Crowdsourcing for Data Management: a Survey

DOI:
10.1007/s10115-017-1057-x

Document Version
Accepted author manuscript

Link to publication record in Manchester Research Explorer

Citation for published version (APA):
Crescenzi, V., Fernandes, A. A. A., Merialdo, P., & Paton, N. W. (2017). Crowdsourcing for Data Management: a Survey. *Knowledge and Information Systems*, 1-41. https://doi.org/10.1007/s10115-017-1057-x

Published in:
Knowledge and Information Systems

Knowledge and Information Systems manuscript No.
(will be inserted by the editor)

Crowdsourcing for Data Management: a Survey

**Valter Crescenzi · Alvaro A.A. Fernandes ·
Paolo Merialdo · Norman W. Paton**

Received: date / Accepted: date

Abstract Crowdsourcing provides access to a pool of human workers who can contribute solutions to tasks that are challenging for computers. Proposals have been made for the use of crowdsourcing in a wide range of data management tasks, including data gathering, query processing, data integration and cleaning. We provide a classification of key features of these proposals, and survey results to date, identifying recurring themes and open issues.

Keywords Data Management · Crowdsourcing

1 Introduction

In *outsourcing*, work is contracted out to a specific third party. In *crowdsourcing*, work is made available to potentially numerous third parties, so that tasks are carried out by potentially distributed networked workers, referred to as a *crowd* [46]. Crowdsourcing is an emerging area, with relationships to other fields, such as social computing, that combine human intelligence and computational techniques to complete tasks that are challenging for computers [86]. As such, the potential roles of crowdsourcing, and the most effective models for its application, remain to be determined. Nonetheless, there has been significant interest both in generic issues relating to crowdsourcing (such as experiment design [79], crowd marketplaces [51] and information quality [52]), and in its application to different domains (as diverse as language transcription [70] and behavioral research [71]).

Valter Crescenzi · Paolo Merialdo
Dipartimento di Ingegneria
Università degli Studi Roma Tre
Via della Vasca Navale, 79
00146 – Roma (Italy)
E-mail: crescenz, merialdo@dia.uniroma3.it

A.A.A. Fernandes · N.W. Paton
School of Computer Science,
University of Manchester,
Manchester M13 9PL, UK
E-mail: alvaro.a.fernandes,norman.paton@manchester.ac.uk

In data management, proposals have been made for crowdsourcing support for different stages of a data management lifecycle, such as data extraction (e.g. [22]), integration (e.g. [72]), cleaning (e.g. [100]), querying (e.g. [37]) and analysis (e.g. [65]). As a result, there is now a significant literature describing the application of largely independently-developed crowdsourcing techniques across a range of long-standing data management challenges, each addressing specific features and encountering issues that transcend individual problems. It seems timely, therefore, to take a step back from the individual proposals with a view to characterizing contributions to date and eliciting their distinctive features and shared concerns. We aim to review the state-of-the-art in crowdsourcing for data management by:

1. providing a classification of features that characterize proposals for the use of crowdsourcing in data management (Section 2);
2. applying the classification in (1) to a wide range of proposals addressing different parts of the data management lifecycle (Section 3);
3. providing detailed descriptions of selected proposals to illustrate the state-of-the-art in a range of areas (Section 3); and
4. identifying areas in which the experience in (2) has yielded lessons that can be more widely applied, and proposing areas that seem to require further investigation (Section 4).

Crowdsourcing systems vary in terms of the level of skills and engagement required from workers, the relationship between the task commissioners and the workers, the types of task undertaken, and the forms of reward obtained. In data management, most of the research has investigated the use of crowdsourced microtasks, as supported by platforms such as Amazon Mechanical Turk (AMT)[1] and CrowdFlower[2]. Such platforms support tasks that may come from different domains, come from a dynamically varying collection of commissioners, rarely require workers to contribute specific expertise, normally require little time to complete, and generally give rise to financial rewards. Here we focus principally on the sort of crowdsourced microtasks that are supported by such platforms, as these have provided the context for most work of relevance to data management.

There have been several other surveys on different aspects of crowdsourcing. Several of these have been broader in scope. Quinn and Bederson [86] relate crowdsourcing to other forms of human computation, and classify proposals in terms of the motivation for engagement, quality control techniques, ways of aggregating results from multiple participants, and the relationship between human and computational processes. Doan *et al.* [30] review crowdsourcing systems on the web, classifying proposals in terms of the nature of the worker engagement with the task, the role of the worker and the problem addressed, with additional consideration of issues such as worker recruitment and evaluation. Yuen *et al.* [106] characterize crowdsourcing systems in terms of application domain, the nature and motivation of the worker, and quality management. Amsterdamer and Milo [5] concentrates on theoretical foundations of crowdsourcing. The book by Marcus and Parameswaran [67] analyses the adoption of crowdsourcing by industry and illustrates academic proposals for a practitioner target. There have also been some more focused surveys. Zhang *et al.* [111] discusses the impact of crowdsourcing on

[1] www.mturk.com/mturk

[2] crowdflower.com

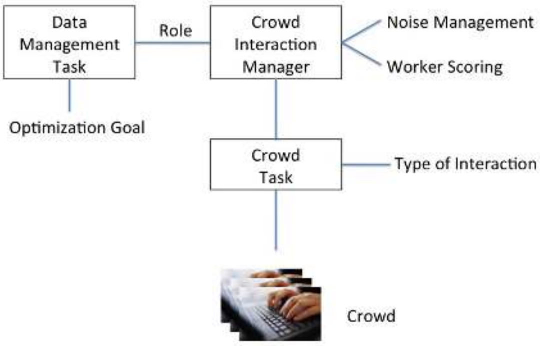


Fig. 1 The dimensions and their relationships.

machine learning, considering both techniques for inferring the ground truth labelling from crowds that contain unreliable workers, and learning models for use with crowdsourced labeled data. Techniques for crowd labelling are also reviewed and evaluated by Muhammadi, *et al.* [76]. These focused surveys drill down on techniques that can be applied to different application domains. The survey by Li, *et al.* [63] discusses how a range of cross-cutting issues such as worker modelling, cost control and latency control have been applied in data management. As such, the survey of Li, *et al.* [63], has a similar scope to that presented here, but with a greater emphasis on the generic techniques that have been applied in data management, and a particular emphasis on operators. This survey is organised around the data management task supported, is wider ranging in the types of system considered, and provides focused case studies of state-of-the-art proposals that support different task types.

2 Dimensions

To support the systematic comparison of proposals, this section introduces a collection of dimensions (and corresponding values) in crowdsourcing for data management, thereby allowing key features of different proposals to be captured using a consistent terminology. The dimensions are summarized in Figure 1 and their values are listed in Table 1.

In Figure 1, the *Data Management Task* is the activity that is being undertaken, to which crowdsourcing is to contribute. These activities, such as data integration or data querying, typically pre-exist crowdsourcing, but have been adapted to exploit the opportunities that crowdsourcing presents. A crowd worker then contributes to the completion of the data management task in some way, for example by providing or labelling data. Thus a crowd worker is contracted to play a *Role* in a data management task. The *Crowd Interaction Manager*, has responsibility for carrying out crowdsourcing on behalf of a *Data Management Task*. The Crowd Interaction Manager must collect requests for crowd workers together into *Crowd Tasks* (sometimes known as Human Intelligence Tasks, or HITs), controlling which

Table 1 Dimensions and their values.

	Dimension	**Values**
Problem	Data Management Task	Data gathering Data integration Data cleaning and validation Operator evaluation Querying Search
	Optimization Goal	Cost Latency Quality
Interaction	Role of the Crowd	Supply data Label data Verify work
	Type of Interaction	Confirm value Select value Provide value
	Noise Management	Majority voting Iterative Error-model
	Worker Scoring	Constant error-rate Control queries Redundant queries Worker comparison Reputation
	Task Composition	Online Offline

workers to employ and when crowd results are considered to be appropriately reliable.

The dimensions are organized over two main categories in Table 1: *problem* and *interaction*. The former groups features that characterize the problem that is addressed with the support of the crowd. The latter aims at identifying characteristics of the interaction with the crowd.

2.1 Data Management Task

The *Data Management Task* within crowdsourcing characterizes the broad objective of the problem tackled by the proposal. We use the following terms to characterize a data management task:

- *data gathering*: the procurement of values that contribute to the pool of instances over which an application acts (e.g. this phase includes tasks such as data extraction);
- *data integration*: the alignment of information from multiple data sources (e.g. this phase includes tasks such as schema matching);
- *data cleaning and validation*: the refinement of the data set to address data quality issues (e.g. this phase includes tasks such as de-duplication);
- *operator evaluation*: the evaluation of a data management operator, such as counting the elements in a collection that have some property, or filtering a collection;

- *querying*: the evaluation of declarative requests where the data accessed or the conclusions drawn are informed by the crowd;
- *search*: the evaluation of keyword queries, potentially over structured sources.

As such, the *data management task* represents the functional requirements of the application that is making use of the crowd. Proposals have been made for the use of crowdsourcing for all of these types of task, and thus for a wide range of data management activities. For the most part, crowdsourcing does not carry out complete data management tasks, but rather is used alongside computational techniques to refine their behavior or results.

2.2 Optimization Goals

Crowdsourcing is most widely applied to address issues where humans perform tasks that cannot be efficiently expressed as algorithms. However, designing solutions that exploit crowdsourcing in a data management task is not trivial because several aspects that deal with the involvement of humans in the process need to be considered: (*i*) people can be subjective and error-prone; (*ii*) people take longer than computers to perform tasks; and (*iii*) people are paid for their work.

Overall, crowdsourcing solutions for data management tasks trade off optimization among:

- *cost*: the total monetary rewards for the workers;
- *latency*: the time interval to obtain the results from the submission of the task;
- *quality*: the quality of the results, such as precision and recall in an entity resolution task, or correctness of values in a data gathering task.

As such, the *optimization goals* represent the non functional requirements of the application that is making use of the crowd, and must be taken into account by the solution. Balancing latency, cost and quality of the results is crucial to the design of effective and efficient data processing algorithms that rely on workers recruited by crowdsourcing platforms. It is important to observe that these three aspects are strongly correlated, and influence the design of the crowd interaction. For example, to reduce latency, parallel tasks can be submitted, or the reward can be increased. However, such strategies may strongly impact on costs. Similarly, to improve the quality of the results and overcome human errors, more workers can be employed on the same task, but this strategy also impacts on costs.

While some solutions consider tradeoff optimizations that involve all these three aspects, other solutions involve just a pair of them, assuming the third as an orthogonal concern or simply ignoring it.

2.3 Role of the Crowd in the Solution

Usually a crowdsourcing task is part of a wider activity and assumes a specific role in the data processing workflow. We consider the following roles:

- *crowds supply data to a system*: the provision of values that are missing in a data set (e.g., providing or completing tuples in a table [85]);

– *crowds label data*: the data to be labelled is typically produced computationally, and the labelled data used as part of a learning process (e.g., candidate tuple pairs are labelled as correct or incorrect matches, for training matchers in entity resolution [42]);
– *crowds verify work produced by other workers or by an algorithm*: interim or candidate conclusions are subject to review (e.g., within schema matching to verify inferred types or constraints [72]).

2.4 Type of Interaction

The *Type of Interaction* dimension indicates the type of activity carried out. Reflecting current practice, we use as values:

– *confirm value*: given a proposal from the system, the worker judges whether it is correct (e.g. *Did Bob Dylan write To Ramona?*);
– *select value*: given a question that admits a set of answers provided by the system, the worker is asked to select the correct answer[3] (e.g. *What is the music genre of the song To Ramona? Country | Folk | Grunge*);
– *provide value*: the worker is asked to provide a missing value (e.g. *Who wrote To Ramona?*).

While the *Data Management Task* defines the problem addressed with the support of the crowd, the *Type of Interaction* defines the kind of activity that crowd workers perform. For example, if the *Data Management Task* is *data cleaning and validation* and the worker is shown candidate equivalent entities, then the *Type of Interaction* is *confirm value*.

2.5 Noise Management

The benefit of crowdsourcing depends on the quality of the information obtained. Crowd-provided information is subject to a great degree of uncertainty: workers can be imprecise, either because they make innocent (or deliberate) mistakes, or because they give incorrect answers to ambiguous questions. Several techniques exist to try to manage or reduce the impact of noise resulting from crowd interactions [2, 52].

– *Majority voting* represents a simple yet popular technique to deal with mistakes and subjectivities possibly introduced by the workers. It consists of assigning the same task to multiple workers and choosing as correct the results returned by the majority of them. A variant of this approach weights the votes with a score associated with the ability of each worker to return the correct answer.
– Other approaches rely on an explicit *error model*, which considers the crowd's answer as a random variable. Simpler models assume that workers make independent random mistakes with a constant error-rate. More sophisticated approaches estimate the error-rate by scoring the workers, or relate the error-rate to the difficulty of the task submitted to the crowd. We describe strategies to score the workers in the following.

[3] Although *confirm value* can be seen as a case of *select value* (in which the admitted values are true or false), we prefer to distinguish it, as it represents the simplest form of interaction.

- A family of approaches, that we call *iterative*, jointly evaluates the correctness of answers from multiple workers and the error-rate of the workers by means of an iterative process. They exploit the mutual dependency between agreement on the results and accuracy of the worker by interleaving, up to convergence, two steps: first, they estimate the correct answer for each task from the results returned by multiple workers, accounting for the quality of each worker; second, they estimate the accuracy of the workers by comparing their answers to the ones computed in the first step. Many iterative approaches build on the expectation maximization (EM) algorithm [27] and try to extend it considering specific features of the crowdsourcing context. For example, Joglekar *et al.* extend EM in order to compute confidence intervals of workers' error rate [57]; Raykar *et al.* distinguish the error-rate for positive and negative answers [88]; Whitehill *et al.* consider also the question difficulty [104]; Das Sarma *et al.* aim to overcome limitations of the EM algorithm, such as the convergence towards a local maximum [25]. Other iterative approaches, such as [59–61], are based on standard belief propagation, or on SVD [24].

Hung *et al.* provide an interesting evaluation of the most important majority voting and iterative approaches [50].

2.6 Worker Scoring

In proposals for crowdsourcing for data management, worker scoring has often been seen as extrinsic. However, several strategies have been applied for evaluating the performance of the workers, as follows:

- *Constant error-rate*: this is the simplest solution. It assumes that workers provide wrong answers with a constant (and independent) error-rate.
- *Control queries*: a simple technique for estimating the error rate of workers is to rely on ground truth information. The submitted tasks include some *control queries*, i.e., queries whose answers are known to the system, and these are used to score the workers. This solution impacts on costs, because of the effort in constructing the ground truth, and because of the payments made for answering control queries rather than real ones.
- *Redundant queries*: an alternative solution that does not make use of ground truth information is based on redundant queries. A number of queries are assigned to several workers, and error rate estimation is then based on agreement between workers. The approach is based on the assumption that independent workers make independent errors, which is indeed realistic for workers recruited on a crowdsourcing platform. Also, this solution impacts on costs, since several workers are paid for performing the same task. As discussed in the previous dimension, when *iterative* approaches are adopted to manage noisy answers, the results of redundant queries are used both to estimate the correct answer and to score the workers.
- *Worker comparison*: this strategy consists of employing several workers but submitting different queries to them. This approach is typically applied when the problem to be solved involves computing statistics on large item sets: it is assumed that on average, even on different subsets, the results obtained by multiple workers should be similar. Workers whose results greatly differ from

the others are assigned low scores. Also this strategy, like the previous one, can be used in conjunction with *iterative* approaches to manage noisy answers.
- *Reputation*: worker selection is based on properties of their track record; these scores are usually provided by the crowdsourcing platform.

Observe that a correct scoring of workers is an important issue that influences the quality, the costs and the latency of a crowdsourced solution. If a worker is underestimated, the solution does not trust the worker's responses, and ends up consuming more resources, thus raising the costs and possibly the latency. Conversely, if the worker is overestimated, the solution trusts the worker's responses too much, eventually compromising the quality of the results.

2.7 Task Composition

In crowdsourcing applications, there are often numerous alternative tasks that could be posted to the crowd, and it may not be cost-effective to post all of them. As such, a *Task Composition* strategy determines which tasks are posted, or the order in which they are presented to the crowd. Depending on when decisions are made, *Task Composition* may be characterized as:

- *offline*: task composition decisions are made upfront and not revised in the light of information subsequently obtained; or
- *online*: task composition decisions are made taking into account information as it is obtained.

Several proposals have addressed the online task composition problem [64, 113, 40, 11], treating it as an orthogonal issue. All the approaches assume that the same question is submitted to several workers, in order to overcome the imprecision of the crowd, and that a noise management technique (e.g., *majority voting* or *iterative*) combines the answers to produce a reliable result. The crucial point is that some questions raise a clear agreement of the answers, while others collect conflicting results and then need to involve more workers (for example because the former are easier than the latter). While in offline strategies the same level of redundancy is required for all the questions, online strategies aim at optimizing costs, or quality under budget constraints, composing tasks on the fly by including questions that need more evidence to converge toward a solution.

3 Survey of Proposals

This section applies the dimensions from Section 2 to survey representative proposals for crowdsourcing in data management. The section is organised by the *Data Management Task* supported by the surveyed proposals. In each section we describe a range of representative systems using the dimensions, and drill down into one or a few state-of-the-art proposals, so that key challenges, and approaches to addressing them, can be brought to the fore. It would not have been feasible to provide equally detailed descriptions for all the proposals, but key similarities and differences are highlighted by the dimensions. The values of the dimensions for representative proposals are provided in Table 2.

Table 2 Values of the dimensions for representative proposals.

	Problem		Solution				
Proposal	**Data Management Task**	**Optimization Goals**	**Role of the Crowd**	**Type of Interaction**	**Noise Management**	**Worker Scoring**	**Task Composition**
CrowdDB [37, 39, 101]	Querying, Data Gathering	Cost, Quality, Latency	Supply data, Label data	Provide, Confirm	Majority voting	None	Online
Deco [82]	Querying, Data Gathering	Cost, Quality	Supply data, Label data	Provide	Majority voting	Reputation	Offline
Qurk [69, 68]	Querying, Data Gathering	Cost, Quality, Latency	Supply data, Label data	Provide, Confirm	Iterative	Redundant queries	Offline
CrowdOp [35, 64]	Querying, Data Gathering	Cost, Quality, Latency	Supply data, Label data	Provide, Confirm	Iterative	Redundant queries	Offline
CrowdScreen [81, 80]	Operator (filtering)	Cost, Quality, Latency	Label data	Confirm, Select	Error-model	Control queries, Redundant queries, Reputation	Online
CrowdFind [90]	Operator (finding)	Cost, Latency	Label data	Confirm	Error-model	Control queries	Offline
Davidson *et al.* [26]	Operator (top-k, group-by)	Cost, Quality	Label data	Confirm	Error-model	None	None
Lofi *et al.* [65]	Operator (skyline query)	Cost, Quality	Supply data	Provide	Majority voting	Control queries	Online
Guo *et al.* [43]	Operator (max)	Cost, Quality	Label data	Confirm	Error-model	Constant error-rate	Online
Venetis *et al.* [98]	Operator (max)	Cost, Quality Latency	Label data	Select	Error-model (task difficulty)	None	Online
Verroios *et al.* [99]	Operator (max)	Latency	Label data	Select	None	None	Online
Marcus *et al.* [66]	Operator (count)	Cost, Quality, Latency	Label data	Confirm, Select	Majority voting, Iterative	Worker comparison	Offline
System Q [95]	Data Integration	Cost, Quality	Verify work	Provide	None	None	Offline
McCann *et al.* [72]	Data Integration	Cost, Quality	Verify work	Confirm, Select	Majority voting	Control queries	Online
Zhang *et al.* [107]	Data Integration	Cost, Quality	Verify work	Confirm	Error-model	Reputation	Online
Hung *et al.* [48]	Data Integration	Cost, Quality	Verify work	Select	Iterative	Redundant queries	Offline
Fan *et al.* [34]	Data Integration	Cost, Quality	Verify work	Select	Iterative	Redundant queries	Offline
Osorno-Gutierrez *et al.* [78]	Data Integration	Cost, Quality	Verify work	Confirm	Majority voting	Redundant queries	Offline
Selke *et al.* [91]	Data Integration	Cost, Quality	Label data	Confirm	Majority voting, Reputation	Control queries, Redundant queries	Offline
Data Tamer [94]	Data Integration, Data Cleaning	Quality	Verify work	Confirm	Error-model	Redundant queries	Offline
ZenCrowd [28]	Data Cleaning (entity resolution)	Quality	Verify work	Confirm	Error-model	Reputation	Offline
Silk [53, 54]	Data Cleaning (entity resolution)	Cost, Quality	Label data	Confirm	None	None	Online
Corleone [42]	Data Cleaning (entity resolution)	Cost, Quality	Label data	Confirm	Majority voting	None	Online
Whang *et al.* [103]	Data Cleaning (entity resolution)	Cost, Quality	Open	Confirm	None	None	Online
CrowdER [100]	Data Cleaning (entity resolution)	Cost	Open	Confirm	None	None	Offline
ACD [102]	Data Cleaning (entity resolution)	Cost, Quality	Verify work	Confirm	Error-model	Reputation	Online
CrowdCleaner [96]	Data Cleaning	Cost, Quality	Verify work	Confirm	Error-model	Reputation, Worker comparison	Offline
KATARA [20]	Data Cleaning	Cost, Quality	Verify work	Select	Majority voting	None	Offline
Zhang *et al.* [109]	Data Cleaning	Cost, Quality	Verify work	Confirm	Error-model	Control queries	Online
TripleCheck Mate	Data Cleaning	Quality	Verify work	Confirm, Provide	None	Redundant queries	Offline
Mozafari *et al.* [75]	Data Cleaning (entity resolution)	Cost, Quality Latency	Label data	Select	Majority voting	Control queries, Redundant queries	Online
CrowdFill [85]	Data Gathering	Cost, Quality, Latency	Supply data, Verify work	Provide, Confirm	Majority voting	None	Online
Alfred [22, 23]	Data Gathering	Cost, Quality	Label data	Confirm	Iterative	Redundant queries	Online
Higgins [62]	Data Gathering	Quality	Supply data	Select	Majority voting	None	Offline
CrowdQ [29]	Search Querying	Quality	Supply data Label data Verify data	Confirm Select Provide	–	–	Online
DataSift [83]	Search	Quality	Supply data Label data	Confirm Provide	Majority voting	–	Online
CrowdSearcher [12]	Search	Quality	Supply data Label data	Select Provide	–	–	Offline

```
CREATE TABLE Department (
  university STRING,
  name STRING,
  url CROWD STRING,
  phone STRING,
  PRIMARY KEY (university, name) );
```

Fig. 2 A sample CrowdSQL DDL statement.

3.1 Querying Databases with the Crowd

The opportunity of involving humans to process large amounts of data has motivated several proposals to extend traditional DBMSs with the ability of crowdsourcing data processing tasks.

Some proposals present full-fledged DBMSs that aim to appear to the end user as similar as possible to a conventional database system. They introduce SQL extensions and revisit query evaluation and query optimization techniques, hiding the complexities of dealing with humans as data sources or evaluators of operations, including breaking down large tasks into smaller ones, composing and submitting tasks on a crowdsourcing platform, dealing with latency, errors and inconsistencies introduced by the crowds. Other proposals focused on novel implementation of specific data processing operators (such as selection and aggregate functions) with tasks accomplished by humans recruited on crowdsourcing platforms. In the following, we first illustrate crowd-powered DBMSs, then describe proposals that concentrate on specific operators.

3.1.1 Crowd-Powered DBMSs

CrowdDB

CrowdDB adopts a relational data model: the system evaluates queries expressed in CrowdSQL (an SQL extension) using data stored in conventional tables as well as data acquired by crowdsourcing [37]. In CrowdDB, the database designer can define columns (possibly all the columns of a table) whose values can be gathered by crowdsourcing. The execution of a query that involves one (or more) of these columns triggers a process that transparently submits the data acquisition task to a crowdsourcing platform, treating workers as noisy data sources. The system automatically generates forms to gather values, prepares and submits tasks to a crowdsourcing platform.

Figure 2 shows an example of a CrowdSQL DDL statement: it defines a table, whose column `url` is annotated with the `CROWD` keyword to specify that its values can be acquired from a crowdsourcing platform.

Columns whose values can be crowdsourced admit *CNULL* values, a special CrowdSQL type indicating that the value should be acquired by crowdsourcing when it is first used. These columns play a role during the query processing phase: *CNULL* values are requested from the crowd when they are required to produce a result. To give an example consider the queries in Figure 3. In the first query, the target list includes all the attributes: the result tuples whose `url` column is *CNULL* triggers the creation of tasks to acquire the values from the crowd. Similarly, tasks

```
SELECT *
FROM Department
WHERE name = "Computer Science";

SELECT name, phone
FROM Department
WHERE url LIKE "%berkeley%";
```

Fig. 3 Examples of CrowdSQL queries.

```
SELECT name FROM Department
WHERE university ~= "U of Toronto"
```

Fig. 4 A CrowdSQL query with the `CROWDEQUAL` (∼=) operator.

to crowdsource the `url` values are created for evaluating the second query, where the `url` attribute appears in the `WHERE` clause.

In these tasks, workers are involved to *Provide values*, but CrowdDB can also crowdsource the evaluation of comparison and sorting operators, through specific crowd powered operators, which requires a *Confirm value* type of interaction with the crowd. Figure 4 shows a CrowdSQL query to select the name of departments from the University of Toronto, asking the crowd to match with the possibly different names (such as, "UofT") given for that university in the database.

CrowdDB is built on top of a traditional DBMS, and query processing follows a classical approach: a query is parsed into an algebraic expression which is then optimized and eventually evaluated. The query optimizer is based on query-rewriting rules, and adopts heuristics that aim to reorder operators in order to minimize the cost of crowd tasks. The system adopts a practical approach for bounding costs. First, it allows the designer to define a budget bound by specifying a limit on the number of crowdsourced tuples. Second, it adopts a *majority voting* strategy to control the quality of the results; if no majority occurs, additional workers are involved until the majority agrees or a given number of answers are collected.

A method for executing joins specifically tailored for CrowdDB relies on a two-phase approach [101]: first, an algorithm computes a set of candidate matching pairs; second, crowd workers are involved to decide whether the candidate pairs match. A further optimization step considers transitive relations among pairs. To reduce latency, it composes tasks online, identifying and submitting sets of pairs to be crowdsourced in parallel. The approach does not consider any technique to score the workers, and assumes that they always provide correct answers.

It is worth observing that in the novel scenarios introduced by CrowdDB, in which crowd workers can provide the results of a query, there is a violation of the closed world assumption, which characterizes the traditional query semantics. To address this issue, CrowdDB leverages statistical techniques to estimate the completeness of a query result [39,97], analyzing the cost-benefit tradeoff of acquiring more answers.

Qurk

Qurk is a relational query processing system that includes crowd-based versions of selection (filter), join, and sort operators [68,69]. Qurk relies on SQL standards,

```
SELECT name, findInfo(name).Email,
             findInfo(name).Phone
FROM restaurants

TASK findInfo(String name)
 RETURNS (String Email, String Phone)
 TaskType: Question
 Text: "Find the Email and the Phone for
        the restaurant \%s", name
 Response: Form(
  ("Email", String), ("Phone", String))
```

Fig. 5 A Qurk query (and its associated UDF) for a data gathering task.

```
SELECT c.name
FROM celebs AS c
WHERE isFemale(c.img)

TASK isFemale(Image img)
 RETURNS: BOOLEAN
 TaskType: Filter
 Text: "<img src="\%s"><br/>
        Is this person a woman?", img
 YesText: "Yes",  NoText: "No"
```

Fig. 6 A Qurk query (and its associated UDF) for filtering tuples.

and it adopts user-defined (scalar and table) functions (UDF) to provide a mechanism to crowdsource data acquisition as well as data processing. Through the UDF, the designer specifies the task type (e.g. question, comparison) and information used by the system to generate the user interfaces (HTML forms) for the crowd.

Figure 5 presents an example Qurk SQL query: the crowd is asked to *provide values* of emails and phone numbers for a `restaurants` table. Observe that the query uses a `findInfo()` UDF (shown in the same figure), in which the designer has specified the details required to build the user interface.

UDFs are also used to perform the typical data processing operators used by a DBMS, such as filtering and joins. Figure 6 shows a query which uses a UDF to crowdsource a filtering task for selecting female celebrities from a table storing names and photos of celebrities.

To sort tables, Qurk introduces several crowd-based algorithms. Besides traditional sort algorithms based on pairwise comparison, Qurk implements sort algorithms based on rank: the workers are asked to provide a numerical ranking for each element. This approach requires fewer tasks (and hence incurs smaller costs) than directly comparing pairs of objects, but produces a less accurate ordering. A hybrid algorithm uses ratings to roughly order items, and iteratively improves that ordering by using comparisons to refine the order of objects with similar ratings.

To join tables with crowd support, Qurk generates a HIT interface to ask workers whether elements from two tables satisfy the join condition. The join algorithm is a traditional block nested loop join, which uses the results of comparisons provided by the crowd. The main issue addressed in Qurk for the implementation of a crowd empowered join algorithm is the effectiveness of the interaction with the crowd. Qurk explores several approaches. An approach, called NaiveBatch, consists of preparing HITs that present several pairs vertically: a *Confirmation query* asks the worker whether the two items satisfy the join condition. An alternative approach, called SmartBatch, presents two columns of items, each column from one of the join tables, and asks the worker to select pairs of items that join. If none of the items match the join predicate, the worker can indicate no matches. Observe that, given two tables, R and S, NaiveBatch requires $\frac{|R||S|}{b}$ HITs, where b is the number of item pairs per HIT, while SmarthBatch requires $\frac{|R||S|}{rs}$ HITs, where r and s are the numbers of the items in the first and second columns, respectively. Despite their differences, the experimental activity reported in [69] concludes that the two approaches achieve similar results for accuracy and costs.

Query planning in Qurk is similar to conventional logical-to-physical query plan generation: a query is translated into a plan-tree that processes input tables in a bottom-up fashion. Relational operations that do not require human intervention are pushed down the query plan.

Redundant queries and an *iterative* approach are adopted to score workers and to manage noisy answers, and the experimental results show that the *iterative* approach outperforms *Redundant queries.* Qurk composes the tasks *offline*, allowing database administrators to set the number of tuples per task. Interestingly, in their experimental evaluation the authors observe that although large tasks can reduce the cost of operations such as sorts and joins, larger sizes lead many workers to refuse to perform tasks at all, thus drastically compromising the latency of the system.

Deco

Deco builds over the relational model and offers functionalities for answering declarative queries posed over conventional tables together with data harvested on-demand from the crowd [82]. In Deco, the database designer specifies the database schema as a set of *conceptual relations*, a set of *fetch rules*, and a set of *resolution rules.* Conceptual relations are logical relational tables, possibly in first normal form; the system transparently derives a normalized raw schema, and generates the tables where data are actually stored. Fetch rules are used to specify how to acquire more data by means of procedures that implement access to human workers. In particular, a fetch rule assumes the form $A1 \Rightarrow A2 : P$, where A1 and A2 are sets of attributes from one conceptual relation, and P abstracts a fetch procedure that abstracts a crowd process. The execution of a fetch rule triggers a request for new values for the right set of attributes, given values of the left one. The acquired values are cleansed by the resolution rules, and then used to populate the relations of the raw schema. For example, a resolution can canonicalize some values (such as addresses or phone numbers), or either choose one among several values introduced by the crowd based on a majority voting strategy (e.g. choose the most-cited phone number of a restaurant) or compute an aggregate (e.g. return the average rating of a restaurant over the crowdsourced ratings).

The Deco query language introduces an SQL extension to indicate to the query processor when and how data should be obtained from the crowd. In particular, Deco introduces a clause to specify the minimum number of tuples to be returned by a query. The query processor tries to answer the query using the current content of the local tables; in case it does not retrieve the minimum number of tuples, it gathers additional data from the crowd. Workers are scored by *Reputation*, based on services offered by the crowdsourcing platform, or on worker history. Data gathering is performed by executing the fetch rules and the resolution rules defined by the database designer. Query optimizations in Deco aims to find the query plan that minimizes the estimated monetary costs [84]. One of the main challenges for the query optimizer is the estimation of the cardinalities of the fetch operators, since the database state changes at query execution time, due to the fetch operations.

CrowdOP
CrowdOp is another crowd enhanced RDBMS [35]. It proposes an SQL like language that provides constructs for the following crowdsourcing operators: *FILL*, to gather missing values; *SELECT*, to filter tuples based on conditions that might involve humans (e.g. sentiment of a review, subject of an image); and *JOIN*, to match tuples according to criteria that might require human intervention (e.g. a join based on images from a pair of relations). Overall, the data model and the data manipulation language are similar to those offered by CrowdDB. The most distinctive and interesting feature of CrowsOp is its cost-based query optimizer, which allows the system to minimize latency under user defined budget constraints for all the three crowd-powered operators. To deal with the accuracy of the workers, CrowdOp relies on CDAS, an *iterative* approach [64]. CDAS offers an online processing techniques, which allows for an early termination of the tasks as soon as the system estimates that further answers will not change the results. However, CrowdOp does not consider this feature since it adopts a simpler static cost model. As we shall discuss in Section 4, the study of more involved cost models is an intriguing research direction.

State-of-the-art. From Table 2 it can be seen that, with the exception of Deco, all the Crowd-Powered DBMSs aim at optimizing all the objectives (latency, cost, quality). It is worth noting that all the proposed systems adopt an *offline* task composition strategy. As we discussed in Section 2.7, the task composition problem is mainly seen as an orthogonal issue. However, the integration of a suitable online task composition approach in a Crowd-Powered DBMS could impact on other design choices, and thus represents an interesting research direction.

3.2 Crowd-Based Operators

Crowdsourcing has been applied to implement specific DBMS operators over large data sets where humans perform tasks that cannot be efficiently expressed as algorithms, such as selecting or counting images of “old” persons in a photo dataset.

Maximum
Several solutions have been proposed to address the issue of computing the maximum over a set of items. Guo *et al.* [43] rely on *confirmation queries*: they propose a solution based on a sequence of pairwise comparisons. Their approach consists of asking human workers to pick the greater between a pair of objects. Then, given a set of answers, they consider how to select the maximum and how to improve

the results by requesting further votes. They prove that, although the underlying model is relatively simple (they assume each worker has an error-rate $\eta < 0.5$, which is unaffected by worker identities, object values, or worker behavior), an optimal solution is NP-hard.

Venetis *et al.* [98] follow a different approach, which resembles the *counting* strategy adopted in [66]. Here, a set of items is presented to a group of workers, and each worker is asked to indicate the greatest one. An aggregation function (based on the consensus of the workers) combines the responses and determines the winner. Since the input set can be very large, and a worker can only compare a small set of items, they organize the overall process execution in steps: in every step, a batch of items is submitted to the crowd, with further items being requested in successive steps. The number of steps that is necessary to obtain a reliable result corresponds to the latency of the overall process, while the number of tasks corresponds to the cost. Computing the max is then modeled as an optimization problem: the quality of the result is measured as the probability of finding the maximum item from the whole input set, constrained by cost and time bounds. Subjectivity and mistakes of the workers are addressed by introducing a set of error models which consider the difficulty of the task based on the size of the set of items that compose the task and on the diversity (measured by a distance function) of the items within each set.

Verroios *et al.* [99] focus on the issue of computing the maximum among a set of items, minimizing the latency under an optimal budget allocation. Their solution consists of partitioning the set of items. The subsets are then processed in parallel tournaments, by asking a complete set of pairwise comparisons among the participant items. Several rounds are necessary to compute the overall maximum item. They assume workers always respond correctly, and consider the management of human errors as an orthogonal issue.

Anagnostopoulos *et al.* [6] also present an algorithm for computing the maximum element among a set: they develop a model to make explicit the role of crowdsourcing *experts* and show the benefit wrt a model making use only of ordinary crowdsourcing workers.

Filtering and Finding

Crowdsourcing solutions have also been studied for filtering and finding items in large data sets. The goal of the former problem, filtering, is to select items, from a large collection, that satisfy a given property (e.g. find all the photos that show a cat in a photo database). The latter problem, finding, consists of finding a given number of items (not necessary all) that satisfy a given property (e.g., find 10 photos that contain a cat).

CrowdScreen [81] develops a solution for the *filtering* problem that aims at minimizing costs with constraints on the budget and on the expected error rate. The strategy is similar to the *labeling* solution of Marcus *et al.* [66]: items are evaluated sequentially, and the workers are asked to answer a *confirmation query* about the presence of the target property (e.g. *Does this picture show one dog and two cats? y/n*). In order to improve accuracy, CrowdScreen asks different humans the same question, and develops strategies to dynamically decide whether one or more questions are needed for each item. The solutions developed in [81] assume the same error-rate for every worker. Such an assumption is relaxed in [80], which presents a CrowdScreen extension where the error rates of the workers are estimated either by control queries or by reputation (history of worker performance). This work also relies on prior information about the likelihood that some items can satisfy the filter more often than others, and considers the issue of ranking items.

CrowdFind [90] addresses the *finding* problem and concentrates on the cost-time tradeoff. The problem of finding a given number of items in a dataset has two extreme solutions: a sequential analysis, which consists of posing a query to the crowd item by item, and a parallel analysis, which consists of posing in parallel as many queries as the number of items. The former minimizes the costs because it stops posing queries as soon as the required number of items are found, whereas the latter minimizes the latency. Hybrid solutions pose a sequence of a number of queries in parallel. CrowdFind develops algorithms to turn a cost-optimal sequential algorithm into a hybrid algorithm, with an approximation bound α, that asks at most α more questions than the sequential algorithm, with provable guarantees on the latency of the solution. CrowdFind assumes that the selectivity of the target property, i.e. the percentage of items that satisfy the target property, is given, and develop solutions for a deterministic and for an uncertain scenario. In the deterministic scenario, workers always provide correct answers. In the uncertain scenario, workers can make mistakes, and CrowdFind assumes that their error rates are given, possibly computed by a set of control queries.

Skyline and Top-k

The idea of involving the crowd for data processing is also explored in the context of skyline queries. The goal of a skyline query is to extract relevant tuples from multidimensional databases according to multiple criteria. Lofi *et al.* [65] introduce methods to obtain from the crowd the missing information that has the largest impact on quality of the result set; they propose prediction algorithms to provide approximate values for all the other tuples. *Majority voting* is adopted to address noisy answers; *control queries* are used to discard unreliable workers. Interestingly, the authors experimentally observed that higher quality with lower costs can be achieved by issuing smaller tasks, and then dynamically choosing further tuples to be crowdsourced after each task.

Davidson *et al.* study formal properties of top-k and group-by algorithms, in a scenario where humans are involved to determine whether a pair of items exhibit the same property and to order the items [26]. The proposed algorithms aim at minimizing the number of *confirmation queries* submitted to the crowd to find the exact top-k elements or the exact clusters. Similar to Venetis *et al.* [98], this work models the error-rate considering the difficulty of the task, and not the reliability of the workers. However, this work does not rely on paid workers, but rather on users from an *ad hoc* social network, which stores the individual history of each user. Ciceri *et al.* also propose to submit *confirmation queries* to the crowd for solving pairwise comparisons between tuples with an uncertain score in top-k query processing [21].

Counting

Here the Data Management Task being undertaken is counting; specifically, this means identifying the fraction of the elements in a collection that satisfy a given property. The challenge is to obtain a good estimate of the count, while minimizing latency and cost.

Trushkowsky *et al.* [97] solve a variant of the counting problem, specifically targeted for sets whose cardinality and elements are unknown and can be obtained only with the help of the crowd. Sarma *et al.* [25] focuses on the problem of counting the number of objects in a photo. They show that humans are very good at batch counting only if the batch is not too big, and deal with the problem of segmenting and rearranging the images so that crowd workers are shown images with an appropriate number of objects.

As a case study, we consider Marcus *et al.* [66] that propose solutions for counting items that satisfy a given property.

Case Study: Counting with the Crowd

Marcus *et al.* [66] propose a solution for counting items that satisfy a given property, with the help of the crowd, through four steps: (*i*) designing the crowd microtasks; (*ii*) developing techniques for combining the results from multiple microtasks; (*iii*) estimating the worker reliability; and (*iv*) applying the result of (*iii*) to detect spammers and decide when to stop collecting data. In the paper, examples include identifying the fraction of people in a collection of photographs that are male, or red-haired.

For *designing the crowd microtask*, they develop and compare two alternative strategies. The first strategy, called *labeling*, follows a straightforward approach: it consists of asking the workers to confirm the presence of the desired property item by item. For example, this could involve showing crowd workers individual images and asking them to indicate if the person in the image has red hair. An alternative strategy, called *counting*, exploits human abilities of the workers: instead of asking the workers to label the dataset item by item, it presents the worker a group of items and asks for a rough estimation of the number of items that exhibit the desired property. For example, the workers are shown a collection of images, and then are asked to indicate about how many of the images feature people with red hair.

For *combining the results from multiple microtasks*, the basic formula to calculate $\hat{F}$, an approximation of the overall fraction of items with a given property in the population, is straightforward: $\hat{F} = \frac{\sum_{i,j} Fij}{\sum_{i,j} 1}$, where $F_{ij} = C_{ij}/B_C$ is the ratio between the count C_{ij} reported by worker i on j-th microtask, and B_C is the number of randomly sampled items in it.

However, there are potential issues for *estimating worker reliability*, which are handled in this case by giving a weight $\theta_i \in [0, 1]$ to each worker (0 means a spammer and 1 is an high-quality worker) based on how far their answers are from the mean $\hat{F}$. More precisely, let F_i denote the fraction of items as estimated based only on the responses provided by worker i; the weight θ_i of a worker i is computed as follows:

$$\theta_i = \begin{cases} 1 - |F_i - \hat{F}|, & \text{if } |F_i - \hat{F}| < \lambda \\ 0, & \text{otherwise} \end{cases}.$$

Since θ_i and F_i are mutually dependent on each other by means of $\hat{F}$, their computations are interleaved, adopting an *iterative* approach, until a fixed point is reached. The workers are scored essentially adopting a *Worker comparison* strategy based on the bias of their responses.

The authors also include a step to *detect spammers*, i.e., the workers whose weights are outliers (outside a 95% confidence interval wrt other workers). This leads to a revised formula for calculating the final result (restricted only to the responses of workers not classified as spammer): $\hat{F}_{final} = \frac{\sum_{i,j} \theta_i F_{ij}}{\sum_{i,j} \theta_i}$. □

State-of-the-art. As shown in Table 2, the role of the crowd is to *label data*, with a confirmation query or a selection for the majority of the operators; this is natural as it corresponds to the most natural and effective type of interaction for the unskilled workers that can be recruited on online platforms. Also, it is worth observing that noise management and worker scoring are performed mainly by the simplest approaches: error model (typically uniform) and control queries. As crowdsourcing evolves, the engagement of workers with different levels of expertise, and tasks with different levels of difficulty could inspire novel strategies for implementing data management operators.

3.3 Integration

Data integration is classically mostly a manual activity conducted with tool support. Thus it depends on expert engagement, which limits its applicability in large-scale scenarios, or those where the dynamics of the problem require constant adjustment and refinement. As such, an incremental approach to the problem has been an active topic, referred to as *dataspaces* [38] or *pay-as-you-go* data integration. Crowdsourcing intersects with this trend insofar as crowds are viable sources of information for improving integrations. Data integration typically includes *matching* to identify associations between schema elements, from which *mapping generation* creates views that can be used to translate between representations. In the pay-as-you-go approach, matching and mapping generation are carried out by algorithms in a so-called bootstrapping phase. This is followed by an improvement phase, the most important element of which is refinement, of both matches and mappings.

We first present a detailed case study of crowdsourcing for matching schemas and then discuss further proposals covering other data integration tasks.

Case Study: Correspondence Correctness Questions

Matching [87] is a challenging task in data integration. Support tools typically use ensembles of matching algorithms to assign a similarity score to the correspondences between schema elements identified by the tool. We describe in detail how Zhang *et al.* [107,110] have explored crowdsourced feedback to reduce the uncertainty on such schematic correspondences.

Consider the example schema matching problem in Fig. 7 (adapted from [107]). Two schemas, A and B, both describe information about academic staff that exhibit schematic correspondences (e.g., those indicated by the dotted lines in the figure) that a matching tool could identify, for example by assigning each correspondence a probability.

In Zhang *et al.*, a matching is a set of correspondences, as exemplified by Fig. 8 (left), where probabilities are assigned to three possible matchings, viz., m_1, m_2, m_3. Then, the probability of a correspondence is obtained by summing the probabilities of the matchings in which that correspondence occurs, as exemplified

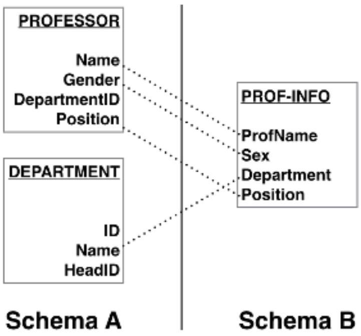
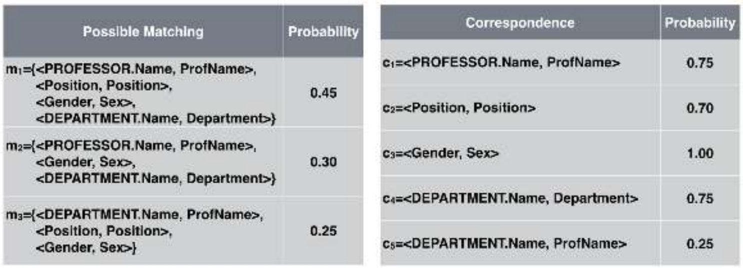


Fig. 7 Example Schema Matching Problem.



Possible Matching	Probability

m₁={<PROFESSOR.Name, ProfName>,
<Position, Position>,
<Gender, Sex>,
<DEPARTMENT.Name, Department>}
	0.45

m₂={<PROFESSOR.Name, ProfName>,
<Gender, Sex>,
<DEPARTMENT.Name, Department>}
	0.30

m₃={<DEPARTMENT.Name, ProfName>,
<Position, Position>,
<Gender, Sex>}
	0.25

Correspondence	Probability
c₁=<PROFESSOR.Name, ProfName>	0.75
c₂=<Position, Position>	0.70
c₃=<Gender, Sex>	1.00
c₄=<DEPARTMENT.Name, Department>	0.75
c₅=<DEPARTMENT.Name, ProfName>	0.25

Fig. 8 Possible Matchings (left) and correspondences (right) of Fig. 7.

by Fig. 8 (right). Thus, correspondence c_1 occurs in matchings m_1 and m_2, so its probability is the sum $m_1 + m_2 = 0.45 + 0.3 = 0.75$.

Broadly speaking, Zhang *et al.* aim to reduce the uncertainty of the set of correspondences that comprise the possible matchings identified by a matching tool. In order to achieve this aim, the approach is to seek feedback from the crowd. Each feedback instance is referred to as a *correspondence correctness question (CCQ)*. For example, the CCQ Q_{c_1} associated with correspondence c_1 would be posed to a crowd worker as follows: "Is the correspondence `<PROFESSOR.Name, ProfName>` correct?".

The problem addressed by Zhang *et al.* can be stated more formally as follows: *Given two schemas S and T, the set RS of possible matchings and the probability assignment Pr function computed by some matching tool, minimize the uncertainty of RS subject to a maximum budget B of the CCQs induced by RS.* The uncertainty of RS is the Shannon entropy computed over the possible matchings in RS, i.e.

$$H(RS) = -\sum_{m_i \in RS} Pr(m_i) \log Pr(m_i)$$

CCQs are published to crowd workers and return an answer. If the worker returns true (resp., false) to a CCQ, this answer reduces to zero the probability of all the matchings in which the associated correspondence does not (resp., does) occur. Since the sum of the probabilities over the set of possible matchings is equal to one, the probabilities of the possible matchings that remain valid are redistributed.

The error rate of a crowd worker can be used to adjust a crowdsourced answer. This enables an approach in which a sequence of answers progressively reduces uncertainty, no matter in which order the sequence is assimilated. In other words, it is the set of CCQs that are chosen for publication that determines the quality of the solution for a given budget.

Given this approach to modelling uncertainty and assimilating crowd feedback, Zhang *et al.* compute the expected reduction in uncertainty resulting from the answer to a given CCQ Q_c in the context of a set of possible matchings RS, i.e., $\Delta H_c = H(RS) - H(RS|Q_c)$.

They first consider an approach they call *Single CCQ* in which one CCQ at a time is chosen to be published to the crowd. At each pass, the CCQ that causes the greatest expected reduction in uncertainty is greedily chosen. The feedback is assimilated and the algorithm iterates, terminating when the budget is spent. Computing the expected reduction in uncertainty from a CCQ is inefficient for larger schemas, so Zhang *et al.* show that choosing the CCQ one is intuitively most uncertain about (i.e., the CCQ with probability closest to 0.5) is an effective strategy even though correspondences are not independent.

The latency of *Single CCQ* could be large, i.e., it fails to exploit the possibility of many crowd workers answering CCQs in parallel. To address this shortcoming, Zhang *et al.* propose an approach they call *Multiple CCQ* in which k CCQs are published simultaneously. They observe the trade-off that arises in this case: rather than publishing the best CCQ every time, they publish k good CCQs at a time.

The *Multiple CCQ* approach is based on the observation that a published CCQ can be in one of three states, viz., *waiting*, if it has not been taken by a worker yet; *active*, if it has been taken by a worker who is working on it; and *answered*, if its answer is available for assimilation. The crowd platform used by Zhang *et al.*, viz., Amazon Mechanical Turk, allows for the withdrawal of CCQs that are published but waiting, thereby saving on budget depletion.

The approach is as follows: the set of best k CCQs is computed and published, then monitored continuously up to consumption of the budget. As answers become available, the following operations take place: (1) all waiting CCQs are withdrawn, (2) the feedback is assimilated causing the probability distribution to be adjusted, (3) the set of best k CCQs taking into account the active CCQs is recomputed and its completion w.r.t. to the latter is published.

In terms of our dimensions, the work on CCQs uses the crowd to feed an algorithm that aims to optimize quality and formalizes a trade-off with cost. The interaction with crowd workers takes the form of tasks, which are composed online in an adaptive fashion, that requires the workers to confirm a value, with answers being adjusted based on their error rate as induced by their reputation. □

We now discuss other representative proposals for the use of crowds to help perform data integration tasks.

In schema matching, automatic techniques identify candidate associations between schema elements, typically building on syntactic measures of similarity [87]. Crowdsourcing can be used to obtain human confirmation of candidate matches. For example, McCann *et al.* [72] adopted a Web 2.0-style reliance on online communities to reduce the schema-matching burden on what the authors refer to as *builders*, defined to be a relatively small set of volunteers that largely shoulders the task of integrating data within a community. The focus is on using a crowd of end users to provide feedback that, upon assimilation, can improve the accuracy of the matches postulated by a matching tool. The tasks are *confirm value* ones, but users can also be asked to confirm or deny postulated predicates or constraints (such as whether two or more attributes stand in a relationship, functional or otherwise). User feedback forms a stream of answers to a question (as a crowd task) posed to users. The stream is closed when a certain number of answers have been collected and the gap between the majority and minority answers is of a given magnitude, in which case the answer is selected using *majority voting*. Algorithmically, McCann *et al.* follow the common approach of using a question builder and an answer analyzer. For a slightly more detailed description of an another example of this approach, see our account of Hung *et al.*'s work on matching networks [48].

With a view to increasing the efficiency of crowdsourcing feedback on matches, Hung *et al.* [48] introduce the notion of *matching networks* as comprising a set of schemas (rather than a pair) with associated pairwise attribute correspondences linking the schemas. Various constraints are identified that apply to such networks, that can be used to reduce the numbers of questions that need to be put to the crowd. In more recent work [49,47], matching networks have been also shown to have a beneficial impact on the effort required by an expert user (note, not a crowd setting anymore) in performing the task of reconciling matches. Algorithmically, Hung *et al.* first use a matching platform to generate pairwise attribute-level matches, then use crowd workers to decide whether the generated matchings are valid. In this second stage, the matches are analyzed by a question builder that generates the questions to the crowd. The answers undergo a probabilistic process of aggregation that aims to take into account worker error. This provides an error rate that, depending on a predefined threshold, may cause the system to go back to the crowd. If all matches, are decided, the process stops.

Fan *et al.* [34] have focussed on matching web tables, which, the authors argue, present particular challenges due to their being, as a rule, incomplete in two main respects: limited instance-level information because a web table is typically extracted from a single web page, and limited schema-level information since schema descriptions are seldom available for extracted tables. To counteract this, Fan *et al.* adopt a concept-based approach (using FREEBASE[4]). In order to decide how to use the crowd to help algorithms infer matches the authors use a utility function based on a column's matching difficulty and influence: the greater the difficulty and influence, the more likely the column is to be crowdsourced. Algorithmically, the approach works in two phases. The first, called concept determination, aims to use the crowd to determine the concept that best represents the values of each column of every web table using a utility function on the information returned. The second, called table match generation, finds all the pairs of columns from distinct tables that are assigned to the same concept, and creates a semantic correspondence between the two columns.

Crowdsourcing user feedback for mapping generation has received less attention than for matching. Belhajjame *et al.* [7], describe how user feedback on the results of queries posed against a mediated schema can be used to select which mappings to use for answering a given query, possibly constrained to a minimum value for precision or for recall. Users are asked to annotate a result tuple as belonging or not to the intended answer. The empirical evaluation in [7] used synthetic feedback, and investigated how many feedback instances sufficed to converge the mapping

[4] freebase.com

selection and refinement problem to a ground truth. Following on from that work, [78] verified that similar convergence behaviour was observed when feedback was crowdsourced rather than synthetically generated. One distinctive contribution here is the more detailed, comparative approach to managing unreliable responses using majority voting for noise management and redundant queries for worker scoring. Algorithmically, the proposal in [78] expands on simpler applications of majority voting to consider not just inter-work reliability but also intra-worker reliability (i.e., the case where a worker may give different answers to the same question at different times, due to the influence of habituation, for example).

A special case of the mapping refinement problem is addressed by the Q System [95,105]. Mappings are built from the result of keyword queries and then exposed as web forms. End users can obtain results through the web forms. The results are annotated with feedback, which is assimilated by the system to improve the mappings. Algorithmically, the Q system builds a set of conjunctive queries that, essentially, interpret the associations as opportunities for joining the sources. The conjunctive queries are trees against the graph (where sources are nodes and association are edges) and by summing up the edge costs in each query, the Q system can select the top-k conjunctive queries and take their iterated outer union.

A complementary task that can also be considered to involve data integration is schema expansion, which supports the addition of attributes to query results. Using an example relating to movies, Selke *et al.* [91] explore how a stored movie collection can be extended with genre information extracted from social web sources, supplemented with input from crowd sourcing to train the extraction process.

Most proposals for crowdsourcing in data integration focus on a single step within the integration process. In contrast, Data Tamer [94] is a proposal for automating an extract-transform-load pipeline, with input from expert users or crowds to refine the results of the automation. In particular, with allocations of tasks to workers guided by Data Tamer Administrators, experts or crowds may be consulted to confirm if pairs of attributes match or if pairs of entities are duplicates. An interesting feature of the proposal is that a consistent approach is followed for managing user features and response quality across the two data management tasks supported by crowdsourcing. Algorithmically, Data Tamer is a more complex proposal in that, in its data integration aspect, it supports two approaches: top-down (where the target schema is known) and bottom-up (where this global knowledge is missing). This leads to three knowledge-availability levels, viz., zero, partial and complete knowledge is available, respectively. The basic strategy in Data Tamer's approach to integration is for it to take an attribute from a data source and pairwise compare it to a collection of other attributes. Data Tamer relies on a built-in collection of algorithms, which they term experts. Their application returns scores that are weight-aggregated to yield a combined score. For attribute mapping, Data Tamer considers the knowledge-availability level in order to decide attribute pairings, where, for all levels, worst-case complexity is quadratic. They propose parallelization as well as a two-pass approach that would use more experts as filter before focussing on a subset of the initial input.

State-of-the-art. As far as data integration goes, from Table 2 it can be seen that the crowd task is predominantly *confirm value*, which means that other forms of crowd interaction have yet to be fully explored to support integration. Furthermore, there

Algorithm 1: ActiveLearning(TrainingData t)

```
while entropy has yet to stabilise do
    forests = LearnForests(t);
    examples = HighEntropyExamples(forests);
    t = t ∪ ConsultCrowd(examples);
return forests;
```

are no examples of one form of crowdsourced information being used to support multiple data integration tasks. It is worth noting that a significant amount of work has been done on pay-as-you-go integration (e.g. [89,55,95]) that addresses different aspects of user feedback without using crowdsourcing as the means to obtain such feedback. It would be interesting to explore how crowdsourcing would enable, improve or complement this kind of dataspace task.

3.4 Cleaning and Validation

To support *Cleaning and Validation*, the crowd can serve as an additional source of evidence for refining the results of automatic or manual processes, or can provide data that can be used to inform the configuration of cleaning tasks.

Entity Resolution (ER), also known as *duplicate detection*, *instance identification* and *merge-purge*, is the task of identifying different records that represent the same real-world entity (relevant surveys include [19,32]). ER may support integration (e.g. link discovery in linked open data [28,54]), or data cleaning within or across data sets, but in either case the principal challenges to be addressed, and the components in a solution, tend to be similar. As it is impractical to perform a detailed all-against-all comparison, which is $O(n^2)$ on the number of records, for large data sets, ER proposals tend to include: (i) *blocking*, whereby pairs of records that are candidate duplicates are identified using inexpensive, approximate comparison schemes; and (ii) *detailed comparison*, whereby pairs (or groups) of records that are associated by *blocking* are subject to more detailed scrutiny.

A number of relevant proposals have been made that could, but do not, crowdsource information, e.g. techniques that learn blocking schemes (such as [73,9]) use training data that could be crowdsourced. Here, we start with a detailed case study, and move on to discuss some further representative proposals that involve some form of community engagement.

Case Study: Corleone

Perhaps the most ambitious entity resolution proposal, in particular in terms of the range of crowd issues addressed, is Corleone [42], which seeks to provide *hands-off crowdsourcing* for entity resolution, whereby the whole entity resolution process is automated, obtaining input from the crowd as needed. An overview of the components of the Corleone workflow is given in Figure 9. In Corleone, *blocking* and *detailed comparison* both involve forests of decision trees, which are learned from examples, which are principally obtained from the crowd. For example, if we assume that *Match* is an operator that applies to an attribute, then Figure 10 illustrates an example decision tree that might be used to match pairs of books. From these

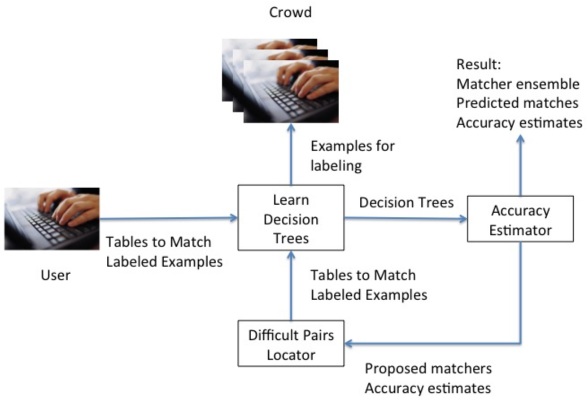


Fig. 9 Corleone workflow.

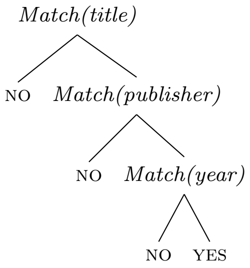


Fig. 10 Example Corleone decision tree for matching books [42].

trees, matching rules with different properties can be extracted. The steps from the workflow in Figure 9 proceed as follows:

1. *Learn Decision Trees:* Decision trees are learned using the random forest learner from the Weka data mining library [44]. In the first instance, trees are learned from two positive and two negative examples provided by the user. Additional positive and negative examples are obtained by *active learning* [92], following Algorithm 1. In active learning, a learning algorithm identifies specific data instances for labelling that are likely to be helpful for improving the result of the learning process. In crowdsourcing, active learning has been widely employed with a view to reducing the number (and thus cost) of interactions with the crowd (e.g. [31,114,54,75,108]). In Algorithm 1, LearnForests applies the existing decision tree learning algorithm to a training data set, consisting of labeled pairwise matches. In each iteration of the active learning, HighEntropyExamples identifies unlabeled pairs over which the existing forests of decision trees disagree the most. These pairs are then labeled by the crowd by way of ConsultCrowd. The algorithm terminates when the entropy of the learned forests has stabilised; in practice this means that recent rounds

of learning have not increased the consistency of the predictions made by the decision trees.

2. *Accuracy Estimator:* With its objective of performing *hands-off crowdsourcing*, given a forest of decision trees, there is a need to predict how effective it is at identifying matching records. Corleone includes a proposal that seeks to provide accurate estimates of the precision and recall of a result without the need to obtain large samples from the crowd. To do this, it requires samples of labeled data that contain a large fraction of true positives, which it obtains by learning decision trees following the approach from (1), but preferring rules that have high precision (i.e. those that return few negative examples). Thus crowdsourcing is also used to obtain samples that can be used to estimate the accuracy of the precision and recall estimates for the matching rules.
3. *Difficult Pairs Locator:* The decision trees learned at step (1) are unlikely to correctly identify all pairs that should match, and this failure could be systematic in some way (for example, the rules may be more effective at distinguishing between fiction books than textbooks). Thus Corleone supports the iterative development of matching rules by identifying examples that have proved difficult to match using the decision trees from (1), and iterating, as illustrated in Figure 9, with a view to learning new rules for these difficult-to-match pairs.

Overall, Corleone addresses some challenges, such as the cost-effective learning of match rules that are shared by other proposals, but is distinctive in using the crowd to support several different tasks with a view to automating all the steps in Figure 9. □

We now discuss other representative proposals for the use of crowds for entity resolution and other data cleaning tasks.

ZenCrowd [28] is another proposal that includes both blocking and more detailed matching of individuals, where the crowd is enlisted to contribute in specific cases. ZenCrowd identifies pairs of instances in linked open data (LOD), with a view to identifying where different publishers have published data about the same entity. A central feature here is the use of a probabilistic *factor graph* to accumulate evidence from different sources, which include automated analyses and responses to crowd tasks. From this graph, a probability is derived that a candidate pair is correct, taking into account worker reliability.

The Silk Link Discovery Workbench [54], like ZenCrowd, focuses on LOD, with a view to generating links between data published by different publishers. The crowd is used to judge whether candidate pairs are duplicates, and the action taken on the basis of the feedback is to refine the collection of detailed comparison rules: feedback provides a subset of the ground truth, which a genetic programming approach uses to evaluate the effectiveness of comparison rules [53]. In this setting, active learning is used to select candidate matches on which to seek feedback (viz. those on which the current rules disagree the most), to drive rule refinement.

Active learning algorithms to support crowd-powered ER tasks have also been developed by Mozafari *et al.* [75]. Even if their approach is generic w.r.t. the learning task (they do not make any assumption on the classification algorithm used), their experimental evaluation over an ER dataset shows competitive results with respect to other approaches specifically devoted to solve ER problems. The proposed algorithms consider *online* composition of tasks: the items to be labeled are chosen based on previous results. Also the degree of redundancy to handle uncertainty of the answers is determined dynamically, based on the difficulty of the task and on worker scores by means of *control queries* or *majority voting.*

More focused proposals target specific features of entity resolution. For example, Whang *et al.* [103] target the selection of candidate record pairs for crowdsourcing. Automated methods yield probabilities for candidate pairs, which in turn provide evidence for alternative clusters of records that represent the same real world entity. These alternative clusters are seen as possible worlds, and the identification of pairs for which the crowd is consulted involves a search for the questions whose answers would yield the highest expected accuracy. Several algorithms are proposed, and empirically evaluated, for exploring this search space. Another focused proposal is CrowdER [100], which concentrates on the construction of crowd tasks. Given candidate pairs, an algorithm is proposed that groups the candidate pairs into tasks such that feedback is obtained on every candidate pair and each task contains a bounded number of records. The purpose of this proposal is to gather the required information while minimising the number of crowd tasks, and thus the expense. In Table 1 for both of these more focused proposals we have indicated that the *Role of the Crowd* is *Open*, as the proposals are more focused on addressing what questions to put to the crowd than on what to do with the answers. Where the entity resolution includes clustering, Adaptive Crowd-based Deduplication (ACD) [102] consults the crowd for confirmation of candidate pairs that are significant for influencing the merging or splitting of candidate clusters.

Other than in ER, there has been some work on the use of crowdsourcing to evaluate the quality of public data resources. For example, CrowdCleaner [96] investigates the repair of errors identified across different web sites that describe versions of the same real world concept; examples discussed include conference deadlines and grocery prices, where the same information may be published (and updated) on several sites. Key features of the proposal include the use of *jury selection* techniques for choosing workers [16,113], and of the use of entropy for analysing the consistency of results. In TripleCheckMate [1], linked data quality is assessed using a combination of stakeholders to find data quality problems in DBpedia, and non-stakeholders to verify the presence of specific issues. In the experiment, stakeholders who are experts on linked data participate in a contest in which there is a prize for the participant who identifies the most quality problems. Then non-stakeholders participate in paid microtasks to verify the results of the stakeholders. The experiment carried out identifies various interesting features (e.g. that different types of activity are better suited to stakeholders than to non-stakeholders), and the details of how crowd tasks were presented and the potential ambiguity in results had a significant impact on reliability. In addition, Mortensen *et al.* [74] discuss how features of ontologies, such as the class hierarchy, can be cast as crowd tasks for verification.

In KATARA [20], tabular data is cleaned, in the light of information in knowledge bases such as DBpedia [10]. The (potentially dirty) tabular database is aligned with a knowledge base, with crowdsourcing used to select between candidate alignments. The crowd is asked questions such as *What is the most accurate type of the highlighted column*, on the basis of several data values from the table. Data items that are inconsistent with the knowledge base, which essentially acts as a reference data set, are candidates for repair. The cleaning of dirty data sets is also addressed by Zhang *et al.* [109], who use the crowd to distinguish between alternative values for attributes of uncertain tuples. In this approach, the candidate values for a tuple each have a probability of being correct, with such probabilities being revised in the light of (uncertain) crowd results using Bayesian inference.

State-of-the-art. As far as crowdsourcing for cleaning goes, one can see that most possibilities suggested by Table 1 have been explored, although most proposals involve the *Confirm value Type of Interaction.* This reflects the fact that automatic techniques are often used to generate candidate solutions, which are then labelled or validated using the crowd. Another notable regularity is that, in data cleaning, most crowdsourcing proposals have targeted entity resolution. In this sense, there is considerable scope for exploring the use of crowds in other data cleaning tasks. For example, there seems not to have been any work on crowdsourcing to support format transformations, which are an important part of data wrangling [58], even though there have been promising results on learning format transformations from examples [93].

3.5 Search

Crowdsourcing has also been explored in the context of search computing and information retrieval. Here, we focus on proposals that involve crowds in search-related tasks from a data management rather than an information retrieval perspective (for an appreciation of the breadth of work on crowdsourcing in information retrieval, [115] is a useful entry point). We discuss a few proposals that mostly address complementary questions but share the concern with bridging the gap between classical structured queries (whose answers can be answered exactly against data) and unstructured, keyword-based queries (for which answers are best-effort and sometimes the result of subjective judgements).

CrowdQ [29] aims to use crowds to understand the semantics of keyword-based queries to the extent that they can be answered against a data repository and also give rise to templates which can be reused, possibly in composition with other templates. The main ensuing challenges are addressed by CrowdQ as follows. Firstly, using part-of-speech tagging (POST), named-entity recognition (NER) and other natural-language processing techniques, CrowdQ generalizes a query into a template (e.g., *birthdate of Barack Obama* may yield *birthdate of [person]*). Then, the crowd is used for two purposes: firstly, to identify the relationships between entities, and, secondly, to identify the type of the expected answers (which is crucial to achieve the goal of composing templates). Finally, CrowdQ derives an executable structured query, generates the corresponding query template, and stores and indexes the latter. The whole approach is bootstrapped by offline mining of templates off search engine query logs to populate the template repository.

An example query (adapted from [29]) illustrates the whole process and the role of the crowd in answering it. Let the query Q be *birthdate of the main actor of forrest gump.* Q is annotated as follows: *[birthdate : NOUN] [main actor of : NOUN PHRASE] [forrest gump : ENTITY].* Let the following be a query template (in SPARQL) in the repository

```
SELECT ?y ?x
 WHERE {?y birthdate ?x .
        ?z stars ?y .
        ?z label [MOVIE]
       }
```

CrowdQ goes to the crowd to ask *Is 'forrest gump' an entity in the query Q, yes or no?*. Assume the answer is *yes*. CrowdQ now goes to the crowd to ask *What is the relationship between: main actor and 'forrest gump'*. Assume the answer is something like *the main actor stars in a movie*. Now, CrowdQ goes to the crowd to ask *Is the relationship between main actor and 'forrest gump' the same as between Leonardo di Caprio and Titanic, and that between Michael J. Fox and Back to the Future, yes or no?*. Assume the answer is *yes*. Finally, CrowdQ goes to the crowd to ask *Does the answer to the query Q include a birthdate, a main actor, 'forrest gump', something else? Pick one or more.* Assume the answer is *birthdate, main actor*.

The answers from the crowd allow CrowdQ to match (*type(?y) = main actor, type(?x) = birthdate*) and instantiate (*MOVIE := 'Forrest Gump'*) in the query template above to yield the following executable query:

```
SELECT ?y ?x
 WHERE {?y birthdate ?x .
        ?z stars ?y .
        ?z label 'Forrest Gump'
       }
```

The DataSift [83] system is a response to perceived shortcomings of modern search engines. Among those that DataSift singles out are: (a) no support for queries that interleave text and images; (b) limited support for queries over non-textual corpora (where, e.g., image tags are used but with less than adequate results); (c) poor performance on queries with many keywords (because of growth in noise levels); (d) no, or somewhat arbitrary, disambiguation (e.g., *brown mustang* might refer to a car or an animal, but the major search engines are in no doubt that, here, the car is meant, and if the user means the animal the keyword *horse* must be added, and some advise this); (d) no adequate support for queries involving human judgement.

These shortcomings impose on the user the burden to reformulate and refine the search terms but it may happen that the user altogether lacks the knowledge to do so. DataSift proposes to turn to the crowd to reduce the reformulation burden and to make up for the lack of knowledge. In doing so, it aims to enable rich queries over corpora as well as to filter the resulting answers.

DataSift relies on there being APIs into a crowdsourcing platform/market (such as Amazon Mechanical Turk) and into a corpus (e.g., Amazon Products, Google Images, etc.). It implements five plug-and-play components: two crowdsourced components (called **Gather** and **Filter**) interact with the crowdsourcing platform, four automated components (called **Retrieve**, **Sort** and **Weighting**) perform functions that are independent of the crowd.

One example query that interleaves text and images (adapted from [83]) and that could be grounded on Amazon Products and Google Images is *type of cable that connects to* ⟨ *IMAGE* ⟩, where ⟨ *IMAGE* ⟩ is a picture of a USB B-female socket in a printer.

DataSift makes use of configurations, which are selected compositions of the above components, to implement workflows such as, e.g., GRFS (for gather, retrieve, filter, sort). In this case, given a query Q, and a desired number s of reformulations from a desired number h of crowd workers, G goes to the market place and, posing the question *Please provide s reformulations of the following query Q*, obtains $h \times s$ reformulations of Q. Next, the retrieve component R, given a desired number n of results and, possibly, weights on terms, goes to the corpus and returns $n/(hs)$ items. Next, F goes to the market place with the items returned and, posing the question *Does the item i satisfy the following query Q, yes or no?*, obtains P positive and N negative answers. Finally, the sort component S returns a range for every item, where the higher the difference $(P - N)$ for an item, the higher its rank. The headline result in [83] is that the $\texttt{GRFW}_1\texttt{RFS}$ configuration produces the best results, with 100-150% gains in precision. In this configuration, the reformulations returned in G are weighted by the number of items for which $P > N$ obtained in F. This then enables the second occurrence of R to focus on the best reformulations, the results of which are then, finally, filtered in the second occurrence of F and ranked in S

CrowdSearcher [12] focusses on one specific shortcoming amongst those identified by DataSift, viz., the lack of adequate support for queries involving human judgement. An example (adapted from [12]) is *Which are the trendiest areas in London?*, which, if reduced to matching answers, may result in outdated answers or answers that do not relate to the user's sense of what trends count for them.

Unlike most other crowdsourcing proposals we survey here, CrowdSearcher seeks human input from social networks like Facebook and Twitter. The key insight is to combine exploratory search with crowdsourcing of information that requires human judgement. In the following account, note that what [12] calls a *query* is often best understood as a *task* sent to crowd.

A CrowdSearcher query is a transformation of an input model to an output model involving a mapping scheme that selects the crowd engines to use, the query representation for each such engine, and the resources to be used in answering the query. The input is a triple (every component of which is optional) $\langle C, N, S \rangle$, where C is the data collection offered for searching, N is a natural language (NL) query, S is a collection of queries for eliciting preferences over, for tagging, for ranking, for grouping and for manipulating elements in C. As an example (adapted from [12]), set C to be a collection of ten football players, each of which is represented by a name and photo, N to be the question *Which players do you admire?*, and S to be the singleton containing the preference query $\{\texttt{like}\}$. In another example (again, adapted from [12]), C could be a set of three new restaurants in town, represented with name, photo and location, N the request *Which restaurants serve the best fish at the best price in town?*, $S = \{\texttt{like, tag, order}\}$. The output is a pair $\langle C', S' \rangle$ where C' is the data collection returned by the crowd, and S' contains the answers to non-NL queries in S. Note that C' captures the fact that some of the queries in the input component S can result in the insertion, deletion and modification of elements in the input component C. For the first example query above, the output would have $C = C'$ and S' the counts of *like*s per player. The mapping model is a quintuple $\langle E, G, H, T, D \rangle$ where E contains the crowd engines to be used (e.g., Facebook), G specifies the crowd workers to be used (e.g., friends of the user), H are constraints on the execution (e.g., at least k answers from n different workers), T is, if the query complexity demands, the method for decomposing queries and recomposing answers, and D is the engine-specific presentation method (e.g., tabular, pins on map, etc.). For the second example above, the mapping could have $E = \{\text{Facebook}\}$, G the friends of the user who live in town, H might be a query timeout at five minute, T is not needed, and D is pins on a map denoting restaurant locations.

Differently from crowdsourcing for integration, there is more variety, if not complementarity among the proposals for crowdsourcing search above. Clearly, CrowdQ is aiming to use crowds to bridge the gap between keyword search and structured queries, with the latter very much the goal. In contrast, both DataSift and CrowdSearcher go out to the crowd in order to acquire information that is hard or impossible to obtain by computation (such as image identification or judgements of merit, respectively). This last pair can also be contrasted in terms of focus. DataSift is more focussed in tackling those shortcomings of search engines which lead users to have to engage in a sequence of reformulations of a request until the desired information emerges. CrowdSearcher is particularly focussed on facilitating the collection and subsequent use of such value judgements. Notably, since such value judgements stem from a group that shares values, it targets not crowd market places (as do most other proposals, including DataSift) but social networks, making use of the APIs exposed by such companies as Facebook and Twitter.

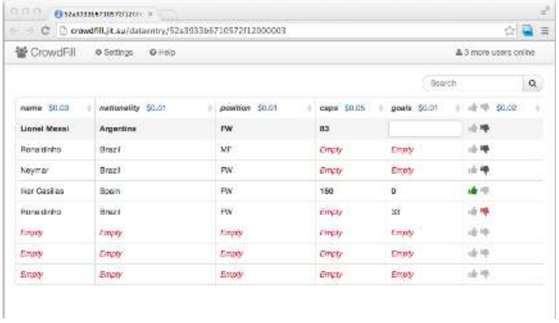
State-of-the-art. In terms of Table 2, it seems clear that the focus on crowdsourcing for search has been on improving quality, under the assumption that the return on investment from much longer response times is significant given the limitations of current search engines. It also worth noting that the surveyed proposals have not shown much concern with noise management or worker scoring. In the case of CrowdSearcher, this is perhaps understandable as their model presumes the identification on the part of the user of the workers to target (e.g., followers of someone in Twitter, friends in Facebook). In the case of DataSift, some concern with noise management is revealed by the use of majority voting.

3.6 Data Gathering

We have seen that the languages of the Crowd-Powered DBMSs in the literature (Section 3.1.1) include constructs to acquire data by crowdsourcing. There are also proposals that have studied other approaches for gathering data by means of crowdsourcing.

Higgins [62] has investigated how to apply crowdsourcing for improving the performance of an information extraction engine to automatically acquire facts, i.e. binary relations between named entities, from a large text corpora. In particular, Higgins leverages the crowd to improve the results: noisy and incomplete results are submitted to a crowd platform. HITs are composed of questions consisting of a pair of entities and a phrase (produced by the information extraction stage) that can suggest the context. Workers are asked to provide the relationship that correlates the entities, by choosing from a set of candidate answers.

Another way to harvest data is to extract data from web pages by means of *wrappers*. For data-intensive websites, a wrapper consists of a set of rules (e.g. XPath expressions) that extract values of interest from the HTML code (see [18] for a survey on wrappers). ALFRED [22] is a supervised wrapper inference system that crowdsources the production of training data with a view to reducing costs and enabling large-scale applications. It starts by generating a pool of candidate



name $0.03	nationality $0.01	position $0.01	caps $0.05	goals $0.01	$0.02
Lionel Messi	Argentina	FW	83		
Ronaldinho	Brazil	MF	Empty	Empty	
Neymar	Brazil	FW	Empty	Empty	
Iker Casillas	Spain	FW	150	0	
Ronaldinho	Brazil	FW	Empty	33	
Empty	Empty	Empty	Empty	Empty	
Empty	Empty	Empty	Empty	Empty	
Empty	Empty	Empty	Empty	Empty	

candidate table

name	nationality	position	caps	goals	↑	↓
Lionel Messi	Argentina	FW	83	37	2	0
Ronaldinho	Brazil	MF	97	33	3	0
Ronaldinho	Brazil	FW	97	33	2	1
Iker Casillas	Spain	GK	150	0	2	0
David Beckham	England	MF	115	17	1	0
Neymar	Brazil	FW			0	1
Zinedine Zidane					0	0
	France	DF			0	0
					0	0
					0	0

final table

name	nationality	position	caps	goals
Lionel Messi	Argentina	FW	83	37
Ronaldinho	Brazil	MF	97	33
Iker Casillas	Spain	GK	150	0

Fig. 11 CrowdFill: data entry interface (left); candidate and final tables (right).

extraction rules, then it poses *confirmation questions* to crowd workers, in order to rank the candidates and select the most accurate one. It also deals with the inherent uncertainty of crowd responses, adopting an *iterative* strategy for noise management combined with a *Worker comparing* strategy for scoring the workers. An important aspect of the system is that it uses a probabilistic model to estimate the quality of the candidate solutions available, so that it can decide at runtime how many workers are needed to produce an accurate solution [23].

Foundational issues in applying crowdsourcing for mining patterns from online users have been studied in [3,4], where a framework is developed to exploit the crowd for learning association rules, although the focus is more on communities of users than on paid microtasks.

Case Study: CrowdFill

CrowdFill [85] introduces an original paradigm for collecting structured data from the crowd. Figure 11(left) shows its user interface: the same partially filled table is shown to all the engaged workers, and they are asked to *concurrently* fill empty values. By providing to every worker the same view, CrowdFill stimulates a *collaborative* behaviour of the participating workers that are required to build on other workers' contributions.

The users are allowed to contribute to a *candidate table*, such as that shown in Figure 11(top-right), by means of the following possible operations: insert a new empty row; fill an empty (column) value; upvote/downvote a complete row. Every row r is then associated with a simple score $f(u_r, d_r)$ based on the number of upvotes (u_r) and downvotes (d_r) collected on it. CrowdFill adopts a simple *majority voting* scheme requiring at least three votes:

$$f(u_r, d_r) = \begin{cases} u_r - d_r & \text{, if } u_r + d_r \geq 2 \\ 0 & \text{, otherwise} \end{cases}$$

Although CrowdFill does not propose any worker scoring model, its compensation scheme, rather than being based on a fixed reward, measures the contribution of each worker to the *final table*, i.e., a table built by aggregating all the contributions and by solving the conflicting and/or incomplete information accumulated from the workers. For example, the final table in Figure 11 has been obtained by removing all incomplete rows, omitting the Beckham row because it is scored zero, and selecting a single Ronaldinho row with the best score.

In order to motivate the workers, keep them engaged and focused on the entering the needed data, a reward for every column is shown to the worker, as visible in the header row of Figure 11(left), and the rewards are given only to workers providing contributions surviving into the final table. Two (non-exclusive) types of contributions are considered: a *direct* contribution is that providing a value that becomes part of a final row; an *indirect* contribution is the first to provide a new value that occurs in the column of a final row, even if its row will not become final. Several budget allocation schemes are presented together with a preliminary experimental evaluation. □

State-of-the-art. As shown in Table 2, the *Role of the Crowd* for data gathering is *Supply data* or, for papers also dealing with *Querying* data management task, *Label data*, eventually in the context of a complete crowd-based DBMS (CrowdDB, Deco, Qurk, CrowdOp). Exceptions to this rule are Alfred [22] and Higgins [62] that use the crowd workers to build data extraction programs, and to improve an information extraction system, respectively. In this sense, there is considerable scope for exploring the use of crowds for augmenting knowledge bases by using both data/information extraction systems working on available sources and data directly provided by the crowd [14].

4 Discussion

The previous section has reviewed representative results from across the data management lifecycle; this section presents a critical discussion of some of the recurring themes and open issues encountered.

4.1 Crowdsourcing Data Management Workflows

The work surveyed uses public crowds, where workers receive financial rewards, so there is a strong incentive to maximize the return on investment. For the full range of data management tasks, proposals have sought to minimize the number of crowd tasks required, by adapting query optimization algorithms (e.g. [69,84, 101]), by deploying established techniques such as active learning (e.g. [22,54]), or by using custom techniques for predicting which crowd tasks will yield the most valuable information (e.g [100]). The state-of-the-art is that there has been widespread experimentation with different techniques on a range of problems, but that the wider applicability of the individual approaches is not easy to establish from the literature. In complex data management issues, such as entity resolution or data integration, effective solutions consist of articulated workflows that include different stages. There has been preliminary work on optimizing workflows, where several of the steps in the workflow could benefit from crowdsourcing [56,42]. Here the challenge is to decide how much of the budget to assign to each of the steps in the workflow. This raises additional questions for investigation, such as estimating the financial return on investment that can be expected from different crowd data management activities, so that crowdsourcing can be deployed by an organisation in a way that yields the maximum return. For example, in the case of pay-as-you-go data integration, there is potential benefit in seeking crowd feedback on the outcomes of more than one stage. Feedback on matching can be propagated to mapping (as when a match is disconfirmed, mappings that rely on it are disconfirmed) and *vice versa* (since evidence that a mapping is correct can be used as evidence that the matches it relies on are also correct). In spite of these opportunities, most work surveyed here has taken such subtasks in isolation. In principle, one would wish to understand better how and when to alternate and interleave the submission of crowdsourcing tasks for collecting feedback on related tasks, e.g., on matches and mappings.

4.2 Leveraging Workers Expertise

Crowdsourcing platforms have started profiling workers by their expertise and grouping tasks by categories. For example Amazon Mechanical Turk has introduced workers with *qualifications* and similarly CrowdFlower allows workers to become *skilled contributors*. AMT qualifications and CrowdFlower certifications are gained by passing admission tests, and tasks that require expert workers usually have higher wages. Also, CrowdFlower proposes *templates* to design tasks according to popular categories, such as, sentiment analysis, data collection, data validation, data categorization, image annotation. However, there is little evidence in the literature on the different types of task that can be carried out by workers with different levels of expertise, or on the implications of such decisions for quality and cost-effectiveness. A few proposals (e.g. [98, 75]) have developed methods to evaluate the error rates based on the characteristics and the difficulties of the task. Also, there are solutions to decide which tasks to assign to which workers (e.g. [60, 45]), as well as approaches that propose the combined use of expert and non-expert workers (e.g. [1, 6, 77]). Nevertheless, there is a need for additional research to make the process of recruiting from, and assigning tasks to, heterogeneous groups of workers more systematic. iCrowd [33] represents a step toward this direction: when a new task is submitted, it is assigned to known workers that performed well on similar tasks, while the accuracy of new workers is profiled by means of a set of suitable tasks. The presence of expert workers has also been exploited to improve the results of complex tasks. For example, Nguyen *et al.* [77] study how to leverage answers provided by expert (and costly) workers to evaluate random workers. There has also been work on identifying when users can share subjective feedback, by clustering workers with similar perspectives [8], but again there is a need to generalise such work beyond a specific case study. Additional research has investigated topics such as modeling the skills of crowd members [112], the identification of experts within social networks [13], and the identification of collections of crowd workers that together are most likely to generate a correct outcome [16]. Results such as these could potentially be combined to improve crowdsourcing platforms, providing comprehensive facilities for crowd workers management and selection.

4.3 Rewarding and Engagement Models

A fundamental aspect that has been neglected in the literature is the study of rewarding and engagement models. Setting the reward associated with a task is not trivial. Excessive rewards can lead to a waste of money, while low rewards do not attract workers. Note that these factors strongly influence the latency of a crowdsourced job, since in a crowdsourcing platform workers are limited, and they are more likely to choose tasks with higher rewards. Iperiotis *et al* [36] proposes a rewarding model based on explicit trade-off between task price and desired completion time. More recently, Gao *et al* [41] have developed a more involved model, which considers the crowd arrival rate and the probability of the crowd to accept the tasks. These proposals adapt to the standard pricing model of current crowdsourcing platforms; however, we believe that it is worth investigating other scenarios. For example, COPE [15] models crowdsourcing platforms as a trade market, where workers act as traders who aim to maximize their profit by submitting their contribution. Another interesting scenario is that presented by WiseMarket [17], which proposes a paradigm to recruit workers from social media platforms in order to have a larger base of workers, including people with variegated experiences from a broader demographic base than those usually found on standard crowdsourcing platforms, and develops a rewarding model based on the achievement of specific results.

Acknowledgements This work has been supported at Manchester by the UK Engineering and Physical Sciences Research Council through the VADA Programme Grant.

References

1. Acosta, M., Zaveri, A., Simperl, E., Kontokostas, D., Auer, S., Lehmann, J.: Crowdsourcing linked data quality assessment. In: ISWC (2), pp. 260–276 (2013)
2. Allahbakhsh, M., Benatallah, B., Ignjatovic, A., Motahari-Nezhad, H., Bertino, E., Dustdar, S.: Quality control in crowdsourcing systems: Issues and directions. Internet Computing, IEEE **17**(2), 76–81 (2013)
3. Amsterdamer, Y., Grossman, Y., Milo, T., Senellart, P.: Crowd mining. In: ACM SIGMOD, pp. 241–252 (2013)
4. Amsterdamer, Y., Grossman, Y., Milo, T., Senellart, P.: Crowdminer: Mining association rules from the crowd. PVLDB **6**(12), 1250–1253 (2013). URL `http://www.vldb.org/pvldb/vol6/p1250-amsterdamer.pdf`
5. Amsterdamer, Y., Milo, T.: Foundations of crowd data sourcing. ACM SIGMOD Record **43**(4), 5–14 (2015)
6. Anagnostopoulos, A., Becchetti, L., Fazzone, A., Mele, I., Riondato, M.: The importance of being expert: Efficient max-finding in crowdsourcing. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, SIGMOD '15, pp. 983–998. ACM, New York, NY, USA (2015). DOI 10.1145/2723372.2723722. URL `http://doi.acm.org/10.1145/2723372.2723722`
7. Belhajjame, K., Paton, N.W., Embury, S.M., Fernandes, A.A.A., Hedeler, C.: Incrementally improving dataspaces based on user feedback. Inf. Syst. **38**(5), 656–687 (2013)
8. Belhajjame, K., Paton, N.W., Hedeler, C., Fernandes, A.A.A.: Enabling community-driven information integration through clustering. Distributed and Parallel Databases **33**(1), 33–67 (2015). DOI 10.1007/s10619-014-7160-z. URL `http://dx.doi.org/10.1007/s10619-014-7160-z`
9. Bilenko, M., Kamath, B., Mooney, R.: Adaptive blocking: Learning to scale up record linkage. In: ICDM, pp. 87–96 (2006). DOI 10.1109/ICDM.2006.13
10. Bizer, C., Lehmann, J., Kobilarov, G., S., Becker, C., Cyganiak, R., Hellmann, S.: Dbpedia - a crystallization point for the web of data. J. Web Sem. **7**(3), 154–165 (2009)
11. Boim, R., Greenshpan, O., Milo, T., Novgorodov, S., Polyzotis, N., Tan, W.C.: Asking the right questions in crowd data sourcing. In: Data Engineering (ICDE), 2012 IEEE 28th International Conference on, pp. 1261–1264 (2012). DOI 10.1109/ICDE.2012.122
12. Bozzon, A., Brambilla, M., Ceri, S.: Answering search queries with crowdsearcher. In: Proc 21st WWW, pp. 1009–1018 (2012)

13. Bozzon, A., Brambilla, M., Ceri, S., Silvestri, M., Vesci, G.: Choosing the right crowd: expert finding in social networks. In: Joint 2013 EDBT/ICDT Conferences, EDBT '13 Proceedings, Genoa, Italy, March 18-22, 2013, pp. 637–648 (2013). DOI 10.1145/2452376.2452451. URL `http://doi.acm.org/10.1145/2452376.2452451`
14. Bühmann, L., Usbeck, R., Ngomo, A.N., Saleem, M., Both, A., Crescenzi, V., Merialdo, P., Qiu, D.: Web-scale extension of RDF knowledge bases from templated websites. In: The Semantic Web - ISWC 2014, pp. 66–81 (2014)
15. Cao, C.C., Chen, L., Jagadish, H.V.: From labor to trader: Opinion elicitation via online crowds as a market. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, pp. 1067–1076. ACM, New York, NY, USA (2014). DOI 10.1145/2623330.2623717. URL `http://doi.acm.org/10.1145/2623330.2623717`
16. Cao, C.C., She, J., Tong, Y., Chen, L.: Whom to ask? jury selection for decision making tasks on micro-blog services. PVLDB **5**(11), 1495–1506 (2012). URL `http://vldb.org/pvldb/vol5/p1495_calebchencao_vldb2012.pdf`
17. Cao, C.C., Tong, Y., Chen, L., Jagadish, H.V.: Wisemarket: A new paradigm for managing wisdom of online social users. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '13, pp. 455–463. ACM, New York, NY, USA (2013). DOI 10.1145/2487575.2487642. URL `http://doi.acm.org/10.1145/2487575.2487642`
18. Chang, C., Kayed, M., Girgis, M., Shaalan, K.: A survey of web information extraction systems. IEEE TKDE **18**(10), 1411–1428 (2006)
19. Christen, P.: A survey of indexing techniques for scalable record linkage and deduplication. IEEE TKDE **24**(9), 1537–1555 (2012). DOI 10.1109/TKDE.2011.127
20. Chu, X., Morcos, J., Ilyas, I.F., Ouzzani, M., Papotti, P., Tang, N., Ye, Y.: KATARA: A data cleaning system powered by knowledge bases and crowdsourcing. In: SIGMOD, pp. 1247–1261 (2015). DOI 10.1145/2723372.2749431. URL `http://doi.acm.org/10.1145/2723372.2749431`
21. Ciceri, E., Fraternali, P., Martinenghi, D., Tagliasacchi, M.: Crowdsourcing for top-k query processing over uncertain data. IEEE Trans. Knowl. Data Eng. **28**(1), 41–53 (2016). DOI 10.1109/TKDE.2015.2462357. URL `http://doi.ieeecomputersociety.org/10.1109/TKDE.2015.2462357`
22. Crescenzi, V., Merialdo, P., Qiu, D.: A framework for learning web wrappers from the crowd. In: WWW, pp. 261–272 (2013)
23. Crescenzi, V., Merialdo, P., Qiu, D.: Crowdsourcing large scale wrapper inference. Distributed and Parallel Databases (2014)
24. Dalvi, N., Dasgupta, A., Kumar, R., Rastogi, V.: Aggregating crowdsourced binary ratings. In: Proceedings of the 22nd international conference on World Wide Web, pp. 285–294. ACM (2013)
25. Das Sarma, A., Parameswaran, A., Widom, J.: Globally optimal crowdsourcing quality management. In: Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data, SIGMOD '16 (2016)
26. Davidson, S.B., Khanna, S., Milo, T., Roy, S.: Using the crowd for top-k and group-by queries. In: Proceedings of ICDT '13, pp. 225–236 (2013)
27. Dawid, A.P., Skene, A.M.: Maximum likelihood estimation of observer error-rates using the em algorithm. Applied statistics pp. 20–28 (1979)
28. Demartini, G., Difallah, D.E., Cudré-Mauroux, P.: Large-scale linked data integration using probabilistic reasoning and crowdsourcing. VLDB J. **22**(5), 665–687 (2013)
29. Demartini, G., Trushkowsky, B., Kraska, T., Franklin, M.J.: Crowdq: Crowdsourced query understanding. In: CIDR (2013)
30. Doan, A., Ramakrishnan, R., Halevy, A.Y.: Crowdsourcing systems on the world-wide web. Commun. ACM **54**(4), 86–96 (2011)
31. Donmez, P., Carbonell, J.G., Schneider, J.: Efficiently learning the accuracy of labeling sources for selective sampling. In: 15th ACM SIGKDD, pp. 259–268 (2009)
32. Elmagarmid, A., Ipeirotis, P., Verykios, V.: Duplicate record detection: A survey. IEEE TKDE **19**(1), 1–16 (2007). DOI 10.1109/TKDE.2007.250581
33. Fan, J., Li, G., Ooi, B.C., Tan, K.l., Feng, J.: icrowd: An adaptive crowdsourcing framework. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, pp. 1015–1030. ACM (2015)
34. Fan, J., Lu, M., Ooi, B.C., Tan, W., Zhang, M.: A hybrid machine-crowdsourcing system for matching web tables. In: IEEE 30th International Conference on Data Engineering,

Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014, pp. 976–987 (2014). DOI 10.1109/ICDE.2014.6816716. URL `http://dx.doi.org/10.1109/ICDE.2014.6816716`

35. Fan, J., Zhang, M., Kok, S., Lu, M., Ooi, B.C.: Crowdop: Query optimization for declarative crowdsourcing systems. IEEE Trans. Knowl. Data Eng. **27**(8), 2078–2092 (2015). DOI 10.1109/TKDE.2015.2407353. URL `http://dx.doi.org/10.1109/TKDE.2015.2407353`
36. Faradani, S., Hartmann, B., Ipeirotis, P.G.: What's the right price? pricing tasks for finishing on time. In: Human Computation, *AAAI Workshops*, vol. WS-11-11. AAAI (2011). URL `http://dblp.uni-trier.de/db/conf/aaai/hc2011.html#FaradaniHI11`
37. Franklin, M., Kossmann, D., Kraska, T., Ramesh, S., Xin, R.: Crowddb: Answering queries with crowdsourcing. In: ACM SIGMOD, pp. 61–72 (2011)
38. Franklin, M.J., Halevy, A.Y., Maier, D.: From databases to dataspaces: a new abstraction for information management. SIGMOD Record **34**(4), 27–33 (2005)
39. Franklin, M.J., Trushkowsky, B., Sarkar, P., Kraska, T.: Crowdsourced enumeration queries. In: Proc. ICDE, pp. 673–684 (2013). DOI 10.1109/ICDE.2013.6544865. URL `http://dx.doi.org/10.1109/ICDE.2013.6544865`
40. Gao, J., Liu, X., Ooi, B.C., Wang, H., Chen, G.: An online cost sensitive decision-making method in crowdsourcing systems. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, SIGMOD '13, pp. 217–228. ACM, New York, NY, USA (2013). DOI 10.1145/2463676.2465307. URL `http://doi.acm.org/10.1145/2463676.2465307`
41. Gao, Y., Parameswaran, A.: Finish them!: Pricing algorithms for human computation. Proceedings of the VLDB Endowment **7**(14), 1965–1976 (2014)
42. Gokhale, C., Das, S., Doan, A., Naughton, J.F., Rampalli, N., Shavlik, J.W., Zhu, X.: Corleone: hands-off crowdsourcing for entity matching. In: SIGMOD Conference, pp. 601–612 (2014)
43. Guo, S., Parameswaran, A., Garcia-Molina, H.: So who won?: Dynamic max discovery with the crowd. In: ACM SIGMOD, pp. 385–396 (2012)
44. Hall, M.A., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: an update. SIGKDD Explorations **11**(1), 10–18 (2009). DOI 10.1145/1656274.1656278. URL `http://doi.acm.org/10.1145/1656274.1656278`
45. Ho, C.J., Jabbari, S., Vaughan, J.W.: Adaptive task assignment for crowdsourced classification. In: ICML (1), pp. 534–542 (2013)
46. Howe, J.: The rise of crowdsourcing. Wired **14**(6) (2006)
47. Hung, N.Q.V., Tam, N.T., Chau, V.T., Wijaya, T.K., Miklós, Z., Aberer, K., Gal, A., Weidlich, M.: SMART: A tool for analyzing and reconciling schema matching networks. In: 31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015, pp. 1488–1491 (2015). DOI 10.1109/ICDE.2015.7113408. URL `http://dx.doi.org/10.1109/ICDE.2015.7113408`
48. Hung, N.Q.V., Tam, N.T., Miklós, Z., Aberer, K.: On leveraging crowdsourcing techniques for schema matching networks. In: DASFAA (2), pp. 139–154 (2013)
49. Hung, N.Q.V., Tam, N.T., Miklós, Z., Aberer, K., Gal, A., Weidlich, M.: Pay-as-you-go reconciliation in schema matching networks. In: IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014, pp. 220–231 (2014). DOI 10.1109/ICDE.2014.6816653. URL `http://dx.doi.org/10.1109/ICDE.2014.6816653`
50. Hung, N.Q.V., Tam, N.T., Tran, L.N., Aberer, K.: An evaluation of aggregation techniques in crowdsourcing. In: International Conference on Web Information Systems Engineering, pp. 1–15. Springer (2013)
51. Ipeirotis, P.: Analyzing the amazon mechanical turk marketplace. XRDS: ACM Crossroads **17**(2), 16–21 (2010)
52. Ipeirotis, P., Provost, F., Wang, J.: Quality management on amazon mechanical turk. In: Proc. ACM SIGKDD Workshop on Human Computation, pp. 64–67 (2010)
53. Isele, R., Bizer, C.: Learning expressive linkage rules using genetic programming. PVLDB **5**(11), 1638–1649 (2012)
54. Isele, R., Bizer, C.: Active learning of expressive linkage rules using genetic programming. J. Web Sem. **23**, 2–15 (2013)
55. Jeffery, S.R., Franklin, M.J., Halevy, A.Y.: Pay-as-you-go user feedback for dataspace systems. In: SIGMOD Conference, pp. 847–860 (2008)
56. Jeffery, S.R., Sun, L., DeLand, M., Pendar, N., Barber, R., Galdi, A.: Arnold: Declarative crowd-machine data integration. In: CIDR 2013, Sixth Biennial Conference on Innovative

Data Systems Research, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings (2013). URL http://www.cidrdb.org/cidr2013/Papers/CIDR13_Paper22.pdf

57. Joglekar, M., Garcia-Molina, H., Parameswaran, A.: Evaluating the crowd with confidence. In: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 686–694. ACM (2013)
58. Kandel, S., Paepcke, A., Hellerstein, J.M., Heer, J.: Wrangler: interactive visual specification of data transformation scripts. In: Proceedings of the International Conference on Human Factors in Computing Systems, CHI 2011, Vancouver, BC, Canada, May 7-12, 2011, pp. 3363–3372 (2011)
59. Karger, D.R., Oh, S., Shah, D.: Budget-optimal crowdsourcing using low-rank matrix approximations. In: Communication, Control, and Computing (Allerton), 2011 49th Annual Allerton Conference on, pp. 284–291. IEEE (2011)
60. Karger, D.R., Oh, S., Shah, D.: Iterative learning for reliable crowdsourcing systems. In: 25th Conf. on Neural Information Processing Systems, pp. 1953–1961 (2011)
61. Karger, D.R., Oh, S., Shah, D.: Budget-optimal task allocation for reliable crowdsourcing systems. Operations Research **62**(1), 1–24 (2014)
62. Kondreddi, S.K., Triantafillou, P., Weikum, G.: Combining information extraction and human computing for crowdsourced knowledge acquisition. In: Data Engineering (ICDE), 2014 IEEE 30th International Conference on, pp. 988–999. IEEE (2014)
63. Li, G., Wang, J., Zheng, Y., Franklin, M.J.: Crowdsourced data management: A survey. IEEE Trans. Knowl. Data Eng. **28**(9), 2296–2319 (2016)
64. Liu, X., Lu, M., Ooi, B.C., Shen, Y., Wu, S., Zhang, M.: Cdas: a crowdsourcing data analytics system. Proceedings of the VLDB Endowment **5**(10), 1040–1051 (2012)
65. Lofi, C., Maarry, K.E., Balke, W.T.: Skyline queries in crowd-enabled databases. In: Proc. 16th EDBT, pp. 465–476 (2013)
66. Marcus, A., Karger, D., Madden, S., Miller, R., Oh, S.: Counting with the crowd. PVLDB **6**(2), 109–120 (2012)
67. Marcus, A., Parameswaran, A.: Crowdsourced data management: Industry and academic perspectives. Foundations and Trends in Databases **6**(1-2), 1–161 (2015)
68. Marcus, A., Wu, E., Karger, D.R., Madden, S., Miller, R.C.: Demonstration of qurk: a query processor for humanoperators. In: SIGMOD Conference, pp. 1315–1318 (2011)
69. Marcus, A., Wu, E., Karger, D.R., Madden, S., Miller, R.C.: Human-powered sorts and joins. PVLDB **5**(1), 13–24 (2011)
70. Marge, M., Banerjee, S., Rudnicky, A.: Using the amazon mechanical turk for transcription of spoken language. In: Intl. Conf. Acoustics Speech and Signal Processing (ICASSP), pp. 5270–5273. IEEE (2010)
71. Mason, W., Suri, S.: Conducting behavioral research on amazons mechanical turk. Behavior research methods **44**(1), 1–23 (2012)
72. McCann, R., Shen, W., Doan, A.: Matching schemas in online communities: A web 2.0 approach. In: Proc. 24th ICDE, pp. 110–119 (2008)
73. Michelson, M., Knoblock, C.A.: Learning blocking schemes for record linkage. In: Proc 21st AAAI, pp. 440–445. AAAI Press (2006)
74. Mortensen, J., Alexander, P.R., Musen, M.A., Noy, N.F.: Crowdsourcing ontology verification. In: ICBO, pp. 40–45 (2013)
75. Mozafari, B., Sarkar, P., Franklin, M., Jordan, M., Madden, S.: Scaling up crowd-sourcing to very large datasets: A case for active learning. Proceedings of the VLDB Endowment **8**(2) (2014)
76. Muhammadi, J., Rabiee, H.R., Hosseini, A.: A unified statistical framework for crowd labeling. Knowledge and Information Systems **45**(2), 271–294 (2015). DOI 10.1007/s10115-014-0790-7
77. Nguyen, Q.V.H., Duong, C.T., Weidlich, M., Aberer, K.: Minimizing efforts in validating crowd answers. In: The 2015 ACM SIGMOD/PODS Conference, EPFL-CONF-204725 (2015)
78. Osorno-Gutierrez, F., Paton, N.W., Fernandes, A.A.A.: Crowdsourcing feedback for pay-as-you-go data integration. In: DBCrowd, pp. 32–37 (2013)
79. Paolacci, G., Chandler, J., Ipeirotis, P.: Running experiments on amazon mechanical turk. Judgment and Decision Making **5**(5), 411–419 (2010)
80. Parameswaran, A.G., Boyd, S., Garcia-Molina, H., Gupta, A., Polyzotis, N., Widom, J.: Optimal crowd-powered rating and filtering algorithms. PVLDB **7**(9), 685–696 (2014)

81. Parameswaran, A.G., Garcia-Molina, H., Park, H., Polyzotis, N., Ramesh, A., Widom, J.: Crowdscreen: Algorithms for filtering data with humans. In: ACM SIGMOD, pp. 361–372 (2012). DOI 10.1145/2213836.2213878. URL `http://doi.acm.org/10.1145/2213836.2213878`
82. Parameswaran, A.G., Park, H., Garcia-Molina, H., Polyzotis, N., Widom, J.: Deco: declarative crowdsourcing. In: Proc. 21st CIKM, pp. 1203–1212 (2012)
83. Parameswaran, A.G., Teh, M.H., Garcia-Molina, H., Widom, J.: Datasift: An expressive and accurate crowd-powered search toolkit. In: Proc. AAAI Conference on Human Computation and Crowdsourcing (2013)
84. Park, H., Widom, J.: Query optimization over crowdsourced data. PVLDB **6**(10), 781–792 (2013)
85. Park, H., Widom, J.: Crowdfill: Collecting structured data from the crowd. In: ACM SIGMOD (2014)
86. Quinn, A.J., Bederson, B.B.: Human computation: a survey and taxonomy of a growing field. In: CHI, pp. 1403–1412 (2011)
87. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB J. **10**(4), 334–350 (2001)
88. Raykar, V.C., Yu, S., Zhao, L.H., Jerebko, A., Florin, C., Valadez, G.H., Bogoni, L., Moy, L.: Supervised learning from multiple experts: whom to trust when everyone lies a bit. In: Proceedings of the 26th Annual international conference on machine learning, pp. 889–896. ACM (2009)
89. Sarma, A.D., Dong, X., Halevy, A.Y.: Bootstrapping pay-as-you-go data integration systems. In: SIGMOD, pp. 861–874 (2008)
90. Sarma, A.D., Parameswaran, A.G., Garcia-Molina, H., Halevy, A.Y.: Crowd-powered find algorithms. In: IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014, pp. 964–975 (2014)
91. Selke, J., Lofi, C., Balke, W.T.: Pushing the boundaries of crowd-enabled databases with query-driven schema expansion. PVLDB **5**(6), 538–549 (2012)
92. Settles, B.: Active learning. Synthesis Lectures on Artificial Intelligence and Machine Learning **6**(1), 1–114 (2012)
93. Singh, R., Gulwani, S.: Transforming spreadsheet data types using examples. In: Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016, pp. 343–356 (2016)
94. Stonebraker, M., Bruckner, D., Ilyas, I.F., Beskales, G., Cherniack, M., Zdonik, S.B., Pagan, A., Xu, S.: Data curation at scale: The data tamer system. In: CIDR 2013, Sixth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings (2013). URL `http://www.cidrdb.org/cidr2013/Papers/CIDR13_Paper28.pdf`
95. Talukdar, P.P., Jacob, M., Mehmood, M.S., Crammer, K., Ives, Z.G., Pereira, F., Guha, S.: Learning to create data-integrating queries. PVLDB **1**(1), 785–796 (2008)
96. Tong, Y., Cao, C.C., Zhang, C.J., Li, Y., Chen, L.: Crowdcleaner: Data cleaning for multi-version data on the web via crowdsourcing. In: 30th International Conference on Data Engineering, ICDE, pp. 1182–1185 (2014). DOI 10.1109/ICDE.2014.6816736. URL `http://dx.doi.org/10.1109/ICDE.2014.6816736`
97. Trushkowsky, B., Kraska, T., Franklin, M., Sarkar, P., Ramachandran, V.: Crowdsourcing enumeration queries: Estimators and interfaces. Knowledge and Data Engineering, IEEE Transactions on **27**(7), 1796–1809 (2015). DOI 10.1109/TKDE.2014.2339857
98. Venetis, P., Garcia-Molina, H., Huang, K., Polyzotis, N.: Max algorithms in crowdsourcing environments. In: Proc. WWW, pp. 989–998 (2012)
99. Verroios, V., Lofgren, P., Garcia-Molina, H.: tdp: An optimal-latency budget allocation strategy for crowdsourced maximum operations. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, pp. 1047–1062. ACM (2015)
100. Wang, J., Kraska, T., Franklin, M., Feng, J.: Crowder: Crowdsourcing entity resolution. Proc. VLDB Endow. **5**(11), 1483–1494 (2012)
101. Wang, J., Li, G., Kraska, T., Franklin, M.J., Feng, J.: Leveraging transitive relations for crowdsourced joins. In: ACM SIGMOD '13 (2013)
102. Wang, S., Xiao, X., Lee, C.: Crowd-based deduplication: An adaptive approach. In: SIGMOD, pp. 1263–1277 (2015). DOI 10.1145/2723372.2723739. URL `http://doi.acm.org/10.1145/2723372.2723739`

103. Whang, S.E., Lofgren, P., Garcia-Molina, H.: Question selection for crowd entity resolution. PVLDB **6**(6), 349–360 (2013)
104. Whitehill, J., Wu, T.f., Bergsma, J., Movellan, J.R., Ruvolo, P.L.: Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In: Advances in neural information processing systems, pp. 2035–2043 (2009)
105. Yan, Z., Zheng, N., Ives, Z.G., Talukdar, P.P., Yu, C.: Active learning in keyword search-based data integration. VLDB J. **24**(5), 611–631 (2015). DOI 10.1007/s00778-014-0374-x. URL `http://dx.doi.org/10.1007/s00778-014-0374-x`
106. Yuen, M.C., King, I., Leung, K.S.: A survey of crowdsourcing systems. In: IEEE International Conference on Social Computing, pp. 766–773 (2011)
107. Zhang, C.J., Chen, L., Jagadish, H.V., Cao, C.C.: Reducing uncertainty of schema matching via crowdsourcing. PVLDB **6**(9), 757–768 (2013)
108. Zhang, C.J., Chen, L., Tong, Y.: Mac: A probabilistic framework for query answering with machine-crowd collaboration. In: Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, pp. 11–20. ACM (2014)
109. Zhang, C.J., Chen, L., Tong, Y., Liu, Z.: Cleaning uncertain data with a noisy crowd. In: ICDE, pp. 6–17 (2015). DOI 10.1109/ICDE.2015.7113268. URL `http://dx.doi.org/10.1109/ICDE.2015.7113268`
110. Zhang, C.J., Zhao, Z., Chen, L., Jagadish, H.V., Cao, C.C.: Crowdmatcher: crowd-assisted schema matching. In: International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014, pp. 721–724 (2014). DOI 10.1145/2588555.2594515. URL `http://doi.acm.org/10.1145/2588555.2594515`
111. Zhang, J., Wu, X., Sheng, V.S.: Learning from crowdsourced labeled data: a survey. Artificial Intelligence Review pp. 1–34 (2016). DOI 10.1007/s10462-016-9491-9
112. Zhao, Z., Wei, F., Zhou, M., Chen, W., Ng, W.: Crowd-selection query processing in crowdsourcing databases: A task-driven approach. In: Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium, March 23-27, 2015., pp. 397–408 (2015). DOI 10.5441/002/edbt.2015.35. URL `http://dx.doi.org/10.5441/002/edbt.2015.35`
113. Zheng, Y., Cheng, R., Maniu, S., Mo, L.: On optimality of jury selection in crowdsourcing. In: Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium, March 23-27, 2015., pp. 193–204 (2015). DOI 10.5441/002/edbt.2015.18. URL `http://dx.doi.org/10.5441/002/edbt.2015.18`
114. Zheng, Y., Scott, S.D., Deng, K.: Active learning from multiple noisy labelers with varied costs. In: 10th ICDM, pp. 639–648. IEEE Computer Society (2010)
115. Zuccon, G., Leelanupab, T., Whiting, S., Yilmaz, E., Jose, J.M., Azzopardi, L.: Crowdsourcing interactions: using crowdsourcing for evaluating interactive information retrieval systems. Inf. Retr. **16**(2), 267–305 (2013). DOI 10.1007/s10791-012-9206-z. URL `http://dx.doi.org/10.1007/s10791-012-9206-z`