Foundations of Crowd Data Sourcing

Yael Amsterdamer and Tova Milo

{yaelamst, milo}@post.tau.ac.il
Tel Aviv University, Tel Aviv, Israel

ABSTRACT

Crowdsourcing techniques are very powerful when harnessed for the purpose of collecting and managing data. In order to provide sound scientific foundations for crowdsourcing and support the development of efficient crowdsourcing processes, adequate formal models must be defined. In particular, the models must formalize unique characteristics of crowd-based settings, such as the knowledge of the crowd and crowd-provided data; the interaction with crowd members; the inherent inaccuracies and disagreements in crowd answers; and evaluation metrics that capture the cost and effort of the crowd. In this paper, we review the foundational challenges in modeling crowd-based data sourcing, for its two main tasks, namely, harvesting data and processing it with the help of the crowd. For each of the two task types, we dive into the details of one foundational line of work, analyzing its model and reviewing the theoretical results established using this model, such as complexity bounds and efficient algorithms. We also overview a broader spectrum of work on crowd data sourcing, and highlight directions for further research.

1. INTRODUCTION

Crowd-based data sourcing is an emerging data procurement paradigm that engages Web users to collectively contribute and process information [8]. Well-known crowd platforms include Wikipedia,[1] websites of reviews and ratings such as IMDB,[2] and many more.

In order to work with the crowd, one has to overcome several challenges, such as dealing with users of different expertise and reliability, and whose time, memory and attention are limited; handling data that is uncertain, subjective and contradictory; and so on. Particular crowd platforms typically tackle these challenges in an ad-hoc manner, which is application-specific and rarely sharable. These challenges along with the evident potential of crowdsourcing have raised the attention of the scientific community, and called for developing sound foundations and provably efficient approaches to crowdsourcing. The present paper surveys *established theoretical foundations of crowdsourcing*, reviews a variety of approaches and results, and highlights remaining questions and directions for future research.

We start, in Section 2, by discussing an essential but challenging aspect of crowdsourcing, namely, *providing formal models for the crowd*. On the one hand, such models are necessary for studying the theoretical foundations of crowdsourcing. On the other hand, the behavior of the crowd may be very unexpected, and hard to formalize. We consider three aspects of crowdsourcing models, namely *the data model* (Section 2.1), *the interface with the crowd* (Section 2.2), and *the target function* of the crowdsourcing process (Section 2.3). For each of the three, we survey the different modeling considerations, and solutions that address them.

For example, consider a dietician that wishes to study the culinary habits of people in a certain population, e.g., combinations of food dishes that are consumed together with caffeine. The culinary habits within the population may not be recorded anywhere, and are individual, so the data must be collected by asking people about their habits. For this purpose, crowdsourcing is a notably powerful tool, as it enables a systematic exploration of a potentially huge data space (relevant combinations of food dishes) in a manner reminiscent of database exploration. Considering the underlying model for such an example, the data model must account for the unrecorded, personal habits of crowd members. One must also define the interface with the crowd, namely what types of questions are posed to crowd members (e.g., "When you eat X, how often do you also eat Y?"), what types of answers are expected (e.g., selecting a frequency for the habit in question from a drop-down list), and how the answers are analyzed by the system. The latter task is par-

[1] `http://en.wikipedia.org/`
[2] `http://www.imdb.com`

ticularly challenging, since the crowd can provide answers that are approximate, contradictory and even malicious. Last, the process of harvesting the culinary habits from the crowd must be guided by some target function. For example, since the crowd is an expensive resource, a typical target function is learning about the habits of crowd members while minimizing their effort. Additional considerations (such as the quality of the computed results) and the tradeoff between them are discussed in Section 2.3.

The crowd could be harnessed for various data-related tasks, which can be divided into two main types. First, the crowd could be engaged in *harvesting new or missing data*; and second, the crowd can *process data* that was already collected, by providing their judgments, comparing, cleaning and matching data items. Both types of tasks have been studied in previous literature, e.g., [3, 6, 7, 9, 12, 13, 14, 15, 17, 19, 24, 26, 28]. However, much of the work has been focused on the technical and practical aspects of crowdsourcing. To gain a deeper understanding of the theoretical foundations of crowdsourcing, we provide a more complete picture for two foundational research studies, one for harvesting data and one for processing it with the crowd.

In Section 3, we consider a recently developed approach for harvesting the crowd, namely, *crowd mining* [3], which focuses on identifying statistically significant patterns within the habits and preferences of the crowd. Crowd mining can be used, e.g., for identifying popular combinations of food dishes (useful to the dietician from our previous example). Crowd mining poses several theoretical challenges: first, its model must account for the individual data of the crowd which is mined in this process, as well as for a notion of *overall significant* data patterns in a given population (rather than in the habits of a single person). We outline this in Section 3.1. Second, similarly to standard data mining techniques, the space of patterns may be huge, and efficient algorithms are required for making this approach feasible. Due to the crowd involvement, the setting is quite different from that of standard data mining, and calls for the development of dedicated solutions, as described in Section 3.2.

To exemplify the second type of crowdsourced tasks, namely, data processing tasks, we consider in Section 4 the crowdsourced implementation of two classic database operators, top-k and group-by, which has been studied in [7]. For example, consider the grouping of photos of individuals by person and finding the most recent photo within each cluster, something which is easy for humans to do but difficult to evaluate by machines (assuming that person name tags and photo dates are unavailable or unreliable). One promising approach for performing such a task is with the help of the crowd. In Section 4.1, we describe the theoretical model defined in [7] for the crowd, and how the top-k and group-by operators can be evaluated with the crowd. Unlike crowd mining, where the considered data is individual, in this setting we assume that there is an (unknown) ground truth, i.e., the true top-k most recent photos of each person; however, this also means that crowd members may make mistakes, e.g., wrongly identify the most recent photo. We describe probabilistic models, which capture the likelihood of errors in crowd answers, as a function of the question difficulty. The target function is then minimizing the number of questions to the crowd, while guaranteeing a low (fixed) overall probability of error. In Section 4.2 we outline the complexity bounds of algorithms that achieve this target, and show that they are quite efficient.

We further discuss additional notable foundational works about crowd data sourcing, focused on data harvesting and processing, at Sections 3.4 and 4.3, respectively. We conclude in Section 5.

2. MODELING THE CROWD

We next discuss the challenges in providing an *adequate formal model* to crowdsourcing, which enables theoretical analysis and the development of efficient algorithms. The discussion is divided into the three aspects of crowdsourcing modeling mentioned in the Introduction, namely *data modeling*, modeling the *interface with the crowd* on top of this data model, and the *target function* of the entire process, which guides the interaction with the crowd.

2.1 Models of Data

The modeling of data in crowdsourcing leverages on existing models such as relational databases [7, 10, 16, 18, 20, 21], tree or graph-shaped data [17], RDF [2], and so on. In cases where the crowd is only employed to *process* the data (e.g., filter, group or sort), standard data models can be used as-is. The novelty here lies in cases when some of the data is harvested with the help of the crowd. One can generally distinguish between procuring two types of data: *general data* that captures "general truth" that typically resides in a standard database, e.g., the locations of places or opening hours; versus *individual data* that concerns individual people, such as their preferences or habits.

To capture harvested general data in crowdsourcing, one can use models of incomplete data, e.g., relational tables with missing values, rows and/or

columns, designated to be filled in by the crowd [20, 21]. In contrast, individual data is typically not recorded in a systematic manner, and can only be collected by posing questions to people. Such data can thus be modeled as per-crowd-member knowledge bases, which are not materialized and are accessed by restricted means of interacting with the crowd [1, 3]. See a concrete such model in Section 3.1.

2.2 Models of Crowd Interface

Interacting with the crowd can be done in various, potentially intricate ways. For instance, in Wikipedia crowd members can add, edit and delete encyclopedic entries, participate in discussions and more. However, to enable exact analysis of crowdsourcing algorithms, this interaction must be defined in a precise manner: what *types of tasks* can be submitted to the crowd, what is the *possible range of answers or actions* of the crowd in response, and how these are *interpreted* by the system. These should reflect, as accurately as possible, the expected behavior of the crowd. For example, common tasks to the crowd include classifying single items [6, 19, 23, 24], comparing sets of items [9, 7, 11, 29], and providing missing data items and values [10, 20, 21, 25] in domains where human judgment is required (e.g., ratings, recommendations, semantic analysis of images or text, etc.). The answers may be interpreted as samples of the full population's knowledge (which allows estimating this full knowledge), as votes (majority vote can be used to reconcile conflicting crowd answers), as uncertain data (see below) or as data annotated with trust levels (based on the crowd members that provided it).

In particular, *potential errors* in crowd answers must be accounted for [10, 14, 20]. This is typically done in crowdsourcing using error probability models. For example, in [7], the error model captures the increase in the error probability of increasingly difficult tasks, as described in Section 4.1. The error probability can be estimated based on preliminary tests with gold-standard data (where the ground truth is known) [6, 19] or by comparing the answers of different users to the same question [3, 14]. Error estimations for individual tasks allow estimating and adjusting the *overall* probability of error, by assigning tasks with higher uncertainty to more crowd members. See Section 4.1.

2.3 Models of the Target Function

Several factors affect the performance of crowd data sourcing processes.

- Traditional computational factors: *computational and space complexity*, *network load*, etc.
- Factors affected by the *number of tasks* posed to the crowd:
  - *Latency* – the response time of the crowd is relatively slow, and may be reduced by executing several tasks in parallel [20].
  - *Monetary cost* – crowd members may be paid for performing tasks, as in crowdsourcing platforms like Amazon Mechanical Turk.[3]
  - The *attention and amount of effort* of a crowd member may be limited. [4]
- Factors reflecting the *output quality*:
  - *Output errors* – may occur because of inaccurate crowd answers [7].
  - *Faithful representation of the trends in the target population* – relevant for the harvesting of individual data, where results are typically computed based on the answers of only a small fraction of the population [3].
  - *Coverage* – the number of collected or processed data items [20].

There exist natural tradeoffs between the aforementioned factors. E.g., computing the optimal questions to the crowd (which may be computationally expensive) may reduce the number of required questions [1]; or, executing many crowd tasks in parallel may reduce the overall latency but lead to the execution of redundant tasks [6, 20]. This means that any crowdsourcing process can only optimize some of the factors, depending on the application. Other factors may be fixed or bounded (e.g., the budget). In the sequel, we describe several target functions used in specific crowdsourcing processes.

3. HARVESTING DATA WITH THE CROWD

As promised in the Introduction, in this section we will exemplify the use of crowdsourcing for harvesting data from the crowd, by diving into the theoretical foundations of a particular paradigm, namely, **crowd mining**, which have been studied in [1, 2, 3]. We start by providing a general motivation for crowd mining, then detail the underlying formal model established in [1, 2, 3] and overview some of the main theoretical results. Throughout the section, we use the culinary preferences example from the Introduction as a running example, but note that crowd mining applies to many other real-life scenarios that involve collecting data about the habits and preferences of the crowd [3].

Classic data mining algorithms discover interesting patterns in large data sets. Such algorithms typically assume that the transactions to be mined (sets of co-occurring data items) are stored in a database. In contrast, individual data is typically

[3] `https://www.mturk.com/`

not recorded in a systematic manner for large populations, and thus crowdsourcing may be required for mining such data.

The individual knowledge of a people can be modeled per-person databases, accessed by posing questions to the relevant person. These databases cannot be materialized, since asking a person to recall and report every piece of knowledge is usually impossible. Instead, the model of [3] relies on the common ability of people to recall information about their habits in the form of summaries, as shown by social studies [5]. For instance, people can report how often they eat a specific combination of food dishes. Crowd mining methods thus ask crowd members to specify their habits and/or provide the frequency at which they engage in these habits. The answers of several users are aggregated to estimate the overall significance of a habit (data pattern) in the population.

In [3], a framework for estimating the confidence in the habit significance is described, and is employed for deciding which question to ask the crowd next (see a review below). In [1], this framework is extended by leveraging semantic connections between data items in order to efficiently explore the space of data patterns and identify the significant ones. E.g., if it is known that few people in a certain population drink coffee, and that espresso is a type of coffee, it is useless to ask people about their espresso drinking habits. Finally, in [2], the crowd mining model is extended to RDF-style facts[4] rather than sets of items, and a novel, rich query language allows specifying complex data patterns of interest. The sound theoretical results of [1] are proved to extend to the more expressive setting of [2].

3.1 Formal Model

We next describe a combined formal model for the crowd mining setting of [1, 3]. For simplicity, we discuss the most basic setting, and mention how it can be extended later, in Section 3.3.

Data model. Let $\mathcal{I} = \{i_1, i_2, i_3, \dots\}$ be a finite set of item names. Define a *database* D as a finite bag (multiset) of *transactions* over $\mathcal{I}$, s.t. each transaction $T \in D$ represents an occasion, e.g., a meal. We start with a simple model where every T contains an itemset $A \subseteq \mathcal{I}$, reflecting, e.g., the set of food dishes consumed in a particular meal. Let $\mathcal{U}$ be a set of users. Every $u \in \mathcal{U}$ is associated with a *personal database* D_u containing the transactions of u (e.g., all the meals in u's history). $|D_u|$ denotes the number of transactions in D_u. The frequency or *support* of an itemset $A \subseteq \mathcal{I}$ in D_u is $\text{supp}_u(A) := |\{T \in D_u \mid A \subseteq T\}| / |D_u|$. This individual significance measure will be aggregated to identify the overall frequent itemsets in the population. For example, in the domain of culinary habits, $\mathcal{I}$ may consist of different food dishes, drinks, etc. A transaction $T \in D_u$ will contain all the items in $\mathcal{I}$ consumed by u in a particular meal. If, e.g., the set {coffee, fruits, yogurt} is frequent, it means that these food and drink items form a frequently consumed combination.

As noted in [22], there may be dependencies between *itemsets* resulting from semantic relations between *items*. For instance, the itemset {chocolate, coffee} is semantically implied by any transaction containing {chocolate, espresso}, since espresso is a (type of) coffee. Such semantic dependencies can be naturally captured by a *taxonomy* [22]. Formally, we define a taxonomy Ψ as a partial order over $\mathcal{I}$, such that $i \leq i'$ indicates that item i' is more specific than i (any i' is also an i).

Based on $\leq$, the semantic relationship between items, we can define a corresponding order relation on itemsets.[5] For itemsets A, B we define $A \leq B$ iff every item in A is implied by some item in B. We call the obtained structure the *itemset taxonomy* and denote it by $\text{I}(\Psi)$. $\text{I}(\Psi)$ is then used to extend the definition of the support of an itemset A to $\text{supp}_u(A) := |\{T \in D_u \mid A \leq T\}| / |D_u|$, i.e., the fraction of transactions that *semantically imply* A.

Crowd interface. In our crowd-based setting, the personal database D_u is not materialized and only models the knowledge of u, so we can only access D_u by asking u questions. As shown in [3], one can ask people for summaries of their personal knowledge, and then interpret them as data patterns – itemset frequencies in our case. We thus abstractly model two types of *crowd questions*, as follows.

- **Closed question.** Parameterized by an itemset $A \subseteq \mathcal{I}$ and asks a user u for $\text{supp}_{D_u}(A)$.
- **Open questions.** Asks a user u to provide some itemset $A \subseteq \mathcal{I}$ along with $\text{supp}_{D_u}(A)$.

Intuitively, closed questions are useful for computing the significance of a particular itemset, whereas open questions are useful for discovering previously unknown items (if the items domain is not given in advance) and for quickly finding itemsets that represent prominent data patterns (which are more likely to be spontaneously recalled by users).

[4] `http://www.w3.org/standards/techs/rdf`

[5] Some itemsets that are semantically equivalent are identified by this relation, e.g., {coffee, espresso} is represented by the equivalent, more concise {espresso} (since drinking espresso is a particular case of drinking coffee), see [1] for full details.

		With respect to the input	With respect to the input and output
Crowd Complexity	**Lower**	$\Omega(\log \lvert S(\Psi)\rvert)$	$\Omega(\lvert mfi\rvert + \lvert mii\rvert)$
	Upper	$O(\log \lvert S(\Psi)\rvert)$	$O(\lvert\Psi\rvert \cdot (\lvert mfi\rvert + \lvert mii\rvert))$
Comput. Complexity	**Lower**	$\Omega(\log \lvert S(\Psi)\rvert)$	EQ-hard
	Upper	$O\Big(\lvert I(\Psi)\rvert \cdot \Big(\lvert\Psi\rvert^2 + \lvert I(\Psi)\rvert\Big)\Big)$	$O\Big(\lvert I(\Psi)\rvert \cdot \Big(\lvert\Psi\rvert^2 + \lvert mfi\rvert + \lvert mii\rvert\Big)\Big)$

Table 1: Summary of crowd mining complexity results, where $|I(\Psi)| \leq 2^{O(|\Psi|)}$ and $|S(\Psi)| \leq 2^{O(|I(\Psi)|)}$

Target function. To compute the *overall significance* of an itemset A, we need to *aggregate* the support of A in the databases of the users in U, and apply a *predicate* for deciding whether the result is significant or not. In [3], *average* is used for aggregation, and the itemset A is considered significant (frequent) if its support exceeds a predefined threshold Θ.

In practice, it is impossible to obtain the answers of all the users about a certain rule. Thus, in [3], a sample-based empirical estimation is employed by posing questions about each data pattern to a random (uniform) sample of the crowd members. In a nutshell, given a small set of user answers regarding a particular itemset, the technique of [3] estimates the *unknown distribution* of the mean of these answers (the aggregated result). As more answers are obtained from the crowd, this distribution converges to the true mean in the entire population. The decision whether an itemset is significant is done based on the probability that the mean support of this itemset exceeds the threshold Θ.

Based on the above setting, the target function of crowd mining can be defined in different ways. In [3], a greedy algorithm is considered, whose target is to choose the next crowd question that minimizes the *overall expected uncertainty* – the single optimization factor. The uncertainty is defined based on estimating, per data pattern, the probability that it was wrongly classified as (in)significant. An alternative target function considered in [1] aims to estimate the significance of *all the itemsets*, while minimizing first the number of questions posed to the crowd (termed *crowd complexity*), and then the computational complexity of selecting these questions. The uncertainty is assumed to be controlled by asking a sufficient number of users about each itemset. I.e., the error is bounded while the crowd and computational complexities are optimized.

3.2 Theoretical Results

The model for crowd mining sets the formal foundations for the development of provably efficient crowdsourcing algorithms, as well as studying their complexity bounds. We next overview such results for the basic model described above, which are fundamental and can be adapted for more complex setting (see Section 3.3). Formally, we define the problem of `CrowdMine` as follows: given a domain of items $\mathcal{I}$, a taxonomy Ψ over its items, find the frequent itemsets in the induced taxonomy of itemsets $I(\Psi)$, by posing closed questions to the crowd. (Open questions are further considered in [3, 2], but the theoretical results below only apply to close questions.) The following theorem summarizes the main complexity results for `CrowdMine` established in [1].

THEOREM 3.1 ([1]). *The complexity bounds of solving* `CrowdMine` *are stated in Table 1.*

In the first column of Table 1, complexity bounds are given in terms the input taxonomy Ψ. In contrast, in the second column, complexity bounds are given in terms of both the input and the output of the mining process, namely, the number of maximal (most specific) frequent itemsets (MFIs) and minimal (most general) infrequent itemsets (MIIs). Intuitively, the MFIs and MIIs are alternative concise descriptions of the frequent itemsets, and thus capture the output of the mining process.

The first row of Table 1 presents *crowd complexity* results, defined as the number of distinct itemsets the crowd is asked about (assuming questions are posed to a sufficient number of users to determine the itemset frequency). Given a taxonomy Ψ, `CrowdMine` has a tight bound logarithmic in $|S(\Psi)|$, the number of possible Boolean frequency functions, which depends on Ψ. As reflected in the inequalities at the bottom of Table 1, $\log |S(\Psi)|$ is at most exponential in $|\Psi|$. When the output is considered, the lower complexity bound is the sum of the numbers of MFIs and MIIs, and the upper bound adds the taxonomy size as a multiplicative factor (i.e., its complexity nearly but not exactly achieves the lower bound).

```
Algorithm
    Input: R: the most general data pattern
    Output: The set of MFIs M
1   Add R to an empty priority queue Q;
2   M ← ∅;
3   while Q contains unclassified data patterns
    do
4       A ← the minimal pattern in Q ;
5       if ask(A) then
6           while exists unclassified B s.t.
            A ≪ B do
7               if ask(B) then A ← B;
8           add A to M;
9   return M;
```

Algorithm 1: Crowd mining algorithm

This is proved by providing a constructive algorithm, outlined in the sequel.

The second row of the table, presents *computational complexity* bounds, of performing an optimal selection of the itemsets the crowd will be asked about throughout the mining process. The crowd complexity lower bound is trivially a lower bound of computational complexity, but w.r.t. the output a stronger hardness result is obtained by showing that the problem is EQ-hard in the taxonomy size and in the numbers of MFIs and MIIs. EQ is a basic problem in Boolean function learning, not known to be solvable in PTIME [1]. The upper bounds in the bottom row are achieved by the same algorithm that achieves the crowd complexity upper bound, and are polynomial in $|\mathrm{I}(\Psi)|$, and at most exponential in $|\Psi|$. See [1] for the proofs of these theoretical results.

The algorithm used in all the constructive[6] upper bound proofs in [1] is presented next in a slightly simplified manner in Algorithm 1, based on its extension from [2]. Although the algorithm is simple, it is both complexity-wise efficient, useful in practice, and can be adapted for more complex settings [2].

Algorithm 1 maintains a priority queue Q where itemsets are ordered from the most general to the most specific. Q is initialized with the single most general pattern. Iteratively, the algorithm pops out an unclassified data pattern A from Q, which is not known to be (in)significant yet. To compute whether the pattern is significant, the algorithm uses the function ask$(\cdot)$, which abstracts, for simplicity, the process of posing questions about A to several crowd members. The algorithm then explores subsequent patterns in the partial order (using the $\ll$ relation), searching for patterns that are both more specific and significant, until it discovers an MFI. Every call to ask$(\cdot)$ can classify multiple patterns, using *semantic inference*: if $A \leq B$ and B is significant, so is A; and vice versa.

The number of MFIs and MIIs is typically much smaller than the number of itemsets [1]. Recall that the crowd complexity of Algorithm 1 described above is $\mathrm{O}(|\Psi| \cdot (|mfi| + |mii|))$. Intuitively, the inner loop of the algorithm can check at most $|\Psi|$ itemsets, since we can attempt add each item to the current itemset at most once (and if this results in an insignificant itemset B, all the more specific itemsets C s.t. $B \leq C$ are inferred to be insignificant). Since each loop identifies an MFI or an MII (if the first itemset popped from Q is insignificant), we obtain the crowd complexity bound mentioned above.

[6]The exception is the upper crowd complexity w.r.t. the input size, whose proof is a non-constructive one.

3.3 Extensions

The simple model of mining itemsets cannot capture all types of interesting patterns that one might want to mine. E.g., it can express sets of co-occurring items, but not more complex relationships between them. This model is thus extended in [3] to mining *association rules*: given two itemsets A and B, $A \rightarrow B$ is an association rule denoting an implication between A and B, which can be viewed as a likelihood of transactions which contain A to also contain B. For example, $\{\mathsf{coffee},\mathsf{fruits}\} \rightarrow \{\mathsf{yogurt}\}$ can signify that whenever people have coffee and fruits, they are likely to also have yogurt. In [2], even more complex data patterns are considered, namely sets and association rules of RDF-style *facts*. Given a vocabulary of items $\mathcal{I} = \mathcal{E} \cup \mathcal{R}$, a fact $f \in \mathcal{E} \times \mathcal{R} \times \mathcal{E}$ denotes a relationship $r \in \mathcal{R}$ between two items $e_1, e_2 \in \mathcal{E}$, e.g., the fact Coffee drinkAt Starbucks denotes a habit of drinking coffee in Starbucks. The semantic order relation between itemsets is extended in [2] to support sets of facts.

Algorithm 1 supports identifying MFIs within the entire space of itemsets. However, people often may be interested only in specific patterns. To restrict the search, the work of [2] introduces a query language called `OASSIS-QL`, which allows to specifying subset of data patterns that are of interest to the user, and mining only them from the crowd. Extending our example from the beginning of the section, the dietician may only be interested in studying culinary preferences with respect to dishes with caffeine, or meals eaten at restaurants in a certain region. Using a syntax based on SPARQL, the RDF query language, `OASSIS-QL` allows selecting only the relevant food fishes/ restaurants from a knowledge base. The selected items are then used to generate rel-

evant habits on which questions are posed to the crowd. It is proved in [2] that given a query, a notion equivalent to MFI and MII can be defined over the patterns that match it; then, Algorithm 1 is adapted to support finding such MFIs and MIIs, which gives a complexity bound similar to the one obtained in [1] for itemsets and without a query.

3.4 Harvesting General Data

The model and algorithms described above only deal with harvesting individual data from the crowd. Another line of work in crowdsourcing considers harvesting general data from the crowd. For instance, some recent work (e.g., [10, 16, 20, 21]) suggests the construction of declarative frameworks, which outsource the harvesting of certain missing values or missing tuples to the crowd. While most of this work focuses on the practical and technical aspects of the problem, we mention here two example works that focus on the theoretical aspects of general data collection with the crowd.

The first is [4], which makes the closed-world assumption that there is a known set of missing values. Each value could be fetched by posing a specific question to a sufficient (fixed) number of crowd members. The paper considers several target functions, including minimizing quality-related factors such as the max or the sum of uncertainties over all the values, while fixing cost-related parameters such as the overall number of questions asked, the maximal number of questions per user, etc. The results in [4] establish the complexity bounds of this problem and include constructive, efficient algorithms where possible. In particular, all the considered variants of the problem are proven to be in PTIME, except for optimizing the sum of uncertainties given a bounded number of questions per-user, or when the users are asked groups of overlapping questions. These exceptions are proved NP-hard.

The second work we mention is [25], which studies the harvesting of data under an open-world assumption. Many answers for a given question are collected from the crowd, e.g., the crowd can be asked to list ice-cream flavors. The key observation is that the more answers are collected, the next answer is less likely to provide a new value not obtained before. For example, after the prominent ice-cream flavors has been collected (e.g., vanilla, chocolate) a crowd member is less likely to spontaneously choose a flavor that is not in the list. To address this phenomenon, statistical tools are developed in [25], for estimating the future rate of incoming new values based on the rate observed thus far. These tools support various target functions that balance the cost (number of questions) versus coverage (number of unique values obtained) tradeoff.

4. DATA PROCESSING WITH THE CROWD

In addition to harvesting data from the crowd, the crowd has been proved to be very effective in tasks of processing data, such as cleaning, sorting and matching data items (e.g., [9, 11, 15, 23, 24, 27, 29]). In particular, some recent work (e.g., [6, 7, 12, 14, 15, 17, 19, 26, 28]), including some of the declarative crowdsourcing frameworks mentioned before [10, 16, 20], consider the execution of common query operators such as filter, join, count and max. As an example, we elaborate, in this section, on the theoretical foundations of computing **top-k and group-by queries**, studied in [7].

Suppose we have a database of unlabeled photos, `PhotoDB`, and we are interested in executing the following query over it.

```
SELECT Most-recent(photo)
FROM PhotoDB
GROUP BY Person(photo)
```

`PhotoDB` contains only photos of a single person (whose face can be recognized), e.g., Alice in her office or Bob in front of the Louvre, but not of Alice and Bob together. We now wish to: (i) group (cluster) the photos by the person they represent; and (ii) find the most recent photo – or the k most recent photos – of each person (max/top-k).

The query above includes two user-defined functions: `Person` clusters photos of the same person, and `Most-recent` selects the most recent photo within each cluster. Note that the most recent photo is also the one in which the photographed person is the oldest. While the `Person` function might be performed by face recognition software, it is slow and costly. Furthermore, the results may not be impressive for a time span of 20 years during which the person ages from babyhood to being an adult. As for ordering the photos by date, we assume there is no trustworthy date of when the photo was taken. This may be the case due to untuned dates in the camera, scanned photos, etc. Thus, we ask crowd members to identify the person in the photo and compare the photo dates based on the person's age and perhaps other cues.

4.1 Formal Model

The data model in this setting is a standard one, namely, a repository of photos. The user-defined functions we wish to execute with the help of the crowd may be viewed as fetching meta-data about the photos. We next detail the crowd interface

Max/Top-k		Clustering		Correlated Clustering
$\Theta\big(n \log \frac{1}{\delta}\big)$	(constant error)	$\Omega(nJ)$		$\mathrm{O}\big((n \log(\alpha J) + \alpha J) \log \frac{n}{\delta}\big)$
$n + \mathrm{o}\big(\frac{n}{\delta} \log \frac{1}{\delta}\big)$	(variable error in general)	$\mathrm{O}\big(nJ \log \frac{n}{\delta}\big)$	(in general)	
$n + \mathrm{O}\Big(\frac{\log \log n}{\delta^2} \log \frac{1}{\delta}\Big)$	(variable error, $f(\Delta) = \Omega(\Delta)$)	$\mathrm{O}(nJ)$	(when $\varepsilon = \frac{1}{2}$)	
$n + \mathrm{O}\big(\log^2 \frac{1}{\delta}\big)$	(variable error, $f(\Delta) = 2^{\Delta}$)			

Table 2: Summary of top-k and group by complexity results; k is constant; n is the database size; δ is the maximal error; $f(\Delta)$ is the error as a function of the distance between items Δ; J is the number of clusters; $\frac{1}{2} - \varepsilon$ is the user error; α is a correlation factor.

model assumed in this setting, along with the model of crowd errors.

Crowd interface. Two types of questions are used to respectively group and order data items (photos).

- **Type question.** Given two data items i, i' return `TRUE` iff they belong to the same type, e.g., the two photos are of the same person.
- **Value questions.** Given two data items i, i' return `TRUE` iff $i < i'$ by the order relation over items, e.g., the photo i was taken before i'.

Given a perfect oracle that answers these questions, it is clearly possible to group the data items and partially sort them to find the top-k. However, the crowd may make mistakes, i.e. they may not correctly identify two pictures as being of the same person (type error) or of one picture of a person being more recent than another picture of that same person (value error). Two models of error are considered in [7], as follows.

- **Constant error model:** Each type or value question is answered correctly by the crowd with a constant probability $> \frac{1}{2}$.
- **Variable error model:** Models an increase in error probability for items that are closer in the considered ordering.

While the constant error model is more standard (and used in used in previous crowdsourcing work, e.g., [6, 19]), the variable model is novel. It captures the intuition that the closer items are in the order relation, the harder it is for a crowd member to correctly order them. It is much easier, e.g., to compare the dates of two photos of the same person with 10 years apart than photos with one week apart, and the error probability increases accordingly.

Target function. Using either of the error models mentioned above, it is impossible to guarantee perfect clustering or top-k selection. Hence, the techniques described in [7] focus on bounding the total error probability: given an arbitrarily small constant $\delta \in (0, \frac{1}{2})$, the techniques compute the correct answer with probability $> 1 - \delta$. Within this bound, the techniques aim to perform the grouping and/or top-k selection tasks while minimizing the number of questions posed to the crowd.

4.2 Theoretical Results

Let us formally define the problems of computing top-k and group by queries with the crowd: let n be the number of items in the database; k a constant; and $\delta > 0$ the maximal allowed overall probability of error. `TOPk` is the problem of computing the top-k items in the database, `CLUSTER` is the problem of clustering the n items, and `CCLUSTER` is the problem of clustering when the clusters are correlated with an order over the data items. For the three problems, the correct answer must be computed in probability $\geq 1 - \delta$, and the complexity is measured by the number of questions posed to the crowd. The next theorem summarizes the main results for these problems, achieved in [7].

THEOREM 4.1 ([7]). *The complexity bounds of solving `TOPk`, `CLUSTER` and `CCLUSTER` are stated in Table 2.*

We next briefly explain these results.

`TOPk`. It is proved in [7] that, when each value question is answered correctly with probability $\frac{1}{2} + \varepsilon$ for a constant ε, the maximum can be computed with probability $\geq 1 - \delta$ in time $\mathrm{O}\big(n \log \frac{1}{\delta}\big)$; they also show that this bound is tight. In the novel variable error model a much better upper bound can be obtained: suppose the two elements being compared by a value question are Δ apart in the sorted order. Then the probability of error is $\leq \frac{1}{f(\Delta)}$ for a monotone, non-negative error function f, i.e., the error in the answer decreases when the distance Δ between the elements increases. For `TOPk` (with a constant k), $n + \mathrm{o}(n)$ value questions are sufficient given any strictly monotone error function f, where

$o(n)$ denotes a function that is strictly asymptotically smaller than n.

CLUSTER. For the general clustering problem using the fixed cost model, a lower bound of $\Omega(nJ)$ is achieved for discovering J clusters. For the upper bound, a simple algorithm that compares $O(nJ)$ pairs of elements by type questions proves that the lower bound is tight when $\varepsilon = \frac{1}{2}$; if the answers to the type questions are erroneous ($\varepsilon < \frac{1}{2}$), the number of questions increases by a factor of $O(\log n)$.

CCLUSTER. Consider the case when item types are correlated with an order over item values. E.g., suppose we have a database of hotels in a city, to be clustered by districts. Then there may be a high correlation between the district and the hotel rating. Formally, let $\alpha \in [1, n-1]$ be the correlation factor between the value and type (order and clustering), defined as follows: sort the items according to their value; the *distance* between elements x_i and x_j of type T is $|\{x_l | x_i < x_l <, x_l \text{ is not of type } T\}|$. α is the maximal such distance $+1$. When α is small the correlation can be leveraged to reduce the clustering complexity [7].

4.3 Additional Crowdsourced Operators

To complete the picture, we note a few more example works about other data processing operators. The problem of discovering the *maximum element* in a set of data items, which is a sub-problem of [7] (top-1), was also studied in [12]. However, instead of finding the maximum element exactly, [12] focuses on the judgment problem (which element has the maximum likelihood of being the maximum element) and the next vote problem (which future comparisons will be most effective). Finding the exact solution to these problems was shown to be hard, and instead, efficient heuristics were proposed.

The crowd sourcing of *filter* or *selection* operator, namely, using humans to provide a binary (or, in general, n-ary) evaluation of data items, was studied in [19], and later extended in [18]. Such an operator can be used, e.g., to filter images in a database, by asking crowd members to evaluate each image. The main challenges addressed in this work include the modeling the error of the crowd, and computing a strategy for deciding whether to pose some question to more people or make a decision. The model of [19] assumes that the error probabilities are fixed and known, which is later relaxed in [18] to support per-worker and per-question error probabilities. The complexity bounds of the problem were studied in [18, 19] for various target functions, which balance cost and accuracy in different ways.

We also note the work of [17] about crowd-assisted *searching of elements* within a directed acyclic graph, by asking reachability questions. For example, this operator can be used for classifying documents within a hierarchy of categories, by asking "Does document X belong to category Y?", which is interpreted as "Is the most specific category of X reachable from Y?", assuming that directed edges signify category subsumption. Two target functions for this problem were considered in [17], of either minimizing the number of questions while obtaining full coverage, or maximizing the accuracy within a fixed budget of questions. The complexity bounds of computing which questions to ask were studied for different properties of the graph structure, and efficient algorithms were proposed, where possible.

5. CONCLUSION

In this paper, we discussed the foundations of crowd data sourcing, focusing on its two main uses, namely, harvesting data and processing it with the crowd. We have highlighted the various challenges in providing adequate formal models for the crowd, and the general components of existing solutions that alleviate these challenges. We have then reviewed in more detail a few particular examples for crowdsourcing works, in light of their specific challenges, solutions and established results. We note that many other related challenges were considered by literature on crowdsourcing, which are out of the scope of this paper. For instance, these include the construction of data queries with the help of the crowd; the management of reward offered to crowd members; the assignment of tasks based on crowd member expertise and availability; and so on.

The survey of crowdsourcing works reveals some interesting challenges for future work. In particular, harvesting both individual and general data together has not been considered, and may be beneficial, e.g., for dynamically refining a general knowledge base according to harvested individual data. In addition, crowdsourcing problems are typically studied with respect to a predefined target function. A more generic solution could employ a parameterizable target function, to support customizing the balance between different cost and quality factors. Other challenges include dealing with privacy issues; the selection of crowd members according to their profiles; the design of user interface; and the maintenance of crowd-provided data, to account for trust, staleness, etc. Finally, richer crowd and error models that capture, more fully, different aspects of user behavior are also an intriguing research direction.

Acknowledgments. We are very grateful to Antoine Amarilli, Susan B Davidson, Yael Grossman, Sanjeev Khanna, Slava Novgorodov, Sudeepa Roy, Pierre Senellart and Amit Somech, our collaborators on the main work surveyed in this paper. We also thank the anonymous reviewer for useful comments.

This work has been partially funded by the European Research Council under the FP7, ERC grant MoDaS, agreement 291071, and by the Israel Ministry of Science.

6. REFERENCES

[1] A. Amarilli, Y. Amsterdamer, and T. Milo. On the complexity of mining itemsets from the crowd using taxonomies. In *ICDT*, 2014.

[2] Y. Amsterdamer, S. B. Davidson, T. Milo, S. Novgorodov, and A. Somech. OASSIS: query driven crowd mining. In *SIGMOD*, 2014.

[3] Y. Amsterdamer, Y. Grossman, T. Milo, and P. Senellart. Crowd mining. In *SIGMOD*, 2013.

[4] R. Boim, O. Greenshpan, T. Milo, S. Novgorodov, N. Polyzotis, and W.-C. Tan. Asking the right questions in crowd data sourcing. In *ICDE*, 2012.

[5] N. Bradburn, L. Rips, and S. Shevell. Answering autobiographical questions: the impact of memory and inference on surveys. *Science*, 236(4798), 1987.

[6] A. Das Sarma, A. G. Parameswaran, H. Garcia-Molina, and A. Y. Halevy. Crowd-powered find algorithms. In *ICDE*, 2014.

[7] S. B. Davidson, S. Khanna, T. Milo, and S. Roy. Using the crowd for top-k and group-by queries. In *ICDT*, 2013.

[8] A. Doan, R. Ramakrishnan, and A. Y. Halevy. Crowdsourcing systems on the World-Wide Web. *Commun. ACM*, 54(4), 2011.

[9] J. Fan, M. Lu, B. C. Ooi, W. Tan, and M. Zhang. A hybrid machine-crowdsourcing system for matching web tables. In *ICDE*, 2014.

[10] M. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. CrowdDB: answering queries with crowdsourcing. In *SIGMOD*, 2011.

[11] C. Gokhale, S. Das, A. Doan, J. F. Naughton, N. Rampalli, J. W. Shavlik, and X. Zhu. Corleone: hands-off crowdsourcing for entity matching. In *SIGMOD*, 2014.

[12] S. Guo, A. G. Parameswaran, and H. Garcia-Molina. So who won?: dynamic max discovery with the crowd. In *SIGMOD*, 2012.

[13] X. Liu, M. Lu, B. C. Ooi, Y. Shen, S. Wu, and M. Zhang. CDAS: A crowdsourcing data analytics system. *PVLDB*, 5(10), 2012.

[14] A. Marcus, D. Karger, S. Madden, R. Miller, and S. Oh. Counting with the crowd. In *VLDB*, 2012.

[15] A. Marcus, E. Wu, D. R. Karger, S. Madden, and R. C. Miller. Human-powered sorts and joins. *PVLDB*, 5(1), 2011.

[16] A. Marcus, E. Wu, S. Madden, and R. C. Miller. Crowdsourced databases: Query processing with people. In *CIDR*, 2011.

[17] A. Parameswaran, A. Sarma, H. Garcia-Molina, N. Polyzotis, and J. Widom. Human-assisted graph search: it's okay to ask questions. *PVLDB*, 4(5), 2011.

[18] A. G. Parameswaran, S. Boyd, H. Garcia-Molina, A. Gupta, N. Polyzotis, and J. Widom. Optimal crowd-powered rating and filtering algorithms. *PVLDB*, 7(9), 2014.

[19] A. G. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom. CrowdScreen: algorithms for filtering data with humans. In *SIGMOD*, 2012.

[20] A. G. Parameswaran, H. Park, H. Garcia-Molina, N. Polyzotis, and J. Widom. Deco: declarative crowdsourcing. In *CIKM*, 2012.

[21] H. Park and J. Widom. CrowdFill: collecting structured data from the crowd. In *SIGMOD*, 2014.

[22] R. Srikant and R. Agrawal. Mining generalized association rules. In *VLDB*, 1995.

[23] M. Stonebraker, D. Bruckner, I. F. Ilyas, G. Beskales, M. Cherniack, S. B. Zdonik, A. Pagan, and S. Xu. Data curation at scale: The Data Tamer system. In *CIDR*, 2013.

[24] C. Sun, N. Rampalli, F. Yang, and A. Doan. Chimera: Large-scale classification using machine learning, rules, and crowdsourcing. *PVLDB*, 7(13), 2014.

[25] B. Trushkowsky, T. Kraska, M. J. Franklin, and P. Sarkar. Crowdsourced enumeration queries. In *ICDE*, 2013.

[26] P. Venetis, H. Garcia-Molina, K. Huang, and N. Polyzotis. Max algorithms in crowdsourcing environments. In *WWW*, 2012.

[27] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. CrowdER: crowdsourcing entity resolution. *PVLDB*, 5(11), 2012.

[28] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng. Leveraging transitive relations for crowdsourced joins. In *SIGMOD*, 2013.

[29] C. Zhang, L. Chen, H. V. Jagadish, and C. Cao. Reducing uncertainty of schema matching via crowdsourcing. *PVLDB*, 6(9), 2013.