Parallel and Streaming Truth Discovery in Large-Scale Quantitative Crowdsourcing

Robin Wentao Ouyang, Lance M. Kaplan, Alice Toniolo, Mani Srivastava, and Timothy J. Norman

Abstract—To enable reliable crowdsourcing applications, it is of great importance to develop algorithms that can automatically discover the truths from possibly noisy and conflicting claims provided by various information sources. In order to handle crowdsourcing applications involving big or streaming data, a desirable truth discovery algorithm should not only be *effective*, but also be *scalable*. However, with respect to quantitative crowdsourcing applications such as object counting and percentage annotation, existing truth discovery algorithms are not simultaneously effective and scalable. They either address truth discovery in categorical crowdsourcing or perform batch processing that does not scale. In this paper, we propose new parallel and streaming truth discovery algorithms for quantitative crowdsourcing applications. Through extensive experiments on real-world and synthetic datasets, we demonstrate that 1) both of them are quite effective, 2) the parallel algorithm can efficiently perform truth discovery on large datasets, and 3) the streaming algorithm processes data incrementally, and can efficiently perform truth discovery both on large datasets and in data streams.

Index Terms—Crowdsourcing, truth discovery, quantitative task, big data, parallel algorithm, streaming algorithm

1 Introduction

Crowdsourcing is a process of obtaining needed content, information, or services by soliciting contributions from a large group of usually undefined people, rather than from traditional employees or suppliers. It is becoming increasingly popular as it provides an easy, time-, and cost-efficient way to collect a large volume of data for a variety of applications, such as image labeling, image description, sentiment analysis, listing verification, object counting, translation, and logo design [1]–[5].

A typical crowdsourcing application involves a set of crowd participants u_i and a set of targets z_j (illustrated in Fig. 1). A target could be the label of an image, the translation of a sentence, or the number of objects in an image. When a participant u_i works on a target z_j, she makes a claim x_{ij}. These participants, targets, and claims then form an *information network*.

As the quality of data obtained from crowd participants is often much lower than the quality of data collected from traditional employees and experts [2], it is of great importance to develop truth discovery methods that can automatically discover the true values of targets z_j from possibly conflicting and low-quality crowdsourced claims x_{ij}, thus enabling *reliable* crowdsourcing applications.

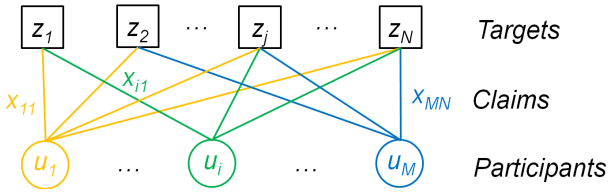


Fig. 1. Information network formed by a set of crowd participants u_i making claims x_{ij} on a set of targets z_j.

In this paper, we address the *effective* and *scalable* truth discovery problem in quantitative crowdsourcing applications involving big or streaming data. This is motivated by the various quantitative applications that crowdsourcing can enable. In sensor network and computer vision applications such as occupancy inference [6] and people counting [7], a large number of labeled data are often needed to train an automatic system. For example, in order to train a sensor-based room-level occupancy inference system, the ground truth people counts from snapshots taken by a camera every 1 to 5 minutes over at least two weeks need to be annotated [6]. That is to say, 3k to 15k snapshots (excluding those at late night) per room need to be annotated. Even for a small-scale deployment with 5 rooms, one needs to annotate 15k to 75k snapshots. Crowdsourcing can thus be used to simplify and speed up such annotation processes [6].

Crowdsourcing can also be used to enable new quantitative applications. For example, eBird[1] crowdsourced the bird populations around the world [8]. It was noted that "birds are notoriously hard to count. While stationary sensors can measure things like carbon dioxide levels and highway traffic, it takes people to note the

- *R. W. Ouyang is with the Department of Computer Science, and M. Srivastava is with the Department of Electrical Engineering and the Department of Computer Science, University of California, Los Angeles, CA, USA.*
  E-mail: {wouyang, mbs}@ucla.edu
- *L. M. Kaplan is with the Networked Sensing & Fusion Branch, US Army Research Laboratory, Adelphi, MD 20783, USA.*
  E-mail: lance.m.kaplan@us.army.mil
- *A. Toniolo and T. J. Norman are with the Department of Computing Science, University of Aberdeen, Aberdeen, UK.*
  E-mail: {a.toniolo, t.j.norman}@abdn.ac.uk

1. http://ebird.org/content/ebird/

type and number of birds in an area". In May 2013, eBird gathered 5.6 million new observations from 169 countries [8]. As another example, Indonesian voters crowdsourced ballot counts to protect against election fraud in 2014 [9]. It was reported that "in just two days, the (General Elections Commissions') website has garnered 1,200 users who 'crowdcounted' over 31,000 documents" [9]. Furthermore, our previous work [10] crowdsourced the waiting line length and the occupancy level in local businesses and services, where physical sensor networks encounter the scalability problem.

However, the lack of an effective and scalable quantitative truth discovery algorithm may hinder the usefulness of these large-scale crowdsourcing applications. Most existing truth discovery methods [11]–[17] are designed for categorical or even binary truth discovery. They rely on the agreement among claims and use the rates of exactly correct claims to capture participants' abilities. These methods are thus not effective or even not applicable for quantitative crowdsourcing applications, where it is not uncommon that each participant provides a different claim for a target, and the rate of exactly correct claims is small for any participant. Moreover, due to the networked nature of crowdsourcing applications (Fig. 1), these methods perform batch truth discovery on the whole networked data, which does not scale.

We recently proposed the Truth, Bias, and Precision (TBP) model [10] for quantitative truth discovery. The TBP model uses biases and precisions to capture crowd participants' abilities in quantity estimation, and it naturally incorporates the similarity between the latent truth and each crowdsourced claim. Although the TBP model has been shown to be more effective than existing truth discovery methods for quantitative crowdsourcing [10], the original model inference algorithm is also not scalable. This is because it is a batch algorithm and the inferences of target-related and participant-related parameters are coupled together such that joint optimization must be performed. Wang et al. [18] recently proposed a recursive truth discovery method that is scalable. However, it is designed for binary truth discovery, and is not applicable to quantitative crowdsourcing.

In this paper, we propose a parallel algorithm and a streaming algorithm to handle truth discovery in large-scale quantitative crowdsourcing applications. We first develop a new batch algorithm based on the TBP model. This algorithm decouples the inferences of target-related and participant-related parameters, and is thus amenable to scalable truth discovery upon further modification. We then examine its structure and exploit the MapReduce framework [19] to develop a parallel algorithm that can efficiently handle big data. This algorithm decomposes the large-scale truth discovery problem into several small-scale map and reduce tasks that can be run in parallel over a cluster of processors. Finally, we exploit the on-line expectation-maximization (EM) algorithm [20] to develop a streaming algorithm, which recursively updates its parameters by processing data incrementally. It can thus efficiently perform truth discovery both on large datasets and in data streams. Through extensive experiments, we demonstrate that both the parallel and the streaming algorithms are simultaneously effective and scalable. To our best knowledge, both the MapReduce framework and the on-line EM algorithm have not been explored for enabling large-scale truth discovery in crowdsourcing applications.

In summary, our main contributions are as follows:

1) We address the effective and scalable truth discovery problem in quantitative crowdsourcing applications involving big or streaming data.
2) We propose a parallel algorithm and a streaming algorithm that are both effective and scalable.
3) We analyze the time and the space complexity of these proposed algorithms.
4) We conduct extensive experiments on both real-world and synthetic datasets to evaluate the performance of these proposed truth discovery algorithms and other state-of-the-art competitors.

The remainder of this paper is organized as follows. We formalize the problem in Section 2 and briefly review the TBP model in Section 3. We then develop new batch, parallel, and streaming truth discovery algorithms in Sections 4, 5, and 6. We analyze the time and the space complexity of these algorithms in Section 7. Experimental results are presented in Section 8. We discuss other research problems and related work in Sections 9 and 10. Finally, we conclude the paper in Section 11.

2 Problem Statement

For ease of illustration, we list the notations used in this paper in Table 1. A typical quantitative crowdsourcing task is to ask crowd participants to count the number of target quantities such as people, vehicles, and animals in images, video frames, local businesses, or other scenarios [6]–[10]. Consider a scenario where M crowd participants u_i make quantitative claims x_{ij} (e.g., 5, 12, and 20) on N target quantities z_j (e.g., the number of people in the jth image). We only observe the crowdsourced claims x_{ij}, but not the true quantity values z_j. The truth discovery problem in quantitative crowdsourcing applications is to automatically recover the true quantity values z_j from crowdsourced claims x_{ij}.

3 Review of the TBP Model

In this section, we briefly review the Truth, Bias, and Precision (TBP) model proposed in [10], summarize its properties, outline the original truth discovery algorithm, and discuss its scalability issue.

3.1 The TBP Model

The structure of the TBP model is shown in Fig. 2. The TBP model is specifically designed to tackle truth discovery in quantitative crowdsourcing applications. It jointly models the following variables: 1) a target's latent true

TABLE 1
Notations.

Notation	Meaning
M	# of crowd participants
N	# of target quantities
K	# of underlying difficulty levels
P	# of processors
X	# of claims
u_i	ith participant
z_j	true value of the jth target quantity
$\mathcal{U}_j$	set of participants who make a claim on z_j
$\mathcal{Z}_i$	set of targets that u_i makes a claim on
π_k	probability that task difficulty is in level k
r_{jk}	difficulty of estimating z_j is in level k
h_{ik}	u_i's bias in difficulty level k
λ_{ik}	u_i's precision in difficulty level k
x_{ij}	claim that u_i makes on z_j
$\mathbf{x}_j$	all the claims on z_j
μ_j, ν_j	hyperparameters for z_j
a_{ik}, b_{ik}	hyperparameters for λ_{ik}

quantity value z_j, 2) a task's underlying difficulty level r_{jk}, 3) a participant's quantity estimation bias h_{ik} and precision λ_{ik}, and 4) a participant's claim x_{ij}. Among these variables, only the claim x_{ij} is observed.

1) Latent truth. It models that the target quantity in each task has a latent true value z_j, which is generated from a Gaussian distribution as

$$p(z_j|\mu_j,\nu_j) = \mathcal{N}(z_j|\mu_j, 1/\nu_j) = \sqrt{\frac{\nu_j}{2\pi}} \exp\left[-\frac{\nu_j}{2}(z_j-\mu_j)^2\right],$$

where μ_j and ν_j are hyperparameters, encoding the prior belief on the mean and the precision (i.e., inverse of the variance) of z_j.

2) Difficulty level. It models that there are K discrete difficulty levels (such as easy, normal, and hard). This is motivated by the observations from Fig. 4 in [10], where multiple modes are observed in participants' quantity estimation errors. By modeling difficulty levels, the errors around each mode can then be well explained as caused by a specific difficulty level and can be conveniently modeled by a parametric distribution. In particular, each task draws an unobserved underlying difficulty indicator r_{jk} from a multinomial distribution as

$$p(r_{jk} = 1|\pi_k) = \pi_k,$$

where π_k is the probability that the task difficulty is in level k. r_{jk} uses 1-of-K coding, where only 1 out of K r_{jk} can be 1 and others are all 0. Using such a coding, when we write r_{jk}, we implicitly imply $r_{jk} = 1$.

3) Participant's bias and precision. It models that each participant u_i has K pairs of (bias, precision) parameters (h_{ik}, λ_{ik}) in quantity estimation. The choice of which pair to use depends on the underlying task difficulty r_{jk}. It also imposes the prior probability on each λ_{ik} as

$$p(\lambda_{ik}|a_{ik},b_{ik}) = \mathsf{Gamma}(\lambda_{ik}|a_{ik},b_{ik})$$
$$= \frac{1}{\Gamma(a_{ik})} b_{ik}^{a_{ik}} \lambda_{ik}^{a_{ik}-1} \exp(-b_{ik}\lambda_{ik}),$$

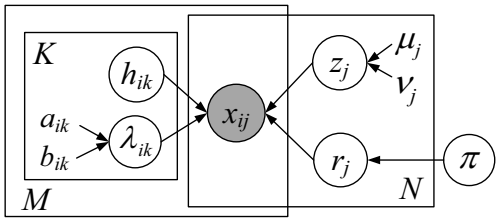


Fig. 2. Structure of the Truth, Bias, and Precision (TBP) model. Each node represents a random variable. Dark shaded nodes represent observed variables, light nodes represent latent variables and model parameters, and nodes without a circle represent hyperparameters.

where a_{ik}, b_{ik} are hyperparameters of a Gamma distribution (the conjugate prior of the precision parameter of a Gaussian distribution), encoding the prior belief on λ_{ik}.

4) Crowdsourced claim. It models the conditional probability that u_i makes a claim x_{ij} on a target z_j as

$$p(x_{ij}|z_j, r_{jk}, h_{ik}, \lambda_{ik}) = \mathcal{N}(x_{ij}|z_j + h_{ik}, 1/\lambda_{ik}). \quad (1)$$

In (1), the underlying task difficulty r_{jk} impacts which (h_{ik}, λ_{ik}) pair u_i will use to generate x_{ij}. x_{ij} is centered at $z_j + h_{ik}$ (instead of z_j in order to reflect the participant's estimation bias), and its spread is controlled by λ_{ik}.

The hyperparameters are set as follows

$$\mu_j = \check{z}_j, \nu_j = \frac{|\mathcal{U}_j|^2}{\sum_{i\in\mathcal{U}_j}(x_{ij}-\check{z}_j)^2}, a_{ik} = 1 + 10^{-4}, b_{ik} = 10^{-4}.$$

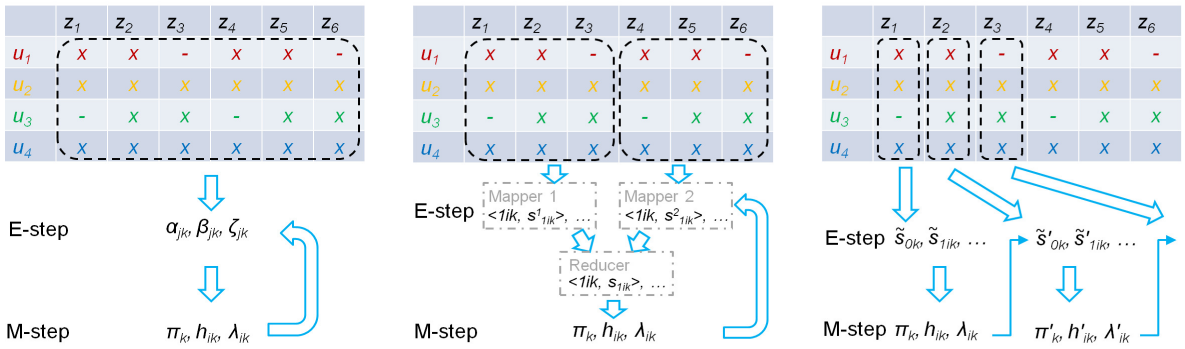
where $\check{z}_j = \text{median}_{i\in\mathcal{U}_j}(x_{ij})$ and $\mathcal{U}_j$ is the set of participants who make a claim on z_j. These hyperparameters can encode prior belief and also help alleviate model overfitting.

The number K of difficulty levels is set empirically, based on the type of tasks (implying different overall difficulties) [10]. K is set to 3 for object counting tasks, and K is set to 2 for percentage annotation tasks.

3.2 Properties

It can be shown that the TBP model holds the following properties [10] which make it more appropriate for quantitative truth discovery than existing categorical truth discovery methods [11]–[17].

1) The TBP model uses biases and precisions to capture participants' abilities in quantity estimation. In contrast, categorical truth discovery methods use the rates of exactly correct claims to capture participants' abilities. Such rates are usually very small for any participant in quantitative crowdsourcing.
2) The TBP model naturally incorporates the similarity between the latent truth and each crowdsourced claim. The more similar a claim (after bias correction) is to the latent truth, the more likely such a claim can be observed. In contrast, in most categorical truth discovery methods, the likelihood of observing a claim is the same for any claim that differs from the latent truth.



(a) Batch truth discovery (b) Parallel truth discovery (c) Streaming truth discovery

Fig. 3. Illustration of different truth discovery algorithms. x denotes a crowdsourced claim and $-$ denotes a missing value (i.e., the corresponding u does not make a claim on the corresponding z). (a) Batch truth discovery. It loads the whole dataset for processing. (b) Parallel truth discovery. It splits the whole dataset into small chunks, which are processed by map tasks in parallel. The outputs of the map tasks are then sorted and input to the reduce tasks. (c) Streaming truth discovery. It loads and processes data incrementally, and updates its parameters recursively.

3) The marginal probability of observing a claim under the TBP model is given by a Gaussian mixture model. Therefore, TBP has the ability to model the estimation errors in crowdsourced claims with complex underlying distributions, such as multi-modal, caused by varying task difficulties.

3.3 Original Truth Discovery Algorithm

Given the TBP model, the original truth discovery algorithm in [10] treats $\mathbf{R} = \{r_{jk}\}$ as latent variables, $\boldsymbol{\theta} = \{\pi_k, z_j, h_{ik}, \lambda_{ik}\}$ as model parameters, and $\mathbf{X} = \{x_{ij}\}$ as observed data. It is developed based on the Expectation Maximization (EM) algorithm [21], [22], and it iterates over an E-step and an M-step until model convergence. The converged z_j values are used as the estimated truths.

In the E-step (tth iteration), it computes the expectations of the latent variables r_{jk} as

$$\gamma_{jk}^{(t)} \equiv \mathbb{E}_{\mathbf{R}|\mathbf{X},\boldsymbol{\theta}^{(t)}}(r_{jk}) \propto \pi_k^{(t)} \prod_{i \in \mathcal{U}_j} \mathcal{N}(x_{ij}|z_j^{(t)} + h_{ik}^{(t)}, 1/\lambda_{ik}^{(t)}).$$

For each j, $\gamma_{jk}^{(t)}$ needs to be normalized over k.

In the M-step, it first computes $\pi_k^{(t+1)} = \frac{\sum_j \gamma_{jk}^{(t)}}{N}$. It then initializes z_j as $z_j^{(t)}$ and iterates over the following system of equations until convergence.

$$\begin{aligned} h_{ik} &= \frac{\sum_{j \in \mathcal{Z}_i} \gamma_{jk}^{(t)} (x_{ij} - z_j)}{\sum_{j \in \mathcal{Z}_i} \gamma_{jk}^{(t)}}, \\ \lambda_{ik} &= \frac{\frac{1}{2} \sum_{j \in \mathcal{Z}_i} \gamma_{jk}^{(t)} + a_{ik} - 1}{\frac{1}{2} \sum_{j \in \mathcal{Z}_i} \left[\gamma_{jk}^{(t)} (x_{ij} - z_j - h_{ik})^2 \right] + b_{ik}}, \\ z_j &= \frac{\nu_j \mu_j + \sum_k \left[\gamma_{jk}^{(t)} \sum_{i \in \mathcal{U}_j} (\lambda_{ik}(x_{ij} - h_{ik})) \right]}{\nu_j + \sum_k \left(\gamma_{jk}^{(t)} \sum_{i \in \mathcal{U}_j} \lambda_{ik} \right)}. \end{aligned} \quad (2)$$

The converged values h_{ik}^*, λ_{ik}^*, and z_j^* are treated as $h_{ik}^{(t+1)}$, $\lambda_{ik}^{(t+1)}$, and $z_j^{(t+1)}$.

3.4 Scalability Issue

It is observed that the original truth discovery algorithm presented in Section 3.3 is unfortunately not scalable. This is because 1) each update needs all the available data x_{ij}, and 2) the optimal h_{ik}^*, λ_{ik}^*, and z_j^* of (2) do not have closed-form solutions, but are coupled together so that joint optimization must be performed (e.g., each optimal h_{ik}^* depends on a set of unknown z_j, and each optimal z_j^* depends on a set of unknown h_{ik} and λ_{ik}).

4 Batch Truth Discovery

In this section, we develop a new batch truth discovery algorithm based on the TBP model. This algorithm decouples the inferences of target-related (z_j) and participant-related (h_{ik}, λ_{ik}) parameters. We illustrate its structure in Fig. 3(a). As a batch algorithm, it is still not scalable. However, it can be extended to handle big data upon further modification, which results in a new parallel algorithm in Section 5 and a new streaming algorithm in Section 6. We also illustrate the structures of these two algorithms in Fig. 3 in order to demonstrate the core ideas and differences of them.

4.1 Overview

The idea behind this new batch algorithm is to move the inference of z_j to the E-step, rather than in the M-step. In other words, z_j is treated as a latent variable rather than a model parameter. This is motivated by the observation from Fig. 2 that crowdsourced claims corresponding to the same target are *conditionally independent* when both the task difficulty r_{jk} and the target value z_j are given (otherwise, they are dependent).

In this new algorithm, we compute the expectations of task difficulties r_{jk}, target-related parameters z_j, and their functions in the E-step, and we only need to update participant-related parameters h_{ik} and λ_{ik} in the M-step. By doing so, the expectations of z_j and the updating rules of h_{ik} and λ_{ik} all have *closed forms*.

Algorithm 1 Batch truth discovery

Input: Crowdsourced claims $\mathbf{X} = \{x_{ij}\}$
Output: Estimated true quantity values $\hat{z}_j$

```
Initialize π_k, h_ik, λ_ik and set t = 0
while Model does not converge do
    {E-step}
    for j = 1, ..., N do
        For all k, calculate α_jk^(t), β_jk^(t), ζ_jk^(t) acc. (5)
    end for
    {M-step}
    For all k, update π_k^(t+1) acc. (8)
    for i = 1, ..., M do
        For all k, update h_ik^(t+1), λ_ik^(t+1) acc. (8)
    end for
    t ← t + 1
end while
{Truth Discovery}
Estimate ẑ_j using converged model paras. acc. (9)
return ẑ_j
```

In particular, we treat $\mathbf{Y} = \{r_{jk}, z_j\}$ as latent variables and $\boldsymbol{\vartheta} = \{\pi_k, h_{ik}, \lambda_{ik}\}$ as model parameters. We can write out the likelihood of observing crowdsourced claims $\mathbf{X} = \{x_{ij}\}$ given model parameters $\boldsymbol{\vartheta}$ as

$$\begin{aligned} p(\mathbf{X}|\boldsymbol{\vartheta}) &= \prod_j p(\mathbf{x}_j|\boldsymbol{\vartheta}) \\ &= \prod_j \int p(z_j) \sum_k \Big[p(r_{jk}|\boldsymbol{\vartheta}) \prod_{i \in \mathcal{U}_j} p(x_{ij}|z_j, r_{jk}, \boldsymbol{\vartheta}) \Big] dz_j, \end{aligned} \quad (3)$$

where $\mathbf{x}_j = \{x_{ij} | i \in \mathcal{U}_j\}$ is the set of claims with respect to the target z_j.

Using the maximum a posteriori (MAP) estimation [22], we can maximize $\ln p(\mathbf{X}|\boldsymbol{\vartheta}) + \ln p(\boldsymbol{\vartheta})$ with respect to $\boldsymbol{\vartheta}$, and obtain the estimates of model parameters. However, direct optimization is difficult due to the summations and integrals in $p(\mathbf{X}|\boldsymbol{\vartheta})$. We thus again resort to the EM algorithm [21] for model inference. We summarize the inference procedure in Algorithm 1.

The derivation of this new batch algorithm is much more complex than that of the original algorithm in [10]. This is because we now have an additional latent variable z_j, in addition to r_{jk}, and we need to compute more expectation terms in the E-step.

4.2 E-step

We first write out the complete-data log-likelihood as

$$\begin{aligned} \ln p(\mathbf{X}, \mathbf{Y}|\boldsymbol{\vartheta}) &= \sum_j \ln p(z_j) + \sum_j \sum_k r_{jk} \ln \pi_k \\ &+ \sum_j \sum_k r_{jk} \sum_{i \in \mathcal{U}_j} \Big(\frac{1}{2} \ln \lambda_{ik} - \frac{1}{2} \lambda_{ik} (x_{ij} - z_j - h_{ik})^2 \Big). \end{aligned} \quad (4)$$

In the E-step, we compute $\mathbb{E}_{\mathbf{Y}|\mathbf{X}, \boldsymbol{\vartheta}^{(t)}}[\ln p(\mathbf{X}, \mathbf{Y}|\boldsymbol{\vartheta})]$, which is the expectation of the complete-data log-likelihood given the current estimate $\boldsymbol{\vartheta}^{(t)}$ of model parameters. After writing out the expression of $\mathbb{E}_{\mathbf{Y}|\mathbf{X}, \boldsymbol{\vartheta}^{(t)}}[\ln p(\mathbf{X}, \mathbf{Y}|\boldsymbol{\vartheta})]$, we find that we need to compute the following expectations

$$\begin{aligned} \alpha_{jk}^{(t)} &\equiv \mathbb{E}_{\mathbf{Y}|\mathbf{X}, \boldsymbol{\vartheta}^{(t)}}(r_{jk}) \propto \pi_k^{(t)} \mathcal{N}(f_{jk}^{(t)} | \mu_j, 1/\nu_j + 1/g_{jk}^{(t)}), \\ \beta_{jk}^{(t)} &\equiv \mathbb{E}_{\mathbf{Y}|\mathbf{X}, \boldsymbol{\vartheta}^{(t)}}(r_{jk} z_j) = \alpha_{jk}^{(t)} \phi_{jk}^{(t)}, \\ \zeta_{jk}^{(t)} &\equiv \mathbb{E}_{\mathbf{Y}|\mathbf{X}, \boldsymbol{\vartheta}^{(t)}}(r_{jk} z_j^2) = \alpha_{jk}^{(t)} \big[(\phi_{jk}^{(t)})^2 + 1/\psi_{jk}^{(t)} \big], \end{aligned} \quad (5)$$

where

$$\begin{aligned} f_{jk}^{(t)} &\equiv \frac{\sum_{i \in \mathcal{U}_j} \lambda_{ik}^{(t)} (x_{ij} - h_{ik}^{(t)})}{\sum_{i \in \mathcal{U}_j} \lambda_{ik}^{(t)}}, \; g_{jk}^{(t)} \equiv \sum_{i \in \mathcal{U}_j} \lambda_{ik}^{(t)}, \\ \phi_{jk}^{(t)} &\equiv \frac{\nu_j \mu_j + \sum_{i \in \mathcal{U}_j} \lambda_{ik}^{(t)} (x_{ij} - h_{ik}^{(t)})}{\nu_j + \sum_{i \in \mathcal{U}_j} \lambda_{ik}^{(t)}}, \; \psi_{jk}^{(t)} \equiv \nu_j + \sum_{i \in \mathcal{U}_j} \lambda_{ik}^{(t)}. \end{aligned} \quad (6)$$

For each j, $\alpha_{jk}^{(t)}$ needs to be normalized over k. We provide the complex derivation in the Appendix.

4.3 M-step

In the M-step, we re-estimate the model parameters $\boldsymbol{\vartheta}$ given the expectations of the latent variables. We perform the MAP estimation [22] and the corresponding M-step is to solve the following optimization problem

$$\begin{aligned} &\boldsymbol{\vartheta}^{(t+1)} = \arg\max_{\boldsymbol{\vartheta}} \mathbb{E}_{\mathbf{Y}|\mathbf{X}, \boldsymbol{\vartheta}^{(t)}}[\ln p(\mathbf{X}, \mathbf{Y}|\boldsymbol{\vartheta})] + \ln p(\boldsymbol{\vartheta}) \\ &\text{s.t.} \sum_k \pi_k = 1, \end{aligned} \quad (7)$$

where $\ln p(\boldsymbol{\vartheta}) = \sum_i \sum_k \ln p(\lambda_{ik})$.

We can derive the solution to (7) as follows

$$\begin{aligned} \pi_k^{(t+1)} &= \frac{\sum_j \alpha_{jk}^{(t)}}{N}, \; h_{ik}^{(t+1)} = \frac{\sum_{j \in \mathcal{Z}_i} \alpha_{jk}^{(t)} x_{ij} - \sum_{j \in \mathcal{Z}_i} \beta_{jk}^{(t)}}{\sum_{j \in \mathcal{Z}_i} \alpha_{jk}^{(t)}}, \\ \lambda_{ik}^{(t+1)} &= \frac{\frac{1}{2} \sum_{j \in \mathcal{Z}_i} \alpha_{jk}^{(t)} + a_{ik} - 1}{\frac{1}{2} \sum_{j \in \mathcal{Z}_i} \Big[\alpha_{jk}^{(t)} x_{ij}^2 + \alpha_{jk}^{(t)} (h_{ik}^{(t+1)})^2 - 2\alpha_{jk}^{(t)} x_{ij} h_{ik}^{(t+1)} \\ &\qquad - 2\beta_{jk}^{(t)} x_{ij} + 2\beta_{jk}^{(t)} h_{ik}^{(t+1)} + \zeta_{jk}^{(t)} \Big] + b_{ik} \end{aligned} \quad (8)$$

where $\mathcal{Z}_i$ is the set of targets that u_i makes a claim on.

As the expectations $\alpha_{jk}^{(t)}$, $\beta_{jk}^{(t)}$, and $\zeta_{jk}^{(t)}$ have been computed in the E-step and the claims x_{ij} are observed, these updating rules in (8) have closed forms, which do not depend on the unknown true values of z_j (but its known expectations).

4.4 Estimating True Quantity Values

Different from the original truth discovery algorithm in [10], we do not directly estimate z_j during model inference. After the EM algorithm converges, we estimate the true quantity value z_j using the minimum mean square error (MMSE) estimation [22] as

$$\begin{aligned} \mathbb{E}(z_j|\mathbf{x}_j) &= \sum_k p(r_{jk}|\mathbf{x}_j) \mathbb{E}(z_j|r_{jk}, \mathbf{x}_j) \\ &= \sum_k \alpha_{jk} \phi_{jk} = \sum_k \beta_{jk}. \end{aligned} \quad (9)$$

Algorithm 2 Parallel truth discovery

Input: Crowdsourced claims $\mathbf{X} = \{x_{ij}\}$
Output: Estimated true quantity values $\hat{z}_j$

Initialize π_k, h_{ik}, λ_{ik} and set $t = 0$
while Model does not converge **do**
  {*Map algorithm*}
  for each chunk c of claims (in parallel) **do**
    Initialize all $s^{c(t)}$ as 0
    for each target j **do**
      For all k, calculate $\alpha_{jk}^{(t)}$, $\beta_{jk}^{(t)}$, $\zeta_{jk}^{(t)}$ acc. (5)
      For all k, update $s_{0k}^{c(t)}$ acc. (12)
      for $i \in \mathcal{U}_j$ **do**
        For all k, update $s_{1ik}^{c(t)}$, $s_{2ik}^{c(t)}$, $s_{3ik}^{c(t)}$ acc. (13)
      end for
    end for
    Output $< key, value >$ pairs acc. (14)
  end for
  {*Reduce algorithm*}
  Compute N, $s_{0k}^{(t)}$; update $\pi_k^{(t+1)}$ acc. (11)
  for Each chunk d of participants (in parallel) **do**
    for each participant i **do**
      Compute $s_{1ik}^{(t)}$, $s_{2ik}^{(t)}$, $s_{3ik}^{(t)}$ and update $h_{ik}^{(t+1)}$, $\lambda_{ik}^{(t+1)}$ acc. (11)
    end for
  end for
  $t \leftarrow t + 1$
end while
{*Truth Discovery*}
Estimate $\hat{z}_j$ using converged model paras. acc. (9) on each mapper (in parallel)
return $\hat{z}_j$

It is observed that $\mathbb{E}(z_j|\mathbf{x}_j)$ can be conveniently computed as the sum over β_{jk}, which has been calculated in (5) in the E-step. We then round $\mathbb{E}(z_j|\mathbf{x}_j)$ to the closest integer as the final estimate $\hat{z}_j$.

5 Parallel Truth Discovery

In this section, we develop a parallel truth discovery algorithm based on the new batch algorithm developed in Section 4 and the MapReduce framework [19] that can efficiently handle big data. This parallel algorithm decomposes the large-scale truth discovery problem into several small-scale map and reduce tasks that can be run in parallel over a cluster of processors. We illustrate the structure of this algorithm in Fig. 3(b).

5.1 Overview

We first examine the structures of the updating rules (8) in the M-step of the new batch algorithm, and define the following new variables representing sufficient statistics

$$
\begin{aligned}
s_{0k}^{(t)} &\equiv \sum_j \alpha_{jk}^{(t)},\ s_{1ik}^{(t)} \equiv \sum_{j \in \mathcal{Z}_i} \alpha_{jk}^{(t)}, \\
s_{2ik}^{(t)} &\equiv \sum_{j \in \mathcal{Z}_i} \Big(\alpha_{jk}^{(t)} x_{ij} - \beta_{jk}^{(t)} \Big), \\
s_{3ik}^{(t)} &\equiv \sum_{j \in \mathcal{Z}_i} \Big(\alpha_{jk}^{(t)} x_{ij}^2 - 2\beta_{jk}^{(t)} x_{ij} + \zeta_{jk}^{(t)} \Big).
\end{aligned} \tag{10}
$$

Using these sufficient statistics, we can rewrite the updating rules (8) in the M-step as

$$
\begin{aligned}
\pi_k^{(t+1)} &= \frac{s_{0k}^{(t)}}{N},\ h_{ik}^{(t+1)} = \frac{s_{2ik}^{(t)}}{s_{1ik}^{(t)}}, \\
\lambda_{ik}^{(t+1)} &= \frac{\frac{1}{2} s_{1ik}^{(t)} + a_{ik} - 1}{\frac{1}{2}\big[s_{1ik}^{(t)} (h_{ik}^{(t+1)})^2 - 2 s_{2ik}^{(t)} h_{ik}^{(t+1)} + s_{3ik}^{(t)} \big] + b_{ik}}.
\end{aligned} \tag{11}
$$

It is observed from (11) that once we have computed the values of the sufficient statistics in (10), the model parameters in the M-step can be easily updated. The problem remaining is how to compute these sufficient statistics in (10) efficiently.

By observing the structures of the sufficient statistics, we exploit the MapReduce framework [19] to compute them in parallel. MapReduce is a programming model and an associated implementation for processing large datasets with a parallel, distributed algorithm on a cluster. A MapReduce job usually splits the input dataset into independent chunks, which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the map tasks, which are then input to the reduce tasks. MapReduce has been used in a wide range of applications, including distributed pattern-based searching, distributed sorting, web access log statistics, inverted index construction, document clustering, and machine learning[2].

In the following, we detail our proposed Map and Reduce algorithms. The overall parallel algorithm is summarized in Algorithm 2.

5.2 Map Algorithm

We first partition the whole set of claims into C chunks with almost equal size such that each chunk can be completely loaded into memory (the partition is target-wise). Each chunk then contains a subset of claims x_{ij} corresponding to non-overlapping targets z_j, but overlapping participants u_i (illustrated in Fig. 3(b)). Denote the number of targets in the cth chunk as N^c.

Assume we have P processors, then P mappers can be run in parallel and each mapper processes a chunk of claims. For the cth chunk, a mapper initializes all the sufficient statistics s^c as 0, and does the following processing for each target z_j and associated x_{ij} (we annotate related variables with superscript c to denote that the result is produced from the cth chunk of data).

1) Compute $\alpha_{jk}^{(t)}$, $\beta_{jk}^{(t)}$, and $\zeta_{jk}^{(t)}$ according to (5).
2) Update $s_{0k}^{c(t)}$ as

$$
s_{0k}^{c(t)} \leftarrow s_{0k}^{c(t)} + \alpha_{jk}^{(t)}. \tag{12}
$$

3) Update $s_{1ik}^{c(t)}$, $s_{2ik}^{c(t)}$, $s_{3ik}^{c(t)}$ for all $i \in \mathcal{U}_j$ as

$$
\begin{aligned}
s_{1ik}^{c(t)} &\leftarrow s_{1ik}^{c(t)} + \alpha_{jk}^{(t)}, \\
s_{2ik}^{c(t)} &\leftarrow s_{2ik}^{c(t)} + \alpha_{jk}^{(t)} x_{ij} - \beta_{jk}^{(t)}, \\
s_{3ik}^{c(t)} &\leftarrow s_{3ik}^{c(t)} + \alpha_{jk}^{(t)} x_{ij}^2 - 2\beta_{jk}^{(t)} x_{ij} + \zeta_{jk}^{(t)}.
\end{aligned} \tag{13}
$$

2. https://en.wikipedia.org/wiki/MapReduce.

Algorithm 3 Streaming truth discovery

Input: Crowdsourced claims $\mathbf{X} = \{x_{ij}\}$
Output: Estimated true quantity values $\hat{z}_j$

for Each target **do**
  $j \leftarrow j + 1$
  {*E-step*}
  For all k, calculate α_{jk}, β_{jk}, ζ_{jk} acc. (5)
  For all k, update $\tilde{s}_{0k}^{(j)}$ acc. (17)
  for $i \in \mathcal{U}_j$ **do**
    $l_i \leftarrow l_i + 1$
    For all k, update $\tilde{s}_{1ik}^{(l_i)}$, $\tilde{s}_{2ik}^{(l_i)}$, $\tilde{s}_{3ik}^{(l_i)}$ acc. (17)
  end for
  {*M-step*}
  If $j > 20$, for all k, update $\pi_k^{(j)}$ acc. (18)
  for $i \in \mathcal{U}_j$ **do**
    If $l_i > 20$, for all k, update $h_{ik}^{(l_i)}$, $\lambda_{ik}^{(l_i)}$ acc. (18)
  end for
end for
{*Truth Discovery*}
Estimate $\hat{z}_j$ using final model paras. acc. (9)
return $\hat{z}_j$

Once a mapper finishes processing the cth chunk of claims, it outputs the following $< key, value >$ pairs

$$< count, N^c >, < 0k, s_{0k}^{c(t)} >,$$
$$< 1ik, s_{1ik}^{c(t)} >, < 2ik, s_{2ik}^{c(t)} >, < 3ik, s_{3ik}^{c(t)} > . \quad (14)$$

The key $count$ needs to be output only once. After the model converges, each mapper also locally computes $\mathbb{E}(z_j|\mathbf{x}_j)$ according to (9).

5.3 Reduce Algorithm

A reducer sums up the values received from different mappers with respect to the same key (illustrated in Fig. 3(b)). For example, for the key $1ik$, the reducer initializes $s_{1ik}^{(t)}$ as 0 and does $s_{1ik}^{(t)} \leftarrow s_{1ik}^{(t)} + s_{1ik}^{c(t)}$ over c (the reduce step for the key $count$ needs to be done only once). After summing up all the values from all the data chunks, it updates the model parameters $\pi_k^{(t+1)}$, $h_{ik}^{(t+1)}$, and $\lambda_{ik}^{(t+1)}$ according to (11). These updated model parameters are then input to the mappers again.

It is easy to show that

$$s_{1ik}^{(t)} = \sum_c s_{1ik}^{c(t)} = \sum_c \sum_{j \in \mathcal{Z}_i^c} \alpha_{jk}^{(t)} = \sum_{j \in \mathcal{Z}_i} \alpha_{jk}^{(t)},$$

where $\mathcal{Z}_i^c$ is the set of targets in the cth chunk of data that u_i makes a claim on. Therefore, this parallel truth discovery algorithm and the batch truth discovery algorithm in Section 4 produce exactly the same result.

As the aggregation of values (e.g., $s_{1ik}^{(t)}$) with respect to different u_i are independent in the M-step (11), we can also partition the set of participants into several chunks and use several reducers in parallel, each processing a subset of non-overlapping participants.

6 Streaming Truth Discovery

In this section, we develop a streaming truth discovery algorithm based on the new batch algorithm developed in Section 4 and the on-line EM algorithm [20]. This algorithm does not need to load all the data in each iteration. Instead, it loads only one target and associated claims in each iteration, and recursively updates its parameters until all the targets are processed. It can thus efficiently perform truth discovery both on large datasets and in data streams, with limited hardware requirement. We illustrate the structure of this algorithm in Fig. 3(c).

6.1 Overview

The batch truth discovery algorithm (Fig. 3(a)) has three drawbacks when dealing with big data: 1) it needs to load and process the whole dataset, which may cause an "out of memory" problem, 2) it needs to iterate over the whole dataset several times until convergence, which costs lots of time, and 3) when new claims are available, it cannot perform truth discovery incrementally but has to reprocess historical data, in addition to new data. Although the parallel algorithm (Fig. 3(b)) can load and process chunks of data in parallel (which avoids the "out of memory" problem and is much faster), it still cannot perform incremental truth discovery.

These drawbacks motivate the development of a streaming truth discovery algorithm (Fig. 3(c)) in this section. This streaming algorithm only needs to load and process one target and the associated claims in each iteration. It recursively updates its model parameters until all the targets are processed. That is to say, it iterates over the whole dataset only once instead of several times. It thus converges much faster than the parallel and the batch algorithms. It also consumes much less memory and can be executed in memory-limited environments. Moreover, it can enable incremental truth discovery in data streams in real time, where crowdsourced claims arrive over time.

To enable the streaming algorithm, we first consider a classical recursive parameter estimation algorithm [23] which takes the form

$$\hat{\boldsymbol{\vartheta}}_n = \hat{\boldsymbol{\vartheta}}_{n-1} + \rho_n I^{-1}(\hat{\boldsymbol{\vartheta}}_{n-1}) \nabla_{\boldsymbol{\vartheta}} \ln p(x_n|\hat{\boldsymbol{\vartheta}}_{n-1}), \quad (15)$$

where n denotes the nth data, ρ is a step size, and I is the Fisher Information Matrix (FIM) [24]. This algorithm recursively updates its parameters $\hat{\boldsymbol{\vartheta}}$ when each new data point x is obtained, thus achieving streaming model update. However, it is difficult to apply this algorithm to our problem as the data likelihood (3) is rather complex, which makes the computation of the FIM prohibitive.

We thus resort to a more recently proposed *on-line EM* algorithm [20], which does not rely on the computation of the FIM, for streaming truth discovery. This algorithm also processes each new data point only once and recursively updates its model parameters. It has been proved to converge to a local maximum [20], and empirically, it often exhibits comparable performance as the batch EM algorithm [24]. It replaces the E-step in the batch EM algorithm [21] by a stochastic E-step on some sufficient statistics, while keeping the M-step unchanged. To apply

this algorithm, the updating rules in the M-step must be explicit (i.e., have closed forms) [20]. As a result, the original truth discovery algorithm in [10] cannot be extended to streaming processing utilizing this on-line EM algorithm, while the new batch algorithm developed in Section 4 can. We summarize our proposed streaming truth discovery algorithm in Algorithm 3.

6.2 E-step

We consider all the claims $\mathbf{x}_j = \{x_{ij} | i \in \mathcal{U}_j\}$ with respect to a target z_j as a high-dimensional data point. The streaming E-step is performed on each $\mathbf{x}_j$, rather than on each x_{ij}. This is because x_{ij} are correlated for a z_j according to the TBP model, and considering them separately is unreasonable. For applications where the target quantity with respect to a given entity changes over time, the quantities at different time are considered as different targets [10], [18]. For example, people counts in a given room at 3pm and at 4pm on the same day are treated as two different targets. Truth discovery is then performed on the claims for respective targets.

We start by examining the complete-data log likelihood $\ln p(\mathbf{x}_j, r_{jk}, z_j | \boldsymbol{\vartheta})$ with respect to one data point (similar to (4), but only with respect to the jth target). We then identify the sufficient statistics in the log likelihood as $S_j \equiv [r_{jk}, r_{jk}z_j, r_{jk}z_j^2, r_{jk}x_{ij}, r_{jk}x_{ij}^2, r_{jk}z_j x_{ij}]$.

In the jth time step of the streaming E-step, we process the jth data point $\mathbf{x}_j$ and update the expectation of S_j through stochastic approximation as [20]

$$\tilde{s}^{(j)} = (1 - \rho^{(j)})\tilde{s}^{(j-1)} + \rho^{(j)}\mathbb{E}(S_j), \tag{16}$$

where $\rho^{(j)} = j^{-\tau}$ is the step size (i.e., the time order j impacts the step size). We set $\tau = 0.6$ according to [20]. As $\rho^{(1)} = 1$, the initial values of $\tilde{s}$ in (16) are well-defined. According to Section 4.2, $\mathbb{E}(S_j)$ can be readily obtained as $\mathbb{E}(S_j) = [\alpha_{jk}, \beta_{jk}, \zeta_{jk}, \alpha_{jk}x_{ij}, \alpha_{jk}x_{ij}^2, \beta_{jk}x_{ij}]$. After examining the structures of the updating rules in the corresponding M-step, we merge the updating of several sufficient statistics given by (16) and have the updating rules in the streaming E-step as

$$\begin{aligned}
\tilde{s}_{0k}^{(j)} &= (1-\rho^{(j)})\tilde{s}_{0k}^{(j-1)} + \rho^{(j)}\alpha_{jk},\\
\tilde{s}_{1ik}^{(l_i)} &= (1-\rho^{(l_i)})\tilde{s}_{1ik}^{(l_i-1)} + \rho^{(l_i)}\alpha_{jk},\\
\tilde{s}_{2ik}^{(l_i)} &= (1-\rho^{(l_i)})\tilde{s}_{2ik}^{(l_i-1)} + \rho^{(l_i)}(\alpha_{jk}x_{ij} - \beta_{jk}),\\
\tilde{s}_{3ik}^{(l_i)} &= (1-\rho^{(l_i)})\tilde{s}_{3ik}^{(l_i-1)} + \rho^{(l_i)}(\alpha_{jk}x_{ij}^2 - 2\beta_{jk}x_{ij} + \zeta_{jk}).
\end{aligned} \tag{17}$$

In the above rules, we use an additional superscript l_i to denote the time order of x_{ij} with respect to the set of claims by u_i. For example, if u_i makes claims as (x_{i1}, x_{i2}, x_{i5}), then we have $l_i = 3$ for x_{i5} (i.e., x_{i5} is the 3rd claim made by u_i).

It is observed from (17) that these new sufficient statistics such as $\tilde{s}_{0k}^{(j)}$ are recursively updated (rather than the sum over all the data as that in (10)). Moreover, $\tilde{s}_{0k}^{(j)}$ is updated for each new $\mathbf{x}_j$, while $\tilde{s}_{1ik}^{(l_i)}$ to $\tilde{s}_{3ik}^{(l_i)}$ are updated only when u_i makes a claim x_{ij} on z_j (i.e., $i \in \mathcal{U}_j$), and are with respect to the time order l_i of this claim.

6.3 M-step

In the streaming M-step, we compute the new model parameters by maximizing the expectation of the complete-data log-likelihood (over a single data point) plus the prior probabilities on model parameters. It can be shown that the updating rules take the following form

$$\begin{aligned}
\pi_k^{(j)} &= \tilde{s}_{0k}^{(j)},\ h_{ik}^{(l_i)} = \frac{\tilde{s}_{2ik}^{(l_i)}}{\tilde{s}_{1ik}^{(l_i)}},\\
\lambda_{ik}^{(l_i)} &= \frac{\frac{1}{2}\tilde{s}_{1ik}^{(l_i)} + a_{ik} - 1}{\frac{1}{2}\left[\tilde{s}_{1ik}^{(l_i)}(h_{ik}^{(l_i)})^2 - 2\tilde{s}_{2ik}^{(l_i)}h_{ik}^{(l_i)} + \tilde{s}_{3ik}^{(l_i)}\right] + b_{ik}}.
\end{aligned} \tag{18}$$

Comparing (18) and (11), we note that $\pi_k^{(j)}$ in (18) does not need normalization over N, as $\tilde{s}_{0k}^{(j)}$ is already normalized in (17). Moreover, $\pi_k^{(t+1)}$, $h_{ik}^{(t+1)}$, and $\lambda_{ik}^{(t+1)}$ are all updated in the tth iteration in (11). In contrast, when processing a new $\mathbf{x}_j$, $\pi_k^{(j)}$ is updated, while $h_{ik}^{(l_i)}$ and $\lambda_{ik}^{(l_i)}$ are updated only when the corresponding x_{ij} is observed. According to [20], we do not perform the M-step for the first 20 claims of each u_i.

7 Analysis

In this section, we analyze the time complexity (TC) and the space complexity (SC) of the proposed batch, parallel and streaming truth discovery algorithms (in Sections 4, 5, and 6 respectively).

7.1 Time Complexity

Batch algorithm. The batch algorithm needs to process all the available data in each iteration. The TC of computing each α_{jk}, β_{jk}, or ζ_{jk} in each iteration is $O(\bar{M})$, where $\bar{M}$ is the average number of participants on each target. The TC of computing each π_k is $O(1)$. The TC of computing each h_{ik} or λ_{ik} is $O(\bar{N})$, where $\bar{N}$ is the average number of claims by each participant. Assume this algorithm iterates T times until model convergence. Its overall TC is then $O(TK(1 + N\bar{M} + M\bar{N}))$, where K is the number of difficulty levels, M is the total number of participants, and N is the total number of targets. We denote the total number of claims as X. As $N\bar{M} = M\bar{N} = X$, its data processing TC can be rewritten as $O(TK(1 + X))$.

Parallel algorithm. Assume in each iteration, each mapper and each reducer process $1/C$ and $1/D$ of all the data. For each mapper, the TC of computing all the α_{jk}, β_{jk}, and ζ_{jk} is $O(K\bar{M}\frac{N}{C}) = O(K\frac{X}{C})$. The TC of computing all the s_{0k}^c is $O(K\frac{N}{C})$. The TC of computing all the s_{1ik}^c, s_{2ik}^c, and s_{3ik}^c is $O(K\frac{X}{C})$. For each reducer, the TC of computing all the s_{1ik}, s_{2ik}, and s_{3ik} is $O(K\frac{CM}{D})$. The TC of computing all the h_{ik} and λ_{ik} is $O(K\frac{M}{D})$. The TC of computing all the s_{0k} and π_k (on any reducer) are $O(KC)$ and $O(K)$ respectively. This algorithm takes the same number T of iterations as the batch algorithm until model convergence. Its overall data *processing* TC is then $O(TK(1 + C + \frac{X+N}{C} + \frac{CM+M}{D}))$. Moreover, each

mapper needs to send out all the s^c_{1ik}, s^c_{2ik}, and s^c_{3ik}, and each reducer needs to send out $1/D$ of all the h_{ik} and λ_{ik}. This *communication* overhead is then $O(KM(1+\frac{1}{D}))$. Additionally, the system needs to sort the output of all the mappers. This *shuffling* overhead is $O(CKM)$.

Streaming algorithm. For the streaming algorithm, the number of iterations is the same as the number N of targets. In each iteration, the TC of computing all the α_{jk}, β_{jk}, and ζ_{jk} is $O(K\bar{M})$. The TC of updating all the $\tilde{s}_{0k}$ is $O(K)$. The TC of updating all the $\tilde{s}_{1ik}$, $\tilde{s}_{2ik}$, and $\tilde{s}_{3ik}$ is $O(K\bar{M})$. The TC of computing all the π_k is $O(K)$. The TC of computing all the h_{ik} and λ_{ik} is $O(K\bar{M})$. Its overall data processing TC is then $O(NK(1+\bar{M})) = O(K(N+X))$.

7.2 Space Complexity

Batch algorithm. In each iteration, the batch algorithm needs to load all the available data, whose SC is $O(X)$. The SC of caching all the α_{jk}, β_{jk}, and ζ_{jk} is $O(KN)$. The SC of caching all the π_k, h_{ik}, and λ_{ik} is $O(K+KM)$. Its overall SC in each iteration is then $O(X+K(1+M+N))$.

Parallel algorithm. In each iteration of the parallel algorithm, each mapper needs to load $1/C$ of all the data, whose SC is $O(\frac{X}{C})$. The SC of caching all s^c_{0k}, s^c_{1ik}, s^c_{2ik}, and s^c_{3ik} on each mapper is $O(K+KM)$. The SC of caching all the π_k, h_{ik} and λ_{ik} on all the reducers is $O(KM)$. As there are C mappers, the overall SC in each iteration is then $O(X+CK(1+M)+KM)$.

Streaming algorithm. In each iteration, the streaming algorithm needs to load all the claims associated with one target, whose SC is $O(\bar{M})$. The SC of caching all the $\tilde{s}_{0k}$, $\tilde{s}_{1ik}$, $\tilde{s}_{2ik}$, and $\tilde{s}_{3ik}$ is $O(K+KM)$. The SC of caching all the π_k, h_{ik}, and λ_{ik} is $O(K+KM)$. Its overall SC in each iteration is then $O(\bar{M}+K(1+M))$.

8 Experiments

In this section, we present experimental results to demonstrate that 1) the proposed batch, parallel, and streaming algorithms are all more effective than other state-of-the-art algorithms for quantitative truth discovery, and 2) the parallel and the streaming algorithms are additionally scalable, suitable for handling crowdsourcing applications involving big or streaming data.

8.1 Setup

Real-world datasets. We conducted two sets of real-world experiments, which are to count people and to estimate the occupancy level based on given snapshots (400 for each experiment). We posted these snapshots on CrowdFlower as quantitative crowdsourcing tasks to collect crowdsourced claims (20 participants for each task; each task contains a single target quantity). The corresponding datasets have been utilized in [10]. The statistics of these datasets are listed in Table 2.

Synthetic datasets. We further simulated two synthetic datasets (object counting, with $K = 3$ difficulty levels) according to the (generative) TBP model to study the scalability of different algorithms in handling large datasets. In the first synthetic dataset (Syn1), we simulated $M = 100$ participants and $N = 10$k tasks. In the second synthetic dataset (Syn2), we simulated $M = 1,000$ participants and $N = 100$k tasks. Syn2 contains more participants and more tasks than Syn1. Each task has 10 (random) participants. Each negative x_{ij} is discarded and resampled until it is non-negative. π_k is set as $1/K$. The biases of participants are generated as $h_{i1} \sim \text{Uni}(-5,5)$, $h_{i2} \sim \text{Uni}(-15,15)$, and $h_{i3} \sim \text{Uni}(-30,30)$ (Uni denotes the uniform distribution). The precisions of participants are generated as $\lambda_{i1} \sim \text{Uni}(0.01,0.1)$, $\lambda_{i2} \sim \text{Uni}(0.001,0.02)$, and $\lambda_{i3} \sim \text{Uni}(0.0001,0.005)$. The true quantity values are generated as $z_j \sim \text{Uni}(5,100)$. All x_{ij} and z_j are rounded to the closest integers. The statistics of these datasets are also listed in Table 2.

TABLE 2
Statistics of quantitative crowdsourcing tasks (occ - occupancy, pt - participant, avg - average).

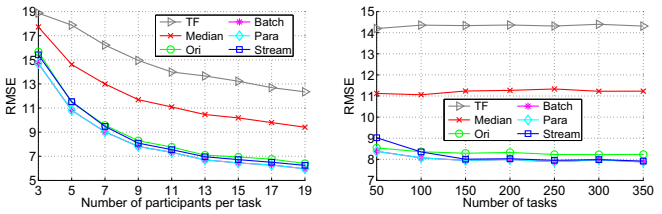
Task	# tasks	# pts	# total claims	avg # pts per task	avg # tasks per pt	ground truth (mean $\pm$ std)
People	400	106	8,000	20	75.5	29.4 ± 16.1
Occ	400	93	8,000	20	86.0	46.9 ± 28.3
Syn1	10,000	100	100,000	10	1,000	52.2 ± 27.4
Syn2	100,000	1,000	1,000,000	10	1,000	52.5 ± 27.5

8.2 Methods in Comparison

We have shown in [10] that the original truth discovery algorithm based on the TBP model outperforms existing algorithms such as majority voting, Average Log [15], Investment [15], Truth Finder [11], and median, in terms of the root mean square error (RMSE). Moreover, Truth Finder and median are the two best-performing competitors. To more clearly visualize the differences of the newly proposed algorithms, we only compare them with the two best-performing competitors.

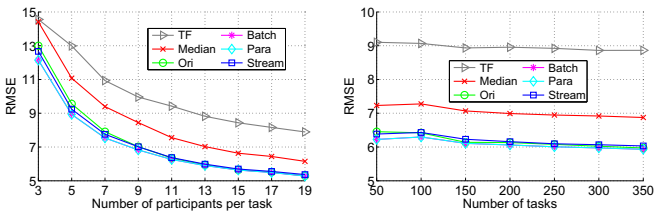
In summary, we compare the performance of the following algorithms for truth discovery in quantitative crowdsourcing applications.

1) `TF`: the Truth Finder method proposed in [11] with implication scores between distinct pairwise claims. We set the implication score between x_{ij} and $x_{i'j}$ as $\exp(-|x_{ij} - x_{i'j}|/20)$, which increases as the difference between x_{ij} and $x_{i'j}$ decreases. It means that similar claims support each other.
2) `Median`: using the median (more robust to outliers than mean) of the claims as the estimated truth.
3) `Ori`: the original truth discovery algorithm in [10].
4) `Batch`: the batch truth discovery algorithm proposed in Section 4.
5) `Para`: the parallel truth discovery algorithm proposed in Section 5.
6) `Stream`: the streaming truth discovery algorithm proposed in Section 6.



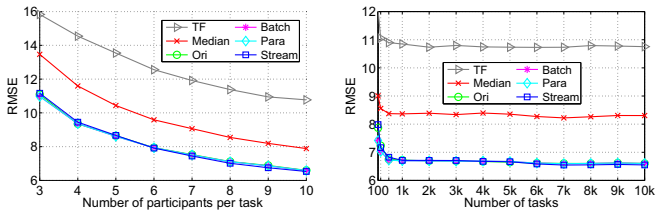
(a) RMSE vs. # of pts per task (b) RMSE vs. # of tasks

Fig. 4. RMSE on the people counting dataset. (a) RMSE vs. the number of participants per task ($N = 200$). (b) RMSE vs. the number of tasks ($M = 74$).



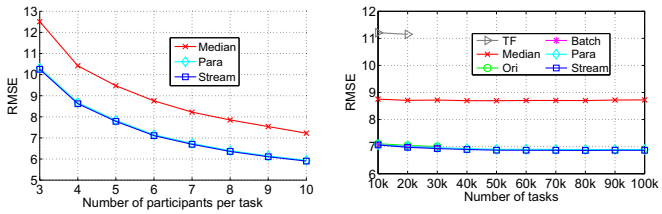
(a) RMSE vs. # of pts per task (b) RMSE vs. # of tasks

Fig. 5. RMSE on the occupancy estimation dataset. (a) RMSE vs. the number of participants per task ($N = 200$). (b) RMSE vs. the number of tasks ($M = 65$).



(a) RMSE vs. # of pts per task (b) RMSE vs. # of tasks

Fig. 6. RMSE on the first synthetic dataset. (a) RMSE vs. the number of participants per task ($N = 5$k). (b) RMSE vs. the number of tasks ($M = 70$).



(a) RMSE vs. # of pts per task (b) RMSE vs. # of tasks

Fig. 7. RMSE on the second synthetic dataset. (a) RMSE vs. the number of participants per task ($N = 50$k). (b) RMSE vs. the number of tasks ($M = 700$).

The last five algorithms round their outputs to respective closest integers.

8.3 Evaluation Metric

Error: We use the RMSE to evaluate the effectiveness of an algorithm. The RMSE takes into account both the mean and the standard deviation of estimation errors. It is defined as $\text{RMSE} = \sqrt{\frac{1}{N}\sum_j (\hat{z}_j - z_j)^2}$. The RMSE penalizes larger errors more and a smaller RMSE indicates better performance (i.e., discovered "truths" are closer to true quantity values on average).

CPU time: We use the CPU time to evaluate the time efficiency of an algorithm. A shorter CPU time implies a faster algorithm.

Space usage: We use the number of non-empty elements in allocated arrays to evaluate the space efficiency of an algorithm. A smaller number implies a more space efficient algorithm.

All the algorithms are implemented in Matlab (`Para` is implemented using Matlab Parallel Computing Toolbox). All the experiments below are conducted on the UCLA Hoffman2 Cluster[3] with a maximum of 8 processors (limited by the user group). Each processor is equipped with an Intel Xeon CPU and 4GB RAM. We are unable to monitor the system memory through a Matlab script. Except `Para`, all the other algorithms are evaluated using a single processor. As the RMSE of `Para` does not depend on the number of processors used, but the CPU time does, we do not indicate the number of processors used when we report its RMSE, but we indicate that number when we report its CPU time. Each experiment is performed 20 times by randomly sampling the desired number of entities. We only retain targets with at least 3 crowdsourced claims and participants with at least 20 claims (due to the requirement of `Stream`).

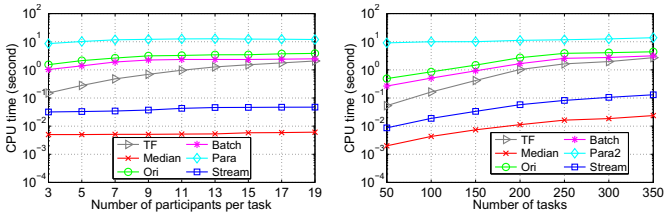
3. http://hpc.ucla.edu/hoffman2/hoffman2.php

8.4 Effectiveness

Real-world datasets. Fig. 4 plots the RMSEs of different algorithms on the people counting dataset. It is observed from Fig. 4(a) that the RMSEs of different algorithms obviously decrease as more participants join a task (we set $N = 200$). `Median` results in smaller RMSEs than `TF`. `Ori`, `Batch`, `Para`, and `Stream` result in even smaller RMSEs than the strong competitor `Median`, showing their effectiveness. When the number of participants per task is 19, the RMSEs of these four algorithms are all about 30% smaller than that of `Median`.

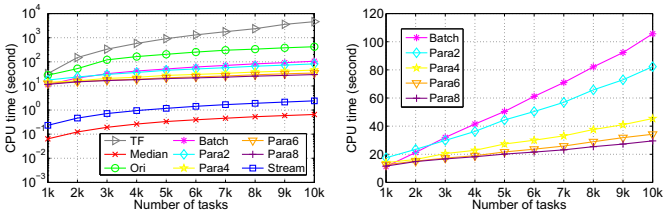
`Batch`, `Para`, and `Stream` exhibit comparable performance as `Ori`, as they are different model inference algorithms for the same TBP model. We also observe that `Para` and `Batch` have exactly the same performance, as we have shown in Section 5.3 that `Para` only differs from `Batch` in how the data are partitioned and processed, but not in the final updating rules. We also observe that `Batch` performs better than `Ori`. This is because in `Batch`, the true quantity value z_j is treated as a latent variable, while in `Ori`, z_j is treated as a model parameter. As a consequence, `Ori` needs more data than `Batch` to reliably infer the values of more model parameters. `Stream` exhibits slightly better performance than `Ori`, but slightly worse performance than `Batch` and `Para`.

It is observed from Fig. 4(b) that the RMSEs of different algorithms are relatively stable or slightly decrease as more tasks are involved (we use 70% participants). These results show that participants' ability parameters can be accurately estimated when a sufficient number



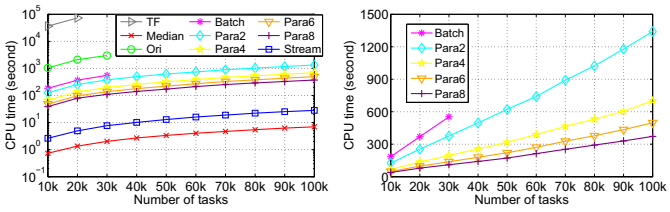
(a) CPU time vs. # of pts per task (b) CPU time vs. # of tasks

Fig. 8. CPU time on the people counting task.



(a) All the CPU time (b) CPU time of Batch and Para

Fig. 9. CPU time on the first synthetic dataset.



(a) All the CPU time (b) CPU time of Batch and Para

Fig. 10. CPU time on the second synthetic dataset.

of tasks are performed, and adding more tasks will not lead to significant reduction in the RMSE.

Fig. 5 plots the RMSEs of different algorithms on the occupancy estimation dataset. We observe similar trends as those in Fig. 4.

Synthetic datasets. We also observe similar trends of RMSEs on the two synthetic datasets as those on the real-world datasets. Fig. 6 plots the RMSEs of different algorithms on the first synthetic dataset. It is observed in Fig. 6(a) that `Ori`, `Batch`, `Para`, and `Stream` all exhibit almost identical performance regardless of the number of participants per task (we set $N = 5$k). It is observed in Fig. 6(b) that the RMSEs of different algorithms quickly decrease in the beginning when more tasks are involved (we set $M = 70$). But these RMSEs become relatively stable after $N = 1$k. This is because each task in this synthetic dataset is worked on by only 10 participants (rather than 20 in the real-world datasets), and thus truth discovery algorithms need more tasks to reliably infer participants' ability parameters.

Fig. 7 plots the RMSEs of different algorithms on the second synthetic dataset. It is observed that `TF`, `Ori`, and `Batch` run into computational limitations (discussed next) on this dataset, while the relative performance of other algorithms is similar to that in Fig. 6.

8.5 Time Efficiency

Real-world datasets. Fig. 8 plots the CPU time (log scale) of different algorithms on the people counting dataset (we set $N = 200$). As this dataset is small, we test `Para` with 2 processors. It is observed from Fig. 8(a) that all the CPU time increases slightly as more participants join a task. `Median` is the most time efficient, which can finish processing 200 tasks, each with 19 participants in less than 0.01 second. `Stream` follows, it can finish processing these tasks in around 0.05 second. As `Para` has additional communication and shuffling overhead, its CPU time is longer than `Batch` on this small dataset. `Batch` is more time efficient than `Ori`, as its M-step has closed forms, while `Ori`'s does not.

It is observed from Fig. 8(b) that all the CPU time obviously increases as more tasks are involved (we use 70% participants), except `Para`, whose CPU time seems to be dominated by its communication overhead.

Synthetic datasets. Fig. 9 plots the CPU time of different algorithms on the first synthetic dataset. `Para#` denotes `Para` with `#` processors, where `#` ranges from 2 to 8. It is observed from Fig. 9(a) that `Median` is the most time efficient, followed by `Stream`. When processing 10k tasks, `Median` and `Stream` take around 0.6 second and 2.4 seconds respectively. `TF` is the most time-consuming as it needs to assign pair-wise implication scores between distinct claims. Its CPU time increases quickly as more tasks are involved, which exceeds 1,000 seconds when processing 6k tasks. As a consequence, `TF` is not suitable for processing big data. The CPU time of `TF` can be much shorter if we do not assign implication scores. However, the resulting RMSEs are much larger.

To more clearly observe the differences between `Batch` and `Para#`, we plot in Fig. 9(b) their CPU time on the linear scale. It is observed that `Para2` is more time efficient than `Batch` when the number of tasks is larger than 3k, and `Para4`, `Para6`, and `Para8` are more efficient than `Batch` when the number of tasks is larger than 2k. `Para8` is the most efficient when the number of tasks is larger than 5k. It can process 10k tasks in 29 seconds, much shorter than 106 seconds of `Batch`.

Fig. 10 plots the CPU time of different algorithms on the second synthetic dataset. It is observed that `TF` has excessively long CPU time which exceeds 20 hours when processing 20k tasks. `Ori` has a CPU time of 50.0 minutes and `Batch` has a CPU time of 9.2 minutes when processing 30k tasks. `TF` is forced to terminate by the UCLA Hoffman2 Cluster when the number of tasks is larger than 20k, as its CPU time exceeds 24 hours. `Ori` and `Batch` cause an "out of memory" error when the number of tasks is larger than 30k. As a result, these methods are not able to handle large datasets. In contrast, `Median`, `Para`, and `Stream` still work when the number of tasks is 100k, showing their scalability. `Para8` is the most time efficient among the variants of `Para`. `Para8` takes 373 seconds (6.2 minutes) to process 100k tasks, compared with 1340 seconds (22.3 minutes) of `Para2`.

Note that, when new claims arrive over time, `Median` and `Stream` can perform incremental truth discovery.

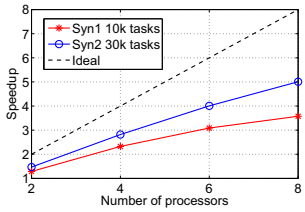
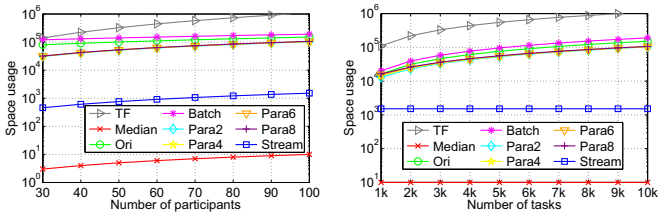


Fig. 11. Speedup curves (the speedup at P is defined as the ratio of the CPU time for one processor to the CPU time for P processors).



(a) Space usage vs. # of pts (b) Space usage vs. # of tasks

Fig. 12. Space usage on the first synthetic dataset. (a) Space usage vs. the number of participants ($N = 10$k). (b) Space usage vs. the number of tasks ($M = 100$).

Their CPU time depends on the number of incremental tasks processed. In contrast, other algorithms need to process all the cumulative data. Their CPU time depends on the number of cumulative tasks processed.

Speedup. We now examine the speedup of using multiple processors (`Para`) over using one processor (`Batch`). In our problem, we define the speedup at P as the ratio of the CPU time for one processor to the CPU time for P processors. Fig. 11 shows the speedup on the first synthetic dataset (Syn1) with 10k tasks, the second synthetic dataset (Syn2) with 30k tasks, and the ideal curve (i.e., a straight line with slope 1). It is observed that the speedup curve of Syn2 with 30k tasks is closer to the ideal curve. This makes sense as there is more processing workload versus communication overhead for Syn2. In other words, the speedup is more salient when using multiple processors for heavier processing workload.

8.6 Space Efficiency

Fig. 12 plots the space usage (in terms of the number of non-empty elements in allocated arrays) of different algorithms on Syn1. The space usage of `Para#` is the overall usage across all the processors. It is observed that `Median` is the most space efficient, as it is a model-less method that works on one target and associated claims each time. `Stream` follows, as it also works on one target and associated claims each time. However, it needs to cache sufficient statistics and model parameters. `Para` consumes more space than `Stream` as it needs to load all the data chunks over the processors. The space usage of the variants of `Para` does not differ too much, as the space used by the raw data dominates. `Batch` and `Ori` consume more space than `Para`, as they need to cache expectations on all the targets while `Para` needs to cache sufficient statistics on all the participants. Since usually $M \ll N$, `Para` is more space efficient. Finally, `TF` is the most space-consuming as it performs matrix operations on all the distinct claims for all the targets.

9 Discussion and Future Work

Which algorithm to choose? Given a computing system, we can first run `Batch` on a small dataset and examine the memory usage and the execution time. We can then predict them given a large dataset. If the predicted memory usage exceeds the available system memory or the execution time exceeds expectation, we should not choose `Batch`. If we have enough hardware and prefer good performance, we might choose `Para`. If we have limited hardware and we can tolerate slight performance loss, we might choose `Stream`.

If truth discovery needs to be performed in data streams where claims about time-varying target quantities arrive over time, we might choose `Stream` as it can perform incremental truth discovery without processing historical data. If the target quantities do not change over time while the number of claims per target is unbound, then all these algorithms need to be rerun on both historical and new data from time to time.

Exploiting alternative parallel processing frameworks. We currently exploit the MapReduce framework [19] to enable parallel truth discovery. There are also alternative parallel processing frameworks with different design principles such as the resilient distributed datasets [25] that can be exploited in the future.

Theoretical analysis. Results on the effectiveness of the streaming algorithm in this paper are empirical. In the future, we may analyze the theoretical performance of the streaming algorithm, compared with others.

Alternative modeling. Besides modeling the task difficulty in the TBP model, we can also consider to model direct dependency of participants' ability parameters on the latent truth in the future.

10 Related Work

Research in crowdsourcing has gained rapid growth in recent years [1]. Crowds have been explored to perform various Human Intelligence Tasks (HITs) such as large-scale image classification [13], transcription [26], and word processing [3]. Besides performing such HITs on crowdsourcing platforms such as the Amazon Mechanical Turk (AMT)[4] and CrowdFlower[5], crowds have also been utilized to detect and report events in the physical world, e.g., to detect earth quakes [27], to detect desired flora on campus [28], and to detect social disorder in public places [29].

However, these useful applications may be impaired by unskilled or sloppy crowd participants who provide low-quality data. As a consequence, it is important to

4. https://www.mturk.com/mturk/welcome.
5. http://www.crowdflower.com.

develop truth discovery algorithms which can automatically discover the truths from possibly conflicting and noisy crowdsourced data.

In the domain of truth discovery from conflicting Web information, Yin et al. [11] proposed truth finder, which is a transitive voting algorithm with rules specifying how votes iteratively flow from sources to claims and then back to sources. Pasternack and Roth [15] proposed AverageLog, Investment, and PooledInvestment algorithms. Zhao et al. [17] proposed a more principled probabilistic approach which can automatically infer true claims and two-sided source quality.

In the domain of aggregating conflicting claims in crowdsourcing applications, Dawid and Skene [30] modeled the generative process of the claims by introducing source ability parameters. Whitehill et al. [12] further included the task difficulty in the model. Welinder et al. [14] proposed a model consisting of worker compatibility for each task. Wang et al. [16] proposed a model for truth discovery in social sensing. They further extended their model to consider potentially dependent information sources in [31]. Baba et al. [32] proposed a model for truth discovery in general crowdsourcing tasks such as article writing and logo design. Ouyang et al. [33] proposed a model for truth discovery in crowdsourced detection of spatial events.

Nevertheless, these methods are mainly designed for categorical or even binary truth discovery, and thus they do not directly apply or are not effective for quantitative truth discovery problems. Although we have proposed the TBP model in [10] and have shown that TBP is more effective than existing methods for quantitative truth discovery, the original model inference algorithm is unfortunately not scalable. Recently, Wang et al. proposed a recursive truth discovery algorithm in [18]. However, it is for binary truth discovery and is not applicable to our problem. Moreover, it utilizes the classical recursive parameter estimation algorithm [23], which is difficult to be applied to our problem (discussed in Section 6).

In this paper, we first develop a new batch truth discovery algorithm based on the TBP model, which decouples the inferences of target-related and participant-related parameters. We then examine its structure and exploit the MapReduce framework [19] to develop a parallel algorithm. Finally, we exploit the on-line EM algorithm [20] to develop a streaming algorithm. We demonstrate that the parallel and the streaming algorithms are both effective and scalable.

11 Conclusion

In this paper, we propose new parallel and streaming truth discovery algorithms for quantitative crowdsourcing applications involving big or streaming data. Through extensive experiments, we demonstrate that both algorithms are effective. Moreover, the parallel algorithm can efficiently perform truth discovery on large datasets, and the streaming algorithm can efficiently perform truth discovery both on large datasets and in data streams. They can thus support effective and scalable truth discovery in large-scale quantitative crowdsourcing applications.

Acknowledgements

This research is based upon work supported in part by the U.S. ARL and U.K. Ministry of Defense under Agreement Number W911NF-06-3-0001, and by the NSF under award CNS-1213140. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views or represent the official policies of the NSF, the U.S. ARL, the U.S. Government, the U.K. Ministry of Defense or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

[1] Alexander J Quinn and Benjamin B Bederson. Human computation: a survey and taxonomy of a growing field. In *CHI*, pages 1403–1412. ACM, 2011.

[2] Aniket Kittur, Ed H Chi, and Bongwon Suh. Crowdsourcing user studies with mechanical turk. In *CHI*, pages 453–456. ACM, 2008.

[3] Michael S Bernstein, Greg Little, Robert C Miller, Björn Hartmann, Mark S Ackerman, et al. Soylent: a word processor with a crowd inside. In *UIST*, pages 313–322. ACM, 2010.

[4] Greg Little, Lydia B Chilton, Max Goldman, and Robert C Miller. Exploring iterative and parallel human computation processes. In *Proceedings of the ACM SIGKDD workshop on human computation*, pages 68–76. ACM, 2010.

[5] Omar F Zaidan and Chris Callison-Burch. Crowdsourcing translation: Professional quality from non-professionals. In *HLT*, pages 1220–1229. ACL, 2011.

[6] Aftab Khan, James Nicholson, Sebastian Mellor, Daniel Jackson, Karim Ladha, et al. Occupancy monitoring using environmental & context sensors and a hierarchical analysis framework. In *BuildSys*. ACM, 2014.

[7] Cong Zhang, Hongsheng Li, Xiaogang Wang, and Xiaokang Yang. Cross-scene crowd counting via deep convolutional neural networks. In *CVPR*. IEEE, 2015.

[8] Jim Robbins. Crowdsourcing, for the birds. http://www.nytimes.com/2013/08/20/science/earth/crowdsourcing-for-the-birds.html, 2013. [Online; accessed 16-Dec-2015].

[9] Enricko Lukman. Indonesian voters are crowdsourcing ballot counts to protect against election fraud. https://www.techinasia.com/kawal-suara-indonesia-voters-crowdsourcing-ballot-counts-protect-election-fraud, 2014. [Online; accessed 16-Dec-2015].

[10] Robin Wentao Ouyang, Lance Kaplan, Paul Martin, Alice Toniolo, Mani Srivastava, and Timothy J Norman. Debiasing crowdsourced quantitative characteristics in local businesses and services. In *IPSN*, pages 190–201. ACM, 2015.

[11] Xiaoxin Yin, Jiawei Han, and Philip S Yu. Truth discovery with multiple conflicting information providers on the web. *IEEE TKDE*, 20(6):796–808, 2008.

[12] Jacob Whitehill, Ting-fan Wu, Jacob Bergsma, Javier R Movellan, and Paul L Ruvolo. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *NIPS*, pages 2035–2043, 2009.

[13] Vikas C Raykar, Shipeng Yu, Linda H Zhao, Gerardo Hermosillo Valadez, Charles Florin, et al. Learning from crowds. *The Journal of Machine Learning Research*, 99:1297–1322, 2010.

[14] Peter Welinder, Steve Branson, Pietro Perona, and Serge J Belongie. The multidimensional wisdom of crowds. In *NIPS*, pages 2424–2432, 2010.

[15] Jeff Pasternack and Dan Roth. Knowing what to believe (when you already know something). In *COLING*, pages 877–885. Association for Computational Linguistics, 2010.

[16] Dong Wang, Lance Kaplan, Hieu Le, and Tarek Abdelzaher. On truth discovery in social sensing: a maximum likelihood estimation approach. In *IPSN*, pages 233–244. ACM, 2012.

[17] Bo Zhao, Benjamin IP Rubinstein, Jim Gemmell, and Jiawei Han. A bayesian approach to discovering truth from conflicting sources for data integration. *VLDB Endowment*, 5(6):550–561, 2012.

[18] Dong Wang, Tarek Abdelzaher, Lance Kaplan, and Charu C Aggarwal. Recursive fact-finding: A streaming approach to truth estimation in crowdsourcing applications. In *ICDCS*, pages 530–539. IEEE, 2013.

[19] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[20] Olivier Cappé and Eric Moulines. On-line expectation–maximization algorithm for latent data models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 71(3):593–613, 2009.

[21] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38, 1977.

[22] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Springer New York, 2006.

[23] D Michael Titterington. Recursive parameter estimation using incomplete data. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 257–267, 1984.

[24] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[25] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *NSDI*. USENIX Association, 2012.

[26] Matthew Marge, Satanjeev Banerjee, and Alexander I Rudnicky. Using the amazon mechanical turk for transcription of spoken language. In *ICASSP*, pages 5270–5273. IEEE, 2010.

[27] Takeshi Sakaki, Makoto Okazaki, and Yutaka Matsuo. Earthquake shakes twitter users: real-time event detection by social sensors. In *WWW*, pages 851–860. ACM, 2010.

[28] Sasank Reddy, Deborah Estrin, and Mani Srivastava. Recruitment framework for participatory sensing data collections. In *Pervasive Computing*, pages 138–155. Springer, 2010.

[29] Robin Wentao Ouyang, Animesh Srivastava, Prithvi Prabahar, Romit Roy Choudhury, Merideth Addicott, and F Joseph McClernon. If you see something, swipe towards it: crowdsourced event localization using smartphones. In *UbiComp*, pages 23–32. ACM, 2013.

[30] Alexander Philip Dawid and Allan M Skene. Maximum likelihood estimation of observer error-rates using the em algorithm. *Applied Statistics*, pages 20–28, 1979.

[31] Dong Wang, Md Tanvir Amin, Shen Li, Tarek Abdelzaher, Lance Kaplan, et al. Using humans as sensors: An estimation-theoretic perspective. In *IPSN*, pages 35–46. IEEE, 2014.

[32] Yukino Baba and Hisashi Kashima. Statistical quality estimation for general crowdsourcing tasks. In *KDD*, pages 554–562. ACM, 2013.

[33] Robin Wentao Ouyang, Mani Srivastava, Alice Toniolo, and Timothy J. Norman. Truth discovery in crowdsourced detection of spatial events. In *CIKM*, pages 461–470. ACM, 2014.

Robin Wentao Ouyang received the Ph.D. degree in electronic and computer engineering from Hong Kong University of Science and Technology (HKUST), Hong Kong, China, in 2011, and the B.E. degree in electronic engineering from Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in 2007. He is currently a postdoctoral associate with the Department of Computer Science, University of California, Los Angeles (UCLA), USA. His current research interests include human computation, data mining and mobile computing.

Lance M. Kaplan received the B.S. degree with distinction from Duke University, Durham, NC, in 1989 and the M.S. and Ph.D. degrees from the University of Southern California, Los Angeles, in 1991 and 1994, respectively, all in Electrical Engineering. From 1987-1990, Dr. Kaplan worked as a Technical Assistant at the Georgia Tech Research Institute. He held a National Science Foundation Graduate Fellowship and a USC Dean's Merit Fellowship from 1990-1993, and worked as a Research Assistant in the Signal and Image Processing Institute at the University of Southern California from 1993-1994. Then, he worked on staff in the Reconnaissance Systems Department of the Hughes Aircraft Company from 1994-1996. From 1996-2004, he was a member of the faculty in the Department of Engineering and a senior investigator in the Center of Theoretical Studies of Physical Systems (CTSPS) at Clark Atlanta University (CAU), Atlanta, GA. Currently, he is in the Networked Sensing and Fusion branch of the U.S. Army Research Laboratory (ARL). Dr. Kaplan serves as Editor-In-Chief for the IEEE Transactions on Aerospace and Electronic Systems (AES) and he is Vice President of Conference for the International Society of Information Fusion (ISIF). In addition, he also served on the Board of Governors of the IEEE AES Society, 2009-2014, and on the ISIF Board, 2012-2014. He served as Technical Co-Chair (with Neil Gordon) for the 2011 ISIF/IEEE International Conference on Information Fusion in Chicago, IL. From 2004-2014, he served as the Remote Sensing Co-Organizer (with Peter Kahn) for the IEEE Aerospace Conference in Big Sky, MT. He is a three time recipient of the Clark Atlanta University Electrical Engineering Instructional Excellence Award from 1999-2001. Dr. Kaplan has published over 180 technical articles. His current research interests include signal and image processing, information/data fusion, resource management, and network science.

Alice Toniolo is a postdoctoral researcher in the Computing Science Department at the University of Aberdeen (UK). Her interest is in computational models of argumentation for reasoning and dialogue. She was awarded her PhD in Computing Science by the University of Aberdeen (UK) in 2013.

Mani Srivastava received the B.Tech. degree from the Indian Institute of Technology Kanpur, Kanpur, India, in 1985, and the M.S. and Ph.D. degrees from the University of California Berkeley, Berkeley, in 1987 and 1992, respectively.

He was with Bell Laboratory Research, Murray Hill, NJ. He joined the University of California, Los Angeles, as a Faculty Member, in 1997, where he is currently a Professor of the Electrical Engineering and Computer Science Department. His current research interests include embedded systems, low-power design, wireless networking, and pervasive sensing.

Dr. Srivastava is affiliated with the National Science Foundation Science and Technology Center on Embedded Networked Sensing, where he co-leads the System Research Area. He currently serves as the Steering Committee Chair of the IEEE TRANSACTIONS ON MOBILE COMPUTING.

Timothy J. Norman is a professor of computing science at the University of Aberdeen, Scotland, UK. His research interests are in multi-agent systems, computational models of trust, policies and argumentation, and how these methods are applied to problems of information interpretation and management. Dr. Norman has a PhD in Computer Science from University College London, UK.