Assignment Techniques for Crowdsourcing Sensitive Tasks

L. Elisa Celis
École Polytechnique Fédérale
de Lausanne, Switzerland
elisa.celis@epfl.ch

Sai Praneeth Reddy
Indian Institute of Technology
Delhi, India
cs5110294@iitd.ac.in

Ishaan Preet Singh
Indian Institute of Technology
Delhi, India
cs5110280@iitd.ac.in

Shailesh Vaya
Xerox Research Centre India
Bangalore, India
shailesh.vaya@xerox.com

ABSTRACT

Protecting the privacy of crowd workers has been an important topic in crowdsourcing, however, task privacy has largely been ignored despite the fact that many tasks, e.g., form digitization, live audio transcription or image tagging often contain sensitive information. Although assigning an entire job to a worker may leak private information, jobs can often be split into small components that individually do not. We study the problem of distributing such tasks to workers with the goal of maximizing task privacy using such an approach.

We introduce information loss functions to formally measure the amount of private information leaked as a function of the task assignment. We then design assignment mechanisms for three different assignment settings: PUSH, PULL and a new setting Tug Of War (TOW), which is an intermediate approach that balances flexibility for both workers and requesters. Our assignment algorithms have zero privacy loss for PUSH, and tight theoretical guarantees for PULL. For TOW, our assignment algorithm provably outperforms PULL; importantly the privacy loss is independent of the number of tasks, even when workers collude. We further analyze the performance and privacy tradeoffs empirically on simulated and real-world collusion networks and find that our algorithms outperform the theoretical guarantees.

Author Keywords

Crowdsourcing; Microtasks; Privacy; Social Networks

ACM Classification Keywords

H.1.m Information systems: Models and Principles; K.4.m Computing Milieux: Computing and Society

INTRODUCTION

Crowdsourcing, a term first coined by Wired magazine in 2006, is "the act of taking a job traditionally performed by a designated agent (usually an employee) and outsourcing it to an undefined, generally large group of people in the form of an open call." [18] Often, companies have large volumes of relatively simple jobs which can be easily completed by humans but not (yet) computers; e.g., image labelling, object recognition, video tagging, text digitisation or translation and transcription.

However, given the fact that there is little control over crowd workers, revealing private information can have risks and information contained in tasks has been a target of malicious attacks [16, 17]. For example, a worker (or worse – a colluding group of workers) may be able to decipher the name and prescription on a medicine bottle given an image to label, or steal the identity of a person from a form in a transcription task [14]. There are often ethical, institutional, financial or legal reasons to preserve private information embedded in tasks, especially with user sourced or healthcare data. For example, crowdsourcing the evaluation of standardised tests can be problematic since student work is protected by privacy laws [7]. Such legal issues or business best practices (such as six sigma) mean that along with practical safety, theoretical guarantees are necessary. Even if we construct algorithms that perform well in *practice*, legal and ethical considerations must be satisfied before implementation in real world applications, making theoretical guarantees a *prerequisite* in many scenarios. Traditional approaches for achieving data security, such as encryption, do not work here since a crowdworker must have access to *some* data in order to complete their task. Similarly, recent approaches towards data privacy such as k-anonymity and differential privacy cannot be applied since they function by processing of the data (e.g., by modifying some of the entries in a database) whereas when crowdsourcing we are still in the process of *acquiring* the data.

Instead, we use the following insight: if the private data is broken down into enough components, no single component would allow the worker to reconstruct the original content of the data. In this manner, we can prevent the release of *private* information while still releasing all of the data. As long as no *single* worker has access to multiple components, privacy is maintained. Such approaches have been used in practice, e.g., for digitisation tasks; however, no formal guarantees have been provided. Moreover, all proposed solutions of this form assume that there is no *collusion* between work-

ers. We instead provide a solution for cases with a known underlying collusion network in which nodes collude with their neighbours. For common assignment settings our approach works for such graphs that have maximum degree in $o(\sqrt{n})$, where n is the number of workers, and tasks that can be split into $o(\sqrt{n})$ components.

Our Contributions

In this work, we address the challenge of creating and posting tasks on a crowdsourcing platform while maintaining privacy with respect to task content. We consider a generic class of tasks where each task can be split into multiple *components* such that each component can be independently completed by a crowdworker, and no single component leaks confidential information.[1] We make three main contributions:

1. We formalise the notion of a *information loss* $\mathcal{I}$, and introduce the *Reliable Information Ratio* $\mathcal{I}_{RIR}$ as a flexible and robust metric of effective information loss.
2. We consider two common task assignment settings (PUSH and PULL) and introduce a third hybrid setting (TOW), and present task assignment algorithms for all three. We provide theoretical guarantees and analyse their tradeoffs as a function of the collusion network and task properties.
3. We empirically evaluate our algorithms on the class of Watts-Strogatz collusion graphs, and a real-world social network.

Rekatsinas et al. [40] consider partitioning large data amongst adversarial data centres and state that assuming no collusion is "necessary in application domains that require release of private user information". Hence, the fact that we can provide *any* bounds on the amount of information lost when collusion occurs is of interest. Moreover, the TOW assignment setting should be of independent interest; it would be simple for a platform to implement, would not change the worker or requester experience, and could allow for better guarantees including, but not limited, to privacy.

APPROACH

Private Information Loss

We first introduce a general definition of information loss which can be adapted to specific domains. We call a single *unit* of private information a *gem*. The exact definition will be domain specific and depend on the type of information which is to be protected; e.g., the identity of a person in a photograph, the matching of an illness to a patient, uncovering the grade of a student. We define an *information loss function* $\mathcal{I} : \mathcal{A} \to \mathbb{R}^+$ as a mapping from a given *assignment* of tasks to workers $A \in \mathcal{A}$ to a measure of the number of gems lost. The ideal scenario would be to have an assignment mechanism that produces an assignment $A^\star$ such that $A^\star = \text{argmin}_{A \in \mathcal{A}} \mathcal{I}(A)$. When not attainable, we instead provide theoretical guarantees that *upper bound* the information lost when using our assignment mechanism.

[1]We assume that multiple jobs are independent; i.e., no sensitive information can be gleaned from components that come from different jobs. Our approach (with appropriate modifications to the guarantees) can be extended to the dependent case.

Intuitive information loss functions exist, such as the number of *gems* revealed (we call this $\mathcal{I}_1$). More complex metrics can also be used; in this paper we study an additional information loss function $\mathcal{I}_{RIR}$ that captures the *effective* information loss in settings where the identification of a gem is a noisy process.

Collusion Networks

Recent evidence has shown that crowdworkers may collude maliciously [46]. In a collusion network, vertices are crowdworkers and edges represent a pair of workers that can *collude*, i.e., share task content and, hence, *collectively* discover gems. Here, we will assume that collusion means every worker has access to it's neighbours datasets, which is reasonable in practice [11]. If collusion is possible between vertices up to distance k, we can simply take the augmented graph $G + G^2 + \cdots + G^k$.

The social network graph of crowd workers can be inferred in a worst-case manner (i.e., over-estimating link probabilities) using information such as location, interests and groups [24, 48]. Such information about crowdworkers is surprisingly easy to attain, even in restrictive settings [31]. Alternatively, we could use a more restricted platform (such as an actual social network), or use auxiliary information about workers (such as their expertise, and past performance) to construct such a graph [5, 42, 19]. We might have an overestimate of the collusion network, but we can use it as a worst-case scenario in order to provide the strongest possible guarantees. Of course, if the whole graph is colluding (such as a fully connected graph), we have no hope of giving a good privacy guarantee. However, in social networks it is believed that the degree is $O(\log(n))$, which more than suffices for guarantees.

Recall that the information loss is a function of the task assignment. If there is no collusion, the information loss does not change if we permute the assignment. However, with collusion the topology of the network comes into play and the assignment mechanism must take it into account.

Assignment Settings

Our main results fall into three categories of *assignment settings*. Different crowdsourcing paradigms allow different amounts of control over 1) the selection of a worker, and 2) the assignment of a task. We consider three such settings which capture the essence of the available degrees of freedom.

The PULL setting is common in microtask assignment, popularised by platforms such as Amazon Mechanical Turk. In this setting, tasks are posted in a *queue*, and crowd workers arrive in an online manner and *"pull"* a task off the queue. The requester cannot directly control which worker gets which task, other than potentially placing a blanket ban on a certain kind of worker, e.g., with geographic or reputation-based restrictions, or limit the number of tasks a single worker can accept. Given the restrictiveness of this setting, the best strategy we can hope for is simply to randomise the order of the tasks in the queue. By connecting this assignment algorithm to the well-studied balls-and-bins problem, we can upper bound information loss with high probability.

PUSH is the traditional employment setting where an employer has all the tasks and workers, and can assign a task to any worker (who may have a work *capacity*). We show that *zero* information is lost, as long as there are *enough* workers. The lower bound on the number of workers depends on the topology of the collusion network (which we assume is known). We prove that we require roughly $m\Delta$ workers, where Δ is the maximum degree of the network and m is the number of components in a task. Importantly, this bound does not depend on the number of tasks to be crowdsourced. Empirically, our algorithms perform even better and the number of workers required seems to depend on the *average* rather than the maximum degree.

We introduce a hybrid setting, which we call "tug-of-war" (TOW), that interlaces worker and requester preferences (similar in spirit to [9]). Workers arrive in an online manner (as in PULL) and must be assigned a task, but the task can be chosen depending on the worker's attributes (as in PUSH). Hence, we can use their history, or social graph etc. to minimise $\mathcal{I}$. Allowing the hybrid TOW assignment setting would require one additional layer (already present in some platforms) that gathers information about crowdworkers, either through verified social networks, financial information, or task history. The problem in this case is that when we are *almost out of tasks*, our options are limited and we may be forced to assign multiple components of the same task to a single worker (or to neighbouring workers). However, we show that this can only occur roughly $C\Delta$ times where C is the maximum number of times a worker arrives. The above bound can be large. However, while not achieving optimality as with PUSH, the number of gems lost is still independent of the number of tasks. This is a strong contrast to PULL where the dependence is linear. Hence, this task assignment setting proves to be a good compromise between PUSH and PULL, and is of practical interest when privacy is a concern in crowdsourcing. More generally, we expect the added flexibility to the requester while maintaining the freedom for workers to arrive at will strikes a natural balance which could be useful to applications beyond privacy where requesters can use the worker's task history to their advantage when assigning tasks.

RELATED WORK

Crowdsourcing is of growing public interest and has established itself in mainstream business practice and research methodology, using a variety of methods to engage humans to solve large or complex problems. Parts of almost all large jobs can be split into simpler tasks and crowdsourced [25]. This has been especially popular in areas such as document processing [22], data labelling [6], audio transcription/translation, judging the relevance of search results and many others that require minimal skills. Moreover, there are many frameworks that allow tasks to be completed on online marketplaces [26].

There is an increasing concern and study of the privacy of individuals in online social networks [15], and more recently, of crowd workers [31]. Also, a number of questions have been raised about the privacy of the information in tasks to be crowdsourced [22]. A bottleneck to the expansion of crowdsourcing to many domains has been the privacy of the data given to workers [27, 50]. An analysis of Amazon's Mechanical Turk found it vulnerable to coordinated attacks by crowdworkers [30]. There are many platforms that assist users in their daily lives such as Solent which helps edit documents, or PlateMate that informs users of the calorie content of food through user uploaded pictures. These systems use the power of the crowd but operate on possibly sensitive user generated content [29]. Some protocols that trade performance and quality of work with privacy have been proposed and studied [20]. For example, specialised approaches exists that degrade the quality of images to be labelled using noise and random perturbations [45], while [14] blurs videos so that crowdworkers can not identify faces. Some approaches that provide k-anonymity and differential privacy have been proposed [51, 41] but these require processing of the data which is often not possible. For example, anonymising form data before crowdsourcing transcription would require knowledge of data filed in forms, which would only by available after transcription.

Simultaneously, there have also been many advances in matching users to tasks, to better improve the work quality [43, 23, 12]. We extend this idea, of using some minimal information about users, to actually preserve data privacy. [40] considers the related problem of data sharing, and presents a framework for partitioning private data across multiple non-colluding adversaries with the goal of distributing data such that an adversary cannot glean any private information. However, their focus is on partitioning the data set into components; we instead assume such a partition is given or easy to attain, but that we may be forced to give workers multiple components.

APPLICATIONS

Along with healthcare related applications, any crowdsourcing task that is based on user generated content along with many miscellaneous applications such as labelling of satellite imagery, or reviewing closed circuit videos, are tasks that would benefit from crowdsourcing but deal with sensitive data.

Form digitisation in the context of crowdsourcing has been widely studied, due in part to its wide use in industry [32, 8]. Insurance forms are often handwritten and need to be digitised (2009's HITECH Act alone sanctioned over 20 billion dollars as stimulate to encourage electronic health records[4]). The number of such forms can be large, has high variance, and require little to no skilled labor [37] making it ideally suited for crowd work. However, health insurance forms could reveal very sensitive information about a patient. In the US, the Health Insurance Portability and Accountability Act (HIPAA) guarantees privacy to patients, and outlines very stringent requirements for processing of healthcare related forms [36], which is a significant barrier for crowdsourcing.

Concretely, if we distribute the contents of a form such that one worker get the first name of the patient, another gets the last name of the doctor, another gets the billing code and so on, then workers have no context and thus have no private

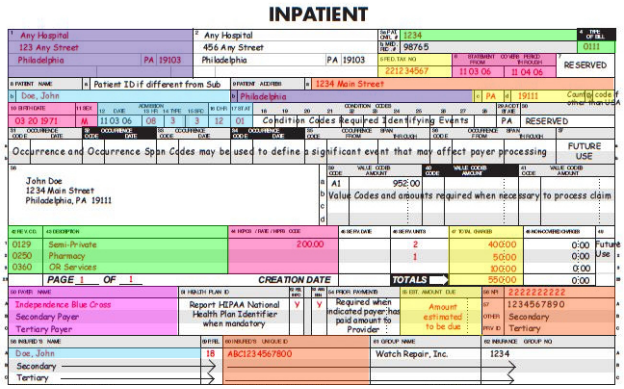


Figure 1. An example insurance form.

information. Figure 1, shows a snippet of a patient record, which is split into components so that it can be digitised. Fields with the same color form a component. For example, the *pink* fields include Birthdate and Sex. These together do not reveal any personal information. However, if you combine this with the name from the *blue* fields, then we could learn who was admitted to a hospital. If all the components are digitised separately, they can be combined to give the complete digitised form. Note that the number of such parts into which a task can be split obviously needs to be less than the number of workers to guarantee perfect privacy; otherwise we necessarily have to assign multiple parts of the same task to at least one worker. For our results, we assume that the number of parts is $o(\sqrt{n})$ where n is the number of workers. In practice, we expect the number of parts to be in the 10s for a task, while the number of workers is conservatively in the 10000s (for example, Amazon Mechanical Turk has over half a million workers).

The application of our techniques will of course be domain specific but can be done in a number of different ways; for example, splitting tasks temporally may be of interest. In a system such as PlateMate [34], a single picture from a user may not compromise privacy, but a week's pictures together may reveal information about a user's location or habits. Similarly, in audio transcription or translation, crowdworkers getting a single word or phrase to transcribe won't learn any personal information. Crowdsourced assistive technology applications, e.g., Legion:Scribe [28], create real time captions from audio for deaf workers. To improve latency, it splits the task into many small independent task. This has the additional benefit of providing privacy as long as the segments are small enough and no single worker (or set of colluders) is assigned multiple segments.

While the idea of splitting tasks in order to improve privacy is not new [2, 3], guarantees on the performance (and hence the satisfaction of HIPPA-like requirements) have not been validated theoretically or empirically. Moreover, existing work has not considered the possibility of colluding workers. Our work takes a concrete step towards closing these gaps by providing strong statistical guarantees on the amount of information loss, even when accounting for colluding behavior.

PRELIMINARIES

Notation

We consider a particular type of *task* that is to be crowdsourced, and let an *instance* be a specific embodiment of that task; e.g., if the task is digitising handwritten health insurance forms, an instance would be a particular filled form (we will use this as a running example to clarify notation). Hence, in this context, we consider one task, but many instances that are to be completed. A *component* is a portion of a task that can be completed independently, possibly a single field in the form. Form fields (such as name, age etc.), or groups of fields are components that can be digitised individually and multiple fields can be combined to get the completed final task.

Recall that we call a single unit of private information a *gem*. A *Privacy Preserving Partition* (or simply *Partition*) of a task is a collection of components such that no single component can be used to find a gem, and if all components are completed then we can reconstruct a completed task. If the identity of the person who filled the form is a *gem*, then the first name field along with gender and age may form one set of components, while last name and weight might be a second. Individually, a component does not reveal a *gem* but both sets combined may leak the identity. Let the number of components in such a partition be m. [2]. Let n be the number of crowd workers, f the number of instances and Δ the maximum degree in the graph $G = G' + G'^2$, where G' is the collusion network of crowdworkers. Hence, G is the graph such that if (u, v) is an edge $\in G$, then u can reach v in two steps. This simulates the situation where a node has access to all gems assigned to itself and those assigned to it's neighbours (in G').

n	number of crowdworkers
f	number of instances of a task
m	number of components in a task
C	maximum capacity of a worker
G	the collusion graph
Δ	maximum degree of G
p	probability of a true positive
q	probability of a false positive

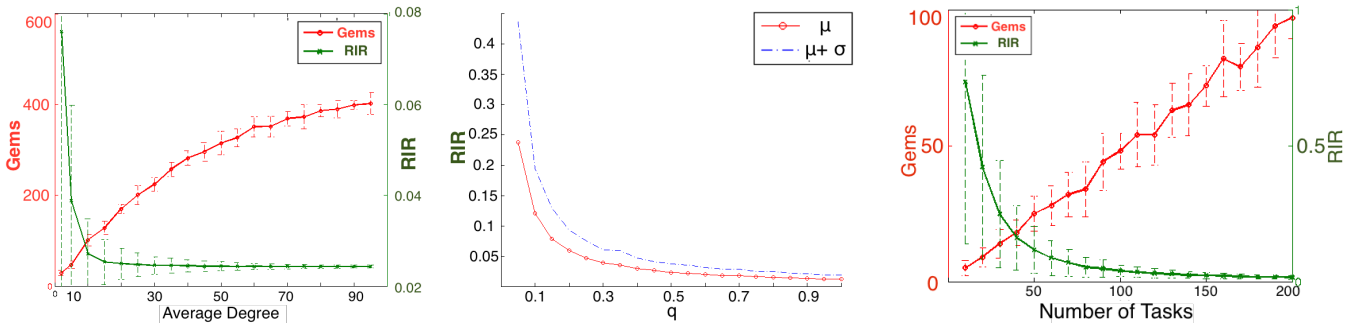
Table 1. Notation

Modelling Information Loss

Recall that for an assignment A of tasks to workers, an *information loss function* $\mathcal{I} : \mathcal{A} \to \mathbb{R}^+$ is a measure of the number of gems lost. Our goal is to find an assignment that minimizes the information loss function. The first information loss function which comes to mind is simply a count of the number of gems lost. We call this

$$\mathcal{I}_1 = \sum_{\text{task j}} \#\text{ gems lost from } j. \tag{1}$$

[2]Note that our results improve as m decreases, so it is important in a practical setting to design small partitions. This is outside the scope of this paper but studied in [40]



(a) $f = 100$, $q = 0.3$, varying k (b) $f = 100$, $k = 5$, varying q (c) $k = 5$, $q = 0.3$, varying f

Figure 2. PULL experiments: The number of gems discovered and $\mathcal{I}_{RIR}$, $n = 100$, $m = 5$, $\beta = 0.4$, $p = 0.9$ on a Watts-Strogatzcollusion network.

The cost of information loss can of course be non-linear. Hence, we define a generalisation

$$\mathcal{I}_g = \sum_{\text{task j}} g(\#\text{ gems lost from } j)\,, \tag{2}$$

where g is a positive monotonically increasing convex function. We have been considering tasks where an instance is compromised if two or more components of the same instance are given to a particular worker. Let $\mathcal{I}_s$ be a variant of I_g which equals the total number of times each task is compromised. A simple metric such as $\mathcal{I}_s$ is a useful measure of how many gems have spread to how many people. Formally,

$$\mathcal{I}_s = \sum_{\text{worker } i} \sum_{\text{task } j} 1_{\{i \text{ has gem from } j\}}. \tag{3}$$

While functions of the form above capture *worst case* bounds, they implicitly assume that every crowd worker is malicious, and, moreover, can perfectly identify gems. However, we can often present an anonymised version of a task, or partition a task such that it is a priori not clear which parts of the task go together [40]. For example, given a last name field and first name field in a form, it would not be clear whether they were actually from the same form; snippets of an audio stream would not seem to be from the same source as long as their pitch/tone and other aspects were changed. While we lose a gem if the worker can de-anonymise the task or piece together multiple components of the same job, these processes can be noisy. Hence, inspired by the signal-to-noise ratio, we introduce the *reliable information ratio (RIR)*:

$$\mathcal{I}_{RIR} = \sum_{\text{worker } i} \mathcal{I}^i_{RIR} = \frac{\mathrm{E}\left[\#i\text{'s gem true positives}\right]}{\mathrm{E}\left[\#i\text{'s gem false positives}\right]} \tag{4}$$

A true positive refers to when the worker learns some information correctly from a pair of components of a task (say first name and last name) while a false positive refers to incorrect information learnt by the worker (if the first name and last name were actually from different forms). The greater the number of false positives, the less reliable and discernible the true positives become. Clearly, if $\mathcal{I}^i_{RIR} \ll 1$, then the identification of a gem is such a noisy process that it renders it effectively useless.

This fact comes into focus when one considers the *utility* of detecting a gem. Let $\#^i_c$ be the number of gems worker i identifies correctly and $\#^i_{\bar{c}}$ be the number of gems worker i identifies incorrectly (with a λ penalty for each such false positive). Then, if the worker's utility for detecting gems is $\mathrm{E}[\text{util}] = \mathrm{E}[\#^i_c] - \lambda\mathrm{E}[\#^i_{\bar{c}}]$ for some $\lambda \in [0, 1]$

$$\mathrm{E}[\text{util}] = \mathrm{E}[\#^i_c]\left(1 - \frac{\lambda}{\mathcal{I}^i_{RIR}}\right). \tag{5}$$

If $\mathcal{I}^i_{RIR} \leq \lambda$, the utility of trying to find a gem is negative. λ can be set based on the level of security required.

REMARK 1. *For this paper, we consider a generic class of tasks that can be split into* components *such that a gem is revealed if and only if two or more components of the same job are given to a particular worker. Form digitisation, among others, falls into this category.*

When calculating the *Reliable Information Ratio* $\mathcal{I}^i_{RIR}$ for a worker i, we let p_i be the probability of a true positive given a pair of components from the same instance of a task and $q_i > 0$ be the probability of a false positive given a pair from different instances. If a worker is not malicious, $p_i = 0$ and hence $\mathcal{I}_{RIR} = 0$.

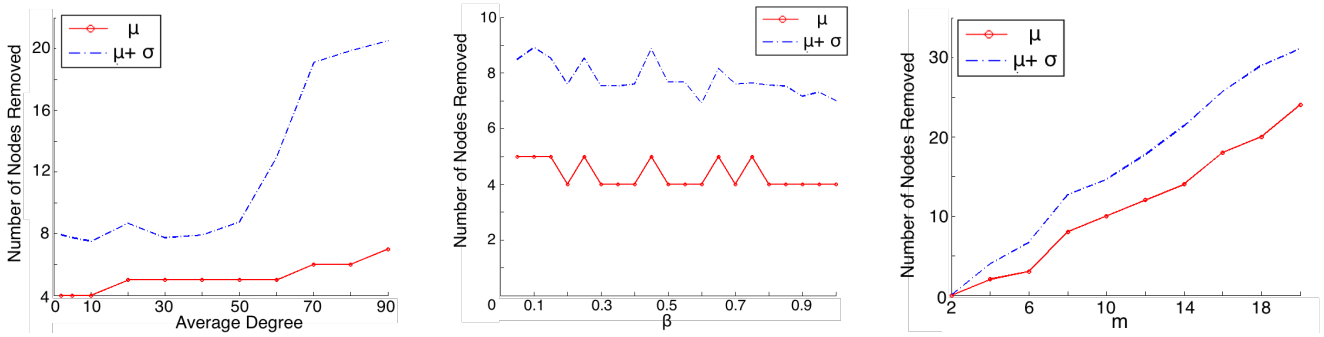
THEORETICAL RESULTS

PULL Assignment Model

In the PULL assignment model, crowd workers arrive in an online manner and take the next task from the queue. This is the assignment model used in a majority of crowdsourcing platforms including Amazon Mechanical Turk. This gives the requester almost no control with which to ensure privacy. Crowdworkers can arrive multiple times in an arbitrary order. However, we neutralise this by randomly permuting the order of the tasks; this effectively gives a random permutation of the crowd workers.

Connection to Balls and Bins

Imagine the components of a task instance as balls, where the balls corresponding to job j are of "color" j. Further, imagine the crowd-workers as bins. Then, one can see a one to one correspondence between task assignment in the PULL model and throwing of sets of colored balls into the bins at random.



(a) $m = 5, \beta = 0.4$, varying k (b) $m = 5, k = 5$, varying β (c) $k = 5, \beta = 0.4$, varying m

Figure 3. PUSH experiments: The number of nodes (workers) removed, $n = 100$, $f = 100$ on a Watts-Strogatz collusion network.

We will use the fact that the balls and bins process is well approximated by a Poisson process to prove our results; it is well known that after throwing m balls into n bins, the distribution of the number of balls in a given bin is approximately Poisson with mean $\frac{m}{n}$. This analogy can be carried further and the *joint* distribution of the number of balls in all bins is well approximated by a process where the load at each bin is an independent Poisson random variable with mean $\frac{m}{n}$. Formally, let $X_1^m, \ldots, X_n^m$ denote the number of balls in the i^{th} bin after throwing m balls, and let $Y_1^m, \ldots, Y_n^m$ be independent Poisson random variables with mean $\frac{m}{n}$. Adapting an argument by Gonnet [13], one can reconstruct the following result from [1]:

THEOREM 1 (THEOREM FROM [1]). *Consider a non-negative function $f : \mathbb{Z}_+^n \to \mathbb{R}_+$. Then, $E[f(X_1, \ldots, X_n)] \leq \sqrt{2\pi en} E[f(Y_1, \ldots, Y_n)]$. Further, if $E[f(X_1, \ldots, X_n)]$ is either monotonically increasing or monotonically decreasing with m. $E[f(X_1, \ldots, X_n)] \leq 4E[f(Y_1, \ldots, Y_n)]$.*

In particular, it is often convenient to consider a function that is a 0/1 event which depends on the load of each bin.

DEFINITION 1. *Load Based Event: A* load based event *is an indicator function $L : \mathbb{Z}_+^n \to \{0, 1\}$.*

Then, the Corollary below follows from Theorem 1.

COROLLARY 1. *Let L be a load based event. Then $\Pr[L(Y_1^m, \ldots, Y_n^m)] \leq p \implies \Pr[L(X_1^m, \ldots, X_n^m)] \leq 5\sqrt{n}p$. Moreover, if the probability of the event is monotonically increasing or decreasing with m, then $\Pr[L(X_1^m, \ldots, X_n^m)] \leq 4p$.*

Simple Information Function $\mathcal{I}_s$

We consider the simple function $\mathcal{I}_s$ to show how Theorem 1 and Corollary 1 can be useful in getting bounds on information functions. To illustrate this point, we assume all workers arrive an equal number of times, although the proof can be extended as long as no worker arrives more than $\log(n)$ times using standard techniques from the balls and bins literature.

THEOREM 2. *The function $\mathcal{I}_s$ defined above is less than $10nf\left[1 - e^{-m/n}\left(1 + \frac{m}{n}\right)\right]$ with high probability when $m \leq n$.*[3]

[3]We believe this assumption to be reasonable since we would need at least as many workers as parts in order to ensure no collisions.

PROOF. Let $L(X_i^m)$ be the load based event for the event that two or more balls of a fixed color c fall into bin i. Note that it is monotonic in m. Hence, we can approximate it via a Poisson process to get a bound on $\mathrm{E}[\mathcal{I}_s]$:

$$\mathrm{E}[\mathcal{I}_s] = \mathrm{E}\left[\sum_{\text{worker i}} \sum_{\text{task j}} 1_{i \text{ has gem from } j}\right] \quad (6)$$

$$= \mathrm{E}\left[\sum_{\text{bin i}} \sum_{\text{color c}} L(X_i^m)\right] \quad (7)$$

$$\leq 4\mathrm{E}\left[\sum_{\text{bin i}} \sum_{\text{color c}} L(Y_i^m)\right] = 4nf\mathrm{E}[L(Y_i^m)]. \quad (8)$$

Since Y_i^m is a Poisson random variable with mean $\frac{m}{n}$ and $\mathrm{E}[L(Y_i^m)]$ is simply the probability that Y_i^m is greater than or equal to 2, we get that

$$\mathrm{E}[\mathcal{I}_s] \leq 4nf\left[1 - e^{-m/n}\left(1 + \frac{m}{n}\right)\right]. \quad (9)$$

Note that in many practical settings $m \ll n$ (we expect m to be around 10 for a task, while the number of workers is conservatively in the 10000s), so $\mathrm{E}[\mathcal{I}_s] \approx 5mf$, which, interestingly, does not depend on n.

A similar argument using Corollary 1 shows that $\mathcal{I}_s \leq 2\mathrm{E}[\mathcal{I}_s]$ with high probability, giving the desired bound. □

Reliable Information Ratio $\mathcal{I}_{RIR}$

We first bound $\mathcal{I}_{RIR}$ for a generic probability p of a true positive and probability q of a false positive.

THEOREM 3. *With high probability, $\mathcal{I}_{RIR} \leq 10m\sqrt{n} \cdot \frac{p}{q}$.*

The proof of this theorem is similar in structure to the proof of Theorem 2 and is omitted due to space constraints.

Application to Digitisation Tasks

Recall the form digitisation task we introduced earlier. Note that in this setting, handwriting similarity can make it more difficult for a worker to distinguish between forms.[4] Hence, we can cluster tasks in a way that increases q, the probability

[4]There is a vast literature on handwriting similarity recognition.

of a false positive, by presenting components of tasks with similar handwriting to the same worker. Combining this with some additional randomisation techniques, we get the following Corollary of Theorem 3:

COROLLARY 2. *If $q \geq \frac{10m}{\lambda\sqrt{n}}p$, then $\mathcal{I}_{RIR} \leq \lambda n$ with high probability.*

Note that this means that the *average expected utility* for a worker is negative since $\mathcal{I}^i_{RIR} \leq \lambda$. Hence, we would like to *boost* q in order to cross this bound. A similar boosting approach could be used with any type of task, as long as a relevant similarity metric can be defined. In audio, this might be the pitch and tone of speech; in images this could be the clarity or colour palette.

PUSH Assignment Model

With regards to the amount of control the requester gets, the PUSH assignment setting is at the opposite end of the spectrum compared to PULL. We have the maximum amount of knowledge about the workers and control over the assignment. This is closer to the standard employment model where we have a fixed number of workers, and know their characteristics and networks, and can simply assign tasks at will.

We assume we are given an undirected collusion graph G' with a vertex for each crowdworker, and an edge (u, v) meaning that u and v know each other and may collude. We create the graph $G' + G'^2$. This is necessary since a node u in G' might have access to tasks from two of her neighbours v and w and may get a gem if both of them are assigned components from the same task. Now, we lose a *gem* if two or more components from the same task are assigned to a single crowdworker *or*, in the collusion case, to two neighbouring crowdworkers. Informally, our assignment algorithm ensures no gems are lost as long as the the degree of the graph is small relative to the number of workers.

THEOREM 4. *Given a collusion graph G' and $G = G' + G'^2$, assume $n \geq m(\Delta+1)$. Then there exists an algorithm that assigns components to crowd workers in linear time such that $\mathcal{I}_s = \mathcal{I}_{RIR} = 0$.*

Note that, if there is no collusion, then $n \geq m$ suffices.[5]

Note that an edge uv in this graph implies that we should not assign two components of the same task to $\{u, v\}$ since either we would assign to the same node, or the two nodes can share information. However, if uv is not an edge, we can safely assign to these vertices. Hence, given a proper coloring of G, if we assign the components of a task to vertices of the same color, we know that no information is leaked on that task.[6]

Algorithm 1 Graph coloring

Coloring: Color the graph with $\Delta + 1$ colors.
Pruning: Let $C(m)$ be the set of colors that have been used fewer than m times. $\forall\ c \in C(m)$, remove all vertices with color c from the graph and remove c as a color.

[5]In most settings, m is a constant and $\Delta \in o(n)$ so this condition is satisfied.
[6]Note that we have switched from PULL and are now coloring *workers*, whereas earlier we were coloring *tasks*.

Algorithm 1 is a simple algorithm that performs optimally for large enough graphs.

LEMMA 1. *Algorithm 1 removes at most $(m-1)(\Delta+1)$ vertices, and produces a proper coloring of the pruned graph where at least one color has $\geq m$ vertices.*

PROOF. After the coloring step, let $k_i = |C_i| \mod m$ where C_i is the set of vertices of color i. Hence, $\sum_{i=1}^{\Delta+1} k_i \leq (m-1)(\Delta+1)$. This means we have cut out at most $(m-1)(\Delta+1)$ vertices. Since $n \geq m(\Delta+1)$, there exists at least one color with at least m vertices. □

We use Algorithm 2 to assign tasks given such a coloring. Since for every group $s_i > m$, no worker gets two components from the same task. Once a group has been completely assigned, we have also assigned tasks to all workers in the corresponding group. Now, we continue the same process with the next group and its tasks and so on until all the tasks are assigned.

Algorithm 2 Distributing Tasks

Let s_i be the number of nodes of color i; $\{J_1, J_2, \ldots, J_f\}$ be the set of tasks with components $j'_{i,1}, j'_{i,2}, \ldots, j'_{i,m}$ each.

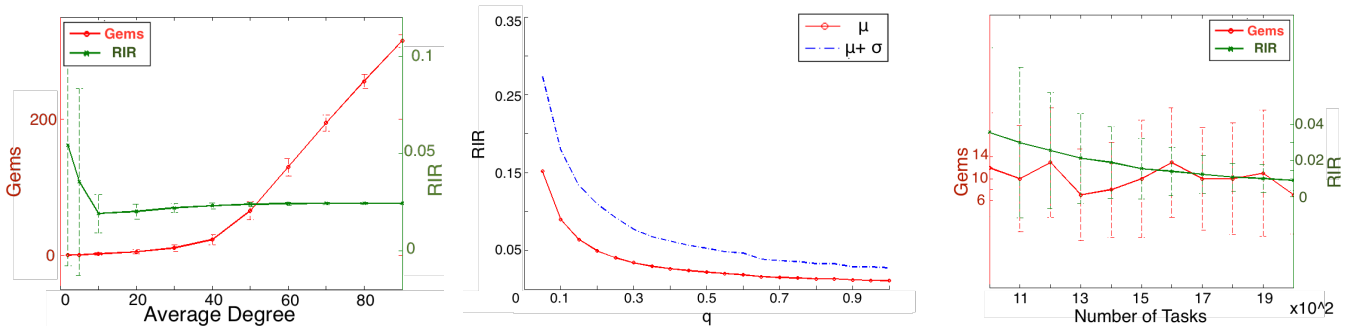
1. **Divide tasks:** Divide the tasks into sets proportional to the size of each group: to color 1 give tasks $\{J_1, J_2, \ldots, J_{fs_1/n}\}$, to color 2 give $\{J_{1+fs_1/n}, \ldots, J_{f*(s_1+s_2)/n}\}$, and so on.
2. **Assign tasks:** Now, within the tasks assigned to a group (e.g., group 1 is $\{J_1, J_2, \ldots, J_{fs_1/n}\}$) create an ordered list of components of tasks:
$$j'_{1,1}, j'_{2,1}, \ldots, j'_{fs_1/n,1}$$
$$\vdots$$
$$j'_{1,m}, j'_{2,m}, \ldots, j'_{fs_1/n,m}$$
In other words, the first components of all the tasks in the partition, followed by the second components of all the tasks and so on.
3. Assign the first $\frac{fm}{n}$ components in the above ordering to the first worker of group 1, the next $\frac{fm}{n}$ to the second worker of the group and so on.

The proof of Theorem 4 follows as we are assigning the m different components of a task to m workers of the same color. Since we take a greedy coloring, and straightforwardly distribute tasks it is clear the process is linear in the size of the inputs.

REMARK 2. *In our empirical analysis, we see that the dependence seems to be on the* average *rather than* maximum *degree.*

Worker Capacities

The most generous assumption we made in the PUSH model is that we can assign tasks at will. Though this can be done in a balanced manner, we now consider the case where workers have (potentially) different known capacities which bound the number of components they are able or willing to complete.



(a) $f = 100$, $q = 0.3$, varying k (b) $f = 100$, $k = 5$, varying q (c) $k = 5$, $q = 0.3$, varying f

Figure 4. TOW experiments: The number of gems discovered and $\mathcal{I}_r$, $n = 100$, $m = 5$, $\beta = 0.4$, $p = 0.9$ on a Watts-Strogatz collusion network.

For a worker u, let c_u be her capacity. We begin with an extension of the above coloring algorithm to get the following theorem.

THEOREM 5. *Given a collusion graph G, if $n \geq mf(\Delta+1)$, there exists a linear-time algorithm that assigns components to crowd workers such that $\mathcal{I}_1 = \mathcal{I}_2 = \mathcal{I}_{RIR} = 0$.*

The proof is straightforward and omitted due to space constraints. A problem with the above approach is that it seems that we are not taking advantage of vertices with very high capacity, and hence require more vertices overall. However, there are two issues with trying to achieve a better bound; 1) there must be *enough* high capacity vertices, and 2) they must not be clustered in the graph in a way that we cannot group at least m of them into the same color. Hence, both the distribution of capacities and the graph structure come into play and we can construct examples which show this bound is tight. Our empirical results suggest, however, that we can dramatically outperform this bound.

TOW Assignment Model

We now consider the intermediate case where the graph and the capacities are a priori unknown. Using information about crowdworkers has been suggested in some works to improve work quality, and we propose a platform in which the workers' social network (or information from which a collusion graph can be generated) is available. As in the PULL model, workers arrive in an online manner, potentially multiple times. However, this time we know their edges when they arrive, and we can use this information to assign a task. While this gives us more control over the assignment, we also weaken our assumptions on the arrival of workers and allow it to be arbitrary (rather than uniform).

Without loss of generality, we assume a worker arrives and asks for exactly one job; a worker who wants more than one job can arrive again after their initial job is completed.[7] We also make the *worst case* assumption that workers continue to arrive until all tasks are completed; as we will see in the proof, in this case fewer assignments means fewer collisions.

[7]This process is similar to the process on Amazon Mechanical Turk and many other platforms

THEOREM 6. *Consider a collusion graph G', and $G = G' + G'^2$ as before. Let $C = \max_u c_u$ and assume workers continue to arrive until all jobs are completed. Then there exists an algorithm that assigns components to crowd workers such $\mathcal{I}_1 \leq C(\Delta+1)$, $\mathcal{I}_s \leq mC(\Delta+1)$ and $\mathcal{I}_{RIR} \leq m^2C\Delta \cdot \frac{p}{q}$ and runs in linear time.*

PROOF. Recall that when a worker arrives, her edges are known. For each worker we maintain an *array* of colors of size equal to the number of arrivals. Maintain an online multi-coloring as workers arrive such that no two entries in the array are of the same color. Moreover, a valid coloring is one in which the intersection of the colors at the two endpoints of an edge is empty.

We will multi-color the vertices greedily as they arrive. Let C be the maximum number of arrivals of a single vertex (i.e., the maximum number of tasks assigned to a single worker). Hence, we may require up to $C(\Delta+1)$ colors. For each active color (i.e., color which has been used at least once), we let $\mathcal{J}_a$ be the *current* job being assigned to vertices with color a. Whenever a vertex arrives and is colored with a, the next component of $\mathcal{J}_a$ is given to that vertex. If $\mathcal{J}_a$ finishes, we set a new previously unassigned job as $\mathcal{J}_a$.

Given color a, we will always have a component in $\mathcal{J}_a$ to assign *unless* we have run out of jobs. Consider the case where we have no more jobs. When a vertex arrives, if possible, we assign to it a color a which has components remaining in $\mathcal{J}_a$. In the case that no such color is available, we must still give it a component. Note that this will create a collision (otherwise we could have colored the vertex with the color of that component to begin with). In the worst case, this can occur for up to $C\Delta$ colors, where each has all m components of the same job. The bounds on $\mathcal{I}_1$ and $\mathcal{I}_s$ are just the number of such violations/components.

To bound $\mathcal{I}_{RIR}$, we observe that the worst case is when a vertex arrives for the first time, all Δ (at most) of its neighbors have arrived C times already and are colored identically; summing over all vertices gives us the bound on $\mathcal{I}_{RIR}$. □

Importantly, in Theorem 6 we give worst case bounds. As in Remark 3, we can often take $C = 1$. Moreover, if we consider the random worker arrival model as in PULL, we

regain those bounds when $f = C\Delta$, hence the probability of collision is very small and once again $\mathcal{I}_s \approx \mathcal{I}_{RIR} \approx 0$. We consider this random arrival model in our empirical results, and indeed show the performance is indeed improved.

REMARK 3. *In many cases, as with Amazon Mechanical Turk (AMT), the requester is allowed to* cap *the number of times a worker is allowed to complete a hit. In such a setting,* $\mathcal{I}_s = \mathcal{I}_{RIR} = 0$. *Note that this is not an improvement on Theorem 4 since here we assume that workers continue to arrive as long as we have tasks available, so effectively,* $n = \infty$ *and the conditions required trivially hold.*[8]

EXPERIMENTAL RESULTS

We first evaluate our task assignment algorithms with a real world network as the underlying collusion network. We assume the worst case, that all nodes with a social connection *will* collude. Nonetheless, we obtain favourable results which are often considerably better than theoretical bounds. Further, to evaluate the variation of results as we change the characteristics of the collusion network, we use the Watts-Strogatz small world network model. Note that each simulation was run 50 times and graphs have error bars or show both the mean (μ) and mean plus variance ($\sigma + \mu$).

Assumptions on Parameters

Our theoretical results are phrased in terms of parameters of the collusion graph, workers, and tasks; we briefly discuss our assumptions here. For TOW, we need not make any assumptions on the parameters. However, the information loss turns out to be a function of m, Δ, and C. Again, we expect C to be small; in fact, C can often be controlled directly by the requester (e.g., on Amazon Mechanical Turk) by setting a maximum number of tasks a worker is allowed to take. For PUSH and PULL we require that $m \ll \sqrt{n}$ and $\Delta \ll \sqrt{n}$.

Worst Case Graphs

While we expect the above assumptions to hold in practice, we briefly consider worst-case graphs; in particular graphs with high maximum degree Δ. Such graphs could force us to lose many gems, and our theoretical guarantees for certain kinds of information loss ($\mathcal{I}_1$ and $\mathcal{I}_s$), would not hold, although others ($\mathcal{I}_{RIR}$) would actually improve. Note, however, that an edge assumes collusion, meaning bidirectional sharing of information. Unidirectional sharing (for example, by posting tasks on a public forum) does not suffice for collusion since the user that posted their tasks is not given *additional* information; instead, an (overestimate) of colluding edges could be drawn by making all workers who publicly post tasks into a clique. While this could lead to a higher maximum degree than what would be expected in a social network, at least $\Omega(\sqrt{n})$ workers would have to post their tasks before it violates our assumptions.[9] Note that graphs with high localised connectivity, e.g., one with many communities that are highly interconnected within the community but sparsely connected across communities, do not pose a problem, nor do graphs with cliques (as long as the cliques are of size $o(\sqrt{n})$). Indeed, our synthetic experiments use Watts-Strogatz graphs, which are clique-heavy. [10]

Real World Graph

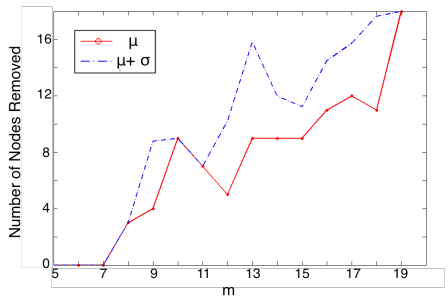


Figure 5. No. of nodes removed in PUSH, with a 1 hop collusion network.

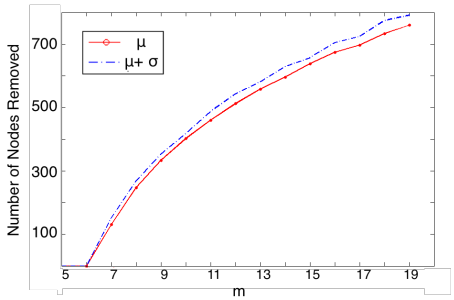


Figure 6. No. of nodes removed in PUSH, with a 2 hop collusion network.

We first test our algorithms for PUSH and TOW using the social network of an online community for students at UC Irvine (from [38]). We assume that two nodes are connected if they have sent a message to each other. The resulting graph has an average degree of 14, and maximum degree of 255. For PUSH, we proved that we will need to remove at most $(m-1)\Delta$ nodes where Δ is the max degree of the graph used in the colouring. In Figure 5 we see a relation with the average degree instead of the maximum degree and in Figure 6, we notice that the variation is with the square of the average degree. Hence, for social network like graphs, our algorithm performs much better than the bounds we established.

For TOW, we assumed a 2 hop collusion network and achieved almost the same results as in the 1 hop case. We observe that as the number of tasks is varied, as expected from our theoretical results, the number of gems compromised remains constant, i.e., independent of the number of tasks; this is a stark contrast to PULL[11], in which the number of gems compromised increases linearly with the number

[8] An alternate practical strategy on AMT-like platforms is to simply *remove* the tasks once fewer than $C\Delta$ remain, and re-post them with the next batch. This would reduce the information loss to 0.

[9] An alternative approach would be to collapse a large clique into an "effective" node (at the expense of n), which, in some cases, would once again satisfy the requirements of our algorithms.

[10] If cliques or strongly knit communities are known and causing a problem with parameters, they could be compressed into a single node for assignment purposes.

[11] Note that this is equivalent to comparing with a random assignment of tasks to workers conditioned on the same set of workers arriving in PULL and TOW.

of tasks (see Figure 7). Moreover, we observe that $\mathcal{I}_{RIR}$ for TOW decreases as we increase the number of tasks (see Figure 8).

Comparing the performance of our assignment with random assignment, we see that there is considerable improvement and the number of gems compromised in random assignment increases with the number of tasks, but is nearly constant in TOW.

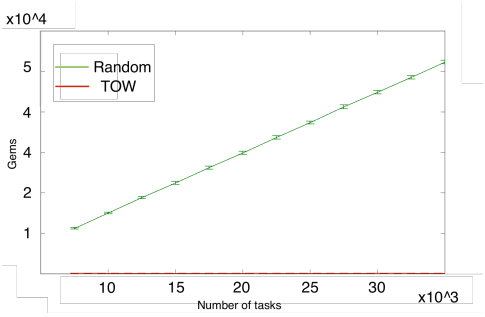


Figure 7. Gems compromised in PULL vs TOW.

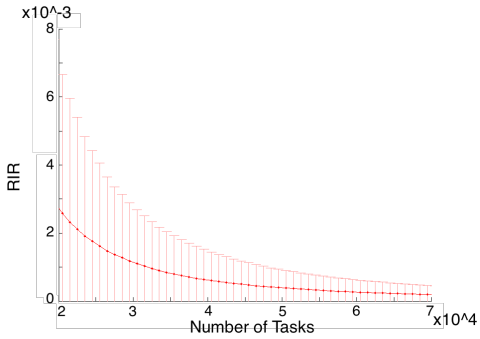


Figure 8. RIR in TOW, with m=5.

Synthetic Experiments

The well-studied Watts-Strogatz small world network model [49] has been shown to have many properties of real world social networks. Hence, we used this model to create graphs with different characteristics so as to see the effect of graph structure on our algorithms

DEFINITION 2. *$G_{WS}(n,k,\beta)$ Define a* Watts-Strogatz *random graph $G_{WS}(n,k,\beta) = G(V,E)$ as follows: Let $|V| = n$. Label the vertices $0,\ldots,n-1$, and let $(i,j) \in E$ if and only if $i \in [j-k/2, j+k/2] \mod n$. Now, select and randomly rewire each edge independently with probability β.*

n	number of nodes in the network
f	number of tasks to be completed
m	number of components in a task
k	average degree in the G_{WS}
β	probability of rerouting an edge in G_{WS}
p	probability of getting a true positive
q	probability of getting a false positive

Table 2. Parameters

PULL

Though the number of gems revealed $\mathcal{I}_1$ can be high, the Reliable Information ratio is very low (see Figure 2). $\mathcal{I}_1$ increases sub-linearly with respect to k and linearly with respect to f, whereas $\mathcal{I}_{RIR}$ decreases exponentially. While this may seem surprising, note that the more tasks in the system, the higher the load of any given worker; in particular, this means the variance in the system disappears so all workers have relatively high load and hence high noise.

PUSH

While we proved theoretically that at most $(m-1)(\Delta+1)$ would need to be removed for a perfect coloring, we see a dependence on the *average degree*, i.e., k in G_{WS} as well as on m. In particular, the number is independent of f. The number of nodes removed varies almost linearly with both m and k but is constant when varying Δ while maintaining the average degree. We accomplish this by varying β, which does not affect the average degree, but changes Δ. Also, the required number of nodes is considerably lower than our worst case bound. In Figure 3 c) the worst case bound is $\sim 8m$, while one standard deviation away from the mean is $\sim 1.7m$.

TOW

As we would expect, the number of gems which can be leaked increases with the degree of the graph since more people can collude. However, *RIR* quickly reaches a value less than 0.025 (Fig 4). Hence, most leaks are not effective. While perhaps initially counterintuitive, this arises from the fact that the number of non-compromised tasks given to each worker is increasing and adding noise to the system. However, a practical value of k may be low. Hence we also vary the number of jobs distributed for a low value k. $\mathcal{I}_{RIR}$ remains low in this case, and importantly, $\mathcal{I}_1$ does not depend on the number of tasks. Also, the number of gems compromised in a random assignment is over a 100 times more. Hence, though we cannot achieve 0 information loss as in PUSH, it is a significant improvement over PULL since it depends only on the network structure and not on the amount of work.

CONCLUSION AND FUTURE WORK

We formalised the concept of privacy in crowdsourcing and presented provable guarantees on information loss while crowdsourcing jobs, which could open up crowdsourcing to a myriad of relatively sensitive applications including but not limited to healthcare data and crowdsourcing on user generated or personal content.

We believe that a further important contribution of this work is introducing the hybrid TOW assignment setting where workers *PULL* a task but requesters can *PUSH* the specific instantiation of the task to the worker. This type of assignment mechanism, though not used in practice, maintains the benefits of an on-demand work force while allowing requesters the flexibility of assigning tasks in a manner which maintains privacy. In fact, the applications of such a setting are not limited to guaranteeing privacy but could be used to provide quality assurances, specific demographics, and other properties requesters could desire.

REFERENCES

1. Micah Adler, Soumen Chakrabarti, Michael Mitzenmacher, and Lars Rasmussen. 1998. Parallel Randomized Load Balancing. *Random Structures and Algorithms* (1998), 159–188.
2. Chithralekha Balamurugan, Shourya Roy, and Sujit Gujar. 2013. Methods and systems for creating tasks of digitizing electronic document. (May 29 2013). US Patent App. 13/904,319.
3. Chithralekha Balamurugan, Shourya Roy, Jacki O'neill, and Sujit Gujar. 2014. Method and system for a text data entry from an electronic document. (Oct. 21 2014). US Patent 8,867,838.
4. David Blumenthal. 2010. Launching HIteCH. *New England Journal of Medicine* 362, 5 (2010), 382–385.
5. Alessandro Bozzon, Marco Brambilla, Stefano Ceri, Matteo Silvestri, and Giuliano Vesci. 2013. Choosing the right crowd: expert finding in social networks. In *Proceedings of the 16th International Conference on Extending Database Technology*. ACM, 637–648.
6. Jonathan Bragg, Daniel S Weld, and others. 2013. Crowdsourcing multi-label classification for taxonomy creation. In *First AAAI conference on human computation and crowdsourcing*.
7. Bo Brinkman. 2013. An analysis of student privacy rights in the use of plagiarism detection systems. *Science and engineering ethics* 19, 3 (2013), 1255–1266.
8. Kuang Chen, Akshay Kannan, Yoriyasu Yano, Joseph M Hellerstein, and Tapan S Parikh. 2012. Shreddr: pipelined paper digitization for low-resource organizations. In *Proceedings of the 2nd ACM Symposium on Computing for Development*. ACM, 3.
9. Djellel Eddine Difallah, Gianluca Demartini, and Philippe Cudré-Mauroux. 2013. Pick-A-Crowd: Tell Me What You Like, and Ill Tell You What to Do. In *Proceedings of the 22nd international conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 367–374.
10. Whitfield Diffie and Martin E Hellman. 1976. New directions in cryptography. *Information Theory, IEEE Transactions on* 22, 6 (1976), 644–654.
11. Noah E Friedkin. 1983. Horizons of observability and limits of informal control in organizations. *Social Forces* 62, 1 (1983), 54–77.
12. Gagan Goel, Afshin Nikzad, and Adish Singla. 2014. Allocating tasks to workers with matching constraints: truthful mechanisms for crowdsourcing markets. In *Proceedings of the companion publication of the 23rd international conference on World wide web companion*. International World Wide Web Conferences Steering Committee, 279–280.
13. Gaston H Gonnet. 1981. Expected length of the longest probe sequence in hash code searching. *Journal of the ACM (JACM)* 28, 2 (1981), 289–304.
14. Mitchell Gordon, Walter S Lasecki, Winnie Leung, Ellen Lim, Steven P Dow, and Jeffrey P Bigham. 2014. Glance Privacy: Obfuscating Personal Identity While Coding Behavioral Video. In *Second AAAI Conference on Human Computation and Crowdsourcing*.
15. Ralph Gross and Alessandro Acquisti. 2005. Information revelation and privacy in online social networks. In *Proceedings of the 2005 ACM workshop on Privacy in the electronic society*. ACM, 71–80.
16. Christopher G Harris. 2011. Dirty deeds done dirt cheap: a darker side to crowdsourcing. In *Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third International Conference on Social Computing (SocialCom)*. IEEE, 1314–1317.
17. Kashmir Hill and Zack O'Malley Greenburg. 2010. *The Black Market Price of Your Personal Info*. Forbes Magazine. http://www.forbes.com/2010/11/29/black-market-price-of-your-info-personal-finance.html
18. Jeff Howe. 2008. *Crowdsourcing: How the power of the crowd is driving the future of business*. Wired Magazine, Random House.
19. Srikanth Jagabathula, Lakshminarayanan Subramanian, and Ashwin Venkataraman. 2014. Reputation-based worker filtering in crowdsourcing. In *Advances in Neural Information Processing Systems*. 2492–2500.
20. Hiroshi Kajino, Yukino Baba, and Hisashi Kashima. 2014. Instance-Privacy Preserving Crowdsourcing. In *Second AAAI Conference on Human Computation and Crowdsourcing*.
21. Ravi Kannan, Santosh Vempala, and Adrian Vetta. 2004. On clusterings: Good, bad and spectral. *Journal of the ACM (JACM)* 51, 3 (2004), 497–515.
22. Ehud D Karnin, Eugene Walach, and Tal Drory. 2010. *Crowdsourcing in the document processing practice*. Springer.
23. Roman Khazankin, Harald Psaier, Daniel Schall, and Schahram Dustdar. 2011. Qos-based task scheduling in crowdsourcing environments. In *Service-Oriented Computing*. Springer, 297–311.
24. Ashiqur R KhudaBukhsh, Jaime G Carbonell, and Peter J Jansen. 2014. Detecting Non-Adversarial Collusion in Crowdsourcing. In *Second AAAI Conference on Human Computation and Crowdsourcing*.
25. Aniket Kittur, Jeffrey V Nickerson, Michael Bernstein, Elizabeth Gerber, Aaron Shaw, John Zimmerman, Matt Lease, and John Horton. 2013. The future of crowd work. In *Proceedings of the 2013 conference on Computer supported cooperative work*. ACM, 1301–1318.

26. Aniket Kittur, Boris Smus, Susheel Khamkar, and Robert E Kraut. 2011. Crowdforge: Crowdsourcing complex work. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM, 43–52.

27. Nicolas Kokkalis, Thomas Köhn, Carl Pfeiffer, Dima Chornyi, Michael S Bernstein, and Scott R Klemmer. 2013. EmailValet: Managing email overload through private, accountable crowdsourcing. In *Proceedings of the 2013 conference on Computer supported cooperative work*. ACM, 1291–1300.

28. Walter Lasecki, Christopher Miller, Adam Sadilek, Andrew Abumoussa, Donato Borrello, Raja Kushalnagar, and Jeffrey Bigham. 2012. Real-time captioning by groups of non-experts. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*. ACM, 23–34.

29. Walter S Lasecki, Mitchell Gordon, Jaime Teevan, Ece Kamar, and Jeffrey P Bigham. 2015. Preserving Privacy in Crowd-Powered Systems. (2015).

30. Walter S Lasecki, Jaime Teevan, and Ece Kamar. 2014. Information extraction and manipulation threats in crowd-powered systems. In *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*. ACM, 248–256.

31. Matthew Lease, Jessica Hullman, Jeffrey P Bigham, Michael Bernstein, Juho Kim, Walter Lasecki, Saeideh Bakhshi, Tanushree Mitra, and Robert C Miller. 2013. Mechanical turk is not anonymous. *Social Science Research Network* (2013).

32. Greg Little and Yu-An Sun. 2011. Human OCR: Insights from a complex human computation process. In *Workshop on Crowdsourcing and Human Computation, Services, Studies and Platforms, ACM CHI*. Citeseer.

33. R Manmatha, Chengfeng Han, Edward M Riseman, and W Bruce Croft. 1996. Indexing handwriting using word matching. In *Proceedings of the first ACM international conference on Digital libraries*. ACM, 151–159.

34. Jon Noronha, Eric Hysen, Haoqi Zhang, and Krzysztof Z Gajos. 2011. Platemate: crowdsourcing nutritional analysis from food photographs. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM, 1–12.

35. Ali Nosary, Laurent Heutte, Thierry Paquet, and Yves Lecourtier. 1999. Defining writer's invariants to adapt the recognition task. In *Document Analysis and Recognition, 1999. ICDAR'99. Proceedings of the Fifth International Conference on*. IEEE, 765–768.

36. U.S. Department of Health & Human Services. 2000. *Summary of the HIPPA Privacy Rule*. http://www.hhs.gov/ocr/privacy/hipaa/understanding/summary/

37. Jacki O'Neill, Shourya Roy, Antonietta Grasso, and David Martin. 2013. Form digitization in BPO: from outsourcing to crowdsourcing?. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 197–206.

38. Tore Opsahl and Pietro Panzarasa. 2009. Clustering in weighted networks. *Social networks* 31, 2 (2009), 155–163.

39. Tony M Rath and Rudrapatna Manmatha. 2007. Word spotting for historical documents. *International Journal of Document Analysis and Recognition (IJDAR)* 9, 2-4 (2007), 139–152.

40. Theodoros Rekatsinas, Amol Deshpande, and Ashwin Machanavajjhala. 2013. SPARSI: Partitioning Sensitive Data Amongst Multiple Adversaries. *Proc. VLDB Endow.* 6, 13 (Aug. 2013), 1594–1605. DOI: http://dx.doi.org/10.14778/2536258.2536270

41. Pierangela Samarati and Latanya Sweeney. 1998. Generalizing data to provide anonymity when disclosing information. In *PODS*, Vol. 98. 188.

42. Cristina Sarasua and Matthias Thimm. 2013. Microtask available, send us your CV!. In *Cloud and Green Computing (CGC), 2013 Third International Conference on*. IEEE, 521–524.

43. Benjamin Satzger, Harald Psaier, Daniel Schall, and Schahram Dustdar. 2013. Auction-based crowdsourcing supporting skill management. *Information Systems* 38, 4 (2013), 547–560.

44. Imran Ahmed Siddiqi and Nicole Vincent. 2007. Writer identification in handwritten documents. In *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*, Vol. 1. IEEE, 108–112.

45. Lav R Varshney. 2012. Privacy and reliability in crowdsourcing service delivery. In *SRII Global Conference (SRII), 2012 Annual*. IEEE, 55–60.

46. Lav R Varshney, Aditya Vempaty, and Pramod K Varshney. 2014. Assuring privacy and reliability in crowdsourcing with coding. In *Information Theory and Applications Workshop (ITA), 2014*. IEEE, 1–6.

47. Louis Vuurpijl and Lambert Schomaker. 1996. Coarse writing-style clustering based on simple stroke-related features. *Progress in Handwriting Recognition* (1996), 37–44.

48. Gang Wang, Tianyi Wang, Haitao Zheng, and Ben Y Zhao. 2014. Man vs. machine: Practical adversarial detection of malicious crowdsourcing workers. In *23rd USENIX Security Symposium, USENIX Association, CA*.

49. Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of small-worldnetworks. *nature* 393, 6684 (1998), 440–442.

50. Stephen M Wolfson and Matthew Lease. 2011. Look before you leap: legal pitfalls of crowdsourcing. *Proceedings of the American Society for Information Science and Technology* 48, 1 (2011), 1–10.

51. Sai Wu, Xiaoli Wang, Sheng Wang, Zhenjie Zhang, and Anthony KH Tung. 2014. K-anonymity for crowdsourcing database. *Knowledge and Data Engineering, IEEE Transactions on* 26, 9 (2014), 2207–2221.