualify them if they answer randomly. The answering profile of a greedy spammer is a profile where the probability to answer x when the ground truth is x is a constant 0.5, regardless of the difficulty of the task. Using

the same parameters as for standard workers proposed in Section 3, the generative functions for Type 1 spammers are:

$$Pr(l_{ij} = y_j | d_j, \alpha_i, y_j = 1) = 0.5 \text{ and } Pr(l_{ij} = y_j | d_j, \beta_i, y_j = 0) = 0.5 \quad (13)$$

- **Type 2 spammers**: the objective of spammers of this type is to impact efficiency of crowdsourcing platforms. If this type of spammer become majoritary in pool of worker, then the accuracy of the platform can be severely compromised. Though we are not aware of denial of service attacks of this form, assuming existence of this type of spammers and considering various proportions of Type 2 spammers is a way to study the *maximal impact* that spammers can have on a system. The generative functions for Type 2 spammers are:

$$Pr(l_{ij} = 0 | d_j, \alpha_i, y_j = 1) = 1 \text{ and } Pr(l_{ij} = 1 | d_j, \beta_i, y_j = 0) = 1 \quad (14)$$

  Note that the probabilities of incorrect answers from Type 2 spammers do not depend on their recall or specificity. We hence use this type of spammer as a worst case measure of spammers impact, as answering incorrectly a question supposes the ability to know the correct answer. In some sense, this is a high (but evil) competence level that should be very rare.
- **Type 3 spammers**: the objective of spammers of this type is to force the results returned by a crowdsourcing platform. In [22], attacks of these spammers are called *sybil attacks*. Regardless of the ground truth, these spammers return the same answer. This type of attack is not a purely theoretical view: It was demonstrated that robots had rigged the results of a famous talent show [2] in 2019. The behavior of a Type 3 spammer is to return systematically the answer he wants to see in the aggregated result. For instance, if worker i is a spammer willing to favor answer 1 to question j, his behavior will be defined by the following generative functions.

$$Pr(l_{ij} = 1 | d_j, \alpha_i, y_j = 1) = 1 \text{ and } Pr(l_{ij} = 1 | d_j, \beta_i, y_j = 0) = 1 \quad (15)$$

  Of course, one can write symmetric generative functions when the preferred answer of spammer i is 0. Depending on the chosen answer, a spammer will have a good recall and poor specificity, or the converse. We will call Type 3.1 spammers the malevolent workers that want to force the system to return final answer 0, and Type 3.2 spammers the malevolent workers that want to force the system to return final answer 1.

We illustrate the generative function for different types of spammers of Figure 5. We compare the probability of a correct answer when ground truth is 1 (diagram on the left) for an average user which recall is $\alpha = 0.75$. For a genuine

[2] `https://en.wikipedia.org/wiki/The_Voice_Kids_(Russian_season_6)`

worker of this kind, the probability for a correct answer is represented with a mixed line, and decreases to 0.5 while difficulty increases. On the other hand, the probability of a correct answer for Type 1 spammers is always 0.5. One can see on the figure that the difference between Type 1 spammer and genuine user decreases while difficulty increases. Type 2 spammers (intentional wrong answer, represented by yellow line) and Type 3.1 spammers (constant answer 0 represented by a black dotted line) both have a probability 0 to give the correct answer. The diagram on the right represents similar curves when the ground truth is 0 and the specificity of a genuine user is $\beta = 0.75$.

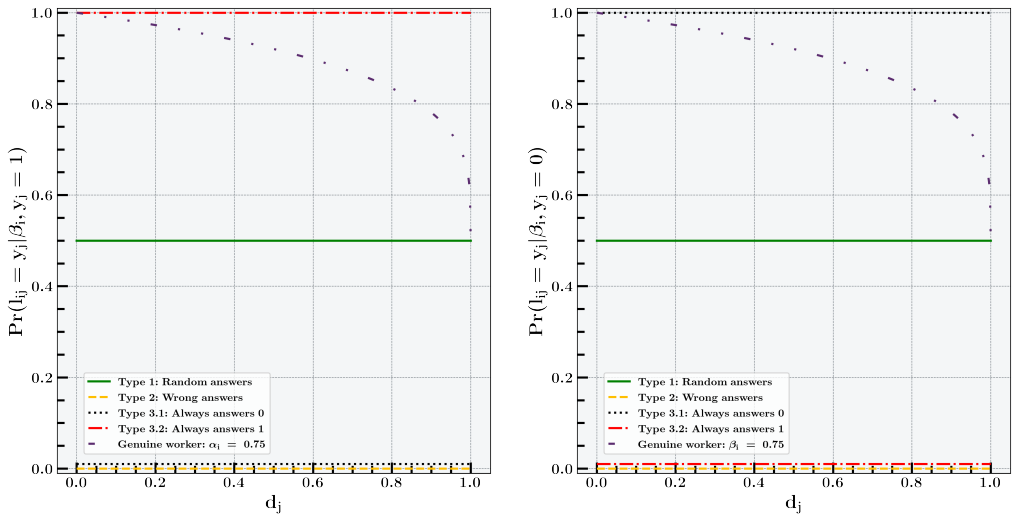

Fig. 5: Generating functions depicting the probability of correct answers for honest workers (with recall $\alpha = 0.75$ and specificity $\beta = 0.75$) and Type 1, Type 2, and Type 3 spammers.

One can notice that the probabilities of correct answers are very different for genuine workers and for spammers. However, detection of a spammer is not as straightforward as Figure 5 suggests, as ground truth is not known, and correct answers, recalls and specificities are estimated from observed answers.

Though the objective of this section is to study the impact of spammers on efficiency of CrowdInc, one can easily build on the experiments showed later in this section to imagine a simple detection algorithm. Fixing a lower bound $th_R^{l,i}$ and an upper bound $th_R^{u,i}$ for recall (resp. $th_S^{l,i}$ and an upper bound $th_S^{u,i}$ for specificity) for each spammer type i, and claim that a worker w_j is a spammer of type i if $th_R^{l,i} \leq \alpha_j \leq th_R^{u,i}$ and $th_S^{l,i} \leq \beta_j \leq th_S^{u,i}$. Though this algorithm is simplistic, it achieved good detection scores in the experiment showed below.

6.2 Experimentation of CrowdInc with spammers

The objective of the experiment was to measure the effect of spammers on the performance of our aggregation algorithm, study the impact of spammers on the cost and accuracy and give ways to detect spammers in a pool of workers.

Datasets. We consider the datasets used in Section 5, namely *Duck Identification* and *Sentiment Analysis*, and analyze the effect of spammers on CrowdInc's performance. We do not consider the Product Identification dataset as it associates only three unique workers per task. This does not allow conclusive experiments, as replacing a worker by a spammer immediately leads to a situation with 33% of spammers.

The experimentation was organized as follows: we randomly generated spammers with their own characteristics (type), and replaced genuine workers with spammers. This allowed us to compare the results achieved with an original dataset with a biased result of identical size. For Type 1 spammers, we followed an uniform distribution to generate spammers answers for each tasks. For Type 2 spammers, we replaced the former Boolean answer with the negation of the ground truth. For Type 3.1 spammers, we set the answer to tasks to 0 for each spammer and conversely for Type 3.2 spammers, we set the answer to tasks to 1 for each spammer.

We have considered two types of spammer sets in our experiment: sets composed of spammers of a *S type* and sets composed of spammers of *Mixed types*. We have then analyzed the impact of spam with growing percentage of corrupted workers of a single type, and with growing percentage of spammers of mixed types. The mixed type spammer sets included all types of spammer and for simplicity, we kept an equal proportion of each type. We have performed experiments for each set of spammers, replacing a growing proportion of genuine workers by corrupted workers. We let the proportion of spammers range from 10% to 60% to study the effect of spam on the performance of aggregation. Our experiment did not exceed 60% of spammers, becasue the performance of Crowdinc was already too low with this proportion of spam. Overall, the two datasets and the various composition of genuine workers and spammers sets results in 57 experimentation environments, shown in Table 4. Recall that answers of genuine workers are random values following a probability law, and are sampled according to a profile that depends on workers recall and specificity. This means that the influence of spammers depend on correctness of the answers they returned. We hence ran each experiment 30 times to avoid bias.

6.3 Spammer detection with thresholds

One way to detect spammer is to show that they have a poor expertize, than can only be justified by a malevolent behavior. As explained before (see in particular Figure 5), each spammer type has characteristics that can be observed from the values of recall and specificity. We can hence set thresholds for recall and specificity, and decide by comparing the actual expertize of a worker whether

Dataset Name	Duck Identification												Sentiment Identification								
Genuine Workers	39												20								
Spammer Type	Individual (Type 1, Type 2, Type 3.1, Type 3.2)						Mixed (Equal number of all type of spammers)						Individual (Type 1, Type 2, Type 3.1, Type 3.2)						Mixed (Equal number of all type of spammers)		
Spam Workers (% compared to genuine workers (approx))	4 (10)	8 (20)	12 (30)	16 (40)	20 (50)	24 (60)	4 (10)	8 (20)	12 (30)	16 (40)	20 (50)	24 (60)	2 (10)	4 (20)	6 (30)	8 (40)	10 (50)	12 (60)	4 (20)	8 (40)	12 (60)

Table 4: Spam dataset parameters.

he is a spammer or not, and even determine the type of spammer that was discovered this way. Now, the difficulty is to set the appropriate values for these thresholds.

We perform a grid search on the values of recall and specificity to find thresholds allowing identification of spammers. We use the Duck identification dataset with 10% of spammer and consider each type of spammer independently. We show the results in Figure 6. Both recall and specificity range from 0.0 to 1.0. For each plot, the recall value ranges from 0.0 to 1.0 and is represented on the horizontal axis. Likewise, specificity is represented on the y-axis and ranges from 0 to 1. We choose a step size of 0.1 for recall and specificity. Each cell hence represent an interval of values for recall and specificity. The obtained grid is hence a 10×10 grid. Now for each cell, we compute a *detection score*.

Let n_s be the total number of spammers, n_g be the number of genuine workers, n_s^d the number of spammers detected as spammers and n_g^d the number of genuine workers classified as genuine by some spammer detection technique. Then the detection score is

$$\frac{1}{2} \cdot \left(\frac{n_g^d}{n_g} + \frac{n_s^d}{n_s} \right)$$

The maximal detection scores reached for the most accurate thresholds are rather high (above 90%), but do not achieve a 100% correct classification of workers. The graphics of Figure 6 shows the achieved detection score with a color. For Type 1 spammers, we achieve the best detection score when the recall and specificity values ranges between 0.45 and 0.60. As Type 1 spammers make a random guess to return their answers, their recall and specificity lie between 0.45 and 0.60. This is below the values achieved by the complete pool of genuine honest workers (0.89 and 0.90) in the experiment of Section 5. The Type 2 spammer always gives wrong answers. This results in a low value of recall and specificity. We can observe that the detection score to correctly detect Type 2 spammers is high with a threshold for recall and specificity set to 0.2 (or less). The Type 3.1 spammers always return answer 0. In this case, the specificity value is very high, however, the recall value is very low. We can observe that, with a very high specificity threshold > 0.9 and a small recall threshold < 0.2, we get the highest detection score for Type 3.1 spammers. Conversely, the Type 3.2 spammers always answer 1 that leads to very high recall and low specificity. The score to detect Type 3.2 spammers is highest when we have a threshold for recall set to > 0.9 and a threshold for specificity < 0.2.

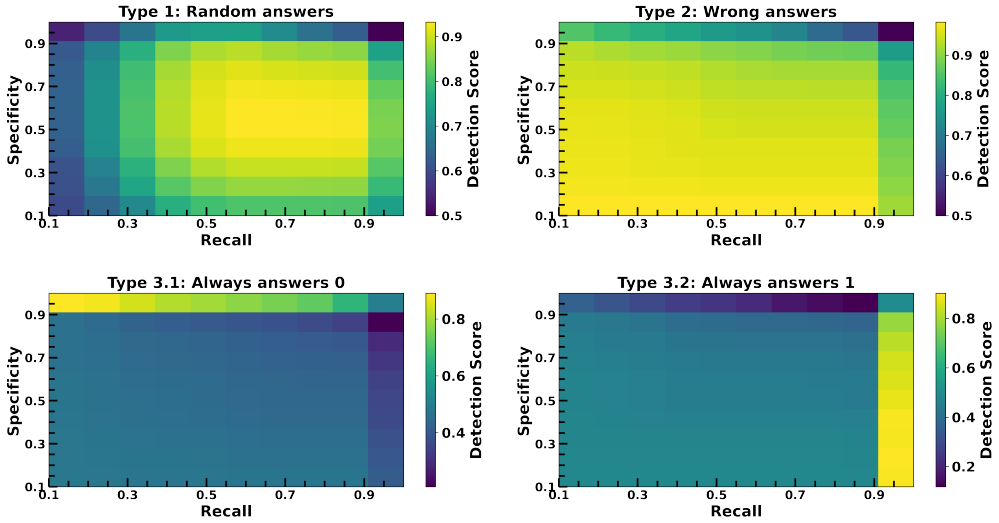

Fig. 6: Threshold search to detect spammers for four spammers type: Type 1, Type 2, Type 3.1 and 3.2.

Overall, the experiments conducted allowed to find appropriate thresholds for detection of all spammers. The optimal synthesized values of thresholds allowing to detect spammers of each category are represented in Table 5.

Recall Value Threshold	Specificity Value Threshold	Spammer Type	Description
Low ($0.45 \leq \alpha \leq 0.60$)	Low ($0.45 \leq \beta \leq 0.60$)	Type 1	Random answers
Very low ($\alpha \leq 0.2$)	Very low ($\beta \leq 0.2$)	Type 2	Wrong answers
Very low ($\alpha \leq 0.2$)	High ($0.9 \leq \beta$)	Type 3.1	Always answer 0
High ($0.9 \leq \alpha$)	Very low ($\beta \leq 0.2$)	Type 3.2	Always answers 1

Table 5: Thresholds to detect spammers of Type 1, Type 2, Type 3.1 and 3.2.

6.4 Effect of spammers on accuracy

We now analyze the effect of spammers on accuracy of aggregation. We consider spammer sets composed of spammers of a single type and spammer sets with mixed types. In each experiment, we increase the percentage of spammers from 10% to 60% and use the CrowdInc algorithm proposed in Section 4 to aggregate the answers. We do not consider cases with more than 60% of spammers because accuracy is already very low with this percentage of malevolent workers. Let us recall that the achieved recall, specificity and global accuracy achieved by CrowdInc are respectively 0.89, 0.91 and 0.9 for Duck Identification, and 0.94, 0.95 and 0.94 for Sentiment Analysis. We observe the following outcomes.

Unsurprisingly, introducing spammers degrades the overall accuracy of aggregation. This was expected as the spammers, unlike genuine workers whose answer

are based on their belief, try to trick the system. Additionally, as the number of spammer increases accuracy of aggregation decreases. This result follows intuition: as more spammers try to trick the system, confidence in aggregated answers decreases. This tendency was visible in all simulation environments. The accuracies, recalls and specificities achieved with 10% spammers are shown in Figures 7, 8, 9, 10, 11, 12, 13, and 14.

Let us discuss more precisely the obtained results for each spammer type. For Type 1 spammers, accuracy falls steadily as shown in Figure 7 and 10. The Type 1 spammers give random answers, and in this case still have a sufficient probability to return a correct answer. Such a type of spammer affects the overall accuracy of the system very gradually, as their disagreement is easily corrected by genuine workers. All graphics (Figures 7,8,9,10,11,12) show that one can still achieve an acceptable accuracy, recall and specificity with up to 60% of workers returning random answers.

On the other hand, for Type 2 spammers, accuracy decreases very quickly. Observe the results in Figure 7. With only 20% of Type 2 spammers, accuracy is already very low, and with 30% of spammers, the performance of the system is close to 0. The reason is that Type 2 spammers always return a wrong answer, and hence influence the final result. The recall and specificity for Duck Identification Figure 8 and 9 show the same trend: a proportion of 30% of spammers make the system unusable. For Sentiment Analysis (Figures 10,11,12), the limit lays a little further, allowing up to 40% of corrupted workers.

For Type 3.1 and Type 3.2 accuracy falls gradually as shown in Figure 7, 10. Note that, such spammers affect negatively the balanced results when most of the tasks have ground truth as 0 and on the contrary affect it positively the ground truth is 1 for a majority of workers. Indeed, specificity of aggregated answers with Type 3.1 spammers approaches 1 (refer Figure 8, 11) and the recall falls to 0 (refer Figure 9, 12) when the number of spammers increases. This behavior is expected as Type 3.1 spammers always return answer 1 (and hence achieve 100% correct answers on images with ducks). Symmetric results are obtained with Type 3.2 spammers as depicted in Figures 8 and 9.

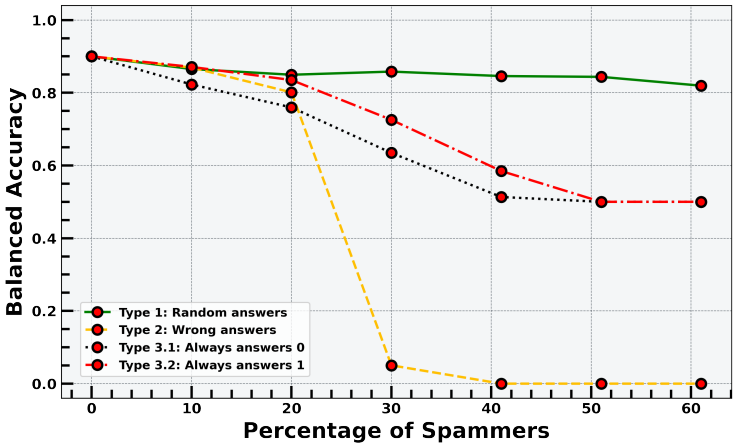

Fig. 7: Effect of individual type spammers: Type 1, Type 2, Type 3.1 and Type 3.2 on balanced accuracy for Duck Identification dataset.

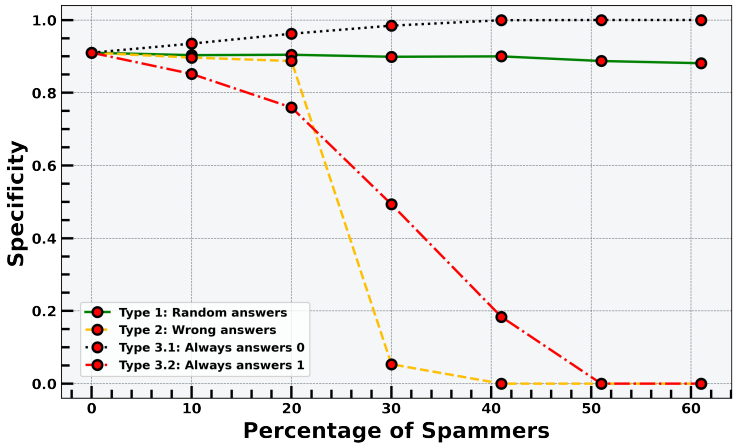

Fig. 8: Effect of individual type spammers: Type 1, Type 2, Type 3.1 and Type 3.2 on specificity for Duck Identification dataset.

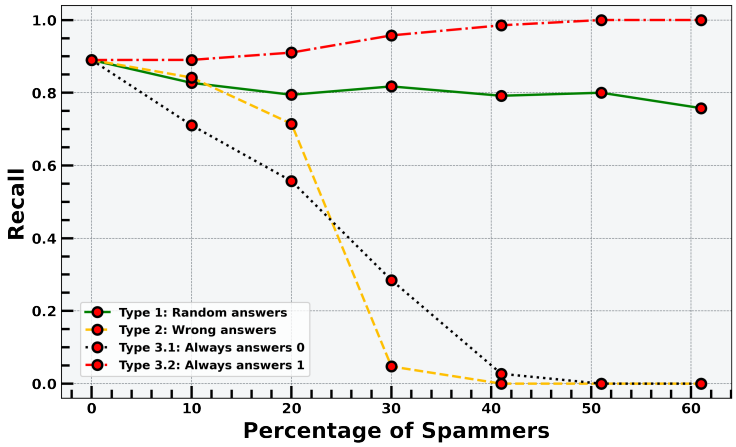

Fig. 9: Effect of individual type spammers: Type 1, Type 2, Type 3.1 and Type 3.2 on recall for Duck Identification dataset.

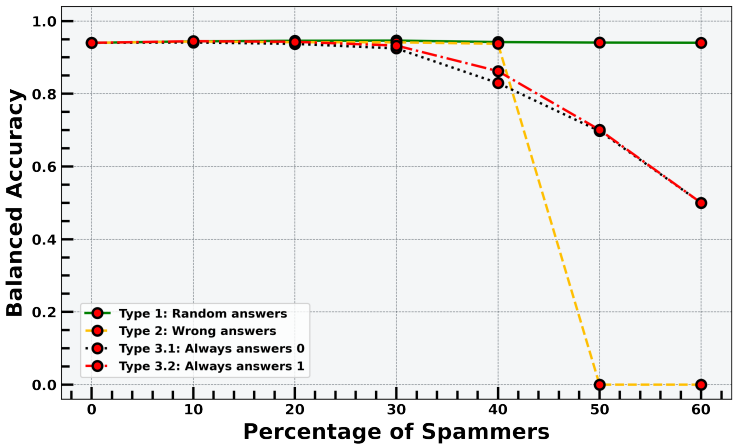

Fig. 10: Effect of individual type spammers: Type 1, Type 2, Type 3.1 and Type 3.2 on balanced accuracy for Sentiment Analysis dataset.

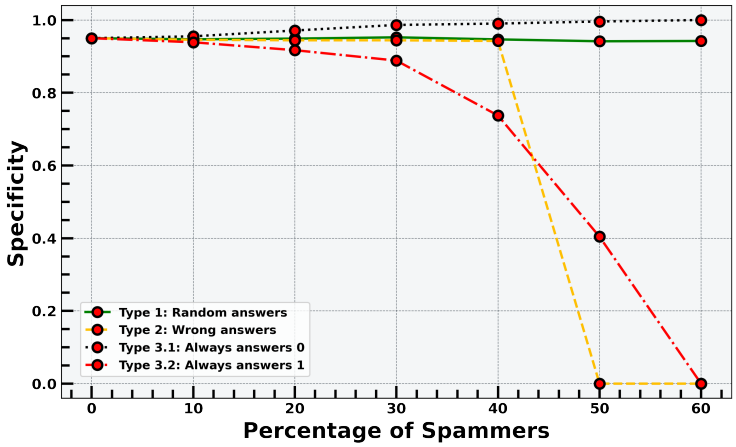

Fig. 11: Effect of individual type spammers: Type 1, Type 2, Type 3.1 and Type 3.2 on specificity for Sentiment Analysis dataset.

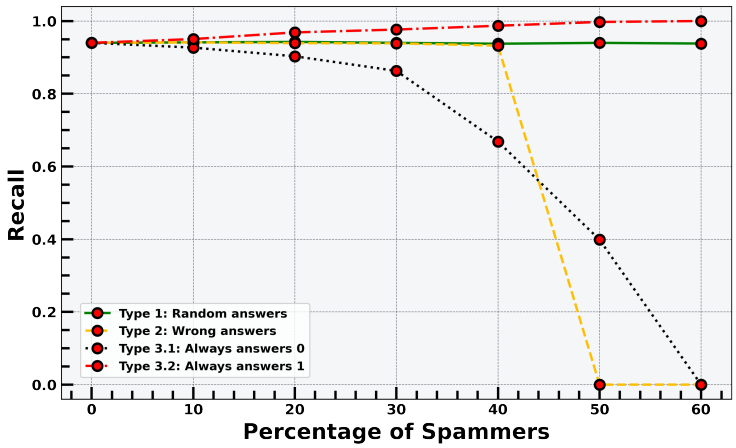

Fig. 12: Effect of individual type spammers: Type 1, Type 2, Type 3.1 and Type 3.2 on recall for Sentiment Analysis dataset.

Let us now consider mixed set of spammers. We introduced an equal share of spammers of each type and incrementally increased the percentage of spammers among workers. We find that in this case, recall, specificity and the balanced accuracy falls gradually as shown in Figures 13 and 14. An intuitive explanation is that introducing an equal percentage of spammers of each type among workers amounts to increasing the number of random answers: the behaviors of Type 3.1 and 3.2 spammers cancel each other and the impact of Type 2 spammers is hence less significant, resulting in an behavior close to that of a set of workers with a smaller number of (Type 1) spammers.

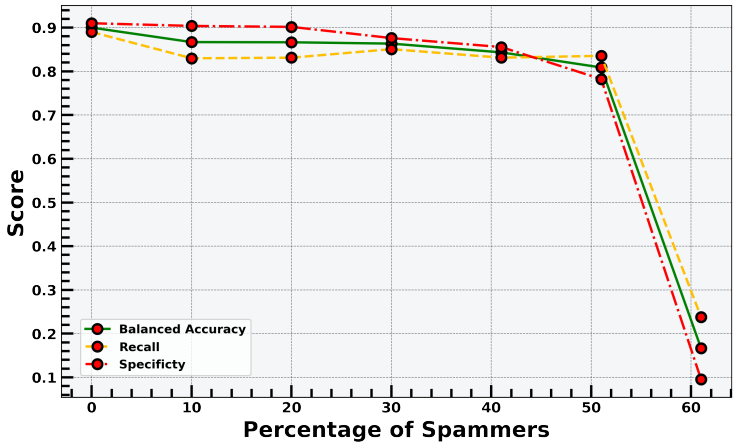

Fig. 13: Effect of Mixed spammers: equally distributed Type 1, Type 2, Type 3.1 and Type 3.2 for Duck Identification dataset.

6.5 Effect of spammers on costs

The objective of our last experimentation is to study how spammers affect accuracy but also the costs of our CrowdInc algorithm. As for the previous experiments, we again consider the Duck Identification and Sentiment Analysis datasets. We set up two types of experimentation. In the first case, we insert individual spammers of each type and increment the percentage of spammers from 10% to 60%. On the other hand, we insert a mixed set of spammers with an equal proportion of spammers of all type. Here, we also insert spammers incrementally from 10% to 60%. We compare the performance of CrowdInc to varying degree of spammers in both cases.

We first show compared costs and accuracies for sets of spammers of a single type, for growing ratios of malevolent workers. We show the achieved results for Duck Identification in Figure 15 and Sentiment Analysis in Figure 16. In each

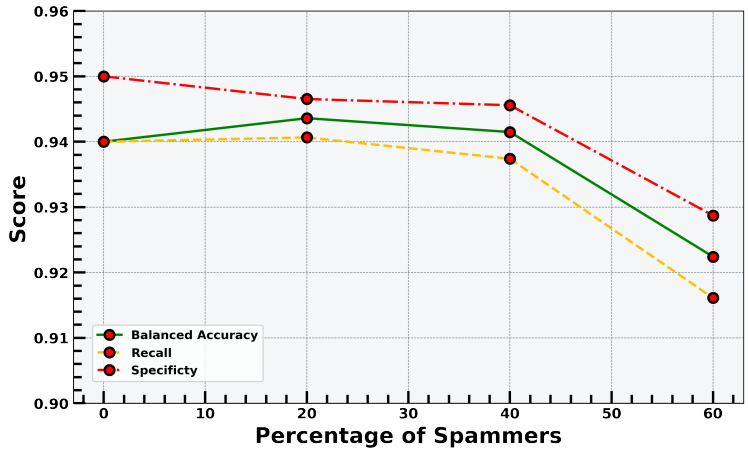

Fig. 14: Effect of Mixed spammers: equally distributed Type 1, Type 2, Type 3.1 and Type 3.2 for Sentiment Analysis dataset.

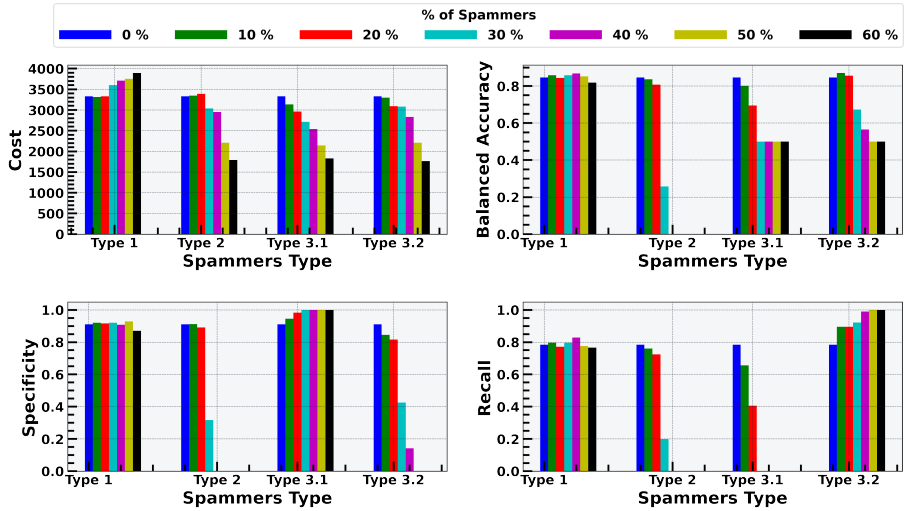

Fig. 15: Comparison of cost vs. accuracy (Individual Spammers): Type 1, Type 2, Type 3.1 and Type 3.2 with varying percentage of spammers for Duck Identification dataset.

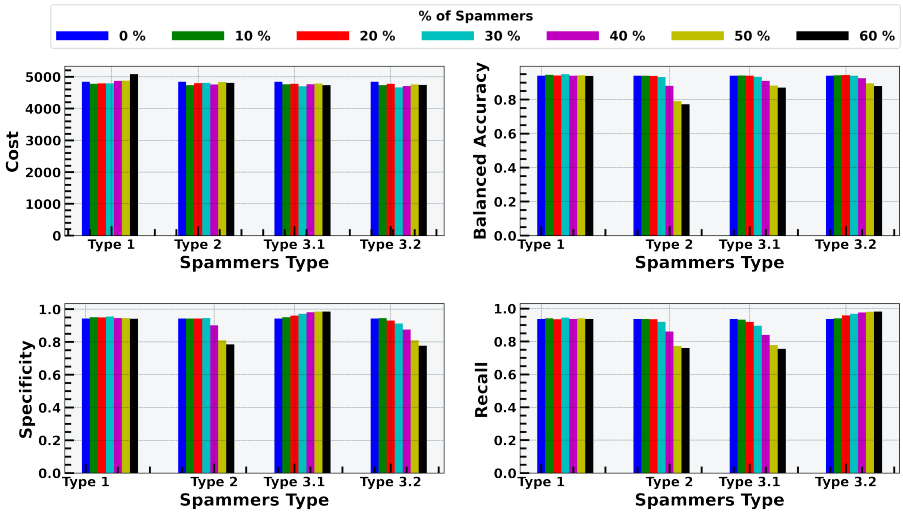

Fig. 16: Comparison of cost vs. accuracy (Individual Spammers): Type 1, Type 2, Type 3.1 and Type 3.2 with varying percentage of spammers for Sentiment Analysis dataset.

Figure, the top left bar plot represents the cost, the top right bar plot represents the accuracy, the bottom left bar plot represents the specificity and the bottom right bar plot represents the recall, for each spammer type and for varying percentages of spammers. Let us first consider Duck Identification (Figure 15). For Type 1 spammers, the cost of aggregation increases with the percentage of spammers, while accuracy, specificity and recall remain quite stable. The explanation is that, when spammers give random answers they may be correct or wrong. Such random answering by workers increases the number of steps needed to converge to a final consensual answer, but one error compensates the other, and the final results are not affected. The Type 2 spammers provide always incorrect answers. We observe that in Duck Identification, the cost decreases as well as accuracy, recall and specificity (which are almost 0 and not visible on plots for a percentage of spammer $\geq 40\%$). Note that compared to Type 1 spammer, performance decreases faster. For Type 3.1 spammers, we observe that cost as well as accuracy decreases when the number of spammers increases. Specificity increases and approaches 1 with a high number of spammers. In contrast, the recall values fall very sharply and reach 0. The intuitive reason is the following: for tasks with ground truth is, the spammers return 1. Hence, specificity approaches to 1 when the percentage of Type 3.1 spammers increases. For the tasks with ground truth 0, spammers return a wrong answer, so recall decreases rapidly. The situation for Type 3.2 spammers is symmetric. It is also interesting to note that the accuracy in the case of Type 3.1 and Type 3.2 spammers depends upon the proportion of ground truth with 0 or 1. If the dataset consists of a greater

proportion of tasks with ground truth 0, we get greater accuracy when we have Type 3.1 spammers. In contrast with a higher proportion of tasks with ground truth as 1, we get greater accuracy with Type 3.2 spammers than with Type 3.1 spammers. Though it might be intriguing that cost decreases with Type 2, Type 3.1 and Type 3.2 spammers, there is an intuitive explanation: as spammers all provide the same biased answer, they allow to reach a (possibly wrong) consensus within a smaller number of rounds. For Sentiment Analysis, the costs remain almost stable, while accuracy decreases. In fact, even when the number of spammers increases, the impact on cost and accuracy also depends upon how the genuine workers are answering. If there is a lot of agreement among the answers provided by the genuine workers, the impact of spammers on cost and accuracy is limited. In the case of Sentiment Analysis, there is a lot of agreement among workers answers. As a result, the effect of spammers answers is lighter than for the Duck Identification dataset.

Let us now analyze results for a mixed set of spammers with an equal proportion of each spammer type. We find that when the percentage of spammers increases, the cost of CrowdInc increases and the performance metrics (accuracy, specificity and recall) decrease. The results are shown in Figure 17 and 18. As explained earlier, mixed composition of spammer sets shows identical performance as spammers sets with only Type 1 spammers. Here also, changes in cost and accuracy depend on answers of genuine workers. As a result, we can observe that the effect of spammers is less important on the Sentiment Analysis dataset than on the Duck Identification dataset. This observation extends to costs.

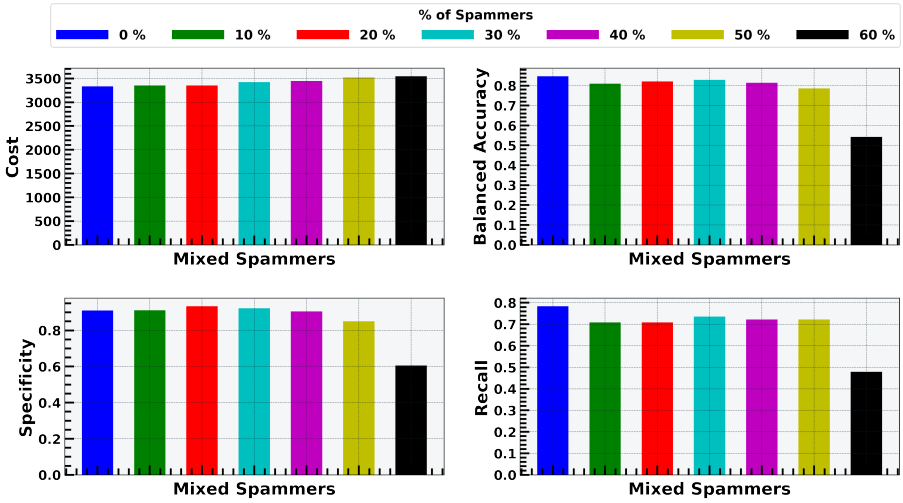

Fig. 17: Comparison of cost vs. accuracy (Mixed Spammers): equally distributed Type 1, Type 2, Type 3.1 and Type 3.2 with varying percentage of spammers for Duck Identification dataset.

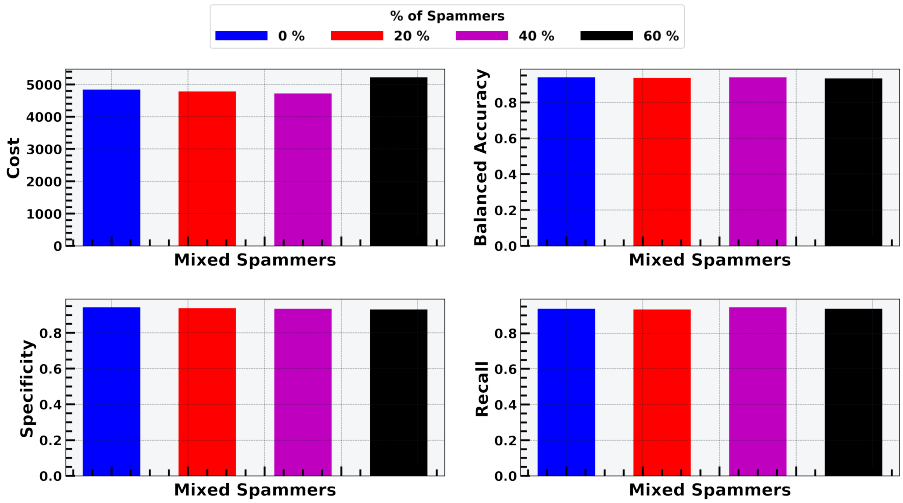

Fig. 18: Comparison of cost vs. accuracy (Mixed Spammers): equally distributed Type 1, Type 2, Type 3.1 and Type 3.2 with varying percentage of spammers for Sentiment Analysis dataset.

Overall, the experimentation showed a very marginal increase is the cost of CrowdInc (and hence also on the number of rounds) for a percentage of spammers below 20%. Accuracy of answers is still good with 20% of spammers of all types. This means that Crowding stops with a confidence in forged data that is still high. Accuracy starts to fall dramatically (and costs starts to increase) for Duck Identification with 30% of spammers, while results are still acceptable with 40% of spammers with Sentiment Analysis. This shows that spammer impact, just like performance of aggregation mechanisms, is data-dependent. Though it is difficult to find general rules explaining this difference, one can notice that the number of answers in Sentiment Analysis is larger. An hypothesis is that it makes this dataset more robust to changes introduced by malevolent workers. Following the remarks in [26], who concluded that there is not aggregation technique that can be considered as the ultimate one, we believe that spammer sensitivity is a parameter that may vary with size and difficulty of data. This is not only a characteristics of data, but also of workers culture and beliefs: [23] showed on the Duck Identification dataset that workers had difficulties to differentiate ducks and grebes, due to a cultural habit associating ducks with a short neck. Yet, a general tendency showed by our experiment is that Type 1 spammers (returning random answers) have little effect on accuracy of the results. This is good news, as it means that influencing the results of a crowdsourced vote requires coordination among workers (at least to agree on the answer that has to be forced).

7 Conclusion and discussions

In this paper, we introduced an aggregation technique for crowdsourcing platforms. Aggregation is based on expectation maximization and jointly estimates the answers, the difficulty of tasks, and the expertise of workers. Using difficulty and expertise as latent variables improves the accuracy of aggregation in terms of recall and specificity. We also proposed CrowdInc an incremental labeling technique that optimizes the cost of answers collection. The algorithm implements a worker allocation policy that takes decisions from a dynamic threshold computed at each round, which helps achieving a trade off between cost and accuracy. We showed in experiments that our aggregation technique outperforms the existing state-of-the-art techniques. We also showed that our incremental crowdsourcing approach achieves the same accuracy as EM with static allocation of workers, better accuracy than majority voting, and in both cases at lower costs.

In a second part of the paper, we have studied the impact of malevolent workers on performance of CrowdInc. In the considered cases, our algorithm can accept 10 to 20 % of spammers without affecting too much its accuracy and cost. As answering profiles of spammers are rather different from standard behaviors, one can rely on recall and specificity estimation to detect spammers when the percentage of corrupted workers is not too high.

The ideas proposed in this paper can lead to several improvements that will be considered in future work. In the paper, we addressed binary tasks for simplicity, but the approach can be easily extended to tasks with a finite number m of answers. For each worker i, and for a given difficulty, one can specify the joint probability of ground truth $y_j = v$ and of an answer $L_{ij} = v' \in \{1, \ldots, m\}$ for $v, v' \in \{1, \ldots, m\}$. The role of the EM algorithm remains to estimate the ground truth and parameters such as task difficulty and workers expertise. The difficulty of each task t_j remains a parameter d_j. Expertise is the ability to classify a task as m when its ground truth is m. An EM algorithm just has to consider probabilities of the form $Pr(L_{ij} = v' | y_j = v, \alpha_i, \beta_i, d_j)$ to derive hidden parameters and final labels for each task. Another easy improvement is to consider incentives that depend on workers accuracy. This can be done with a slight adaptation of costs in the CrowdInc algorithm. Another possible improvement is to try to hire experts when the synthesized difficulty of a task is high, to avoid hiring numerous workers or increase the number of rounds.

Last, we think that the complexity of CrowdInc can be improved. The complexity of each E-step of the aggregation is linear in the number of answers. The M-step maximizes the log likelihood with an iterative process (truncated Newton algorithm). However, the E and M steps have to be repeated many times. The cost of this iteration is visible in Table 3, where one clearly see the difference of running time between a linear approach such as Majority Voting (third column), a single round of EM (second column), and CrowdInc. Using CrowdInc to reduce costs results in an increased duration to compute final answers. Indeed, the calculus performed at round i to compute hidden variables for a task t is lost at step $i + 1$ if t is not stopped. An interesting idea is to consider how a part of computations can be reused from a round to the next one to speed up

convergence. However, building an incremental version of EM is far from being trivial. Indeed, EM converges towards optima, that can sometimes be local. It is kwown that the choice of initial values for hidden parameters need not influence the final result. One can, for instance, reuse the recall and specificity computed for workers in a round, and expect this quality to remains stable in subsequent rounds, and hence speed up convergence of EM. However, this particular initialization of round does guarantee improvement of CrowdInc in all cases, and experimental validation is needed to show that, on the average, remembering parameters speeds up convergence.

Acknowledgements: We would like to thank anonymous reviewers for their careful reading and for useful comments on a preliminary version of this work.

References

1. P. Dai, C. H. Lin, and D. S. Weld. POMDP-based control of workflows for crowdsourcing. *Artificial Intelligence*, 202:52–85, 2013.
2. F. Daniel, P. Kucherbaev, C. Cappiello, B. Benatallah, and M. Allahbakhsh. Quality control in crowdsourcing: A survey of quality attributes, assessment techniques, and assurance actions. *ACM Computing Surveys*, 51(1):7, 2018.
3. A.Ph. Dawid and A.M. Skene. Maximum likelihood estimation of observer error-rates using the EM algorithm. *J. of the Royal Statistical Society: Series C (Applied Statistics)*, 28(1):20–28, 1979.
4. G. Demartini, D.E. Difallah, and Ph. Cudré-Mauroux. Zencrowd: leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking. In *Proceedings of the 21st World Wide Web Conference (WWW'12)*, pages 469–478. ACM, 2012.
5. A.P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.
6. P.A. Flach. *Machine Learning - The Art and Science of Algorithms that Make Sense of Data*. Cambridge University Press, 2012.
7. H. Garcia-Molina, M. Joglekar, A. Marcus, A. Parameswaran, and V. Verroios. Challenges in data crowdsourcing. *Transactions on Knowledge and Data Engineering*, 28(4):901–911, 2016.
8. Baocheng Geng, Qunwei Li, and Pramod K. Varshney. Prospect theory based crowdsourcing for classification in the presence of spammers. *IEEE Trans. Signal Process.*, 68:4083–4093, 2020.
9. H. Halpin and R. Blanco. Machine-learning for spammer detection in crowd-sourcing. In *Proceedings of the 4th Human Computation Workshop, HCOMP@AAAI 2012*, volume WS-12-08 of *AAAI Workshops*. AAAI Press, 2012.
10. D.R. Karger, S. Oh, and D. Shah. Iterative learning for reliable crowdsourcing systems. In *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems (NIPS'11)*, pages 1953–1961, 2011.
11. J. Le, A. Edmonds, V. Hester, and L. Biewald. Ensuring quality in crowdsourced search relevance evaluation: The effects of training question distribution. In *SIGIR 2010 workshop on crowdsourcing for search evaluation*, volume 2126, pages 22–32, 2010.

12. Q. Li, Y. Li, J. Gao, B. Zhao, W. Fan, and J. Han. Resolving conflicts in heterogeneous data by truth discovery and source reliability estimation. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, pages 1187–1198. ACM, 2014.
13. C. Miao, Q. Li, L. Su, M. Huai, W. Jiang, and J. Gao. Attack under disguise: An intelligent data poisoning attack mechanism in crowdsourcing. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018*, pages 13–22. ACM, 2018.
14. S. G. Nash. Newton-type minimization via the lanczos method. *SIAM Journal on Numerical Analysis*, 21(4):770–788, 1984.
15. B. Pang and L. Lee. A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of ACL'04, the 42nd annual meeting on Association for Computational Linguistics*, pages 271–278. Association for Computational Linguistics, 2004.
16. V. Raykar and P. Agrawal. Sequential crowdsourced labeling as an epsilon-greedy exploration in a markov decision process. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, volume 33 of *Proceedings of Machine Learning Research*, pages 832–840. PMLR, 2014.
17. V. C. Raykar, S. Yu, L.H. Zhao, G.H. Valadez, C. Florin, L. Bogoni, and L. Moy. Learning from crowds. *J. of Machine Learning Research*, 11(Apr):1297–1322, 2010.
18. Vikas C. Raykar and Shipeng Yu. Eliminating spammers and ranking annotators for crowdsourced labeling tasks. *J. Mach. Learn. Res.*, 13(1):491–518, February 2012.
19. L. Tran-Thanh, M. Venanzi, A. Rogers, and N.R. Jennings. Efficient budget allocation with accuracy guarantees for crowdsourcing classification tasks. In *Proceedings of the 12th International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '13*, pages 901–908. International Foundation for Autonomous Agents and Multiagent Systems, 2013.
20. M. Venanzi, J. Guiver, G. Kazai, P. Kohli, and M. Shokouhi. Community-based bayesian aggregation models for crowdsourcing. In *23rd International World Wide Web Conference, WWW '14*, pages 155–164. ACM, 2014.
21. J. Wang, T. Kraska, M.J. Franklin, and J. Feng. Crowder: Crowdsourcing entity resolution. *Proceedings of the VLDB Endowment*, 5(11):1483–1494, 2012.
22. Y. Wang, K. Wang, and C. Miao. Truth discovery against strategic sybil attack in crowdsourcing. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 95–104. ACM, 2020.
23. P. Welinder, S. Branson, P. Perona, and S.J. Belongie. The multidimensional wisdom of crowds. In *Proceedings of NIPS'10, Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010*, pages 2424–2432. Curran Associates, Inc., 2010.
24. J. Whitehill, T. Wu, J. Bergsma, J.R. Movellan, and P.L. Ruvolo. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *Proceedings of NIPS'09, Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems*, pages 2035–2043. Curran Associates, Inc., 2009.
25. A. Xu, X. Feng, and Y. Tian. Revealing, characterizing, and detecting crowdsourcing spammers: A case study in community q&a. In *Proceedings of INFOCOM 2015, Conference on Computer Communications*, pages 2533–2541. IEEE, 2015.
26. Y. Zheng, G. Li, Y. Li, C. Shan, and R. Cheng. Truth inference in crowdsourcing: Is the problem solved? *Proceedings of the VLDB Endowment*, 10(5):541–552, 2017.