DBCrowd 2013

First VLDB Workshop on Databases and Crowdsourcing

Reynold Cheng Anish Das Sarma Silviu Maniu
Pierre Senellart

Riva Del Garda, Trento, August 26, 2013

DBCrowd 2013
First VLDB Workshop on Databases and Crowdsourcing
Riva del Garda, Trento, August 26, 2013
DBCrowd
VLDB Workshop

http://dbweb.enst.fr/events/dbcrowd2013/

Contents

Organization

Co-Chairs

Reynold Cheng	The University of Hong Kong, Hong Kong
Anish Das Sarma	Google, USA
Pierre Senellart	Télécom ParisTech, France

Vice-Chair

Silviu Maniu	The University of Hong Kong, Hong Kong

Program Committee Members

Sihem Amer-Yahia	CNRS, France
T.-H. Hubert Chan	The University of Hong Kong, Hong Kong
Michael Chau	The University of Hong Kong, Hong Kong
Lei Chen	Hong Kong University of Science and Technology, Hong Kong
Jiefeng Cheng	Shenzhen Institutes of Advanced Technology, China
Susan Davidson	University of Pennsylvania, USA
Michael J. Franklin	U.C. Berkeley, USA
Ada Waichee Fu	Chinese University of Hong Kong, Hong Kong
Donald Kossmann	ETH Zürich, Switzerland
Tim Kraska	Brown University, USA
Guoliang Li	Tsinghua University, China
Eric Lo	The Hong Kong Polytechnic University
Samuel Madden	MIT, USA
Amélie Marian	Rutgers University, USA
Atsuyuki Morishima	University of Tsukuba, Japan
Zaiqing Nie	Microsoft Research Asia, China
Jian Pei	Simon Fraser University, Canada
Mauro Sozio	Télécom Paristech, France
Martin Theobald	University of Antwerp, Belgium

External Reviewers

Jia Wang	Chinese University of Hong Kong, Hong Kong
Ben Ng	The University of Hong Kong, Hong Kong

Logo Design

Siyu Ray Lei	The University of Hong Kong, Hong Kong

Part I.

Invited Keynotes

Multi-Platform, Reactive Crowdsourcing

Invited Keynote

Stefano Ceri
Politecnico di Milano
Milan, Italy

ABSTRACT

In recent years, we developed CrowdSearcher, which integrates a conceptual framework, a specification procedure and a reactive execution control environment for designing, deploying, and monitoring crowd-based applications on top of social systems, including social networks and crowdsourcing platforms.

We show how social platforms, such as Facebook or Twitter, can be used for crowdsourcing search-related tasks, side by side with traditional crowdsourcing platforms; and we show how controlling the quality of performers and of results can lead to increased performance and interoperability.

The contribution of this talk is a broad vision that brings together crowdsourcing, social networking, expertise finding, reactive rules and multi-platform system integration, at the purpose of increasing effectiveness of crowd-based applications.

BIOGRAPHY

Stefano Ceri is professor of Database Systems at the Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB) of Politecnico di Milano. He was visiting professor at the Computer Science Department of Stanford University (1983-1990), and he is the director of Alta Scuola Politecnica, the school of excellence for top-level master students selected from Engineering, Architecture, and Design Faculties of Politecnico di Milano and Politecnico di Torino. His research work covers over three decades (1976–2013) and has been generally concerned with extending database technologies in order to incorporate new features: distribution, object-orientation, rules, streaming data; with the advent of the Web, his research has been targeted towards the engineering of Web-based applications and search systems. More recently he turned to crowdsearching and to genomic computing. He was awarded an advanced ERC Grant on Search Computing (November 2008 – October 2013), described in `http://www.search-computing.it`. He is national coordinator of the PRIN Project GenData 2020, focused on building query and data analysis systems for genomic data as produced by fast DNA sequencing technology (February 2013 – January 2016). He is author of about 300 publications on international journals and conferences (H index 57) and of 10 international books; the book *Web Information Retrieval* is in print (Springer-Verlag). He is co-editor in chief (with Mike Carey) of the book series "Data Centric Systems and Applications" (Springer-Verlag). He is the recipient of the ACM-SIGMOD "Edward T. Codd Innovation Award" (New York, June 26, 2013).

Mining the Crowd

Invited Keynote

Tova Milo
Tel Aviv University
Tel Aviv, Israel

ABSTRACT

Harnessing a crowd of Web users for data collection has recently become a wide-spread phenomenon. A key challenge is that the human knowledge forms an open world and it is thus difficult to know what kind of information we should be looking for. Classic databases have addressed this problem by data mining techniques that identify interesting data patterns. These techniques, however, are not suitable for the crowd. This is mainly due to properties of the human memory, such as the tendency to remember simple trends and summaries rather than exact details. Following these observations, we develop here a novel model for crowd mining. We will consider in the talk the logical, algorithmic, and methodological foundations needed for such a mining process, as well as the applications that can benefit from the knowledge mined from crowd.

BIOGRAPHY

Tova Milo received her Ph.D. degree in Computer Science from the Hebrew University, Jerusalem, in 1992. After graduating she worked at the INRIA research institute in Paris and at University of Toronto and returned to Israel in 1995, joining the School of Computer Science at Tel Aviv University, where she is now a full Professor and the Head of the Department. Her research focuses on advanced database applications such as data integration, XML, and semi-structured information, Data-centered Business Processes, and Crowdsourcing, studying both theoretical and practical aspects. Tova served as the Program Chair of several international conferences, including PODS, ICDT, VLDB, XSym, and WebDB. She is a member of the VLDB Endowment and the ICDT executive board and is an editor of *TODS*, the *VLDB Journal*, and the *Logical Methods in Computer Science* Journal. She has received grants from the Israel Science Foundation, the US–Israel Binational Science Foundation, the Israeli and French Ministry of Science, and the European Union. She is an ACM Fellow and a recipient of the 2010 ACM PODS Alberto O. Mendelzon Test-of-Time Award and of the prestigious EU ERC Advanced Investigators grant.

Part II.

Research Papers

Wrapper Generation Supervised by a Noisy Crowd

Valter Crescenzi, Paolo Merialdo, Disheng Qiu

Dipartimento di Ingegneria
Università degli Studi Roma Tre
Via della Vasca Navale, 79 – Rome, Italy

{crescenz, merialdo, disheng}@dia.uniroma3.it

ABSTRACT

We present solutions based on crowdsourcing platforms to support large-scale production of accurate wrappers around data-intensive websites. Our approach is based on supervised wrapper induction algorithms which demand the burden of generating the training data to the workers of a crowdsourcing platform. Workers are paid for answering simple membership queries chosen by the system. We present two algorithms: a single worker algorithm (ALF$_\eta$) and a multiple workers algorithm (ALFRED). Both the algorithms deal with the inherent uncertainty of the responses and use an active learning approach to select the most informative queries. ALFRED estimates the workers' error rate to decide at runtime how many workers are needed. The experiments that we conducted on real and synthetic data are encouraging: our approach is able to produce accurate wrappers at a low cost, even in presence of workers with a significant error rate.

1. INTRODUCTION

The abundance of data contained in web pages has motivated many research efforts towards the development of effective methods and tools for generating web wrappers, i.e., rules that allow the extraction of data from web pages. Supervised approaches to infer web wrappers have limited scalability, mainly because they require a set of training data, typically provided as labeled values. Unsupervised approaches (e.g. [3, 6]) have been investigated as an attempt to "scale-up" the wrapper generation process by overcoming the need of training data. Unfortunately, they have limited applicability because of the low precision of the produced wrappers.

Crowdsourcing platforms represent an intriguing opportunity to "scale-out" supervised wrapper inference approaches. These platforms support the assignment of mini-tasks to people recruited on the Web, and thus allow the engagement of a large number of workers to produce massive amounts of training data.

However, generating wrappers with the support of crowdsourcing platforms introduces a number of challenges that were not addressed in the literature: the mini-tasks submitted to the platform should be extremely simple, since they are performed by non-expert workers; their number should be minimized to contain the costs.

We are developing a framework, ALF [7], that addresses the above issues to let the crowd effectively and efficiently supervise the wrapper generation process.[1] ALF progressively infers a wrapper by posing *membership queries* (*MQ*), which are the simplest form of queries [1], since they admit only a yes/no answer (e.g., "*Is the string* `'John Wayne'` *a value to extract?*"); ALF implements an active learning algorithm to select the queries that more quickly bring to the generation of an accurate wrapper, thus reducing the costs [14]; and, finally, here we extend ALF to adopt a probabilistic model that considers errors (wrong answers) introduced by inaccurate workers [2, 15].

We have experimentally observed that with perfect workers, i.e., workers that do not make any mistake in answering the proposed membership queries, ALF generates the correct extraction rules reducing on average the number of queries by 4× with respect to a random choice [7]. However, it is well known that the workers recruited on crowd sourcing platforms are far from being perfect. On an empirical evaluation that we conducted on a popular crowdsourcing platform, we experienced a significant number of incorrect answers (around 10% on average) even for the simple membership queries posed by our system.

This paper extends our framework to manage workers that may return incorrect answers. First, we introduce ALF$_\eta$, which extends the underlying model of ALF to deal with the noisy answers of a single worker. The presence of errors introduces the challenging issue of deciding when to stop the learning process. Intuitively, when a worker is inaccurate, the costs of acquiring her answers may become not justified by the increment of quality in the inferred wrapper that these answers produce. ALF$_\eta$ needs an estimation of the worker's error rate to reason on the optimal number of queries that should be assigned to the worker. Unfortunately, at most a rough estimation of the error rate is available when the worker is engaged.

Then, to overcome this issue, we introduce ALFRED (ALF with REDundancy), an algorithm that builds on ALF$_\eta$ to find the best solution according to the training data provided by multiple workers. It adopts the conventional technique of facing the presence of errors by engaging multiple workers

---

[1] A demo is available at http://alfred.dia.uniroma3.it.

for solving the same tasks. However, ALFRED decides the number of workers during the learning process, at runtime, and minimizes the costs engaging only the workers actually needed to achieve the desired quality. ALFRED exploits the weighted consensus among multiple workers to estimate their error rates so that better stopping conditions can be crafted to take into account both the cost (quantified as the number of queries) and the quality of the wrapper (as estimated by a probabilistic model).

Contributions. Overall, the paper makes several contributions: (*i*) we extend our crowd based wrapper inference framework [7] in order to manage noisy answers; (*ii*) we propose a principled approach to decide at runtime how many workers should be engaged to deal with the presence of noisy answers; (*iii*) we show how to estimate the workers' error rates during the learning; (*iv*) we set several termination strategies aiming at a fair trade-off between output quality and cost; (*v*) we report a set of preliminary experimental results with both synthetic and real answers collected from the crowd.

Paper outline. The paper is organized as follows: Section 2 formalizes our setting and presents the extension to our previous probabilistic model to deal with noisy answers; Section 3 introduces ALF$_\eta$, the active learning algorithm for a single noisy worker and Section 4 presents ALFRED, its generalization to multiple workers. Section 5 reports the experimental results. Section 6 discusses related work, and Section 7 concludes the paper.

2. MODELING WRAPPER QUALITY AND NOISY WORKERS

We focus on data-intensive websites whose pages are generated by scripts that embed data from an underlying database into an HTML template. Let $U = \{p_1, \dots, p_n\}$ be an ordered set of pages generated by the same script. Given an attribute of interest published in the pages, its values can be extracted by means of an *extraction rule* (or simply *rule*). The value extracted by a rule r from a page p, denoted by $r(p)$, is either a string occurrence from the HTML source code of p, or a special *nil* value. A rule r over the pages in U returns the ordered set of values $r(p_1), \dots, r(p_n)$ and represents a concrete tool to build a vector of values, denoted by $r(U)$, indexed by the pages of U.

We propose a wrapper induction process that requires as input the set U of pages to be wrapped, and only a single initial annotation v_0 (which is assumed correct) of the attribute value to extract.[2]

The inference process starts by generating a space of hypothesis, i.e., a set $\mathcal{R}_{v_0}$ of candidate rules that extract the given initial annotations v_0. We consider extraction rules defined by means of expressions belonging to a simple fragment of XPath; namely, we use *absolute* and *relative* XPath expressions that specify paths to the leaf node containing the value to be extracted. Absolute rules specify paths that start either from the document root or from a node having an 'id'; relative rules start from a *template* node working as pivot. Textual leaves that occur once in every input page are considered template nodes [3].

The inference process evaluates the candidate rules in $\mathcal{R}_{v_0}$ by posing questions to workers recruited from a crowdsourcing platform: they are shown a page and asked whether a given value v from that page corresponds to a value of the target attribute. The goal is to select the rule working not only on the annotated sample page from which it has been generated, but also for all the other pages in the input collection U. Each query is formed by picking up the value in the set $V^{\mathcal{R}}_{v_0}(U)$ of values extracted from pages in U by the candidate rules $\mathcal{R}_{v_0}$.

Figure 1 shows an example: suppose that we are interested to generate a wrapper that extracts the Title from the fictional set of movie pages $U = \{p_1, p_2, p_3\}$ whose DOM trees are sketched in Figure 1(left). Assume that the initial annotation $v_0 =$'City of God' is supplied on the sample page p_1. Figure 1(right) shows the set $\mathcal{R}_{v_0} = \{r_1, r_2, r_3\}$ of candidate rules generated from this initial annotation. The queries composed by the inference process use the values $V^{\mathcal{R}}_{v_0}(U)$ that appear as elements of the vectors extracted by the rules in $\mathcal{R}_{v_0}$ from the pages in U.

The binary answer l, with $l \in \{-, +\}$, supplied by a worker adorns the queried value v with either a positive or a negative label, producing a *labeled value* denoted by v^l. An ordered set of k labeled values composes a *training sequence* (t.s.) denoted L^{k-1}, so that $L^0 = \{v_0^+\}$ and $L^{k+1} = L^k \cup \{v_k^l\}$.

Given a t.s. L^k, we introduce a probabilistic model for estimating the probability $P(r|L^k)$ of each candidate rule $r \in \mathcal{R}_{v_0}$ of being correct for the whole set of input pages U, and the probability $P(\overline{\mathcal{R}}_{v_0}|L^k)$ that the correct rule is not present in $\mathcal{R}_{v_0}$. These probabilities are updated after each new labeled value v_k^l is observed, i.e., a worker labels with l the value provided by a *MQ* on v_k and the t.s. is expanded to $L^{k+1} = L^k \cup \{v_k^l\}$.

Bayesian update rules compute the posterior probabilities $P(r|L^{k+1})$ and $P(\overline{\mathcal{R}}_{v_0}|L^{k+1})$ starting from the probabilities, $P(r|L^k)$ and $P(\overline{\mathcal{R}}_{v_0}|L^k)$ respectively, available before observing v_k^l. The process can be repeated treating each posterior probability as the priors required by the next iteration. The whole process is triggered using as a base case the *priors* $\mathcal{P}(\mathcal{R}_{v_0})$ of having generated the correct rule in $\mathcal{R}_{v_0}$, and the probability $\mathcal{P}(r)$ that r is a correct rule. Assuming that $\mathcal{R}_{v_0}$ is generated from a class of rules sufficiently expressive to include the target rule, we can fix $\mathcal{P}(\overline{\mathcal{R}}_{v_0}) = 0$,[3] and uniformly set $\mathcal{P}(r) = \frac{1}{|\mathcal{R}_{v_0}|}$.

Bayesian update rules require the definition of the p.d.f. $P(v_k|r, L^k)$. This is usually obtained by introducing a probabilistic *generative model* to abstract the actual process leading to the generation of t.s.. For the sake of simplicity, we adopt the *Classification Noise Process* (*CNP*) model [2] to describe inaccurate workers that may produce independent and random mistakes with an expected error rate η, and we assume that the next value to query is randomly chosen according to a uniform p.d.f. over all values $V^{\mathcal{R}}_{v_0}(U) \setminus L^k$.

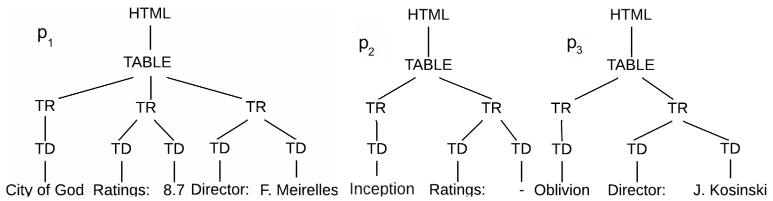
Let $P_\eta(\cdot)$ denote a p.d.f. over all possible t.s. in presence of noise, it follows:

$$P_\eta(v_k^l|r, L^k) = P(v_k^l|r, L^k) \cdot (1-\eta) + P(v_k^{-l}|r, L^k) \cdot \eta \qquad (1)$$

---

[2] Such input annotation may be supplied either manually or automatically by looking up in the page a golden value from an available database. The approach can be easily generalized to deal with multiple initial annotations.

[3] In [7], where we study the possibility of dynamically tuning the expressiveness of the class, we have developed the opposite assumption.



rules \ pages		U		
		p_1	p_2	p_3
$\mathcal{R}_{v_0}$	r_1	City of God	Inception	Oblivion
	r_2	City of God	Inception	*nil*
	r_3	City of God	*nil*	Oblivion

r_1 =/html/table/tr[1]/td
r_2 =//td[contains(.,"Ratings:")]/../../tr[1]/td
r_3 =//td[contains(.,"Director:")]/../../tr[1]/td

Figure 1: (Left) DOM of three sample pages; (Right) Extraction rules in $\mathcal{R}_{v_0} = \{r_1, r_2, r_3\}$ with v_0 ='City of God' and the set of values $V_{v_0}^{\mathcal{R}}(U)$ they extract from pages in $U = \{p_1, p_2, p_3\}$.

where $(1-\eta)$ is the probability that a worker correctly labels with l the provided value v_k, and η is the probability that she wrongly provides the opposite label, here denoted by $-l$. $P(v_k^l|r, L^k)$ is the corresponding noise-free p.d.f. and can be expressed as in [7]:

$$P(v_k^l|r, L^k) = \begin{cases} \frac{1}{|V_{v_0}^{\mathcal{R}}(U)|-|L^k|} & \text{, iff } v_k \in V^l(r) \setminus L^k \\ 0 & \text{, otherwise} \end{cases}$$

where $V^l(r)$ is the subset of values in $V_{v_0}^{\mathcal{R}}(U)$ that should be labeled l if r is the correct rule. In our generative model the values composing the t.s. L^k cannot be queried again, therefore $V_{v_0}^{\mathcal{R}}(U) \setminus L^k$ is the set of values that can be used to generate new queries.

As we discuss in the next section, these probabilities are used to effectively choose the next question, and to establish a termination condition for the learning process.

3. LEARNING EXTRACTION RULES

The probabilistic model developed in the previous section aims at computing, observed a t.s. L^k, the probability $P(r|L^k)$ that a given extraction rule r within a set of candidate rules $\mathcal{R}_{v_0}$ is correct. In this section we present alf$_\eta$, an active learning algorithm that exploits these probabilities to minimize the number of queries to the crowd workers.

Listing 1 alf$_\eta$: Active Learning Algorithm for a Single Noisy Worker

Input: a set of pages U
Input: the set of candidate rules $\mathcal{R}_{v_0}$
Input: a worker w and its associated error rate η_w

Output: a teaching sequence L^k

```
1: let $k = 1$; let $L^1 = \{v_0^+\}$;
2: while (not halt$_{ALF}(L^k)$) do
3:     $v_k \leftarrow$ chooseQuestion($L^k$);
4:     $l \leftarrow$ getAnswer($w, v_k$);
5:     $L^{k+1} \leftarrow L^k \cup \{v_k^l\}$;
6:     compute $P(r|L^{k+1})$, $\forall r \in \mathcal{R}_{v_0}$;
7:     $k \leftarrow k + 1$;
8: end while
9: return $L^k$;
```

Listing 1 contains the pseudo-code of the alf$_\eta$ algorithm: it processes a t.s. L^k built by actively asking to a worker (here modeled by means of the subprogram getAnswer()) the label of a value chosen by the subprogram chooseQuestion(); alf$_\eta$ computes a p.d.f. describing the probability of correctness over the rules in $\mathcal{R}_{v_0}$.

In every iteration (lines 2–8), the worker is asked to label a new value v_k (lines 3–4) and the t.s. is expanded (line 5). Then the probability $P(r|L^{k+1})$ is updated (line 6).

chooseQuestion() selects the next value to be labeled by the worker, i.e., the next membership query. The chosen value is that on which rules most disagree, appropriately weighted according to their probability. This is equivalent to compute the *vote entropy* [14] for each $v \in V_{v_0}^{\mathcal{R}}(U)$:

$$H(v) = -[P(v^+|L^k)\log P(v^+|L^k) + P(v^-|L^k)\log P(v^-|L^k)]$$

where: $$P(v^+|L^k) = \sum_{r \in \mathcal{R}_{v_0}: r(p_v)=v} P(r|L^k)$$

and $$P(v^-|L^k) = \sum_{r \in \mathcal{R}_{v_0}: r(p_v)\neq v} P(r|L^k)$$

are the probabilities that v is respectively either a value to extract or an incorrect value (p_v denotes the page containing v). Intuitively, the entropy measures the uncertainty of a value and querying the value with the highest entropy removes the most uncertain value:

$$\text{chooseQuestion}(L^k)\ \{\ \textbf{return}\ \ \text{argmax}_{v \in V_{v_0}^{\mathcal{R}}(U)} H(v);\ \}$$

Since different termination policies can be appropriate depending on the budget constraints and on the quality targets, we propose several implementations of halt$_{ALF}(L^k)$.

halt$_r$: A simple policy is to stop when the probability of the best rule overcomes a threshold λ_r:

$$\text{halt}_r(L^k)\ \{\ \textbf{return}\ (\max_{r \in \mathcal{R}_{v_0}} P(r|L^k) > \lambda_r);\ \}$$

The main limitation of this strategy is that it does not take into account the costs.

halt$_{MQ}$: A simple policy to upper bound the costs is to stop as soon as the algorithm runs out of a "budget" of λ_{MQ} membership queries.

$$\text{halt}_{MQ}(L^k)\ \{\ \textbf{return}\ (|L^k| > \lambda_{MQ});\ \}$$

The main limitation of this strategy is that it does not consider the quality of the output rules.

halt$_H$: A trade-off between quality and cost can be set by posing only queries that contribute enough to the quality of the inferred rules, and stopping as soon as the costs are considered not correctly rewarded by the increment of quality in the output rules. It turns out that this can be easily modeled in term of the maximum entropy. halt$_H$ terminates as soon as the maximum entropy of the values is below a threshold λ_H, i.e., no value is uncertain enough to deserve a query:

$$\text{halt}_H(L^k)\ \{\ \textbf{return}\ (\max_{v \in V_{v_0}^{\mathcal{R}}(U)} H(v) < \lambda_H);\ \}$$

Whichever is the termination policy adopted by ALF_η, its efficacy in achieving an optimal trade-off between quality and the costs for the learning tasks is strongly affected by a correct estimation of the worker's error rate, η. An incorrect evaluation of η can lead to a waste of queries (when the error rate is overestimated), or to a quality loss (when it is underestimated), as confirmed by the experiments with different termination policies (reported in Section 5.2).

In our experimental evaluation, we use as a termination policy the following combination:

$$\text{HALT}_{\text{ALF}_\eta}(W, L^k) = \text{HALT}_H(L^k) \textbf{ or } \text{HALT}_{MQ}(L^k).$$

This policy leverages the same trade-off between quality and cost as in HALT_H, but focuses on the cost side by limiting through HALT_{MQ} the budget allocated for each worker.

4. INFERENCE WITH A NOISY CROWD

We now introduce another algorithm, ALFRED, that follows the conventional approach based on redundancy [15] to deal with inaccurate workers. ALFRED improves ALF_η robustness by dynamically recruiting additional workers to whom it dispatches redundant tasks. It combines the answers provided by a set W of multiple workers on the same task to estimate the workers' error rate, as well as to find the most likely rule, at the same time.

Our probabilistic model can easily deal with multiple workers by appending the t.s. L_w of every worker $w \in W$ into a unique t.s., denoted by $L = \biguplus_{w \in W} L_w$, which is then used to train ALF_η. However, this approach raises two issues: (*i*) it is not clear how many workers should be involved in the learning process to achieve a good trade-off between output quality and cost; (*ii*) a correct estimation of the workers' error rates strongly affects ALF_η performances, and each worker could have an error rate radically different from others.

To overcome these issues, ALFRED dynamically chooses the amount of redundancy needed, and it exploits the cumulative knowledge both to find the solution on which most of the weighted workers consensus converges, and to estimate the workers' error rates.

Listing 2 illustrates ALFRED pseudo-code: the main loop (lines 2–15) alternates two phases. First, the workers are engaged (lines 3–8): each worker w trains an ALF_η instance, thus producing a t.s. L_w. In this phase, the worker is associated with an initial error rate (line 5).[4] The second phase (lines 10–14) starts as soon as the first phase has accumulated enough workers (at least W_0). The goal of this phase is to leverage all the t.s. provided by the workers in order to compute both the p.d.f. of the rules $P(r|L)$ and the individual error rate η_w of each worker $w \in W$.

Our solution is inspired by the work on *Truth Finding Problems* [10], and exploits the mutual dependency between the probability of correctness of the rules and the error rates of the workers answering the *MQ*. The error rate is defined in term of probability as follows:

$$\eta_w = \frac{\sum_{v_k^l \in L_w} \begin{cases} 1 - P(v_k^+|L) & \text{, iff } l = + \\ P(v_k^+|L) & \text{, iff } l = - \end{cases}}{|L_w|}$$

and it can be interpreted as the probability of the worker of correctly answering a *MQ*, estimated by averaging over all the *MQ* in the t.s. L_w that she has provided. Since also the probability is defined in term of error rates, as shown in Eq. 1, their computation is interleaved (lines 11 and 13) until their values do not significantly change anymore.[5]

Listing 2 ALFRED: Active Learning Algorithm with Multiple Noisy Workers

Input: a set of pages U
Input: the set of candidate rules $\mathcal{R}_{v_0}$

Parameter: number of initial workers W_0
Parameter: a threshold λ_η for error rates convergence

Output: the most probable extraction rule $r \in \mathcal{R}_{v_0}$

```
 1: let $W = \emptyset$; // set of engaged workers
 2: repeat
 3:     repeat
 4:         let $w$ = ENGAGEWORKER();
 5:         let $\eta_w$ = GETERRORRATE($w$);
 6:         let $L_w = \text{ALF}_\eta(U, \mathcal{R}_{v_0}, w)$;
 7:         $W \leftarrow W \cup \{w\}$;
 8:     until ($|W| < W_0$); // wait for workers
 9:     let $L = \biguplus_{w \in W} L_w$; // append all t.s.
10:     repeat
11:         compute $P(r|L)$ by using $\eta_w$, $\forall r \in \mathcal{R}_{v_0}$ ;
12:         let $\eta_w^{prev} = \eta_w$, $\forall w \in W$ ; // save previous $\eta$
13:         compute $\eta_w$ by using $P(r|L)$, $\forall w \in W$;
14:     until ($\sum_{w \in W} (\eta_w - \eta_w^{prev})^2 > \lambda_\eta \cdot |W|$);
15: until ($\text{HALT}_{\text{ALFRED}}(W, L)$);
16: return $\text{argmax}_{r \in \mathcal{R}_{v_0}} P(r|L)$;
```

The termination condition of ALFRED can be set according to different policies by specifying $\text{HALT}_{\text{ALFRED}}$. In our implementation, in order to take into account both budget constraints and quality targets, we used the following combination:

$$\text{HALT}_{\text{ALFRED}}(W, L) = \text{HALT}_r(L) \textbf{ or } \text{HALT}_{MQ}(L)$$

Observe that the recruitment of new workers negatively influences the latency of the system. A strategy to contain the latency of the system is to recruit bulk workers, but this is beyond the scope of the present paper.

5. EXPERIMENTAL EVALUATION

We use a dataset composed of 5 collections of pages: actor and movie pages from www.imdb.com; band and album pages from www.allmusic.com; stock quote pages from www.nasdaq.com. For each collection we selected about 10 attributes, for a total of 40 attributes. Then we manually crafted a golden XPath rule for each attribute. We randomly selected a training sample set of 500 pages from each collection of pages, and another (disjoint) test set of 2,000 pages. Our algorithm was run on the training sample, and the output extraction rules were evaluated on the test set. We compared the (non-null) values extracted by the golden rules against those extracted by the rules generated by our

---

[4] We use the average error rate empirically estimated in our experiment with real workers; another option is to derive this value from the workers ranks provided by the crowdsourcing platform.

[5] We have empirically observed the convergence of the algorithm. A formal proof is left as future work.

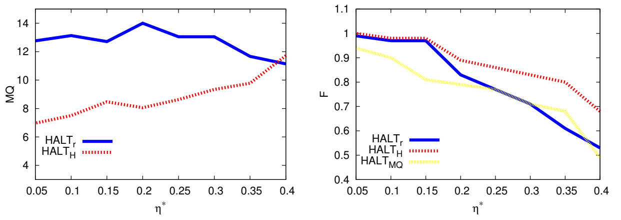


Figure 2: ALF$_\eta$ ($\eta = 0.09$) sensitivity to worker error rate η^*: Cost (left) and quality (right) of the output wrapper.

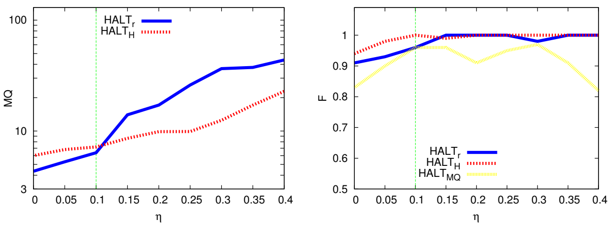


Figure 3: ALF$_\eta$ sensitivity to the expected worker error rate η with a noisy worker $\eta^* = 0.09$: Cost (left) and quality (right) of the output wrapper.

algorithm. For every generated rule r, and given a test set of pages U, we computed precision (P), recall (R), and F-measure (F) at the value level w.r.t. the corresponding golden rule r_g, as follows: $P = \frac{|r_g(U) \cap r(U)|}{|r(U)|}$; $R = \frac{|r_g(U) \cap r(U)|}{|r_g(U)|}$; $F = 2\frac{P \cdot R}{P+R}$.

We report the results of two sets of experiments: the first one was conducted to test ALF$_\eta$ with a single worker, in the second set of experiments we consider ALFRED in presence of multiple workers. We evaluate our algorithms by using synthetic workers following the *CNP* probabilistic model [2], i.e., workers making random and independent errors with a fixed error rate η^*.

In order to estimate the error rate distribution over a population of real workers, we also conducted a preliminary experiment with workers engaged by means of CrowdFlower, a meta platform that offers services to recruit workers on AMT.

5.1 Experiments with AMT Workers

The main intent of this preliminary experiment was to evaluate the error rate distribution of a population of real workers recruited on a crowdsourcing platform. We submitted 100 tasks to 100 distinct workers. Each task was paid 10¢ and consisted on a set of 20 *MQ* to generate the extraction rules for several attributes of our dataset. The ground truth was straightforwardly obtained by the results of our golden extraction rules. We used ALF$_\eta$ configured with HALT$_r$ as termination condition ($\lambda_r = 0.8$), and with $\eta = 0.1$.

The average error rate of an AMT worker was $\widehat{\eta} = 0.09$, with a standard deviation of $\widehat{\sigma}_\eta = 0.11$. About 1/3 of the workers responded correctly to all the queries, and the response time for each *MQ* was around 7 seconds. On average the number of *MQ* posed by the system to infer a rule was 4, and each task contained enough queries to learn 5 rules.

The information obtained from this experiment was then used to set up realistic configurations of the synthetic workers for the other experiments: the average error rate empirically observed $\widehat{\eta}$ is used to set ALF$_\eta$'s parameter $\eta = \widehat{\eta}$ as well as the initial worker error rate estimation of ALFRED, $\eta_w = \widehat{\eta}$; also, the synthetic workers used to test ALFRED are created using the same error rate distribution observed on real workers.

5.2 Single Noisy Worker

The main goal of the experiment was to evaluate the effects of the workers' mistakes on ALF$_\eta$, our algorithm in absence of redundancy. In figure 2 we show the effects of an inaccurate worker with different termination strategies, by setting $\eta = \widehat{\eta} = 0.09$. We simulated workers with an error rate η^* increasing from 0.05 to 0.4. The thresholds of the three termination conditions considered, HALT$_r$, HALT$_{MQ}$ and HALT$_H$ were set to $\lambda_r = 0.8$, $\lambda_{MQ} = 5$ and $\lambda_H = 0.2$, respectively.

As the error rate η^* of the worker increases, the results degrade with every termination strategy, as expected. However, HALT$_H$ detects the wider presence of uncertain values, and tries to compensate with a greater number of queries; conversely, since HALT$_r$ focuses only on the most likely rule, it poses a rather steady number of queries, and the output quality is more seriously compromised.

We then empirically evaluated how an incorrect setting of the parameter η, i.e., the expected worker error rate, influences ALF$_\eta$ performances. We used a single worker with $\eta^* = \widehat{\eta} = 0.09$, and repeated several inference processes, configuring ALF$_\eta$ with η ranging from $\eta = 0$ to 0.4 as reported in Figure 3.

When the system overestimates the accuracy of worker ($\eta < \eta^*$) we observe a reduction of the number of *MQ*, but the quality of the wrapper drops. The system trusts the workers and terminates quickly, thus posing less questions than actually needed. When the system underestimates the worker accuracy ($\eta > \eta^*$), some *MQ* are wasted since the system does not trust the worker. With an η larger than η^* by +0.3, HALT$_r$ requires more than 40 *MQ*, i.e., 5× those required when $\eta = \eta^*$. Observe that many *MQ* are wasted since the F-measure gain is less than 5%.

	average			max		
	$\lvert W\rvert$	F	$\#MQ$	$\lvert W\rvert$	$\#MQ$	σ_F
ALF$_\eta$	1	0.92	7.58	1	11	0.17

Table 1: ALF$_\eta$ inference with synthetic workers

Table 1 reports ALF$_\eta$ results when the termination policy HALT$_{\text{ALF}_\eta}$ has been instantiated by setting the parameters $\lambda_H = 0.2$ and $\lambda_{MQ} = 10$. ALF$_\eta$ requires just a few queries ($\#MQ = 7.58$) to learn rather accurate wrappers ($F = 0.92$). However, there is a significant standard deviation ($\sigma_F = 17\%$) in the output quality that makes the algorithm not that robust to workers' errors.

5.3 Multiple Noisy Workers

As discussed in Section 4, ALFRED builds on ALF$_\eta$, and recruits additional workers to estimate their error rate and to find the correct rule at the same time. We rely on the termination strategy of the outer algorithm (HALT$_{\text{ALFRED}}$) to achieve the target quality, while for the inner ALF$_\eta$ instance we use the same termination policy (HALT$_{\text{ALF}_\eta}$) focused on

	average				max		
	$\lvert W\rvert$	F	$\#MQ$	$\lvert\eta_w - \eta^*\rvert$	$\lvert W\rvert$	$\#MQ$	σ_F
ALFRED$_{no}$	2.33	1	18.6	—	9	83	0.01
ALFRED	2.07	1	16.1	0.8%	4	44	0.01
ALFRED*	2.05	1	16.07	0%	4	40	0.01

Table 2: ALFRED inference with synthetic workers

the costs as in the previous experiment. To evaluate ALFRED performances, we organized as many tasks as the number of attributes of our experiment (40). For each task we executed ALFRED (with $\lambda_\eta = 10^{-4}$) by recruiting workers from a virtual population with the same error rate distribution observed over the real workers (the results have been averaged over 20 executions).

ALFRED's results in Table 2 demonstrate the role played by the workers' error rate estimation. We compare the algorithm against a baseline (ALFRED$_{no}$) in which the error rate estimation is disabled (we just set $\eta_w = \widehat{\eta}$), and against a bound (ALFRED*) in which an oracle sets $\eta_w = \eta^*$. The workers' error rate estimation is precise ($|\eta_w - \eta^*| = 0.8\%$ when the learning terminates), and it allows the system to save queries (16.1 vs 18.6 on average). The average number of *MQ* posed by ALFRED to learn the correct rule is only a negligible amount larger than the lower bound set by ALFRED*. The costs are more than twice those paid running ALF$_\eta$ with a single worker (16.1 vs. 7.58). However, notice that ALFRED always concluded the tasks with a perfect result, and that it is robust to workers' error rates ($\sigma_F = 1\%$).

ALFRED terminates in most of the cases (94%) engaging only 2 workers, and seldom recruited 3 and 4 workers (5% and 1%, respectively). Overall, ALFRED was able to recruit more workers, thus paying their answers, only when needed to achieve the target quality of the output wrapper.

6. RELATED WORK

Wrapper induction for extracting data from web pages has been subject of many researches [5]. A wrapper inference approach tolerant to noise in the training data has been proposed in [8], however it applies only for domains where it is possible to automatically obtain a set of annotations.

Active learning approaches [14] have recently gained interest as they can produce exponential improvements over the number of samples wrt traditional supervised approaches [4]. The advent of crowdsourcing platforms has led to new challenges. The main issue is to learn from noisy observations generated by non expert users.

Wrapper induction techniques that rely on active learning approaches have been proposed in [11, 13]. These studies rely on a more complicated user interaction than ours, since the user has to choose the correct wrapper within a set of ranked solutions. Also, they do not consider the presence of noise in the training data. Many works have studied the problem of learning with noisy data coming from crowdsourcing platform workers, e.g., [9, 12, 15]. [15] shows that when labeling is not perfect, selective acquisition of multiple good labels is crucial and that repeated-labeling can improve label quality and model quality. [12] proposes a crowdsourcing assisted system, CDAS, for data analytics tasks; the system faces the presence of noisy answers by submitting redundant tasks and adopts a voting strategy to estimate the correct solution. Compared to our approach, the number of workers as well as their accuracies are statically determined, based on the workers' historical performances.

In our previous work [7], we studied wrapper inference with a single and perfect worker.

7. CONCLUSIONS

We presented wrapper inference algorithms specifically tailored for working with the support of crowdsourcing platforms. Our approach allows the wrappers to be generated by posing simple questions, membership queries, to workers engaged on a crowdsourcing platform. We proposed two algorithms that consider the possibility of noisy answers: ALF$_\eta$ recruits a single worker, ALFRED can dynamically engage multiple workers to improve the quality of the solution. We showed that ALFRED can produce high quality wrappers at reasonable costs, and that the quality of the output wrapper is highly predictable.

8. REFERENCES

[1] D. Angluin. Queries revisited. *Theor. Comput. Sci.*, 313(2):175–194, 2004.

[2] D. Angluin and P. Laird. Learning from noisy examples. *Mach. Learn.*, 2(4):343–370, Apr. 1988.

[3] A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *SIGMOD 2003*.

[4] M.-F. Balcan, S. Hanneke, and J. W. Vaughan. The true sample complexity of active learning. *Machine Learning*, 80(2-3):111–139, 2010.

[5] C.-H. Chang, M. Kayed, M. R. Girgis, and K. F. Shaalan. A survey of web information extraction systems. *IEEE Trans. Knowl. Data Eng.*, 18(10):1411–1428, 2006.

[6] V. Crescenzi and P. Merialdo. Wrapper inference for ambiguous web pages. JAAI, 22(1&2):21–52, 2008.

[7] V. Crescenzi, P. Merialdo, and D. Qiu. A framework for learning web wrappers from the crowd. In *WWW 2013*.

[8] N. N. Dalvi, R. Kumar, and M. A. Soliman. Automatic wrappers for large scale web extraction. *PVLDB*, 4(4):219–230, 2011.

[9] A. Doan, R. Ramakrishnan, and A. Y. Halevy. Crowdsourcing systems on the world-wide web. *Commun. ACM*, 54(4):86–96, Apr. 2011.

[10] X. Dong, L. Berti-Equille, Y. Hu, and D. Srivastava. Solomon: Seeking the truth via copying detection. *PVLDB*, 3(2):1617–1620, 2010.

[11] U. Irmak and T. Suel. Interactive wrapper generation with minimal user effort. In *WWW 2006*.

[12] X. Liu, M. Lu, B. C. Ooi, Y. Shen, S. Wu, and M. Zhang. Cdas: a crowdsourcing data analytics system. In *VLDB 2012*.

[13] I. Muslea, S. Minton, and C. A. Knoblock. B. Settles. Active learning with multiple views. *JAIR 2006*.

[14] Active learning literature survey. CS Tech. Rep. 1648, University of Wisconsin–Madison, 2009.

[15] V. S. Sheng, F. Provost, and P. G. Ipeirotis. Get another label? Improving data quality and data mining using multiple, noisy labelers. In *KDD 2008*.

Crowdsourcing to Mobile Users: A Study of the Role of Platforms and Tasks

Vincenzo Della Mea Eddy Maddalena Stefano Mizzaro
Department of Mathematics and Computer Science
University of Udine
Udine, Italy
vincenzo.dellamea@uniud.it, eddy.maddalena@uniud.it, mizzaro@uniud.it

ABSTRACT

We study whether the task currently proposed on crowdsourcing platforms are adequate to mobile devices. We aim at understanding both (i) which crowdsourcing platforms, among the existing ones, are more adequate to mobile devices, and (ii) which kinds of tasks are more adequate to mobile devices. Results of a user study hint that: some crowdsourcing platforms seem more adequate to mobile devices than others; some inadequacy issues seem rather superficial and can be resolved by a better task design; some kinds of tasks are more adequate than others; and there might be some unexpected opportunities with mobile devices.

Categories and Subject Descriptors

H.4.m [**Information systems applications**]: Miscellaneous

General Terms

Experimentation, Measurement.

Keywords

Crowdsourcing, mobile devices.

1. INTRODUCTION AND AIMS

Among the phenomena that are acquiring increasing importance in the information technology landscape, two are the subjects of this paper: (i) crowdsourcing, and (ii) mobile devices and applications.

Crowdsourcing, i.e., the outsourcing of tasks typically performed by a few experts to a large crowd as an open call, has been shown to be reasonably effective in many cases, like Wikipedia, the Chess match of Kasparov against the world in 1999, and several others (see, e.g., [4] or even `http://en.wikipedia.org/wiki/Crowdsourcing`). Several *crowdsourcing platforms* (Amazon Mechanical Turk being probably the most known) have also appeared on the Web: they allow requesters to post the tasks they want to crowdsource and workers to perform those tasks for a small reward (usually a few cents).

Meanwhile, mobile devices (phones, smartphones, tablets, and in the near future glasses, watches, and so on) have become ubiquitous and are used to access the Web. According to several statistics, in the next few years there will be more Web accesses by mobile devices than by classical desktop/laptop computers (see, e.g., [6]).

In this paper we study the intersection of mobile and crowdsourcing. We aim at understanding whether the task currently proposed on crowdsourcing platforms are adequate to mobile devices. By "adequate" we mean that they can be performed effectively by using a mobile device in place of a desktop/laptop computer. We specifically seek to answer two research questions:

Q1 Which crowdsourcing platforms, among the existing ones, are more adequate to mobile devices?

Q2 Which kinds of tasks are more adequate to mobile devices?

Besides the above mentioned statistics on increasing mobile usage, this research is also justified by the fact that today quite often people access the Web on their mobile phones for short periods of time, for example while commuting to work on train or underground, while waiting for a bus or for a friend, while in a car (and not driving), while standing in a queue, etc. In other terms, there is plenty of human workforce available for a few minutes (or seconds) bursts, and this kind of workforce seems perfect for the crowdsourcing scenario, where the tasks are usually short and the reward is usually low. Moreover, some crowdsourcing tasks could be more adequate to a mobile scenario than to a classical desktop one. For example, taking pictures of some point of interest (like a monument, a paint, or a billboard), describing a real life scene, or even recording movements, destinations, and trajectories in an urban traffic setting. However, to fruitfully exploit this workforce, it is necessary that the platforms are adequate and tasks are feasible. This consideration also underlies our choice of focussing on the worker side and neglecting the requester part.

The paper is structured as follows. In Section 2 we briefly survey the related work on mobile and crowdsourcing, trying to focus on the research involving both aspects. In Sections 3 and 4 we describe two experiments aiming at answering the two research questions above. In Section 5 we draw conclusions and sketch future developments.

2. RELATED WORK

Although crowdsourcing commercial platforms seem designed with a desktop/laptop user in mind, there has already been some work on the idea of having workers using mobile devices. We briefly survey it in this section.

Musthag and Ganesan[7] focus on mobile micro-task market and present some statistics on mobile workers behavior.

The mCrowd platform [11] is an iPhone based mobile crowdsourcing platform that enables mobile users to act as both requester and workers, and focuses on tasks like geolocation-aware image collection, road traffic monitoring, etc., that exploit the rich array of sensors available on iPhones.

Eagle [2] describes txteagle, a mobile crowdsourcing marketplace used in Kenya and Rwanda for tasks like translations, polls, and transcriptions.

Location-based distribution of tasks to mobile workers is proposed in [1]. Some design criteria for mobile crowdsourcing platforms are also presented and discussed. A similar approach, focused on the specific domain of news reporting is presented in [9]: SMS messages are used for location based assignment for crowdsourcing news.

Narula and colleagues [8] focus on low-end mobile devices and present MobileWorks, a platform for OCR tasks specifically aimed at users from the developing world. Experimental results demonstrate a high rate of task completion (120 per hour) and a high accuracy (99%). A similar approach is presented in [3], where the mClerk system is described. Some experimental results again witness the feasibility of the approach. Some discussion of the viral diffusion of the system among workers is also discussed.

As a different approach, the CrowdSearch system, an image search service for mobile phones that relies on Amazon Mechanical Turk, is presented in [10]. It is interesting because, although it does not exploit a mobile crowd, it is an example of exploiting a crowd in (almost) real time.

3. EXPERIMENT 1

3.1 Aims

The first experiment aims to verify the suitability of existing crowdsourcing platforms to mobile devices (see question Q1 in Section 1). We asked the participants to estimate the difficulty of performing a task on both a mobile device and a desktop/laptop computer.

3.2 Participants

Sixteen participants were involved in the experiment. All of them were italian students, aged between 16 and 30. We required a good knowledge of English and familiarity with computers and smartphones. Participants were randomly subdivided into 4 groups (U_1,U_2,U_3,U_4), each one containing four participants.

3.3 Data

We selected four among the most popular crowdsourcing platforms (see Table 1). We downloaded some randomly selected tasks from these platform, for a total of 2717 tasks (the exact number for each platform is shown in the third column in Table 1). The download has been performed in October and November 2012. The downloaded tasks are among those that can be performed by any requester, i.e., without any qualification. These are not huge samples: for example, on mTurk one can count hundreds of thousands of tasks available per month [5]. Though, the samples are neither negligible, since they count around $1\% - 5\%$. For each task we extracted: identifier, title, required proof, remuneration, time needed, requester identifier, and description. The task collection is available upon request. Three examples of tasks in our collection are (errors included):

id	**Platform name**	**URL**	**# of tasks**
mTurk	Amazon Mechanical Turk	`mturk.com`	1154
micW	Micro Workers	`microworkers.com`	1302
minW	Minute Workers	`minuteworkers.com`	86
shortT	Short Task	`shorttask.com`	175

Table 1: Platforms

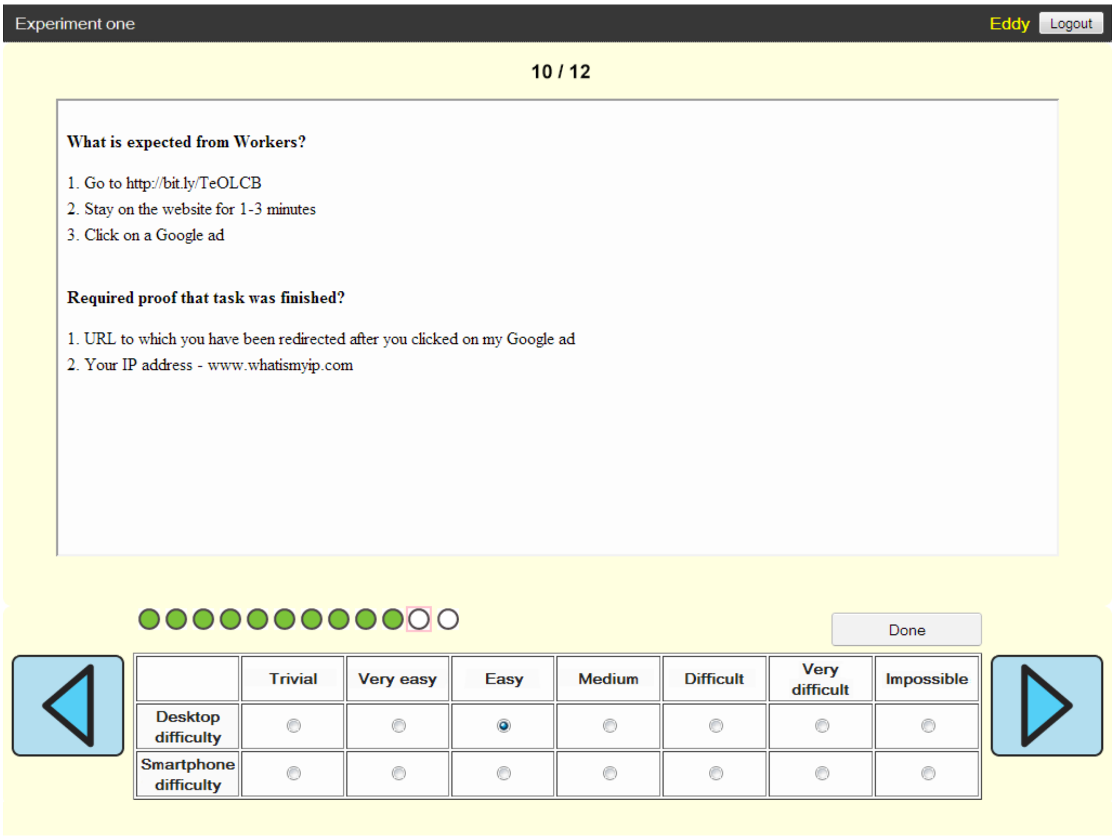
- **Task example 1:**
  1. *Go to http://goo.gl/Dlzk*
  2. *Click the link to go to the download*
  3. *Complete a survey/offer on Sharecash and download the file*
  4. *Send proof*
- **Task example 2:**
  1. *Go to http://OneDollarRiches.com/5737*
  2. *Click on Join Now button*
  3. *Invest 1 dollar by logging in into your Alertpay account*
  4. *After that enter you personal details and login.*
  5. *Join and finish signing up*

  While Sign up use same e-mail of your Alertpay account. because when u make ur refferaf there 1$ sing up go direct into ur alterpay account.
- **Task example 3:** *Find the details for this Restaurant*
  - *For this restaurant below, enter the details below*
  - *You must confirm that the restaurant is still open*
  - *Include the full address, e.g. http://www.thecheesecakefactory.com*
  - *Do not include URLs to city guides and listings like Citysearch*

  Restaurant : Akasha Organics 160 North Main St. Ketchum
  Fill in the text fields with this information: Still open, Restaurant name, Website Address, Phone number, Street Address, City, State, Zip code.

3.4 Methods

We randomly extracted 48 tasks, 12 from each platform, and divided them into 4 groups (T_1, T_2, T_3, T_4). Each group contains 12 tasks (3 tasks from each of the 4 platforms). Task group T_i was assigned to user group U_i (e.g., task group T_1 was assigned user group U_1). We developed a web application to show to each participant the group of 12 tasks assigned to his/her user group (see Figure 1). By using this



What is expected from Workers?

1. Go to http://bit.ly/TeOLCB
2. Stay on the website for 1-3 minutes
3. Click on a Google ad

Required proof that task was finished?

1. URL to which you have been redirected after you clicked on my Google ad
2. Your IP address - www.whatismyip.com

	Trivial	Very easy	Easy	Medium	Difficult	Very difficult	Impossible
Desktop difficulty	○	○	●	○	○	○	○
Smartphone difficulty	○	○	○	○	○	○	○

Figure 1: The interface used in the first experiment (translated into English)

application, each participant recorded two estimates of difficulty for each task, one for a desktop and one for a mobile device (see the bottom part of the figure). Tasks were presented in random order and participants did not know from which platform the tasks were extracted.

Difficulty was provided on a seven points scale ranging from trivial to impossible. For each task we therefore obtained 4 estimates (from the participants in the same group). We then converted the labels into the [0..6] range and calculated the average of difficulty estimates.

3.5 Results

Figure 2 shows the averaged estimated difficulty, on desktop and mobile, for each platform. Tasks from mTurk are estimated slightly more difficult than MicroWorkers, MinuteWorkers, and ShortTask. The difference of difficulty estimates between desktop and mobile is also shown in Figure 3: difficulty estimation is consistently higher on mobile devices, both in absolute terms and as a percentage of the desktop difficulty.

By manually analyzing the task collection we realized that some of them are inadequate to mobile devices for some typical reasons:

- too long description;
- technical obstacles like scrolling problems, unsupported audio formats and/or plugins, pages with Adobe Flash, etc.;
- use of frame attribute in html pages;
- bad layout in a small resolution display;
- need of a high power CPU.

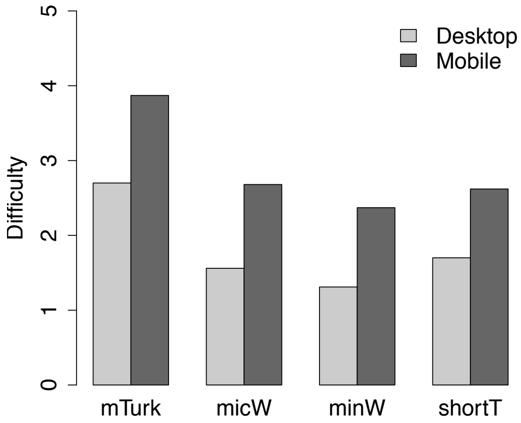


Figure 2: Estimated difficulty

Some of these task issues seem due to the task content, while some other depend on how the Web interface is realized. Many of them seem rather superficial and can be overcome by a better task design and/or better user interfaces.

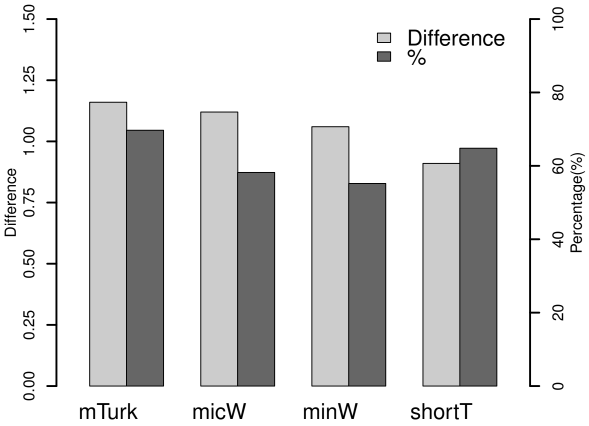


Figure 3: Mobile-desktop difference of estimated difficulty, as absolute time (bars on the left) and as a fraction (right)

4. EXPERIMENT 2

4.1 Aims

The aim of the second experiment is to identify which task kinds are more adequate for mobile devices (see question Q2 in Section 1). We therefore now focus on task features, and not on platforms. Also, in place of asking estimates to participants, we required them to actually perform the tasks on both desktop and mobile devices and we measured the time spent on each task. Participants used two prototype platforms that we built ad hoc for the experiment: one for desktop devices using Google Web Toolkit, and the other specifically made for mobile devices, by means of an Android application. Figure 4 shows the resulting user interfaces.

4.2 Participants and Data

The 16 participants (the same as in the previous experiment) were subdivided into 4 groups labeled U_1, U_2, U_3, U_4.

To identify the kinds of task in a somehow objective way, we relied on the task categories usually requested in crowdsourcing marketplaces. More in detail, we started from the 11 categories suggested by Amazon Mechanical Turk when creating a new task (see `https://requester.mturk.com/create/projects/new`): Categorization, Data Collection, Moderation of an Image, Sentiment, Survey, Survey Link, Tagging of an Image, Transcription from A/V, Transcription from an Image, Writing, and Other. To obtain an amenable number of categories in our experiment, we excluded 5 Mechanical Turk categories: Data collection, Survey and Survey link (considered somehow similar to Sentiment), Transcription from A/V (to avoid technical issues on mobile devices), and Other. We therefore selected 6 task categories, those shown in Table 2. Then we created 4 new tasks for each category, for a total of 24 tasks, and grouped them in four task groups (labeled T_a, T_b, T_c, T_d), each group containing six tasks, one from each category.

Using artificial tasks (i.e., tasks created by ourselves) allowed to remove any platform bias and those issues discussed at the end of Section 3.5, that might have affected the results. Also, their classification was easier (sometimes it is not clear how to classify real tasks). Finally, this allowed us to create task descriptions written in Italian, thus removing any language issue from the experiment (all participants were Italian native speakers). The created tasks are in all respects similar to real tasks.

4.3 Methods

We took the usual special care to avoid any order and learning bias. Each participant performed 6 tasks (one for each of the categories in Table 2) on the desktop platform and 6 other tasks (again, one for each category) on the mobile one. His/her tasks were selected from two task groups, depending on the user group the participant was assigned to. To further avoid bias, participants in each group alternatively started from desktop or from mobile. Therefore, each participant performed a total of 12 different tasks, half on desktop and half on mobile. Each task was performed by 8 participants in two user groups, half of which performed it on mobile and half on desktop.

Statistics have been calculated as follows. At first, the average time needed for task completion has been calculated for each task separately for mobile and desktop performance (i.e., averaged on 4 subjects each). Then category averages have been calculated from task averages, again separately for mobile and desktop devices.

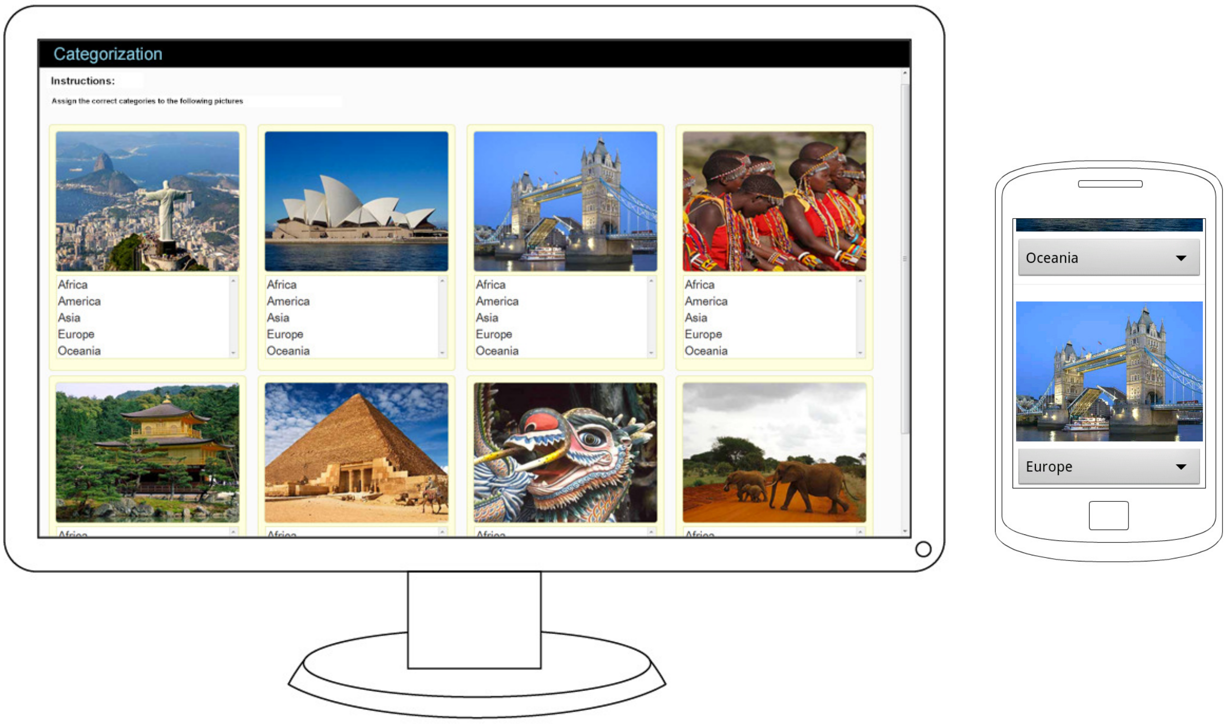
4.4 Results

Figure 5 shows the average time to complete for a task, for each category and on both mobile and desktop devices. Figure 6 shows the differences in average time to complete. Some tasks are quicker: Cat, Mod, Sen required less than one minute on average, on both desktop and mobile. ImT and Tra are a bit longer, between one and two minutes on average, and Wri is even longer. As expected, all tasks are faster on desktop, with the only exception of Wri: in it, the participants autonomously decided to use the voice-to-text functionality when on mobile, and this turned out to be quicker than writing with a keyboard (although we did not investigate the quality of transcription). As highlighted in Figure 6, ImT and Tra show a higher mobile-desktop difference, both on absolute time and percentage, probably because they require multiple texts in more fields, a cumbersome activity if carried out by mobile.

Looking at the percentage differences in Figure 6, one can notice that Cat small difference in absolute terms is actually quite high in percentage: this means that even if the difference in time is rather small, since Cat tasks are quite short (as can be seen in Figure 5), this small value is important in percentage terms. Conversely, looking at the two rightmost bars, the percentage difference in Wri looks smaller than the absolute time difference; this is again due to the average length of the Wri task, which is quite high (see Figure 5). Though, the improvement on mobile is still important, being around 20%.

5. CONCLUSIONS AND FUTURE WORK

The work described in this paper is a first exploration of the opportunities and challenges of outsourcing tasks to a mobile crowd. Results provide preliminary evidence on the inadequacy of current crowdsourcing platforms for mobile devices, even if task complexity would be adequate for being



Instructions:

Assign the correct categories to the following pictures

Figure 4: The interface used in the second experiment: desktop (left) and mobile (right)

Id	Category	Description
Cat	Content categorization	Some images are proposed to the worker, which is required to assign each of them to the correct category.
Mod	Moderation of an image	The worker is required to flag adult contente pictures that are inappropriate for children.
Sen	Sentiment	Some sentences are proposed to the worker, which is required to record his agreement by means of a Likert scale.
ImT	Image tagging	Some images are proposed to worker, which is required to tag each of them with keywords.
Tra	Transcription from an image	The worker is required to extract and write the textual content from a picture.
Wri	Writing	The worker is required to write a short text about a specific topic.

Table 2: Task categories

carried out on mobile scenarios. More in detail, results are fourfold:

- Experiment 1 results show that, according to user perception of difficulty, some crowdsourcing platforms might be slightly more adequate to mobile devices than others.
- Some inadequacy issues seem rather superficial and can be resolved by a better task or interface design.
- Experiment 2 shows that tasks of different kinds, as defined by mTurk categories, might present different difficulties when carried out on desktop or on mobile devices. This might hint a first specialization of task assignment, although examining features of easy and difficult tasks might provide a better ad-hoc specialization, perhaps even independent of the kind of task.
- Experiment 2 also confirms that mobile devices might offer some unexpected opportunities, like the voice-to-text unexpected (by us) solution, autonomously adopted by participants.

We carried out two separate experiments, although sharing subjects, in order to study two different aspects of mobile crowdsourcing: crowdsourcing platform effects, and task category effects. The experiments are preliminary and results are not final, but this is consistent with our aims, that were to begin to study the general issue of mobile crowdsourcing. This exploratory attitude is also a motivation for having two experiments performed with different methodologies (asking to the participants an estimate of difficulty and having participants performing the actual tasks). Of course, these experiments, or similar ones, could have been run by means of some crowdsourcing platform themselves. We preferred a more traditional approach and started with

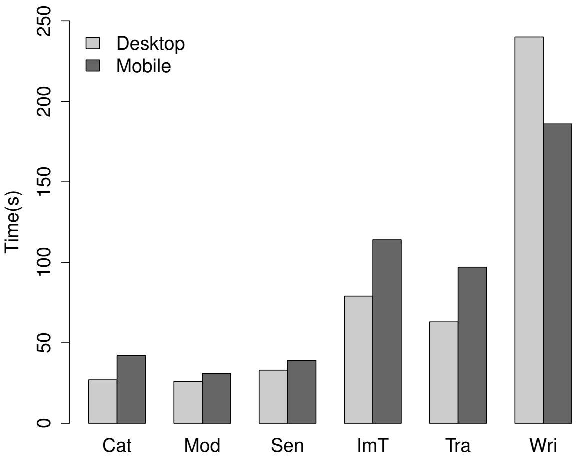


Figure 5: Average time to complete for each task category on both mobile and desktop devices

classical user studies, but we do plan to do that in the future.

To further develop this work, other experiments can be imagined. For example, the same experiments described here could be repeated in real-world scenarios (on the train, road, school rooms, or crowded places) to have more realistic results. It is also feasible to imagine an extended crowdsourcing platform that on the basis of the context of a worker (time, date, geolocation, habits and preferences, mobile device sensors, etc.), automatically filters and selects tasks tailored for a specific context.

6. REFERENCES

[1] F. Alt, A. S. Shirazi, A. Schmidt, U. Kramer, and Z. Nawaz. Location-based crowdsourcing: extending crowdsourcing to the real world. In *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries*, NordiCHI '10, pages 13–22, New York, NY, USA, 2010. ACM.

[2] N. Eagle. txteagle: Mobile crowdsourcing. In *Proceedings of the 3rd International Conference on Internationalization, Design and Global Development: Held as Part of HCI International 2009*, IDGD '09, pages 447–456, Berlin, Heidelberg, 2009. Springer-Verlag.

[3] A. Gupta, W. Thies, E. Cutrell, and R. Balakrishnan. mClerk: enabling mobile crowdsourcing in developing regions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 1843–1852, New York, NY, USA, 2012. ACM.

[4] J. Howe. *Crowdsourcing: Why the Power of the Crowd Is Driving the Future of Business*. Random House Inc, 2008.

[5] P. G. Ipeirotis. Analyzing the amazon mechanical turk marketplace. *XRDS*, 17(2):16–21, Dec. 2010.

[6] M. Meeker and L. Wu. Internet Trends D11 Conference — The annual Internet Trends Report, 2013. `http://www.slideshare.net/kleinerperkins/kpcb-internet-trends-2013`.

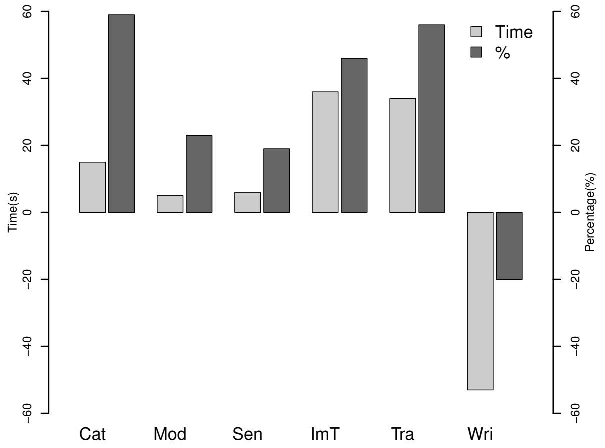


Figure 6: Mobile-desktop differences in average time to complete for each task category, as absolute time (bars on the left) and as a fraction (right)

[7] M. Musthag and D. Ganesan. Labor dynamics in a mobile micro-task market. In W. E. Mackay, S. A. Brewster, and S. Bødker, editors, *CHI*, pages 641–650. ACM, 2013.

[8] P. Narula, P. Gutheim, D. Rolnitzky, A. Kulkarni, and B. Hartmann. MobileWorks: A mobile crowdsourcing platform for workers at the bottom of the pyramid. *Proc. HCOMP11*, 2011.

[9] H. Väätäjä, T. Vainio, E. Sirkkunen, and K. Salo. Crowdsourced news reporting: supporting news content creation with mobile phones. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*, MobileHCI '11, pages 435–444, New York, NY, USA, 2011. ACM.

[10] T. Yan, V. Kumar, and D. Ganesan. Crowdsearch: exploiting crowds for accurate real-time image search on mobile phones. In *MobiSys '10: Proceedings of the 8th international conference on Mobile systems, applications and services*, pages 77–90. ACM Press, 2010.

[11] T. Yan, M. Marzilli, R. Holmes, D. Ganesan, and M. Corner. mCrowd: a platform for mobile crowdsourcing. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, pages 347–348, New York, NY, USA, 2009. ACM.

Condition-Task-Store: A Declarative Abstraction for Microtask-based Complex Crowdsourcing

Kenji Gonnokami
University of Tsukuba
s1320687@u.tsukuba.ac.jp

Atsuyuki Morishima
University of Tsukuba
mori@slis.tsukuba.ac.jp

Hiroyuki Kitagawa
University of Tsukuba
kitagawa@cs.tsukuba.ac.jp

ABSTRACT

Microtasks have been widely adopted by many crowdsourcing platforms as a unit for human computation. Recently, tools to support programmers to implement complex crowdsourcing applications with microtasks have been proposed. One approach is to provide a library of functions that can be called by programs written in imperative programming languages. Another approach is to allow SQL queries to invoke microtasks. The former approach provides large expressive power, while the latter allows declarative descriptions with limited expressive power. This paper proposes the Condition-Task-Store (CTS) abstraction, which is an alternative declarative approach to implement complex data-centric crowdsourcing with microtasks. The CTS abstraction is unique in that it has *all* the following features: (1) it naturally extends the task template adopted by many crowdsourcing platforms to define microtasks, (2) it allows declarative descriptions of crowdsourcing systems, and (3) it has large expressive power.

1. INTRODUCTION

As computer network technologies evolved, crowdsourcing became popular in many application domains. Software systems that take the crowdsourcing approach are called *crowdsourcing systems* [2].

Crowdsourcing systems are often constructed on *crowdsourcing platforms*, which provide fundamental functions for implementing crowdsourcing systems. For example, the Amazon Mechanical Turk (MTurk) [3] is a crowdsourcing platform that provides a market in which *workers* perform *microtasks* (called *HITs* in MTurk) with a small payment amount per task. Crowdsourcing platforms often provide APIs for *requesters* to register microtasks in the platforms.

Because there are frequent patterns appearing in programs for crowdsourcing systems, software tools have been developed to support the implementation of complex crowdsourcing systems. For example, Turkit [4] provides a library of functions to define and call tasks from general-purpose programming languages and introduces the crash-and-rerun programming model for minimizing the cost of re-running programs. Recently, the *declarative approach* to the development of crowdsourcing systems has emerged because declarative abstractions have an affinity toward crowdsourcing applications. Declarative descriptions do not impose unnecessary timing constraints, and we can adopt many well-known optimization techniques. For example, there are proposals that use SQL-like languages to describe data-centric crowdsourcing systems [7] [8] [13]. However, they provide limited expressive power (see Section 4).

In this paper, we offer the following two key contributions:
(1) Alternative approach to declarative crowdsourcing. We first introduce the Condition-Task-Store (CTS) abstraction, which is an alternative declarative approach for implementing complex data-centric crowdsourcing with microtasks. The CTS abstraction describes a crowdsourcing system as a set of *CTS rules*, each of which is a natural extension of task template adopted by many crowdsourcing platforms to define microtasks. Therefore, programmers who are familiar with task templates can easily write simple programs with the CTS abstraction.

The CTS abstraction is unique in that it has *all* the following features: (1) it naturally extends the task template, (2) it allows declarative descriptions of crowdsourcing systems, and (3) it has large expressive power.
(2) Novel criterion for the expressive power of languages. Next, we discuss the expressive power of the CTS abstraction. We introduce a novel criterion to measure the expressive power of programming languages for crowdsourcing; the criterion focuses on the class of interactions with humans we can implement with the language. The criterion is important for the following two reasons. First, complex crowdsourcing often requires various types of interactions with humans. For example, one of such interactions of crowdsourcing is the iterative collaboration [11], which is not necessarily supported by every existing framework. Second, the class of interactions is closely related to the class of games in game theory: because human behavior is affected by the incentives and rules defined by the game structure, the class represents the size of *mechanism design space*, i.e., the set of possible mechanisms we can implement to exploit the wisdom of the crowd. In fact, the change of game structure affects the quality of the data produced by crowdsourcing systems [5]. Our examples in Sections 3 and 4 also suggest how game structure is important in crowdsourcing. Then, we theoretically show that our CTS abstraction is not only Turing complete, but can also support a wide range of game structures.

The remainder of the paper is organized as follows. Section 2 explains related work. Section 3 introduces the CTS abstraction. Section 4 discusses its expressive power. Section 5 explains a prototype to support the software development using the CTS abstraction. Section 6 is the summary.

2. RELATED WORK

Many approaches to support the development of complex crowdsourcing systems have been proposed. They differ from one another in the abstraction they use to describe crowdsourcing systems.

(1) Imperative programming languages. TurKit [4] provides a function library for implementing crowdsourcing, which can be used via codes written in imperative programming languages. It supports the crash-and-rerun model to avoid re-performing costly operations.

(2) MapReduce-like abstraction. CrowdForge [14] is a MapReduce-like framework for describing complex tasks on MTurk. It models a crowdsourcing system as a set of tasks to implement partition, map, and reduce functions. As we discuss in Section 4.3, the expressive power of the CTS abstraction is larger than that of CrowdForge.

(3) Control/data flows. CrowdLang [11] is a language for describing crowdsourcing systems in terms of basic operators, data items, and control flow constructs. Currently, it seems that CrowdLang is used as a language for writing crowdsourcing systems at a high-abstraction level and that it does not provide the means to describe the details required to directly execute the code.

(4) Rule-based abstraction. The CTS abstraction is not the first rule-based abstraction. CyLog [9] is a Datalog-like, rule-based language for describing crowdsourcing systems. A weakness of CyLog is that it requires programmers to be familiar with programming by Horn clauses even for simple crowdsourcing. In contrast, the core component of the CTS abstraction is a task template. Therefore, although the CTS abstraction borrows concepts from logic-based languages, programmers can start with a set of simple task templates and then naturally proceed to more complex crowdsourcing.

(5) SQL-like abstraction. CrowdDB [8], Qurk [7], and Deco [13] use SQL to describe crowdsourcing systems. They propose novel query processing and optimization schemes and we believe that some of the proposed techniques can be applied to the CTS abstraction. As shown in Section 4.3, the expressive power of the CTS abstraction is larger.

To our knowledge, this paper is the first to investigate the CTS abstraction. The abstraction models crowdsourcing systems as a set of CTS rules, each of which is similar to a task template. Technically, a CTS rule can be implemented by combining two ECA rules [12]: one generates microtasks and the other stores results in the database (and can be implemented by using imperative languages). The CTS abstraction provides a higher-level, user-friendly abstraction designed for describing crowdsourcing systems, which has a well-defined semantics and proven large expressive power.

Our discussion on the expressive power is related to game theory. Recently, the literature on algorithmic game theory has addressed various aspects involving both algorithms and games, such as complexities of computing equilibrium of games [15]. To our knowledge, our paper is the first to discuss classes of games that can be implemented by abstractions for crowdsourcing.

3. THE CTS ABSTRACTION

```
<QuestionForm xmlns="http://mechanicalturk.
 amazonaws.com/AWSMechanicalTurkDataSchemas/
 2005-10-01/QuestionForm.xsd">
  <Question>
    <QuestionIdentifier>1</QuestionIdentifier>
    <QuestionContent>
       <Text>How many movies have you seen this month?</Text>
    </QuestionContent>
    <AnswerSpecification>
       <FreeTextAnswer/>
    </AnswerSpecification>
  </Question>
</QuestionForm>
```

Figure 1: Example of a task template

In this section, we first explain task templates. Then, we show examples to give an intuitive explanation of the CTS abstraction. Finally, we explain formal definitions.

3.1 Task Templates

The task template is a popular form for defining and registering microtasks into crowdsourcing platforms. Figure 1 shows an example of a task template in XML format for microtasks (HITs) of MTurk, which asks a worker to enter how many movies he or she watched in a month. The essential components of a task template are the question to be shown (`QuestionContent`) and the type of the values to be received by workers (`AnswerSpecification`). Task templates can contain *variables* (called placeholders) with which we can define many microtasks that are based on the same template but differ in the values bound to the variables.

Requesters first write task templates to define and insert microtasks into the *task pool.* Then, workers perform the tasks that exist in the task pool.

3.2 Overview of the CTS Abstraction

In the CTS abstraction, a crowdsourcing system is described by a set of *CTS rules.* We assume that there exists a relational database. CTS rules read and write data to and from the database.

A CTS rule is the fundamental building block of the CTS abstraction. We first give a simple example and next show another example that requires more than one CTS rule.

Example 1. A Simple Crowdsourcing System

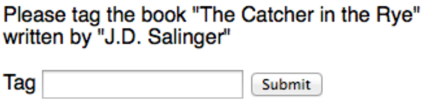
We use the task shown in Figure 2 to ask workers to tag books. The details are as follows:

- The database has the `Book(bid, title, author)` relation to store book information.
- For each book stored in the `Book` relation, tags are given by three workers.
- The result is stored in the `Tag(bid, tag)` relation.
- Workers are paid 10 (cents or any currency) per task, if any other workers entered the same tag.

The crowdsourcing system can be described only by the CTS rule in Figure 3. A CTS rule consists of three parts: condition, task, and store. We explain each part below.

The condition part: We write the condition to generate and insert a task into the task pool. The condition specifies what tuples need to exist in the database for generating a task. For example, the condition in Figure 3 states that a task is generated for each tuple existing in the `Book` relation.

The task part: We write a task template that contains a question to be posed to workers. The question can contain



Please tag the book "The Catcher in the Rye" written by "J.D. Salinger"

Figure 2: Example of a microtask

Condition	`Book(bid, title, author)`	
Task	Question	`Please tag the book "$title" written by "$author"`
	Generator	`Entry(desc:"Tag", var:tag, type:text)`
	Count	3
	Payoff	`PayIf(count(Tag(bid, tag))>=2, 10)`
Store	`Tag(bid, tag)`	

Figure 3: Example of a CTS rule

variables (such as `$author` and `$title`) bound to values in the condition (i.e., the variables are replaced with values each time the task is generated). The task part also specifies additional information, including the variables and its associated types, to store entered values.

Because there are frequently occurring patterns that appear in task template specifications, we provide *template generators* that allow users to describe task templates without specifying implementation details such as HTML tags. The task part in Figure 3 states that we use the `Entry` template generator with the following parameters: (1) the input field is labeled as "Tag," (2) the entered value is to be stored in the `tag` variable, and (3) the type of `tag` is `text`. Count is the number of tasks to be generated for the same value. Payoff describes how much is paid to workers per task. For example, `PayIf(count(Tag(bid, tag)) >= 2, 10)` states that 10 cents will be paid if there are other workers that entered the same tag to the same book.

The store part: We specify the relations and attributes in which the entered values will be stored. The store part in Figure 3 states that we use the `Tag` relation to store `bid` and the entered `tag`. □

As the example above suggests, a CTS rule is a natural extension of the widely-used task template. Because we can omit the condition and task parts, a CTS rule can be used to describe the following four types of processing.

1. Generate a task when the condition is satisfied, and store the result into the database.
2. If the condition part is omitted, generate a task with no condition, and store the result into the database.
3. If the task part is omitted, compute and store values into the database when the condition is satisfied.
4. If both of the condition and task parts are omitted, store values into the database with no condition.

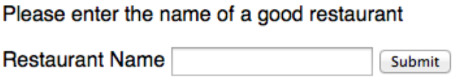
Example 2. More Complex Crowdsourcing.

We consider a crowdsourcing system to rate restaurants, in which workers perform the following two types of tasks:

Task 1: Enter names of restaurants (Figure 4). A 10 cent payment is paid if the average rating by others is higher than 3.

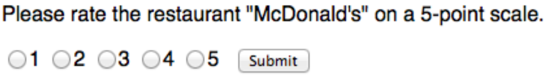
Task 2: Enter an evaluation rating (1 to 5) for the given restaurant (Figure 5). Three workers perform this task for each restaurant, and they receive 10 cents per task.

We assume that the results of Task 1 are stored in `Restaurant(rname)`, and that those of Task 2 are stored in `Rating(rname, value)`. Then, Figure 6 shows CTS rules for Tasks 1 and 2.



Please enter the name of a good restaurant

Figure 4: Example of a microtask: Task 1



Please rate the restaurant "McDonald's" on a 5-point scale.

Figure 5: Example of a microtask: Task 2

Task1:

Condition		
Task	Question	`Please enter the name of a good restaurant`
	Generator	`Entry(desc:"Restaurant Name", var: rname, type: text)`
	Count	
	Payoff	`PayIf(avg(Rating(rname, value))>3, 10)`
Store	`Restaurant(rname)`	

Task2:

Condition	`Restaurant(rname)`	
Task	Question	`Please rate the restaurant ‘‘$rname’’ on a 5-point scale.`
	Generator	`Choice(var: value, type: int, items: [1, 2, 3, 4, 5])`
	Count	3
	Payoff	`Pay(10)`
Store	`Rating(rname, value)`	

Figure 6: CTS rules for Example 2.

The CTS rule for Task 1 has no condition: thus, the task is unconditionally generated. Unless the count number is specified, the number of generated tasks is determined as follows: (1) if the key of the predicate (`rname` of `Restaurant(rname)`) is bound to values by the condition, the task is generated only once for each case in which the condition is satisfied; (2) otherwise the same task is repeatedly generated everytime the task is performed and removed from the task pool.

The condition of the CTS rule for Task 2 states that it generates a task for each tuple stored in `Restaurant` relation. Thus, the task is generated for each restaurant entered in Task 1. Workers receive 10 cents for performing a task. □

Discussion. As the examples above suggest, the description is declarative, and each rule is invoked when its condition is satisfied. Compared to the code written in imperative programming languages, CTS rules naturally describe the parallel and asynchronous processing of computation involving human workers. On the other hand, the CTS abstraction is more expressive than the declarative query languages that do not support the transitive closure, because it essentially supports a loop with a dynamic condition check [1].

An important point is that the incentive structure plays a critical role to appropriately exploit the wisdom of crowd and (at least theoretically) ensure data quality. Because the incentive structure and rules involving multiple humans can be modeled as *games*, we can use game theory to discuss their behaviors. For example, with a simple game-theoretic analysis, the incentive structure of Example 1 guarantees that rational workers enter tags that others can easily come up with. Similarly, the incentive structure of Example 2 guarantees that rational workers for Task 1 enter the names of restaurants that are likely to receive high ratings.

3.3 Formal Definition

This section first defines the CTS rules and then explains the syntactic sugar for the concise description of rules.

3.3.1 Definition of CTS Rules

Definition 1. A program of the CTS abstraction is a set of CTS rules, each of which is modeled as a triple $R_i = (C_i, T_i, S_i)$, running over a relational database schema. Here, T_i is a task template, C_i is the condition to generate a task using T_i, and S_i is the description of how to store the result in the database. The database contains a predefined relation with the schema `Worker(pid, payoff)` to store information on workers and the accumulated values of the payoffs they received so far.

- C_i is a sequence $P_1(x_{11}, \ldots, x_{1n_1}), \ldots, P_m(x_{m1}, \ldots, x_{mn_m})$ of zero or more atoms. Some of the atoms can be arithmetic atoms, such as $x_{11} = 3$.
- T_i is either a triple $(q, \bar{x}, \bar{y})$ or *null*. Here, q is a question for workers, $\bar{x}$ is a sequence of variables bound by C_i, and $\bar{y}$ is a sequence of variables to store the results of performing the task.
- S_i is a sequence $Q_1(y_{11}, \ldots, y_{1n_1}), \ldots, Q_m(y_{m1}, \ldots, y_{mn_m})$ of one or more atoms, where each y_{jk} is any of a variable bound by C_i, a variable that appears in $\bar{y}$, or a constant. Each atom can be followed by either `/update` or `/delete`. □

3.3.2 Syntactic Sugar

We introduce the following variety of syntactic sugars to make the rule description concise.

(1) Attributes of atoms. The notation of atoms, which appear in the condition and store part of CTS rules, are essentially the same as those of Prolog and Datalog. A key difference is that they explicitly specify *attribute names* for their parameters. Each parameter is specified in the form *attribute:variable* or *attribute:constant.* For example, `Restaurant(name:x, zip:305)` is an atom.

There are cases in which parameters and attributes can be omitted in each atom, as described below.

- We can omit a parameter if the rule does not consume the value of the bounded variable. For example, `Restaurant(name:x)` is an atom that omits `zip`.
- We can omit an attribute name if the attribute name is the same as the variable name. For example, `Restaurant(name:name, zip:y)` can be represented as `Restaurant(name, zip:y)` because the attribute and the variable have the same name.

(2) Task part. Because we have frequently occuring patterns in the description, we introduce task-template generators and the following three fields for the task part.

- *Generator* is the name of a task-template generator, associated with its parameters. For example, `Entry` and `Choice` in Figure 6 are task-template generators. They generate actual task templates based on the given parameters and the sentence written in the Question field. These allow users to define task templates without specifying implementation details (such as HTML tags). The Crapid system (Section 5) implements various task-template generators.
- *Count* specifies the number of generated tasks. Let N be the number specified in the field. For every case in which the condition holds, the same task is generated N times and N tuples are inserted into the relation. This is implemented by adding the `count` attribute to the schema of the relation in the store part, and copying the CTS rule N times in the program.
- *Payoff* specifies a function to compute payoff values given to workers. If a function is specified, rules to update `Worker(pid, payoff)` are automatically generated and added to the set of rules. The Crapid system provides pre-defined payoff functions.

3.4 Evaluation Model and Semantics

Given a description d of the CTS abstraction, the evaluation of d on instance *ins* of the database is performed by evaluating each CTS rule in a bottom-up way on *ins*. More precisely,

- For each CTS rule $(C_i, T_i, S_i) \in d$, check if C_i is satisfied with *ins* in the following way: for each atom a_k in C_i (from left to right), check if there exists any tuple to bind variables in a_k to the values that are consistent with the values bound to the variables appeared in $a_0 \ldots a_{k-1}$.
- For every combination V of values that satisfies all the atoms in C_i, perform one of the following:
  - If $T_i \neq \mathit{null}$, replace variables in T_i with values in V and insert the task into the task pool.
  - If $T_i = \mathit{null}$, create new tuples using values in V for the atoms that appear in S_i. Then, perform one of the following: insert the tuples into *ins*, update *ins* with the tuples (when the atom is followed by `/update`), or delete the tuples from *ins* (when followed by `/delete`).
- If a worker completes the generated task, (1) create new tuples for the atoms that appear in S_i, using values in V and the entered values for the task, then (2) apply the insertion, delete, or update operation to *ins* with the new tuples.
- If *ins* is updated, check if there exist rules for which C_i is satisfied with the new *ins*. If such rules exist, process them. Terminate the process if we cannot find such a rule. If there are multiple rules that can be executed at the same time, the rule that appears earlier in the code is evaluated first.

For example, assume that we have the CTS rule shown in Figure 7. We first check if there exists a tuple that matches `Image(i, size, type:"photo")` in the database. Assume that the `Image` relation has tuple $t = (\mathtt{img98}, \mathtt{100}, \mathtt{photo})$. Then, `i` and `size` are bound to `img98` and `100`, respectively. Next, check if there exists a tuple that matches `Large(size)` with `size=100`. If exists, we replace `$i` in the task template with `img98` and insert the task into the task pool to ask workers to choose the category for the image `img98`. If the task is performed, the result of performing the task is stored into `LargePhoto`, and the payoff attribute of the `Worker` relation is incremented by 1.

Given a set d of CTS rules, the semantics of d is defined as a set of *rational consequences* of d, in a similar way as the semantics of CyLog codes [9]. A rational consequence of d is a set of facts that are derived from the rules and the equilibrium [16] of the games, i.e., the states reached by rational workers.

Condition	`Image(i, size, type:"photo"), Large(size)`	
Task	Question	`Please choose a category of $i.`
	Generator	`Choice(var: category, type:string, items:[landscape, portrait, animal, food, others])`
	Count	`1`
	Payoff	`pay(1)`
Store	`LargePhoto(i, category)`	

Figure 7: Two atoms in the condition part

Rule 1

Condition	
Task	
Store	`TuringMachine(id:1, st:s, head:0)`

Rule 2

Condition	`TuringMachine(id, st, head),`
	`Tape(pos:head, sym),`
	`Rule(st, sym, new_st, new_sym, dir),`
	`new_pos = pos + dir`

Task	

Store	`TuringMachine(id, st:new_st, head:new_pos)/update,`
	`Tape(pos, sym:new_sym)/update`

Rule 3

Condition	`TuringMachine(id, head)`
Task	
Store	`Tape(pos:head)/update`

Figure 8: CTS rules implementing a Turing machine

4. EXPRESSIVE POWER

This section discusses the expressive power of the CTS abstraction. First, we show that the CTS abstraction is Turing complete. Then, we introduce a criterion to measure the expressive power of programming languages, which focuses on the class of games the language can implement. Finally, we compare the expressive power of the CTS abstraction with those of other frameworks.

4.1 Turing Completeness

Theorem 1. The CTS abstraction is Turing complete.

Proof Outline. Figure 8 is a set of CTS rules that implements any Turing machine. Formally, a Turing machine consists of a quintuple $(K, \Sigma, \delta, s, H)$ where K is a finite set of states, Σ is an alphabet, $s \in K$ is the initial state, $H \in K$ is the set of halting states, and δ is the transition function [6]. Intuitively, we need the following three components to implement a turing machine.

Element 1. Memory of the machine's inner state

Element 2. Head reading and writing information stored in the tape

Element 3. The long tape in infinitum.

In Figure 8, `TuringMachine(id, st, head)` implements Elements 1 and 2. Here, `st` records the current state whose domain is K, and `head` stores the position of the head. `Tape(pos, sym)` implements Element 3, in which each tuple (p, s) states that symbol s (whose domain is Σ) is written at position p of the tape. `Rule(st, sym, new_st, new_sym, dir)` stores the rules δ to read and write symbols on the tape and move the head.

Rule 1 initializes a Turing machine (the initial state is s). Rule 2 states how the head moves and writes symbols onto the tape according to the rules stored in `Rule(st, sym, new_st, new_sym, dir)`. It states that if the inner state is `st` and the symbol at the current position of the head is `sym`, write `new_sym` at the position, update the inner state by `new_st`, and move the head to `pos+dir`. Rule 3 extends the tape when the head reaches a position that the head never visited before. We need Rule 3 because Rule 2 always requires that `Tape(pos, sym)` exists. We also need a rule to stop the machine if it reaches the halting states $H \subseteq K$. The rule is obvious and omitted due to the space limitation.

Defining the CTS rules to implement a Turing machine proves that the CTS abstraction is Turing complete. □

4.2 Expressive Power in Terms of Games

This section proposes to use the game concept as a measure of the expressive power of programming languages for crowdsourcing, because the class of games that the language can implement affects the way in which the implemented system can exploit the power of the wisdom of crowd. First, we enumerate several classes of games and show the relationship among the classes.

Definition 2. $\mathcal{G}_1$ is a class of games that satisfy the following conditions:

1. Every input from a human is not affected by the inputs from others, and
2. The payoffs are computed by a primitive recursive function of the input values. □

An example of a game in $\mathcal{G}_1$ is one that asks humans to enter tags for a given image without telling them what tags are entered by other humans. Payoffs are defined for each combination of worker inputs. For example, workers receive payoffs when they enter the same tag. Games in $\mathcal{G}_1$ are called *simultaneous games* in game theory.

Definition 3. $\mathcal{G}_N$ is a class of programs in which (1) $N (> 0)$ is known in advance; (2) each game has at most N-phases of interactions, each of which asks a worker to enter data; (3) at each i-th phase workers are shown some information based on what was entered in the first to the $i-1$-th phases; and (4) payoffs are computed by a primitive recursive function of the entered values. □

Each game in $\mathcal{G}_N$ has at most N phases, each of which asks a human to enter data considering some information computed from data in the previous phases. For example, assume that we want to divide a set of cakes into two groups whose total prices are equivallent to each other. Then, a program that (1) asks one worker to divide the cakes into two groups at the first phase, then (2) asks another worker to choose one group, and finally (3) gives to each worker the total price of cakes in his group, belongs to $\mathcal{G}_N$ (with N=2). Note that the game guarantees that the prices of the two groups become the same if workers are rational; it exploits the power of human intelligence to compute how they can make two groups whose prices are equivalent to each other. From the definition, every game in $\mathcal{G}_1$ belongs to $\mathcal{G}_N$.

Definition 4. $\mathcal{G}_*$ is a class of programs in which each program (1) executes a sequence of interactions with workers, with the sequence being generated by a primitive recursive function, (2) shows workers at each interaction the information computed by a function whose parameters are taken from the results of past interactions, and (3) payoffs are computed by a primitive recursive function of the entered values. □

An example of a game in $\mathcal{G}_*$ is to ask workers to write a paragraph that explains a given keyword. Workers update

Table 1: Expressive power

Abstraction /Framework	Turing complete	Class of games
MTurk alone	N	$\subseteq \mathcal{G}_1$
CrowdForge	N	$\subseteq \mathcal{G}_N$
CrowdDB/Deco/Qurk	N	$\subseteq \mathcal{G}_N$
CTS Abstraction	Y	$\mathcal{G}_*$
Imperative programming languages with the MTurk API	Y	$\mathcal{G}_*$

the paragraph until the paragraph is satisfied by the majority of the crowd. When the paragraph is completed, every writer (worker) who contributed to the paragraph receives a payoff, which is computed by dividing a fixed total payment by the number of the contributors.

Theorem 2. $\mathcal{G}_N$ is a proper subset of $\mathcal{G}_*$.

Proof. For every g, $g \in \mathcal{G}_N \Rightarrow g \in \mathcal{G}_*$ and for any given i, there exists $g \in \mathcal{G}_*$ s.t. g has $i+1$ input phases and therefore $g \notin \mathcal{G}_N$. □

Theorem 3. Assume that we allow Turing machines to interact with humans at any step of its execution. Let M be the set of all such machines. M can implement any $g \in \mathcal{G}_*$.

Proof. The sequence of interactions with workers that can be generated by a primitive recursive function can be implemented by some $m \in M$. The information shown and the payoffs can be computed if m is a Turing machine. □

Note that being Turing complete is not a sufficient condition to be able to implement $\mathcal{G}_*$, because the power of interactions with humans does not matter for a language to be Turing complete. $\mathcal{G}_*$ contains indefinite-length sequential games (in game theory) that can be expressed with Turing machines that are able to interact with humans at any step of its execution.

From the definition of the CTS rules, the following holds.

Theorem 4. The CTS abstraction can implement any games in $\mathcal{G}_*$. □

4.3 Comparison of Expressive Powers

Table 1 compares the expressive powers of different abstractions and frameworks. Games implemented by manually registering HITs to MTurk are contained in $\mathcal{G}_1$, whereas code written in programming languages that uses MTurk APIs can implement games in $\mathcal{G}_*$. CrowdForge can implement a part of the games in $\mathcal{G}_N$, while its expressive power is not larger than $\mathcal{G}_N$, because it is not Turing complete. In particular, games implemented by a combination of partition, map, and reduce are contained in $\mathcal{G}_N$ with $N = 3$. Similarly, the class of games that CrowdDB, Qurk, and Deco can implement is not larger than $\mathcal{G}_N$.

5. PROTOTYPE SYSTEM

We implemented the Crapid system, a prototype system to develop crowdsourcing systems using the CTS abstraction. Crapid takes as input a set of CTS rules and outputs executable code. Crapid supports various task-template generators (i.e., Entry, Choice, Decision, and Comparisons as of May 2013) and payoff functions to help users easily define microtasks in CTS rules. Crapid provides a form-based user interface. When a user chooses a task-template generator, Crapid provides an appropriate set of selection boxes and drop-down menus to specify CTS rules. The output code is executable on Crowd4U [10], a crowdsourcing platform deployed at universities.

6. SUMMARY

This paper introduced the CTS abstraction, a declarative approach for implementing complex crowdsourcing with microtasks. The paper also introduced a novel criterion to measure the expressive power of programming languages for crowdsourcing by focusing on the class of games we can implement with the language. The class represents the size of mechanism design space, i.e., the set of possible mechanisms we can implement to exploit the wisdom of the crowd. The CTS abstraction is unique in that it has all the following features: (1) it naturally extends the task template adopted by many crowdsourcing platforms, (2) it allows declarative description, and (3) it has large expressive power.

Future work includes the development of various rewriting techniques for the CTS abstraction. For example, we plan to adapt various optimization techniques for crowdsourcing systems [7] [8] [13] into our context.

Acknowledgements. The authors are grateful to Prof. Shigeo Matsubara of Kyoto university for his helpful comments, and to the contributors of Crowd4U, whose names are partially listed at http://crowd4u.org. This research was partially supported by PRESTO from the Japan Science and Technology Agency, and by the Grant-in-Aid for Scientific Research (#25240012) from MEXT, Japan.

7. REFERENCES

[1] S. Abiteboul, R. Hull, V. Vianu: Foundations of Databases. Addison-Wesley 1995.

[2] A. Doan, R. Ramakrishnan, A. Y. Halevy. “Crowdsourcing systems on the World-Wide Web. Commun.ACM. 2011, vol. 54, no. 4, p. 86-96.

[3] Amazon Mechanical Turk, https://www.mturk.com/.

[4] G. Little, L. B. Chilton, M. Goldman, R. C. Miller. “Turkit: human computation algorithms on mechanical turk”. Proc. UIST (2010), ACM, New York, 57-66.

[5] S. Jain, D. C. Parkes. “A game-theoretic analysis of the ESP game”. ACM Trans. Economics and Comput. 1(1): 3 (2013)

[6] H. R. Lewis, C. H. Papadimitriou. “Elements of the theory of computation.” Prentice-Hall, Englewood Cliffs, New Jersey, 1981

[7] A. Marcus, E. Wu, D. Karger, S. Madden, and R. Miller. “Human-powered sorts and joins”. In VLDB, 2012.

[8] M. J. Franklin., D. Kossmann, T. Kraska, S. Ramesh, R. Xin. “CrowdDB: answering queries with crowdsourcing”. SIGMOD Conference. 2011, p. 61-72.

[9] A. Morishima, N. Shinagawa, S. Mochizuki. “The Power of Integrated Abstraction for Data-centric Human/Machine Computations”. VLDS2011, pp. 5-9.

[10] A. Morishima, N. Shinagawa, T. Mitsuishi, H. Aoki, S. Fukusumi. “CyLog/Crowd4U: A Declarative Platform for Complex Data-centric Crowdsourcing”. PVLDB 5(12): 1918-1921 (2012).

[11] P. Minder, A. Bernstein. “CrowdLang: A Programming Language for the Systematic Exploration of Human Computation Systems”. SocInfo 2012: 124-137

[12] N. W. Paton, O. Di’az. “Active Database Systems”. ACM Comput. Surv. 31(1): 63-103 (1999)

[13] H. Park, R. Pang, A. G. Parameswaran, H. Garcia-Molina, N. Polyzotis, J. Widom. “An overview of the deco system: data model and query language; query processing and optimization”. SIGMOD Record 41(4): 22-27 (2012)

[14] A. Kittur, B. Smus, S. Khamkar, R. E. Kraut. “CrowdForge: crowdsourcing complex work”. UIST 2011: 43-52

[15] T. Roughgarden. “Algorithmic game theory”. Commun. ACM 53(7): 78-86 (2010)

[16] F. Vega-Redondo. Economics and Theory of Games, Cambridge University Press, 2003.

The Palm-tree Index: Indexing with the crowd*

Ahmed R. Mahmood
Purdue University
amahmoo@purdue.edu

Walid G. Aref
Purdue University
aref@cs.purdue.edu

Eduard Dragut
Purdue University
edragut@purdue.edu

Saleh Basalamah
Umm Al-Qura University
smbasalamah@uqu.edu.sa

ABSTRACT

Crowdsourcing services allow employing human intelligence in tasks that are difficult to accomplish with computers such as image tagging and data collection. At a relatively low monetary cost and through web interfaces such as Amazon's Mechanical Turk (AMT), humans can act as a computational operator in large systems. Recent work has been conducted to build database management systems that can harness the crowd power in database operators, such as sort, join, count, etc. The fundamental problem of indexing within crowdsourced databases has not been studied. In this paper, we study the problem of tree-based indexing within crowd-enabled databases. We investigate the effect of various tree and crowdsourcing parameters on the quality of index operations. We propose new algorithms for index search, insert, and update.

1. INTRODUCTION

Crowdsourcing is the practice of solving large problems by dividing them into smaller tasks, each of which is then solved by humans from an online community. The tasks are usually difficult to be performed automatically by a computer as they require human intelligence. Typical crowdsourcing tasks involve labelling, ranking, data cleaning, data filtering, data collection, and entity matching (e.g., see [11, 6, 2, 15, 5, 14]). Websites, e.g., Amazon's Mechanical Turk (MTurk) [1] provide an infrastructure for organizations to submit numerous micro-tasks and collect their results after they are fulfilled by human workers recruited at those websites. In a typical crowdsourced application, tasks are replicated and are answered by multiple people to avoid single user errors. Each person answering a task incurs a (monetary) cost. Thus, a challenging problem for crowdsourcing is that of optimizing the number of tasks while maintaining the accuracy of the results.

Consider the following scenario where a crowdsourcing tree-based index can be useful. Assume that a car repair shop wants to provide an online service that allows users to submit pictures of their damaged cars to get an estimate of repair cost of their car. The repair shop maintains images of cars previously repaired associated their actual repair cost. A good repair estimate would be the actual repair cost of previously repaired car of similar condition. Building a tree index on images of previously repaired cars sorted based on their repair cost allows performing crowdsourcing-based similarity queries on the index. These queries help users to find cars with similar conditions and hence anticipate the same repair cost to their own damaged car. The key to the success of this scenario is that the cost of repair is directly proportional to the condition of the car. The same idea can be applied when estimating the selling price of a used car, the cleaning cost of rental rooms, the placement of a new soccer player in player rankings or the ranking of new scientific publications, etc.

In this paper, we introduce the *palm-tree index*; a crowdsourcing tree-based index. We call our proposed crowdsourcing tree-based index *palm-tree index* as real palm trees also need *humans* in order to move the pollen from one male tree to the other female trees to produce fruits (dates). The palm-tree index aims at indexing keys based on properties that need human intelligence, e.g., to compare images against each other in order to descend and navigate the index. Crowdsourcing-based tree indexing is useful for (1) the ordering of a set of keys based on a subjective property, (2) performing index nested-loops joins, and (3) answering range queries, among other typical uses of an index. The problem of crowdsourcing-based indexing is challenging because of the following issues. First, crowd-based comparisons are subjective and are prone to error. Hence, we need to find strategies that give the most accurate and consistent results. Second, comparison tasks performed by the crowd incur a (monetary) cost. Hence, optimizing the number of tasks while preserving accuracy is very important.

The contributions of this paper are summarized as follows:

- We introduce the problem of crowdsourcing tree-based indexing as a technique for ordering a set of items with "subjective" keys.
- We present a taxonomy for crowdsourcing tree-based indexes.
- We introduce the palm-tree index, a crowdsourced index, along with several accompanying index traversal algorithms.
- We study the quality of the results retrieved using the palm-tree index.

The rest of this paper is organized as follows. Section 2 introduces the taxonomy for crowdsourcing tree-based indexes. Section 3 presents notations used throughout the paper. Section 4 presents the palm-tree index and its search algorithms. Section 5 analyzes the proposed algorithms. Preliminary experimental results

---

*This work was partially supported by the National Science Foundation under Grants III-1117766 and IIS-0964639.

are given in Section 6. Related work is presented in Section 7. Section 8 contains concluding remarks.

2. PROBLEM TAXONOMY

In this section, we introduce a taxonomy for crowdsourcing tree-based indexes. This taxonomy can be seen as extension to the one presented in [8], which addresses only worker motivation, task replication and task assignment. Our taxonomy adds concepts related to tree indexing (e.g., tree height) and to crowdsourcing in general (e.g., worker experience). Figure 1 depicts the taxonomy.

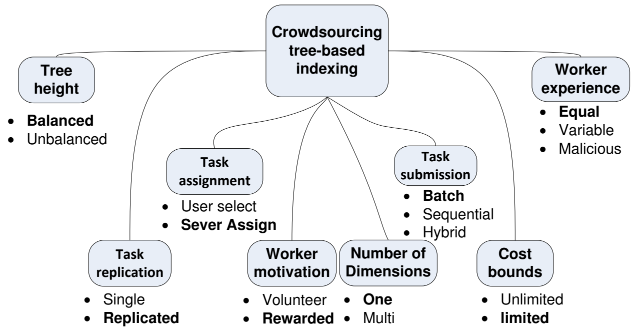


Figure 1: A taxonomy of crowdsourcing tree-based indexing. The concepts utilized in this paper are shown in boldface.

Tree height It specifies whether the tree is balanced or not. The proposed palm-tree index is a balanced B^+-tree-like structure to obtain predictive query cost. More detail about query cost is given in Section 3.

Number of dimensions An index can be either one-dimensional or multi-dimensional. In this paper we consider one-dimensional indexing.

Worker motivation In some situations, workers may be willing to perform tasks for free, e.g., see [8], However, in most instances workers seek rewards for their services. This raises several issues, e.g., budget estimation and the presence of low quality work due to workers solving tasks quickly to obtain more money. In this paper we consider rewarded workers.

Worker experience A simple model to describe workers is to assume that all workers have equal experience, e.g., as in [11]. An alternative and viable model is to differentiate workers based on their level of expertise, e.g., as in [3]. In the palm-tree index, workers are assumed to have indistinguishable, i.e., equal, expertise. We plan to extend our analysis to the general case when workers have variable experience. One special type of workers is a malicious worker. A malicious worker aims at degrading output quality by providing intentionally faulty and misleading answers. Coping with malicious workers is not addressed in this paper.

Task assginment There are two cases to consider: (1) The worker chooses the tasks to work on, or (2) the tasks are automatically assigned to the worker without the workers interference in the selection. This dimension is particularly important when workers are of variable expertise. For example, it is important to avoid assigning difficult tasks to inexperienced workers. In this paper, we assume Case 2, above.

Task replication In a *single* task no-replication model, workers are trusted to deliver high quality answers. In many other situations, workers' responses incur error. The *Replicated* task model allows aggregating multiple workers' opinions to increase the quality of the results. In this paper, we assume the replicated model.

Task submission model A task submission model describes how replications of the same task are submitted to workers. One alternative is to submit all task replicas in a single *batch*. This technique can reduce the response time to get the final results as task replicas are solved by multiple workers concurrently. Nonetheless, one may end up submitting more tasks than needed. In order to reduce the cost needed for simple tasks, the replicas of same task can be submitted *sequentially*. In the sequential model, a replica of a task is submitted only when the outcome of the previous replicas does not produce results that are of acceptable quality. The sequential technique increases the task response time as replicas wait for each other. A *hybrid* submission model is a middle-ground between both alternatives. In the hybrid model, small-sized batches of the same task are submitted sequentially. In this paper, we only consider the batch task submission model, i.e., that replica tasks are submitted in parallel to the designated workers at the same time.

Cost bounds Having *unlimited* cost per worker is a rare occurrence. It may occur when the quality of the result is of major importance. Typically, cost for tasks are *limited* and optimizing the overall task error within cost bounds is of major importance. In this paper, we consider the limited cost case.

3. PRELIMINARIES AND INDEX MODEL

3.1 Problem definition

Let S be a set of N items and q be a query item. The keys of these items are subjective or imprecise (e.g., the item is the photo of a damaged car and the key is the cost of repair for this car). We need to address the following issues:

- Order the items in S according to their keys.
- Construct (build) an index on S.
- Submit query q on S to the crowd.
- Aggregate crowds responses to query q to produce a final result .

3.2 Index model

The structure of the palm-tree index is composed of two components: the index manager and a B^+-tree. Figure 2 gives the main components of the palm-tree index. We distinguish between two main concepts in the palm-tree; *jobs* and *tasks*. A *job* is the unit of work submitted by a palm-tree user to perform either a query or an insertion on the indexed data. A *task* or *HIT* (*H*uman *I*ntelligent *T*ask) is the smallest unit of work assigned to a worker. Typically, a job involves generating multiple tasks.

3.2.1 The B^+-tree Component

The palm-tree index is built on top of a B^+-tree index. Each node in the B^+-tree has n ordered keys and represents one task. [11] observes that human beings are capable of performing n-ary operations with ease. Therefore, we give to a worker a sorted list A of n items, e.g., images or videos, along with the query item q,

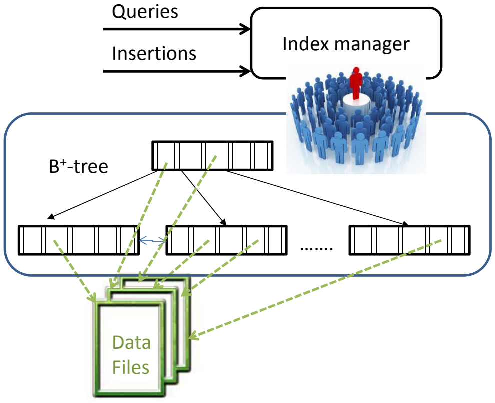


Figure 2: palm-tree index model.

e.g., a query image or video, and ask the worker to place q in A based on how q compares to the items in A. For instance, A can be a list of 5 pictures of cars sorted according to the extent to which the car is damaged, while q can be the picture of a new damaged car. The fanout f of the B$^+$-tree (the maximum allowed children of a node other than the root) naturally captures the maximum n-ary operations that a human can accomplish with ease. Clearly, a human being can handle a lot easier a list A with 5 pictures than one with 20. Determining the optimal f for a given problem requires an initial phase of training. We discuss this issue in more detail in Section 5.

Construction (building) of the palm-tree depends on the type of indexed keys. We have two types of keys: *quantitative* and *qualitative* keys.

DEFINITION 3.1. *A **quantitative key** is a key to be indexed that has two proportional properties, namely, a subjective property and an assessed value. For quantitative keys, the palm-tree index can be constructed by sorting keys based on the assessed values then bulkload the index.*

DEFINITION 3.2. *A **qualitative key** is a key to be indexed based on a subjective property only. It does not have any associated assessed value. For qualitative keys, a palm-tree index can be constructed only using successive insertions and human-based comparisons.*

One example of a quantitative key is images of damaged cars associated with actual repair costs. The degree of car damage is the subjective property of the key while the repair cost is the quantitative key. An example of a qualitative key is images of butterflies with an ordering based on beauty as a subjective property. Regular B$^+$-tree splitting/merging algorithms are used during insertion/deletion into/from the B$^+$-tree. For both types of keys, queries use human intelligence to make comparisons needed to search the B$^+$-tree. All comparisons are based on the subjective property only.

3.2.2 Index manager

The palm-tree index manager is responsible for tree construction and query processing within the palm-tree. It initiates jobs for users' requests, generates tasks for every job, assigns tasks to workers, and aggregates workers' responses. We use majority voting to aggregate responses of workers.

In the palm-tree index, a task consists of items (e.g., pictures) in a node in the B$^+$-tree. A worker is asked to choose the best location for a query item among these items. Figure 3 gives an example task for estimating the repair cost of a car. The image on top represents the query image, while the images below are the items in a node in the tree. The worker assess where the query key fits among the current nodes keys.

Table 1: Palm-tree index operations

Operation	Quantitative keys	Qualitative keys
Query	Crowd-search	Crowd-search
Insert	B$^+$-tree insert	Crowd-search then B$^+$-tree insert
Build	Bulk loading or Successive inserts	Successive inserts
Delete	Query then B$^+$-tree delete	Query then B$^+$-tree delete
Update	Delete then insert	Delete then insert

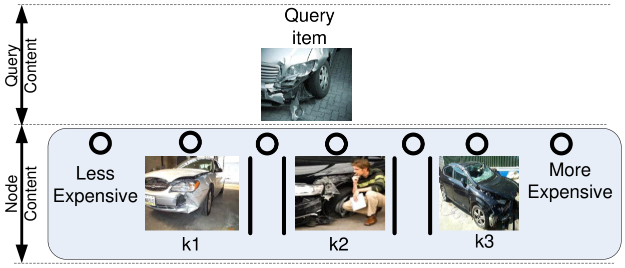


Figure 3: Sample task for choosing the best branch to estimate the car repair cost.

4. ALGORITHMS

Table 1 describes how the build, insert, update, delete, and query operations are performed in the palm-tree index. Regular B$^+$-tree algorithms (i.e, insert-split) are performed directly and do not involve the crowd. We focus on the crowd-search operation (i.e., querying the palm-tree index using the crowd) because the other operations depend on it.

The original search algorithm for the B$^+$-tree index is not directly applicable for the crowdsourced palm-tree index. The palm-tree search algorithm is dependent on the way the palm-tree is traversed and the way the workers answers are aggregated. We identify three search strategies for the palm-tree search algorithm. All three search strategies share the following two steps. The first step is *root_task_generation* that generates tasks at the root level and is common for all three strategies. The second step is the *response_handler* that generates tasks at the lower levels of the tree and varies according to the three strategies. Algorithm 1 describes the *root_task_generation* step.

```
Algorithm 1: generate_root_tasks(Tree tree,Key q)
k← get_replications(tree,h)
solved← 0 // zero solved so far
for $i = 1$ to $k$ do
    w ← choose_worker()
    h← tree.height // tasks at the root level
    t← create_task(tree,h,q,w)
    submit_task(t,response_handler)
end
```

4.1 Leaf-Only Aggregation Strategy

In this technique, a job is replicated to K workers. A worker performs the index search by descending the tree from the root to the leaf based solely on this workers own decisions or assessments to find the best match to the query item. All workers responses and final results are aggregated only at the leaf level. Algorithm 2 gives an outline of the leaf-only aggregation strategy.

Algorithm 2: $handle\_leaf\_only\_aggregation$(Task t)

```
if $t.level \neq 0$ then // task at non-leaf level
    tree ← $get\_subtree$(t.response,t.tree)
    t← $create\_task$(tree,t.level-1,t.q,t.w)
    $submit\_task$(t,$handle\_leaf\_only\_aggregation$)
else
    solved++
    if $solved == k$ then // all workers finished
        result = ← $aggregate\_all\_results$
        return result
    end
end
```

4.2 All-levels Aggregation Strategy

An alternative search strategy is to move down the tree level by level. The path from the root to the final search position is collectively determined by the crowd as follows. The search is started at the root where K_0 tasks are generated at this level. After all workers complete their tasks at the root level, their answers are aggregated, say by majority voting, and the child node that corresponds to the item with the highest vote is selected. Let this node be v. We generate K_1 new tasks for v. We aggregate the answers from the K_1 workers and decide which of the children of v to go to next. We repeat the process until we reach the leaves.

There are two variations of this search technique: *uninformed* and *informed* workers. In the former, a worker is not aware of the other workers' decisions. In the latter, a worker is allowed to view a summary of the other workers' responses to the current task. Algorithm 3 provides an outline of this search procedure.

Algorithm 3: $handle\_all\_levels\_aggregation$(Task t)

```
solved++
if $solved == k$ then // all workers finished at
this level
    result = ← $aggregate\_all\_results$
    if $t.level \neq 0$ then // task at non leaf level
        k← $get\_replications$(tree,t.level-1)
        tree ← $get\_subtree$(result,t.tree)
        for $i = 1$ to $k$ do
            w ← $choose\_worker$()
            t← $create\_task$(tree,t.level-1,t.q,w)
            $submit\_task$(t,$handle\_all\_levels\_aggregation$)
        end
    else
        return result
    end
end
```

4.3 All-levels Aggregation Strategy with Backtracking

The two search procedures above, namely the leaf-only and all-levels aggregation strategies, cannot compensate for an error. Backtracking is one way to correct potentially wrong decisions of the workers. A priority queue of pointers to the unexplored nodes in the index. Unexplored nodes are ordered in the priority queue according to their height (or level) in the tree. Nodes at the same level in the B$^+$-tree index are ordered by the percentage of votes given to these nodes. To detect bad decisions, workers are shown progressively the leaf node bounds of the current sub-tree at hand. If the workers indicate that the position of the query key falls outside the range of the keys in leaf nodes, then the current sub-tree is abandoned and another node is picked from the top of the priority queue to resume the search.

Figure 4 gives an example of the backtracking search strategy. In Figure 4(a), a palm-tree of fanout 2 is used to index keys 1 through 8. Figure 4(b) illustrates the steps of traversing the tree. The search starts at the root node A. Node B is at the top of the priority queue withas 60% of the workers indicate this node. The priority queue keeps track of other alternatives besides B. In next step we show the range (i.e., the min and max) of the keys in the sub-tree rooted at B, that is from 1 to 4. Assume that the workers indicate that the query key is larger than 4; then we have an error. Hence, we backtrack to node C. We insert in the queue node D. Algorithm 4 outlines the search steps.

Algorithm 4: $handle\_backtrack\_all\_levels\_aggregation$(Task t)

```
solved++
if $solved == k$ then // all workers finished at
this level
    result = ← $aggregate\_all\_results$
    if result outside range of current subtree then
        tree ← $pop\_queue$()
    else
        tree ← $get\_subtree$(result,t.tree)
    end
    $push\_queue$(other voting alternatives)
    if $t.level \neq 0$ then // task at non-leaf level
        k← $get\_replications$(tree,t.level-1)
        for $i = 1$ to $k$ do
            w ← $choose\_worker$()
            t← $create\_task$(tree,t.level-1,t.q,w)
            $submit\_task$(t,$handle\_backtrack\_all\_levels\_aggregation$)
        end
    else
        return result
    end
end
```

5. ANALYSIS

In this section, we study how to set the parameters of the palm-tree index: **tree order** and **cost distribution**. We also introduce the main performance metrics of the palm-tree index, mainly, **error** and **cost**.

5.1 Performance metrics

The location of a key is the rank of the key in the ordered list of keys at the leaf level of the tree. Let q be a query key with a ground truth location, say loc_{truth}, at the leaf level, and loc_{ret} be the retrieved location using one of the search strategies.

DEFINITION 5.1. ***Error*** *E is the distance between ground truth location and retrieved location of the query key*
E= $\mid loc_{truth} - loc_{ret} \mid$

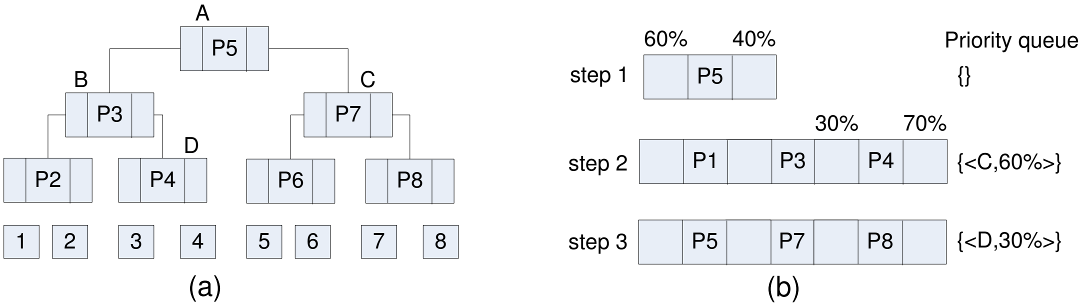


Figure 4: Backtracking all-levels aggregation strategy.

DEFINITION 5.2. ***Cost*** *C is the total number of tasks to complete a job.*

5.2 Tree order and height

The order (fanout) of a regular B$^+$-tree index is usually constrained by the size of the disk page. In contrast, in the palm-tree index, the order of the B$^+$-tree depends on the ability of workers to process at once (in a single task) a specific number of keys (see Section 3).

Notice that increasing the order of the tree increases a worker's probability of making a wrong decision at a given node. However, from a different point of view, in order to get a correct final result, correct decisions must be made at all levels of the tree. Increasing the tree order reduces number of levels required to be correct and hence reduces the final probability of error.

5.3 Cost selection

We adopt majority voting to aggregate workers decisions. Therefore, increasing the number of replicated tasks increases the quality of the aggregated decisions. However, there is almost a saturation point beyond which, increasing the number of replicated tasks would be of little, if any, benefit.

5.4 Cost distribution

Assume that there is a cost bound, say C, to search the index. An important issue is how this cost is distributed among tasks in terms of the number of replicas per tree level as tasks are assigned to workers. In the leaf-only aggregation strategy, every worker has to perform exactly h tasks from the root level to the leaf level. Hence, we can only have $k = \lfloor \frac{C}{h} \rfloor$ workers at most.

In the all-levels aggregation strategy, we propose two techniques to distribute tasks among level, namely *even* and *probabilistic* distribution. In the *even* distribution technique, every level in the palm-tree index is assigned $\lfloor \frac{C}{h} \rfloor$ tasks. *Even* cost distribution assumes equal importance of all the levels of the palm-tree index. However, tree levels are of different importance e.g., when a wrong decision is made at the higher levels (closer to the root level) of the palm-tree index, it would result in a higher final error. This is not the case when wrong decisions are made at the lower levels of the palm-tree index. On the other hand, wrong decisions are more likely to occur at the lower levels than at the higher levels. The reason is that spacing in-between the keys within a node decreases as we descend the tree.

To accommodate for the effect of nodes level on error severity is to replicate tasks in a way that is proportional to the *expected distance error per level*.

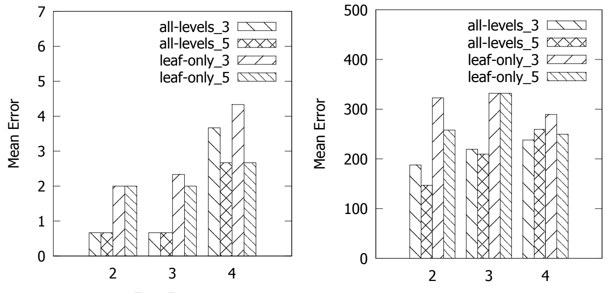
DEFINITION 5.3. ***Probability of distance d error at level l*** *(P_{dl}) is the probability of deviating d branches from the ground truth branch at level l.*

We estimate the final error that would result from a distance d error at level l to be $d \times f^{l-1}$. For example, a distance 1 error at level 1 (i.e., the leaf level) deviates the final result from the correct one by distance 1. A distance 1 error at level h (i.e., the root level) deviates final result about the number of leaf keys in a child subtree of the root node, that is f^{h-1}. We calculate the *expected distance error at level l (EE_l)* to be $\sum_d d \times P_{dl} \times f^{l-1}$.

Techniques for cost distribution for level-by-level voting with backtracking is left for future investigation.

6. EXPERIMENTAL EVALUATION

In order to test the palm-tree index and its search strategies, we implement a web site to submit tasks to workers. This helps in further studying the problem under controlled settings. Two datasets are used in the experiments; the first dataset is a set of 200 square images of different sizes. The second dataset is a set of 1300 used cars images with desired selling prices. A custom-built web crawler is used to collect the dataset of cars from a website of used car ads.



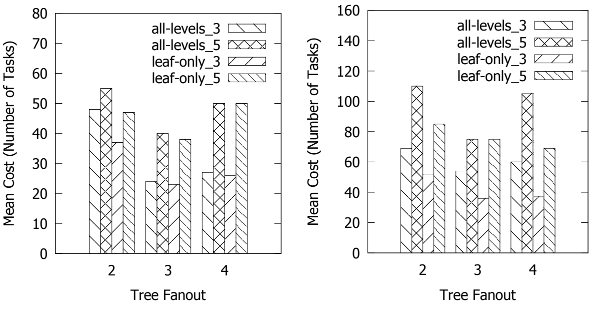
(a) Squares dataset

(b) Cars dataset

Figure 5: Mean error while changing tree fanout.

Figures 5 and 6 give the experimental results of the mean error rate and mean cost for the uninformed all-levels aggregation and leaf-only aggregation search strategies. The parameters of the palm-tree index are set as follows. The fanout ranges from 2 to 4 and task replication is set to 3 and 5.

Figure 5 gives the mean error rates for the two search algorithms when applied to the cars and squares datasets. The error rate for either algorithm is much higher on the cars dataset than on the



(a) Squares dataset (b) Cars dataset

Figure 6: Mean cost while changing tree fanout.

squares dataset. The reason is that it is easier for ordinary users to compare images of squares based on their sizes than it is to compare cars based on their prices. It is also clear that all-levels aggregation outperforms leaf-only aggregation in both datasets. The reason is that in the leaf-only aggregation strategy, if a worker makes a wrong decision at a higher level of the tree, this error affects more significantly his/her answer in the lower levels of the tree.

In the all-levels aggregation, if a worker makes a wrong decision, then other workers can usually compensate for it (We use the majority principle). It is also evident from Figure 5 that increasing the number of workers reduces the error rate.

Figure 6 gives the mean cost needed to perform a query on the palm-tree. The cost for search algorithms is higher on the cars dataset than on the squares dataset. The reason is that the size (and accordingly tree height) of the cars dataset is larger than the that of the squares dataset. It is evident that increasing the number of replications increases the cost needed. It is also evident that increasing the tree fanout generally reduces the cost as the height of the tree decreases. One technique used to reduce the cost is to allow workers to stop at not leaf levels if they agree that a key at a non-leaf node matches the search key.

We are in the process of implementing more extensive experiments that will include larger crowds and other search strategies (i.e., informed all-levels aggregation and all-levels aggregation with backtracking). We plan to study the effect of cost distribution alternatives (even and expected distance errors). We also plan to study the quality of sorts and joins using the palm-tree index.

7. RELATED WORK

Lately, there has been an increasing interest in crowdsourcing. Several crowd-powered database systems, e.g., [6, 15, 12, 11] have been proposed to use human intelegance to perform database operations. These operations mainly focus on sorts and joins, e.g., [11], counting, e.g., [10], and max-finding, e.g., [7, 16]. Other works focus on algorithms to answer top-K, e.g., [4], skyline queries, e.g., [9], and spatial queries, e.g., [8].

The problem of optimizing the number of questions needed to find a set of target nodes within a general directed acyclic graph (DAG) using yes/no questions is studied in [13]. However, the B^+ tree within the palm-tree index posseses more specific properties than a general DAG (i.e., balanced height, and multiple ordered keys within nodes) that requires more specific strategies for tree search and task aggregation methods (Section 4) than in the case when yes/no questions are used.To our knowledge, this is the first work to address the problem of indexing with the crowd.

8. CONCLUSION

In this paper, we study the problem of crowdsourcing-based indexing. We motivate the problem with several application scenarios. We propose a taxonomy of the problem and highlight its main challenges. We propose techniques for index construction and querying. We intend to complete this study with an extensive analysis and experimental evaluation.

Our current work focuses on one-dimensional indexing. We plan to study other variations of crowdsourced indexing, such as multidimensional indexing, spatial and spatio-temporal indexing and fuzzy indexing.

9. REFERENCES

[1] The Amazon Mturk website. `https://www.mturk.com`, 2013.
[2] A. Bozzon, M. Brambilla, and S. Ceri. Answering search queries with CrowdSearcher. In *WWW*, pages 1009–1018, 2012.
[3] A. Bozzon, M. Brambilla, S. Ceri, M. Silvestri, and G. Vesci. Choosing the right crowd: expert finding in social networks. In *EDBT*, pages 637–648, 2013.
[4] S. B. Davidson, S. Khanna, T. Milo, and S. Roy. Using the crowd for top-k and group-by queries. In *ICDT*.
[5] G. Demartini, D. E. Difallah, and P. Cudré-Mauroux. ZenCrowd: leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking. In *WWW*, pages 469–478, 2012.
[6] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. CrowdDB: answering queries with crowdsourcing. In *SIGMOD*, pages 61–72, 2011.
[7] S. Guo, A. G. Parameswaran, and H. Garcia-Molina. So who won?: dynamic max discovery with the crowd. In *SIGMOD*, pages 385–396, 2012.
[8] L. Kazemi and C. Shahabi. Geocrowd: enabling query answering with spatial crowdsourcing. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, pages 189–198. ACM, 2012.
[9] C. Lofi, K. El Maarry, and W.-T. Balke. Skyline queries in crowd-enabled databases. In *EDBT*, pages 465–476, 2013.
[10] A. Marcus, D. Karger, S. Madden, R. Miller, and S. Oh. Counting with the crowd. In *PVLDB*, pages 109–120, 2013.
[11] A. Marcus, E. Wu, D. Karger, S. Madden, and R. Miller. Human-powered sorts and joins. *VLDB Endow.*, 5:13–24, 2011.
[12] A. Marcus, E. Wu, D. R. Karger, S. Madden, and R. C. Miller. Crowdsourced databases: Query processing with people. CIDR, 2011.
[13] A. Parameswaran, A. D. Sarma, H. Garcia-Molina, N. Polyzotis, and J. Widom. Human-assisted graph search: it's okay to ask questions. *VLDB Endow.*, 4:267–278, 2011.
[14] A. G. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom. Crowdscreen: algorithms for filtering data with humans. In *SIGMOD*, pages 361–372, 2012.
[15] H. Park, R. Pang, A. G. Parameswaran, H. Garcia-Molina, N. Polyzotis, and J. Widom. Deco: A system for declarative crowdsourcing. *PVLDB*, 5, 2012.
[16] P. Venetis, H. Garcia-Molina, K. Huang, and N. Polyzotis. Max algorithms in crowdsourcing environments. In *Proceedings of the 21st international conference on World Wide Web*, WWW '12, pages 989–998, 2012.

Crowdsourcing Feedback for Pay-As-You-Go Data Integration

Fernando Osorno-Gutierrez
School of Computer Science
University of Manchester
Oxford Road, Manchester
M13 9PL, UK
osornogf@cs.man.ac.uk

Norman W. Paton
School of Computer Science
University of Manchester
Oxford Road, Manchester
M13 9PL, UK
norm@cs.man.ac.uk

Alvaro A. A. Fernandes
School of Computer Science
University of Manchester
Oxford Road, Manchester
M13 9PL, UK
alvaro@cs.man.ac.uk

ABSTRACT

Providing an integrated representation of data from heterogeneous data sources involves the specification of mappings that transform the data into a consistent logical schema. With a view to supporting large-scale data integration, the specification of such mappings can be carried out automatically using algorithms and heuristics. However, automatically generated mappings typically provide partial and/or incorrect results. Users can help to improve such mappings; expert users can act on the mappings directly using data integration tools, and end users or crowds can provide feedback in a pay-as-you-go fashion on results from the mappings. Such feedback can be used to inform the selection and refinement of mappings, thus improving the quality of the integration and reducing the need for expensive and potentially scarce expert staff. In this paper, we investigate the use of crowdsourcing to obtain feedback on mapping results that inform mapping selection and refinement. The investigation involves an experiment in Amazon Mechanical Turk that obtains feedback from the crowd on the correctness of mapping results. The paper describes this experiment, considers generic issues such as reliability, and reports the results for different mappings and reliability strategies.

1. INTRODUCTION

Large scale data integration, for example over web sources, is challenging due to the heterogeneities that inevitably result from multiple autonomous data publishers. Classical data integration is labour-intensive, and tends to be applied to produce high-quality but high-cost integrations in reasonably stable environments. As a result, there has been a growing interest in pay-as-you-go data integration, where an initial integration is generated automatically, the quality of which is improved incrementally over time [6]. The incremental improvement can take many forms, but is often informed by feedback on the current integration [8].

Crowdsourcing [4] has recently emerged as a way of tapping into human expertise through the web, and systems such as Amazon Mechanical Turk[1] (AMT) and CrowdFlower[2] provide systematic mechanisms for recruiting and paying workers for carrying out specific tasks. This paper explores the hypothesis that crowdsourcing can provide cost-effective feedback of a form that can support data integration. The paper contributes an experiment design that tests the hypothesis, and an analysis of the results of the experiment. Specifically, given automatically generated mappings, we use the crowd to provide feedback on the correctness of the results produced by those mappings. Such feedback has been shown to be useful by several authors. For example, Belhajjame *et al.* [1] showed how such feedback could be used to select between and inform the generation of new mappings; and Talukdar *et al.* [15] used such feedback to identify effective ways of answering keyword queries over structured sources.

The remainder of this paper is structured as follows. Section 2 describes related work on data integration and crowdsourcing. Section 3 describes the data integration context for the experiment. Section 4 presents the design of the experiment including the role of redundancy in validating results. Section 5 presents and analyses the results of the experiment. Section 6 draws some conclusions.

2. RELATED WORK

This section describes work related to that described in this paper, focusing on results in *data integration* and *crowdsourcing for data management*.

In terms of *data integration*, our research builds on the work of Belhajjame *et al.* [1], who use feedback on mapping results to annotate mappings with estimates of their *precision* and *recall*. More specifically, feedback takes the form of *true positive*, *false positive* and *false negative* annotations on tuples returned by mappings, and such feedback allows estimates for precision and recall to be obtained; the more feedback, the more accurate the estimates are likely to be. The estimates of precision and recall are then used to support the selection of mappings for answering a query that meet specific user requirements (e.g. by selecting mappings in a way that maximises recall for a precision above some

---

[1] http://mturk.amazon.com/

[2] http://crowdflower.com/

threshold), and the generation of new mappings whose precision and recall can be estimated in the light of the feedback. Belhajjame *et al.* evaluate the techniques using synthetically generated feedback; this paper explores the collection of such feedback using crowdsourcing.

Our work is one of a growing collection of contributions in crowdsourcing, for which a survey has been carried out by Doan *et al.*[4]. In this survey, a classification of crowdsourcing systems is presented. The application developed in our work would have been classified as a standalone application with explicit collaboration of users. In terms of *crowd sourcing for data management*, there are a range of other approaches that share this classification. Several proposals have been made in which crowdsourcing plays a role in query answering, including CrowdDB [7] and CrSS [14]; such systems extend standard query evaluation over static data sources with techniques for consulting the crowd for information that is not available through other means. In relation to data integration, McCann *et al.* [13] propose using online communities to support the matching of attributes from different sources; such work complements our results, as matches are often used as a foundation for the construction of mappings. At a later stage in the data integration pipeline, CrowdER [16] carries out entity resolution with a technique that combines machine and human work; as in this paper, data is first processed by automatic techniques, the results of which are then verified using the crowd. Our results complement these recent contributions by evaluating the use of the crowd to obtain an additional type of feedback, and by including comparative evaluations of different techniques for ascertaining the reliability of the feedback from the crowd.

3. DATA INTEGRATION CONTEXT

This paper tests the hypothesis that feedback from the crowd can inform the annotation of mappings with information about their quality, where the feedback takes the form of *true positive* or *false positive* feedback on tuples produced by the mappings. This section describes the data integration context for the experiment, including the data that is to be integrated, mapping generation, and the sampling of data on which feedback is to be obtained.

3.1 Experimental data

As music is a well known domain, we use as data sources two music databases (Musicbrainz[3] and Discogs[4]). The schemas of Musicbrainz and Discogs contain a range of information about artists and their recordings. In our experiment we focus on the entity *artist* of each database and the attributes *name*, *real name*, *gender*, *country*, *type* and *begin date year*, that essentially provide simplified views of the artist information from the sources. Given this focus, the tables *musicbrainz_artist*(name, gender, country, type, begin_date_year) and *discogs_artist*(name, realname) were created to form the *source schema* in the experiment.

3.2 Generation of schema mappings

We used Spicy [2] to automatically generate mappings for which feedback is obtained. Spicy is a schema mapping tool that generates candidate schema mappings as SQL views that can be used to map data from a *source schema* into a *global schema*. Spicy requires as input one source schema and one global schema. In the source schema, a foreign key was inserted between the attributes *artist_discogs.name* and *artist_musicbrainz.name*. The global schema consists of a table *artist* that contains the union of the attributes from the tables *artist_musicbrainz* and *artist_discogs*. Figure 1 presents the architecture of Spicy [2].

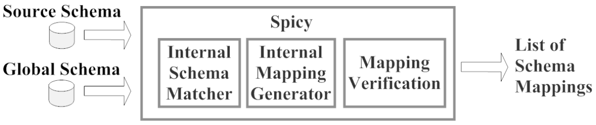


Figure 1: Simplified Spicy architecture.

Mapping	Source schema attribute	Global schema attribute
Mapping 1 (M1)	discogs_artist.name	name
	musicbrainz_artist.country	country
	musicbrainz_artist.type	type
Mapping 2 (M2)	discogs_artist.name	name
	musicbrainz_artist.name	country
	musicbrainz_artist.type	type
Mapping 3 (M3)	musicbrainz_artist.country	name
	musicbrainz_artist.name	country
	musicbrainz_artist.type	type

Table 1: Selected mappings.

Spicy requires sample instance values in the source schemas to generate candidate schema mappings. With a large number of tuples (more than 250), Spicy generates only one mapping. This is insufficient for our experiment, for obvious reasons. Since the reason behind this outcome is that ample information about the sources enables Spicy to generate fewer alternative mappings, we provided the tool with fewer (viz., 200) source tuples. Indeed this caused Spicy to generate several alternative mappings on which we were then able to obtain feedback and proceed with our experimental goals. Also, the data was constrained to artists that started to play in 1980 or later and that are from the United States[5]. This process generated ten candidate mappings. After this process, we selected the attributes *name*, *type* and *country* on which to obtain feedback. These attributes are common to most of the artists, whereas some attributes are relevant only for certain artists. For example, the attribute *gender* is relevant only to *person* artists and not to *group* artists. The three mappings that met the criteria to be used in the experiment are presented in Table 1.

The mappings produced by Spicy are SQL statements inferred from the source schemas and a sample of source tuples. When the Spicy-inferred mappings were run against the complete tuple-content of the Musicbrainz and Discogs databases, we obtained the results that populate the global schema characterized by those mappings. Each mapping,

---

[3] Musicbrainz - http://www.musicbrainz.org/

[4] Discogs - http://www.discogs.com/

[5] This action was in response to a study of the demographics of the workers that participate in Amazon Mechanical Turk. Most workers are from the US and in an age bracket that suggests that their knowledge will be sharper for post-1980 artists and groups.

as a SQL query against the global schema, then produced 4203 rows. However, in our experiment we require more than three mappings to evaluate and we want to have some mappings that are likely to obtain a precision between 0 and 1 (the mappings of Table 1 have precision of either 0 or 1); for this reason, we have guided the generation of additional mappings. M1 was used as the starting point for the generation of additional mappings. The process to generate additional mappings is the following. First, we made a copy of the results from M1, which has a precision of 1. After that, we changed a percentage of tuples in the results to a different value for the *country* attribute from the set {*Canada, Australia, New Zealand, France, United Kingdom*}. By this means, four more mappings were created with different percentages of tuples modified in each mapping. We modified 20% for Mapping 4 (M4), 40% for Mapping 5 (M5), 60% for Mapping 6 (M6) and 80% for Mapping 7 (M7). In total, then, we have seven mappings: three generated directly using Spicy, and four mappings that are variants of one of the Spicy mappings.

3.3 Sampling mapping results

Having defined the mappings, it was necessary to decide on how many tuples should feedback be obtained given that it would be too expensive to obtain feedback from the crowd on all the tuples produced by the mappings. For this purpose, we used a statistical method, simple random sampling [3], to determine the sample size for populations with variables that can take only two values (i.e. *Correct* or *Incorrect*). The method to determine the sample size considers a confidence level and a standard error, which are commonly used in social sciences. For our study we computed the sample size for a confidence of 95% that the mean (i.e. the percentage of values that are annotated as *Correct* or *Incorrect*) would be within 5% of the correct mean. For these requirements, the resulting sample size of a population of 4203 is 352 tuples.

4. HUMAN INTELLIGENCE TASK GENERATION

Having identified the tuples on which feedback is to be obtained, the information is now in place to enable the design of the tasks to be completed by the crowd. For the experiment, we used the AMT crowdsourcing platform, within which user activities are referred to as *Human Intelligence Tasks* (HITs). This section describes how tuples are allocated to HITs, including redundancy and screen design.

4.1 Distribution of result tuples

The seven samples of tuples of size 352 obtained for each mapping are distributed into groups of 25 unique tuples that will feature in questionnaires, such that each questionnaire is a HIT, and each result tuple is the subject of one question. The distribution of tuples considers that one HIT should not have more than one tuple with information about the same artist, and that the number of tuples in a HIT produced from the same mapping is controlled. The number of resulting HITs is presented in Table 2.

Category	**Size**
Mappings	7
Tuples per mapping	352
Total tuples	2464
Tuples per HIT	25
HITs generated after distribution	99

Table 2: Distribution of tuples into HITs.

4.2 Reliability

In our experiment we obtain feedback from humans, who, of course, may fail to provide reliable answers (e.g. answers that contradict those of other users, or even self-contradictory ones). The goal for the investigation is to minimise the risk that unreliable data is obtained, or to manage the unreliability when it is encountered.

A method to estimate the reliability of a single observer is called *Intra Observer Reliability* (IaOR) [10]. With a view to estimating IaOR, some of the questions are asked more than once. To estimate IaOR, each respondent (a worker in AMT) answered two HITs at least two hours apart to reduce the risk that the worker remembered their first answers and answered based on memory. This is called *the practice effect* in social sciences [10].

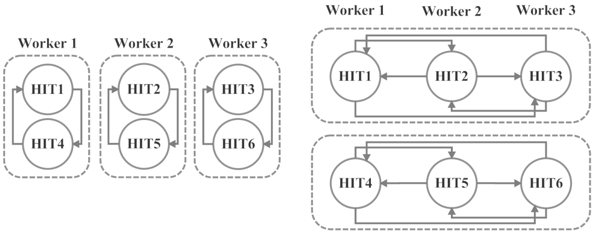
Only a subset of questions in each HIT is redundant. The HITs are organised in pairs such that each HIT contains three random questions from the other HIT in his pair. In Figure 2(a), each arrow represents three questions. Therefore, in each pair there are 6 redundant questions used to assess IaOR. After introducing redundancy for IaOR, each HIT contains 28 questions. In Figure 2(a), HIT1 and HIT4 form one pair, which is answered by *Worker 1*. Then, the reliability is estimated by the percentage of agreement of the 6 redundant questions. The IaOR associated with different numbers of consistent questions is as follows. The worker obtains 16.60% of IaOR for answering consistently 1 question, 33.30% for 2 questions, and so on.

However, there exists the possibility that the answers of a worker are not correct; it is possible to provide (consistently) wrong answers, which would give rise to a high estimate for IaOR. To avoid this situation, we can compare answers between workers, by way of *Inter Observer Reliability* (IrOR). We assume that respondents are reliable if they provide the same answers [10, 5]. To introduce redundancy for IrOR, first we group the HITs in groups of three. Then, inside each group, we choose at random two questions from each HIT to occur in another HIT too, thereby introducing the redundancy required to obtain evidence of IrOR. The selected questions are different from the questions selected for IaOR. Figure 2(b) shows how redundancy was introduced in order to estimate IrOR; each arrow represents 2 questions. In each group there are 6 redundant questions for IrOR.

After introducing redundancy for IaOR and IrOR, each HIT contains 32 questions (25 unique + 3 for IaOR + 4 for IrOR).

In each group of three HITs for IrOR, we estimate the percentage of agreement of each pair in the group. Therefore, we obtain three evaluations, and each worker receives two IrOR evaluations in the group. Then, when reporting results that take into account IrOR in the experiments, we apply an IrOR threshold, and discard the HITs from workers that obtained IrOR evaluations below the threshold.

As an example, consider the group of HITs: HIT1, HIT2 and HIT3, answered by *Worker1*, *Worker2* and *Worker3*,



(a) Redundancy for IaOR   **(b) Redundancy for IrOR**

Figure 2: Redundancy for reliability.

respectively. Then, Worker1 and Worker2 agree on 4 questions and obtain 66.6%; Worker1 and Worker3 agree on 4 questions and obtain 66.6%; finally Worker2 and Worker3 agree on 6 questions and obtain 100%. If we set an inter observer reliability threshold of 100%, in this example we would ignore the answers in this group of HITs from Worker1, who has two reliability evaluations were below the threshold. However, Worker2 and Worker3 are considered reliable because each has only one evaluation of reliability below the threshold, which is not enough to determine that they are unreliable (we assume that it was Worker 1 who provided incorrect answers).

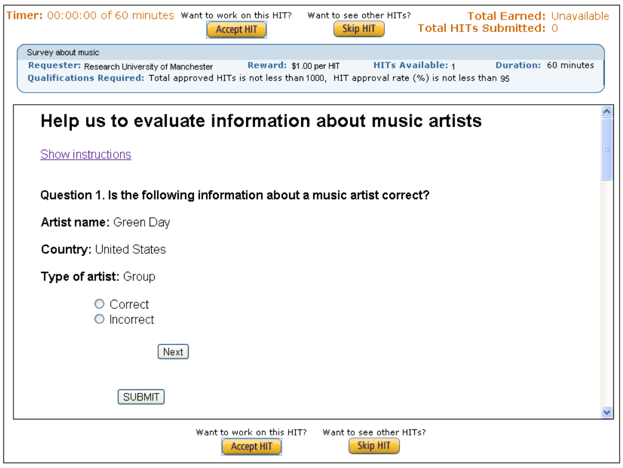
Note that every worker answered two HITs in the experiment. Therefore, they are evaluated twice for inter observer reliability but in different groups. For example, Worker1 answered the HIT1 and HIT4, which are in different groups for IrOR, as illustrated in Figure 2(b).

4.3 HIT Design

An example HIT is presented in Figure 3. The possible answers to a question are *Correct* or *Incorrect*. All the questions in the survey have the same structure. To set the questionnaire length, we carried out pilot tests. We estimated that 32 questions would take users less than 20 minutes to answer. We paid $1.00 (one US dollar) per HIT, which is a higher than average payment per completed task, with the goal of making the HIT attractive to workers [12, 9]. The workers could find the HITs by browsing the AMT tasks list or by searching the keywords music, survey or artists. The HITs in the experiment were available to workers located in the US. We accepted workers that have responded successfully to 1000 HITs before and that have finished successfully 95% of all the HITs that they have ever responded to before (*approval rate*). Thus we have been quite selective in terms of the experience and ratings of participants.

4.4 Experiment Setup on AMT

A crowdsourcing application was developed to control the publishing of HITs in AMT. The application consists of: a *controller* that configures and posts HITs in the AMT platform; a *database* that stores the result tuples that are assigned to the HITs, the AMT IDs of the workers, and data used to control which HITs are assigned to which workers; and a Tomcat Apache *web server* that contains Java Server Pages (JSP) forms for the HITs. The JSP forms use the AMT ID of the worker requesting to view a HIT to retrieve the tuples that are chosen to be part of the HIT assigned to that worker.



Timer: 00:00:00 of 60 minutes

Total Earned: Unavailable
Total HITs Submitted: 0

Survey about music
Requester: Research University of Manchester Reward: $1.00 per HIT HITs Available: 1 Duration: 60 minutes
Qualifications Required: Total approved HITs is not less than 1000, HIT approval rate (%) is not less than 95

Help us to evaluate information about music artists

Question 1. Is the following information about a music artist correct?

Artist name: Green Day
Country: United States
Type of artist: Group

○ Correct
○ Incorrect

Figure 3: HIT example.

The crowdsourcing application went live, and 90 HITs were completed in 40 days. This is a substantial elapsed time compared with that reported by other AMT users (e.g. [9]). We expect that this can be explained by the complex and somewhat unconventional pairing of HITs to enable IaOR, the results of which are discussed further below. On average, the HITs took 12:40 minutes to answer. New HITs were made available every week or when previous HITs were finished in order to appear in the first pages of the AMT task lists. This is a common practice followed by other AMT requesters [9].

5. EXPERIMENTAL RESULTS

5.1 Precision and Error in Precision

The candidate mappings used in our experiment are annotated to indicate if they meet the requirements of users. For this purpose, we annotate the mappings with values of precision and the error in precision as in Belhajjame *et al.* [1]. Precision is the fraction of retrieved tuples that are indicated to be correct by the user [11]. We can estimate the precision of a mapping j after i feedback instances with the formula below.

$$Precision_{ij} = \frac{true\ positives_{ij}}{true\ positives_{ij}\ +\ false\ positives_{ij}} \quad (1)$$

where, for mapping j, $true\ positives_{ij}$ ($false\ positives_{ij}$) is the number of correct (incorrect) tuples retrieved after i feedback instances. The calculation of precision is incrementally updated as the user provides feedback; in the experiment, i changes from *0* to *352*, which is the number of tuples of the mapping evaluated by the workers. We are interested in measuring how user feedback can contribute in the evaluation of the mappings. For this reason, we compare the estimated precision for a mapping j with i feedback instances to a known precision value, which is a *gold standard precision* (GSP) for the mapping j. The *error in precision* can be used for this purpose.

$$Error\ in\ Precision_{ij} = |GSP_j - Precision_{ij}| \quad (2)$$

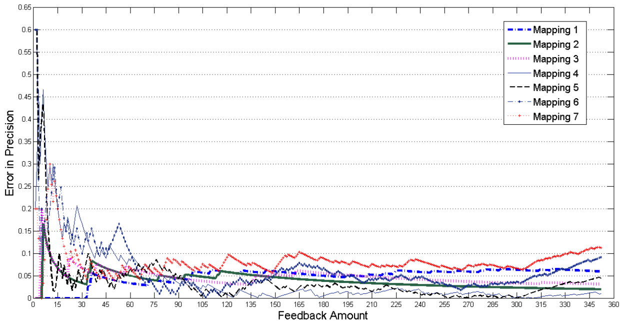


Figure 4: Error in precision of each of the mappings as feedback is collected.

where GSP_j is the gold standard precision of the mapping j. As in Belhajjame *et al.* [1], we use the *average error in precision* (AEP) to measure the quality of an annotation, i.e., the difference between the estimated precision and the GSP.

$$Average\ Error\ in\ Precision_i = \frac{\sum_{j=1}^{K} Error\ in\ Precision_{ij}}{K} \tag{3}$$

where K is the total number of mappings. The AEP is, likewise, incrementally updated as feedback from users arrives.

5.2 Annotation Quality

Using the definitions from Section 5.1, the precision of the mappings was estimated based on the feedback obtained from the crowd. To understand how effective the feedback from the crowd has been at estimating the precision, Figure 4 shows the error in the estimated precision of each of the mappings as the amount of feedback obtained increases. The following can be observed: (i) The error in precision drops rapidly as feedback is collected, such that most mappings have an error of less than 0.1 from around 50 feedback instances. (ii) The errors obtained for mappings M1, M2 and M3, where the ground truth precision is 0 or 1, are in the same range as those obtained for M4, M5, M6 and M7, where the ground truth precision is between 0 and 1. We had expected larger errors in precision for M4 to M7 because these mappings seem to have less obvious errors than those in M2 and M3. In M2 and M3, the error in the mapping is that values are presented in the wrong columns, whereas in M4 to M7 incorrect but plausible values are provided for an attribute. Nevertheless, the users were able to identify errors in nationality with similar reliability to column transposition. (iii) The error in precision for some mappings increases towards the end of feedback collection; this is most likely explained by the effectively random order in which different users provide feedback, with several fairly unreliable users participating late in the experiment.

Abstracting over the plots for the different mappings, Figure 5 shows the AEP from Formula 3 as feedback is collected. We observe that when fewer than 70 feedback instances per mapping have been obtained, the plot is quite unstable, but that thereafter, errors are both quite small and quite stable. This suggests that reasonably reliable estimates of mapping quality can be obtained with quite small amounts of feedback, and thus at a modest financial cost.

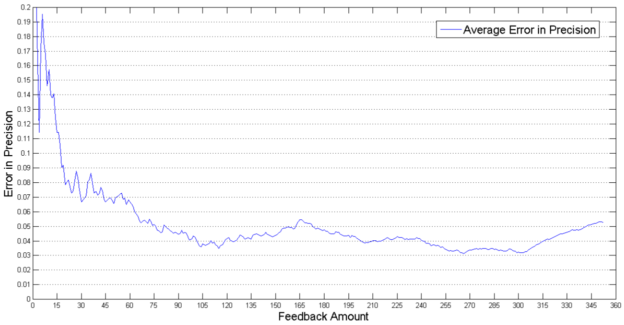


Figure 5: Average Error in Precision (AEP).

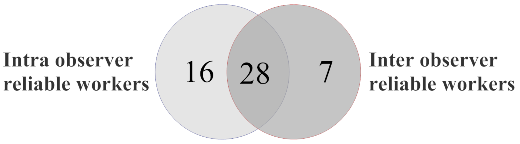

Intra observer reliable workers
16
28
7
Inter observer reliable workers

Figure 6: Distribution of reliable workers.

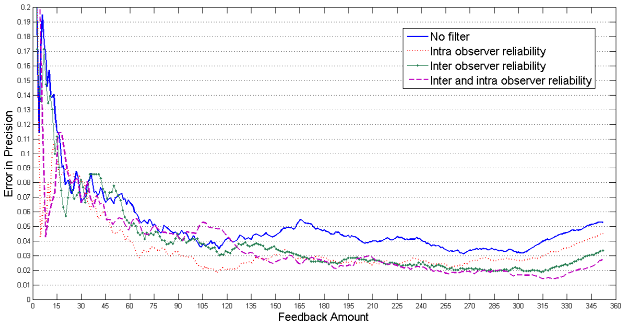


Figure 7: AEP with different reliability filters.

5.3 Feedback Reliability

Applying the reliability techniques from Section 4.2 to the data from Section 5.2, using a reliability threshold of 100%: 44 out of 51 workers are reliable for intra observer reliability; 28 out of 51 workers are reliable for intra and inter observer reliability; and 35 out of 51 workers are reliable for inter observer reliability.

Some users were found to be reliable only for IaOR, some users were found to be reliable only for IrOR, and some users were found to be reliable for both IaOR and IrOR. Figure 6 shows the distribution of the workers that were found reliable against each of the reliability methods.

We estimate the AEP with the feedback obtained filtered to remove the users who are considered to be unreliable by the different techniques. After filtering the feedback, we report the AEP for the results filtered with IaOR, the results filtered with IrOR, and the results filtered with IaOR and IrOR in Figure 7. The results reflect the order in which

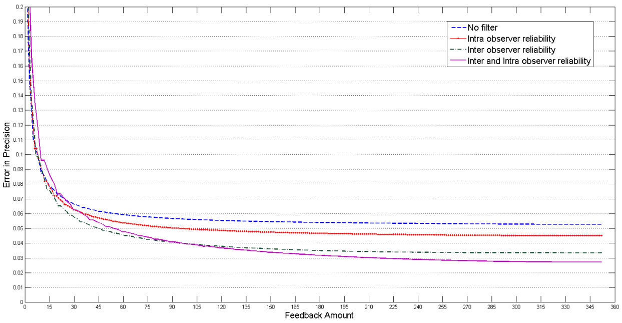


Figure 8: AEP with different reliability filters and randomised order of collection.

the workers provided the feedback. Note that the *Feedback Amount* in the horizontal axis is the total amount of feedback obtained, and that the different reliability schemes all discard some of that feedback. The following can be observed: (i) there is significant variation in the error during the early parts of feedback collection, but this stabilises quite rapidly to a low error as the feedback is increased; and (ii) the different reliability schemes provide better results for different amounts of feedback, reflecting the impact on the conclusions that can be drawn of the order in which users provide feedback. To remove this effect, we have repeatedly randomly changed the order in which the feedback has been obtained from the users, until such time as the additional of further random orderings made no difference to the plot. The resulting plot is provided in Figure 8. This plot shows that while the combined filtering eventually yields the greatest reduction in error, inter observer reliability is almost as effective, and is more effective than intra observer reliability. This is an important observation, because inter observer reliability is much easier to implement, as it does not involve users carrying out repeated tasks at different times.

6. CONCLUSIONS

This paper has studied the use of crowdsourcing to collect feedback on the correctness of query/mapping results; such feedback has been shown to be useful for different data integration tasks, including keyword query evaluation and mapping refinement [1, 15]. The following contributions have been made:

- An experiment has been designed that collects true positive and false positive annotations for query results using the crowd, including techniques for estimating sample sizes and for integrating reliability tests.
- The results of the experiment show that precision estimates derived from crowd feedback improve rapidly as feedback is accumulated, suggesting that the crowd can be used as a cost-effective way of selecting between collections of automatically generated mappings. This confirms the experimental result obtained with synthetic feedback reported by Belhajjame *et al.* [1].
- The experiment design included both inter- and intra-observer reliability. Although simpler for both experimenters and users, inter-observer reliability turned out to be more effective than intra-observer reliability.

Acknowledgment. Fernando Osorno-Gutierrez is supported by a grant from the Mexican National Council for Science and Technology (CONACYT).

7. REFERENCES

[1] K. Belhajjame, N. W. Paton, S. M. Embury, A. A. A. Fernandes, and C. Hedeler. Feedback-based annotation, selection and refinement of schema mappings for dataspaces. In *EDBT*, pages 573–584, 2010.

[2] A. Bonifati, G. Mecca, A. Pappalardo, S. Raunich, and G. Summa. The spicy system: towards a notion of mapping quality. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 1289–1294, New York, NY, USA, 2008. ACM.

[3] D. de Vaus. *Surveys In Social Research (Social Research Today)*. Routledge, 2002.

[4] A. Doan, R. Ramakrishnan, and A. Y. Halevy. Crowdsourcing systems on the world-wide web. *Commun. ACM*, 54(4):86–96, 2011.

[5] A. Fink. *The Survey Handbook*. The Survey Kit. SAGE Publications, 2002.

[6] M. Franklin, A. Halevy, and D. Maier. From databases to dataspaces: a new abstraction for information management. *SIGMOD Rec.*, 34(4):27–33, Dec. 2005.

[7] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. Crowddb: answering queries with crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, SIGMOD '11, pages 61–72, New York, NY, USA, 2011. ACM.

[8] C. Hedeler, K. Belhajjame, A. A. A. Fernandes, S. M. Embury, and N. W. Paton. Dimensions of dataspaces. In *BNCOD*, pages 55–66. Springer, 2009.

[9] P. G. Ipeirotis. Analyzing the amazon mechanical turk marketplace. *XRDS*, 17(2):16–21, Dec. 2010.

[10] M. Litwin. *How to Measure Survey Reliability and Validity*. The Survey Kit. SAGE Publications, 1995.

[11] C. D. Manning, P. Raghavan, and H. Schtze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[12] W. Mason and D. J. Watts. Financial incentives and the "performance of crowds". *SIGKDD Explor. Newsl.*, 11(2):100–108, May 2010.

[13] R. McCann, W. Shen, and A. Doan. Matching schemas in online communities: A web 2.0 approach. In *ICDE 2008*, pages 110 –119, april 2008.

[14] A. Parameswaran and N. Polyzotis. Answering queries using humans, algorithms and databases. In *CIDR 2011*. Stanford InfoLab, January 2011.

[15] P. P. Talukdar, M. Jacob, M. S. Mehmood, K. Crammer, Z. G. Ives, F. Pereira, and S. Guha. Learning to create data-integrating queries. *Proc. VLDB Endow.*, 1(1):785–796, Aug. 2008.

[16] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. Crowder: crowdsourcing entity resolution. *Proc. VLDB Endow.*, 5(11):1483–1494, July 2012.

Part III.

Vision Papers

Crowds, not Drones: Modeling Human Factors in Interactive Crowdsourcing

Senjuti Basu Roy[†], Ioanna Lykourentzou[††], Saravanan Thirumuruganathan[‡,△]
Sihem Amer-Yahia[◇], Gautam Das[‡,△].
[†]UW Tacoma, [††]CRP Henri Tudor/INRIA Nancy Grand-Est, [‡]UT Arlington, [△]QCRI, [◇] CNRS, LIG
senjutib@uw.edu, ioanna.lykourentzou@{tudor.lu,inria.fr},
saravanan.thirumuruganathan@mavs.uta.edu, sihem.amer-yahia@imag.fr,
gdas@uta.edu

ABSTRACT

In this vision paper, we propose `SmartCrowd`, an intelligent and adaptive crowdsourcing framework. Contrary to existing crowdsourcing systems, where the process of hiring workers (crowd), learning their skills, and evaluating the accuracy of tasks they perform are fragmented, siloed, and often ad-hoc, `SmartCrowd` foresees a paradigm shift in that process, considering unpredictability of human nature, namely *human factors*. `SmartCrowd` offers opportunities in making crowdsourcing intelligent through iterative interaction with the workers, and adaptively learning and improving the underlying processes. Both existing (majority of which do not require longer engagement from volatile and mostly non-recurrent workers) and next generation crowdsourcing applications (which require longer engagement from the crowd) stand to benefit from `SmartCrowd`. We outline the opportunities in `SmartCrowd`, and discuss the challenges and directions, that can potentially revolutionize the existing crowdsourcing landscape.

1. INTRODUCTION

Crowdsourcing systems have gained popularity in a variety of domains. Common crowdsourcing scenarios include data gathering (asking volunteers to tag a picture or a video), document editing (as in Wikipedia), opinion solicitation (asking foodies to provide a summary of their experience at a restaurant), collaborative intelligence (asking residents to match old city maps), etc. The action of each worker involved in crowdsourcing can be viewed as *an approximation of ground truths*. In the examples we describe, truth could be a complete set of tags describing a picture, a Wikipedia article, an exhaustive opinion on a restaurant, etc. Truth can be objective (single ground truth) or subjective, where there may be different truths for different users (e.g., youngsters tend to like fast-food restaurants while young professionals may not, photography professionals tend to prefer tags reflecting photo quality as opposed to photo content). In this paper, we are interested in the question of harnessing the crowd to approximate truth(s) effectively and efficiently while taking into account the innate uncertainty of human behavior, named human factors.

Crowdsourcing Today: Existing systems are built on top of private or public platforms, such as Mechanical Turk, Turkit, Mob4hire, uTest, Freelancer, eLance, oDesk, Guru, Topcoder, Trada, 99design, Innocentive, CloudCrowd, and CloudFlower [3]. Tasks are typically small, independent, homogeneous, have minor incentives, and do not require longer engagement from workers. Similarly, the crowd is typically volatile, arrival and departure is asynchronous, with different levels of attention and accuracy.

Limitations of current approaches: There are two primary limitations related to current crowdsourcing approaches. The first refers to the separation and non-optimization of the underlying processes in a dynamic environment. The second limitation is related to the omission of human factors when designing an optimized crowdsourcing solution. In fact, while recent research investigates some of the optimization aspects, those aspects are not studied in conjunction with human factors.

Three major processes involved in the task of ground-truth approximations are - worker skill estimation, worker-to-task assignment, and task accuracy evaluation. Most current commercial crowdsourcing systems (a survey of which can be found in [3]) either do not offer algorithmic optimization, or do that partially and in isolation. Pre-qualification tests, the usage of golden standard data, or hiring of workers based on worker past performance are the norm. Task assignment is completely open and allows self-appointment by the workers, thus undermining quality (workers prefer to increase their individual profit over accomplishing qualitative tasks). Worker wage is often pre-determined and fixed per task, oblivious to the quality of the actual pool of workers who undertake the task in reality. Recent research undertakes some of the challenges unsolved by commercial platforms, and proposes active learning strategies for task evaluation [10, 1, 7], task assignment process [5], adjusting worker wages accordingly to skills [11]. However these works: i) focus on a specific crowdsourcing application type (mostly real-time crowdsourcing with highly volatile crowds) thus losing genericity, and ii) focus on the algorithmic optimization of some but not all of the involved processes (e.g. skill learning, or wage determination, or task assignment).

A more critical limitation refers to the omission or inadequate incorporation of the uncertainty stemming from human factors into the design of the crowdsourcing optimization algorithm. Algorithmic solutions rely on simple, idealized models (e.g. known worker skills or steady worker performance). A recent work [8] proposes probabilistic worker skill estimation models, based on the workers past performance, considering potential deviations in worker performance. Another recent work studies the egoistic profit-oriented objectives of individual workers to incentivize them (e.g. by properly adjusting wages) in order to calibrate algorithms that approximate the ground truth related to the crowdsourcing task [2]. Benefit of explicit feedback and information exchange between workers is studied [4, 6] to improve worker self-coordination, but no existing research incorporates these aspects in a dynamic and interactive environment, nor are there optimized solutions for ground truth discovery, considering human factors.

Opportunities: Future crowdsourcing systems therefore need to, first treat the crowdsourcing problem not in optimization silos, but as an adaptive optimization problem, seamlessly handling the three main crowdsourcing processes (worker skill estimation, task assignment, task evaluation). Secondly and equally important, the uncertainty stemming from human factors needs to be quantified and incorporated into the design of any future algorithm that seeks to optimize the above adaptive crowdsourcing problem. For example, the estimation of every worker parameter that can be influenced by uncertainty needs to be incorporated into the design of the crowdsourcing optimization process. Also, the planning horizon and the optimization boundaries of any algorithm applied to facilitate crowdsourcing need consequently to be determined with this uncertainty in mind. New challenges rise from the above two opportunities, of adopting a seamless crowdsourcing process and of incorporating uncertainty into it.

In summary, crowdsourcing has transitioned from being used as research tool into a research topic on its own. Sooner or later, database researchers have to confront the issues resulting from hybrid processing involving humans and computers. The uncertainties arising due to human factors in crowdsourcing are very different from traditional uncertainty, such as in probabilistic databases [9]. `SmartCrowd` envisions crowdsouring as an adaptive process where human factors are given the significance they deserve. Further, we also introduce a mechanism of crowd-indexing by which workers are organized into groups. Such indices are triggered by human factors, dynamically maintained and provide an efficient way to search for workers.

2. OUR VISION

We propose to rethink crowdsourcing as an adaptive process that relies on an interactive dialogue between the workers and the system in order to build and refine worker skills, while tasks are being completed. In parallel, as workers complete more tasks, the system 'learns" their skills more accurately, and this adaptive learning is used to dynamically assign tasks to workers in the next iteration, by understanding the intrinsic uncertainty of human behavior. Note that, key to the success of these steps is the knowledge on ground truth, which the system is oblivious of (and wishes to discover) in the first place. The primary paradigm shift in `SmartCrowd` is in envisioning the process of ground-truth discovery to be dynamic, adaptive, and iterative in discovering skills required for tasks, evaluating the accuracy of completed tasks, learning skills of involved workers, assigning tasks to workers, determining the number of workers and offered incentives, considering human factors. Interestingly, these intermediate objectives are often inter-dependent, and improving one improves others. The overall objective of this adaptive process is *to maximize accuracy and efficiency while reducing cost and effort.*

2.1 High Level Architecture

The primary distinction of our framework is the deliberate acknowledgement of the importance of human factors in crowdsourcing and how it guides each of our objectives in a dynamic environment. Further, we envision our framework to have an interactive dialogue with the workers to enable adaptive learning, while the workers participate in crowdsourcing tasks. The first two dimensions we tackle are:

- **"who knows what"**, i.e. to evaluate the contributions of workers and based on that to estimate their skills with the least possible error (skill learning process).
- **"who will be asked to contribute to what"**, i.e., by learning required skills for tasks and estimating workers' skills, assign tasks to workers (task assignment process).

`SmartCrowd` functions as follows: workers enter the crowdsourcing platform and complete tasks. Many crowdsourced tasks typically require multiple skills. In the beginning, `SmartCrowd` holds no knowledge over the skills of newcomers. Furthermore, some required skills may be latent, and unknown to `SmartCrowd` in the beginning. As the workers undertake and complete more tasks, `SmartCrowd` discovers latent skills, evaluates workers contribution to the tasks and learns their skills, and therefore assign appropriate tasks to the workers, which in turn achieves higher accuracy and improved efficiency in the process. Moreover, this process is adaptive and iterative, worker skills are "learnt more accurately" and "used more appropriately" over time, ensuring gradual improvement.

Figure 1 shows two primary functionalities that are improved adaptively in `SmartCrowd`: one depicting learning worker skills, and the other depicting completion time of the (ground truth discovery) tasks. More precisely, the steeper the skill estimation error curve gets, the faster we arrive to accurate approximation of workers' skills, i.e., the faster we can profile workers with low error. Also, there is a moment in time when the approximation error in skill estimation is acceptable. This is marked in the figure with a dashed vertical line. Before that, the system is in "cold start" phase, and does not know "much" about workers. Traditionally, this problem is tackled with uniform-prior assumptions, spammer-hammer model, multi-dimensional wisdom of crowd to bootstrap user skills [3]. After that, the framework continues to improve its knowledge on workers' skills and adaptively assigns tasks to workers in iteration, until the system determines that a stopping condition has been reached. Interestingly, faster minimization of skill estimation error leads to earlier termination of cold start period (i.e., the dashed vertical line to the left), which gives rise to better opportunities in designing the task assignment process (task assignment improvement area).

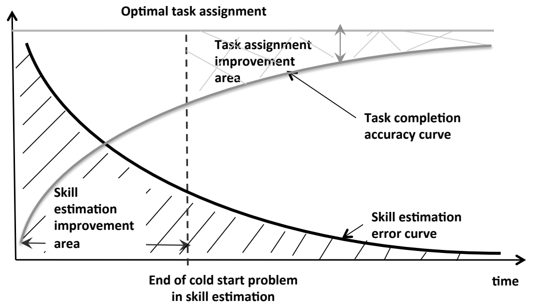

Optimal task assignment
Task assignment improvement area
Task completion accuracy curve
Skill estimation improvement area
Skill estimation error curve
End of cold start problem in skill estimation
time

Figure 1: Tradeoff between Skill Estimation Accuracy and Task Completion Efficiency

As skill estimation improves, task completion efficiency is also expected to improve, since the system can assign tasks more intelligently to workers. However, worker skill estimation is critically related to accurate *task evaluation process*, i.e., to evaluate the accuracy of the completed tasks by the workers. In the absence of explicit ground truth, `SmartCrowd` resorts to uncovering the ground truth using workers themselves. While this interactive process does not necessarily require longer engagement from the workers in the system, it offers opportunities for improved learning. Therefore, the third and final dimension we tackle is:

- **"engaging workers explicitly to improve learning"**, i.e., how to further exploit the learned expertise of workers by engaging them explicitly in evaluating the skill of other workers or by completing more tasks.

Most importantly, these dimensions in `SmartCrowd` are studied in conjunction with two key aspects that are exclusive to crowdsourcing - **human factor** and **scale**. The unpredictability and inconsistency in human behavior are deliberate in the design of `SmartCrowd`. Additionally, `SmartCrowd` envisions the designed solutions to be scalable, i.e., tolerant to the size of the crowd, and its volatility. To the best of our knowledge, `SmartCrowd` is the first ever framework that considers these factors explicitly in crowdsourcing. Finally, `SmartCrowd` could be adapted inside existing systems, since it is designed assuming current crowdsourcing infrastructure.

In summary, to design accurate and efficient crowdsourcing, `SmartCrowd` relies on a formal modeling of the **task evaluation**, **worker skill estimation**, and **task assignment** processes, considering **human factor** and **scale**.

3. CHALLENGES AND DIRECTIONS

While the opportunities foreseen in `SmartCrowd` are novel, the challenges in achieving them are exceptionally arduous. These challenges get further magnified, because of, (1) *Human factor* - which necessitates the key challenges to be modeled and solved considering unpredictability and inconsistency in worker behavior, their volatility, and asynchronous arrival and departure; (2) *Scale* - which necessitates the solutions to be incremental and tolerant to the volatility of the crowd and its size. `SmartCrowd` proposes novel indexing opportunities and reasons that human factor induced crowd-indexing provides a transparent way of achieving the objectives of `SmartCrowd` in conjunction with human factors and scale.

3.1 Human Factors

Human factors, a key distinction of `SmartCrowd`, relates to the uncertainty and non-deterministic nature of the behavior of human workers. For example, there is uncertainty regarding worker availability: workers can enter the crowdsourcing platform when they want, remain connected for as long as they like and they may or may not accept to make a contribution. In the same sense, there is uncertainty regarding the wage that workers may request: worker wage may vary from person to person, even among persons with the same profile for the system, but also wage may vary for the same person in different times, for example due to the person's workload, available time but also due to unseen factors. Finally, uncertainty also goes for skills: the efficiency with which a person completes a task cannot be considered fixed and it is rather uncertain, for example it may decline with the previous workload of the person, or it may depend on the offered wage or on the worker's motivation and personal engagement in the task.

The uncertainty stemming from the human factors does not preclude from designing a crowdsourcing solution with a global optimization target. What it does mean, however, is that, instead of fixed parameter values, `SmartCrowd` needs to study the aforementioned dimensions considering probabilities and confidence boundaries (e.g. we cannot determine the "exact wage" of a person but an approximation, with certain deviation of a central wage value), and be able to update the probabilities, as workers complete more tasks.

3.2 Who Evaluates What and How?

Tasks submitted by workers need to be evaluated for accuracy. Interestingly, the process of evaluating completed tasks is tightly coupled with acquiring each worker's contribution, which in turn helps learning worker skills. A question however is, who evaluates what and how?

A worker's contribution to a task can be evaluated through a *fully-automated and implicit* way by comparing submitted results against each other. In lieu of a known ground truth, a worker's contribution could be measured by computing the divergence of submitted contributions thus far using simple or weighted averages, majority voting, etc. More sophisticated models such as multivariate data analysis could also be used to approximate ground truth. In all cases, implicit evaluation becomes effective when the acquired aggregated data approximates the unknown ground truth. A faster, more reliable but costlier alternative is to *explicitly* designate some of the current workers as the evaluators of submitted tasks.

We envision a *hybrid* method instead; task evaluation is performed by combining system's acquired intelligence augmented with explicit human expertise. This requires complex modeling - 1) how to combine implicit and explicit evaluations together, 2) when and how to hire explicit evaluators, 3) how many explicit evaluators are required. In addition, human factors also contributes multiple new parameters such as 4) what should be the offered incentives, 5) how to model inconsistent attention and arbitrary departure of explicit evaluators, and 6) how to compute this incrementally, as workers enter and exit asynchronously.

3.3 How to Estimate Worker Skills?

Skill estimation pertains to *learning* worker skills accurately and effectively. In `SmartCrowd`, the output of task evaluation (i.e., a worker's contribution to each completed task) is used to estimate worker skills. Therefore, the first challenge is, how to identify and quantify a skill set?

For many complex tasks, some skills may be *latent*. For example, in image moderation, skills might vary for different images. In `SmartCrowd`, we envision learning such latent skills as the tasks are being executed by workers. Discovering a set of latent skills could be formulated as a structure learning problem in machine learning with the objective of uncovering a multi-layer probabilistic model. On the contrary, the problem could also be formulated as a fixed probabilistic model with the objective of learning inference from it. Unlike traditional machine learning problems where the end objective is accurate prediction, one unique requirement for `SmartCrowd` is to make these discovered skills *contextual* and interpretable by the applications.

Irrespective of the specific algorithm used to quantify worker skills, additional challenges in the model involve - 1) determining the minimal number of tasks that workers (or certain groups of workers) need to complete, until their skills can be estimated with high accuracy, considering they may not behave consistently, 2) identifying the "stopping condition" to decide whether a worker's skills have been estimated with adequate certainty or not, and 3) enabling fast and incremental computation (using worker clustering or view maintenance) of skills, as new workers arrive. In addition, human factors causes additional challenges such as identifying declination of skills (possibly due to boredom) or model how worker skill changes over time.

3.4 How to Assign Tasks to Workers?

In `SmartCrowd`, we envision that workers are assigned to tasks based on learned workers' skills and the remaining unfinished tasks. Interestingly, unlike traditional task assignment problems in project management, in `SmartCrowd` , workers' skills are unknown in the beginning, and learned skills evolve as workers engage in more tasks and subject to inconsistency and unpredictability due to human factors.

In `SmartCrowd`, we model assigning tasks to workers as a *probabilistic optimization problem*, with the objective of maximizing accuracy, or minimizing time, or optimizing both at the same time probabilistically. Furthermore, additional factors such as cost (money) could be considered.

Several related questions (or constraints) are required to be factored into this formulation as well - (1) what if a worker declines an assigned task, 2) can multiple tasks be allocated to the same worker, 3) in the case of multiple task allocation, does `SmartCrowd` suggest an ordering tasks to the worker, 4) during task assignment, does `SmartCrowd` need to assign tasks such that there are no idle workers, 5) is there an upper limit on the number of tasks that a single worker can be assigned to in one iteration? 6) how important is the system's benefit vs worker's benefit? Should the system optimize across tasks (i.e., exploit), or give newcomers opportunities (i.e, explore) to prove their skills?

3.5 Crowd-Indexing

Crowdsourcing is an adaptive process - where workers/tasks arrive asynchronously, and the system learns more about workers as they complete assigned tasks. Satisfying the key objectives of worker skill estimation, worker-to-task assignment, and task accuracy evaluation while accounting for human factors at scale, necessitates the development of efficient searching techniques. `SmartCrowd` proposes crowd-indexing to that end, where workers are organized and indexed into *groups*, and the indexes are dynamically maintained.

Interestingly, `SmartCrowd` demands new forms of indexing triggered by human factors, such as predictive skill estimation and task acceptance rate. These factors are dynamic and vary over time, as workers undertake and complete more tasks. Efficient determination of the right group of workers for collaborative tasks is a key question when optimizing cost (time and money). Similarly, selecting explicit evaluator(s) efficiently for task evaluation could benefit tremendously from index design. However, in `SmartCrowd`, we envision incremental indexing strategies, that are adaptive to this dynamic environment.

In contrast to traditional database indexing, crowd-indexing is (a) on-demand indexing where the notion of query workload is akin to tasks arriving at different rates (b) constrained indexing with different objectives such as latency, budget, worker skill diversity (c) alternate indexing as it requires to have a fall-back option (due to the uncertainty of workers accepting a task).

4. CONCLUSION

In this paper, we developed a vision for intelligent crowdsourcing and presented our framework, `SmartCrowd`. In contrast to existing systems, `SmartCrowd` promotes an iterative interaction with workers and an involvement of those workers beyond task completion (they are involved in evaluating each others' contributions), in order to adaptively learn and improve the processes of learning workers' skills and assigning tasks. Both existing (which do not require longer engagement from a volatile and mostly non-recurrent workers) and next generation crowdsourcing applications (which require longer engagement from the crowd) could benefit from our vision. As discussed in this paper, increasing intelligence in `SmartCrowd` comes with several hard challenges. `SmartCrowd` aims to be principled yet efficient in proposing the solution to those challenges.

References

[1] R. Boim, O. Greenshpan, T. Milo, S. Novgorodov, N. Polyzotis, and W. C. Tan. Asking the right questions in crowd data sourcing. In *ICDE*, pages 1261–1264, 2012.

[2] R. Cavallo and S. Jain. Efficient crowdsourcing contests. In *AAMAS*, pages 677–686, Richland, SC, 2012.

[3] A. Doan, R. Ramakrishnan, and A. Y. Halevy. Crowdsourcing systems on the World-Wide Web. *Communications of The ACM*, pages 385–396, 2011.

[4] S. Dow, A. Kulkarni, S. Klemmer, and B. Hartmann. Shepherding the crowd yields better work. In *CSCW*, 2012.

[5] C.-J. Ho and J. W. Vaughan. Online task assignment in crowdsourcing markets. In *AAAI*, 2012.

[6] S.-W. Huang and W.-T. Fu. Don't hide in the crowd!: increasing social transparency between peer workers improves crowdsourcing outcomes. In *CHI*, 2013.

[7] D. R. Karger, S. Oh, and D. Shah. Budget-optimal task allocation for reliable crowdsourcing systems. *CoRR*, abs/1110.3564, 2011.

[8] X. Liu, M. Lu, B. C. Ooi, Y. Shen, S. Wu, and M. Zhang. Cdas: a crowdsourcing data analytics system. *Proc. VLDB Endow.*, 5(10):1040–1051, June 2012.

[9] A. Parameswaran and N. Polyzotis. Answering queries using humans, algorithms and databases. In *CIDR 2011.*

[10] A. Ramesh, A. Parameswaran, H. Garcia-Molina, and N. Polyzotis. Identifying reliable workers swiftly. Technical report, Stanford University.

[11] Y. Singer and M. Mittal. Pricing mechanisms for crowdsourcing markets. WWW '13, pages 1157–1166, 2013.

Cost and Quality Trade-Offs in Crowdsourcing

Anja Gruenheid
Systems Group
ETH Zurich
anja.gruenheid@inf.ethz.ch

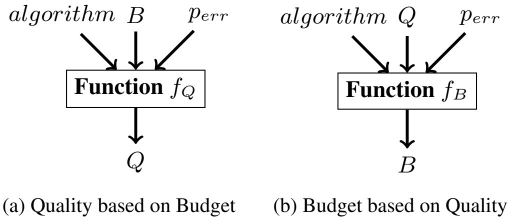
Donald Kossmann
Systems Group
ETH Zurich
donaldk@inf.ethz.ch

ABSTRACT

Algorithms for crowdsourced tasks such as entity resolution, sorting, etc. have been subject to a variety of research work. So far, all of this work has focused on one specific problem respectively. In this paper, we want to focus on the bigger picture. More specifically, we want to show how it is possible to estimate the budget or the quality of an algorithm in a crowdsourcing environment where noise is introduced through incorrect answers by crowd workers. Such estimates are complex as noise in the information set changes the behavior of established algorithms. Using two sorting algorithms, QuickSort and BubbleSort as examples, we will illustrate how algorithms handle noise, which measures can be taken to make them more robust, and how these changes to the algorithms modify the budget and quality estimates of the respective algorithm. Finally, we will present an initial idea of how such an estimation framework may look like.

1. INTRODUCTION

Quality and its influence on the output result of any algorithm using crowdsourcing has been subject to a range of research work. Research focused on topics such as increasing the worker's motivation, quality control mechanisms, or accepting that workers make mistakes and constructing fault-tolerant variations of the original algorithms [4, 15]. This work will focus on none of these quality assurance methods specifically but on assessing the inter-relationships of the crowdsourcing budget, the crowd worker error rate, and the result quality. Our goal is to define an intuition on how these parameters are interleaved, meaning a) how for example adding votes changes the result quality or b) if a certain result quality is required, how many votes are required to meet these quality constraints. These two scenarios are visualized in Figure 1. Obviously, functions f_Q and f_B are closely related for the sake of conformity, where $f_Q(B_i, p_{err}) = Q_i$ and $f_B(Q_i, p_{err}) = B_i$ holds for a specific input budget B_i and input quality Q_i. In other words, f_Q is the reverse calculation of f_B. Note that the crowd worker error rate is a given parameter in both scenarios that obviously influences the result quality and budget requirements.



(a) Quality based on Budget (b) Budget based on Quality

Figure 1: Estimation Functions

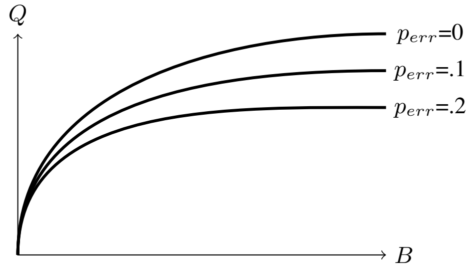


Figure 2: Cost-Quality Trade-Off

It is not the objective of this work to define how this error rate can be determined but we assume that it is a known parameter which can be estimated through worker quality tests or more advanced quality techniques [5, 9]. In addition to the crowd worker error rate and the monetary budget (resp. the required quality), the estimation function takes as input an algorithm which interprets the feedback of the crowd. Depending on this algorithm and its efficiency in a noisy environment, the estimated budget or quality may vary. For the purpose of this paper, we will assume that the algorithm that we want to estimate is a sorting algorithm but the quality and budget estimation techniques that we will present here can also be used for other algorithms in which the crowd is employed, such as entity resolution.

Independent of the algorithm, we assume that the quality of the result decreases if the noise in the crowd answers increases which is depicted in Figure 2. We will observe this behavior when examining two example sorting algorithms, QuickSort and BubbleSort [11]. For these algorithms, we will discuss quality and budget trade-offs, showing how fault-tolerance mechanisms and specific adaptations to the noisy environment can improve result quality effectively. Additionally, we will show that the result quality and budget requirements also highly depend on the applied algorithm and we present examples where algorithms that were thought to be better alternatives will produce results with lower quality.

More specifically, we will first describe, using QuickSort and

BubbleSort as example algorithms, how different algorithms handle noise with and without additional fault-tolerance mechanisms and how they compare in terms of the amount of the budget that they use and the quality of their results. From this specific example, we will then draft the challenges of any estimation framework in this kind of environment.

2. RELATED WORK

Crowdsourcing in context of database systems has been subject to a variety of research recently. One research area focuses on integrating crowdsourcing functionality efficiently into database management systems while another research focus are algorithms suitable for this kind of environment. Systems such as CrowdDB [1], Qurk [7], and Deco [10] connect traditional data storage systems with crowdsourcing platforms such as Amazon Mechanical Turk to gain additional information on queries [12]. To enable users to have similar functionality in those systems as in any comparable relational database management system, research has further focused on optimizing crowd accesses (i.e. reducing the budget spent on the crowd) when implementing algorithms such as ranking and entity resolution. For these algorithms, research has pursued two different directions: The first class of algorithms focused on reducing the budget while assuming that the crowd answers perfectly [13, 14], observing the quality of the answers by the crowd workers as an orthogonal problem. The second class of algorithms introduces the notion of fault-tolerance to be able to handle noisy answers from the crowd [2, 3, 8, 6].

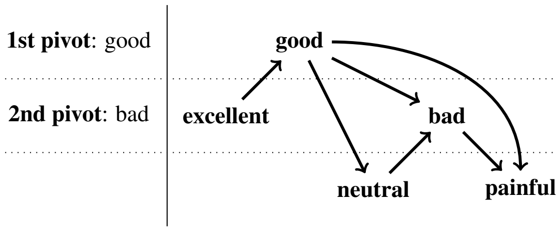
The novelty of our work is that we abstract from the actual implementation of the algorithms in that we want to describe a more general framework for the interdependencies between quality and budget in a crowdsourcing environment independent of the applied algorithm. Hence, we observe rather than construct the behavior of a set of algorithms and show through examples which behavior fits better for noisy environments.

3. IMPACT OF NOISE

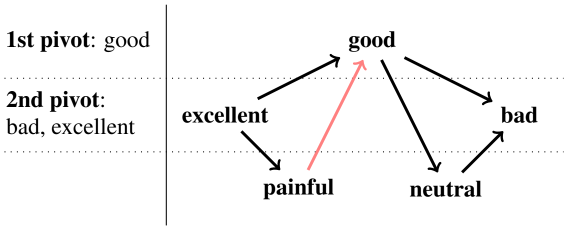
Traditionally, sorting algorithms such as QuickSort and BubbleSort have been evaluated with respect to time and space requirements. The crowdsourcing context now adds another dimension, namely result quality, as a noisy environment influences the quality and correctness of the output of these algorithms negatively. Moreover, temporal constraints become neglible as crowdsourcing itself does not necessarily need immediate response algorithms by design while space requirements can be seen as budget constraints (i.e. the number of comparisons that are required to find a solution). We will show in the following that the choice of algorithm for a certain problem heavily influences the result quality especially in noisy environments. To that purpose, we will first evaluate QuickSort in a crowdsourcing environment after which we will focus on BubbleSort as example algorithms.

3.1 QuickSort

The prerogative of QuickSort is that it can efficiently sort an input set, using $O(n\,log(n))$ comparisons on average. Thus, it is a prime candidate for sorting in a crowdsourcing environment as it allows to reduce the budget spent on the task. On the other hand, noise in the crowdsourcing answers directly propagates if QuickSort is used. An example for this observation is shown in Figure 3. Imagine a crowdsourcing task which requires the workers to order a few words according to their positivity. In the first example answer set, none of the edges (i.e. the votes of crowd workers) is wrong which leads to low budget result with perfect quality. In contrast, in the second example set the edge between 'painful' and 'good' distorts the result in that painful is now ranked higher than three other words due to the transitive propagation of this specific wrong comparison.



(a) QuickSort Without Noise



(b) QuickSort With Noise

Figure 3: QuickSort Behavior Variations

How noise affects the output of the QuickSort algorithm is dominated by the distribution of the noisy answers and whether the implementation of QuickSort used in this context is fault-tolerant. Many of the current crowdsourcing solutions do not provide fault-tolerant algorithms but assume the crowd to answer perfectly [13, 14]. In contrast, we believe that given the application of these algorithms where information is collected in crowdsourcing platforms such as Amazon Mechanical Turk, accepting that the crowd does create a certain amount of noise is only natural. Mechanisms that can make QuickSort fault-tolerant may include strategies such as using the majority of the crowd votes in order to determine an edge or to request that one answer exceeds the alternative by quorum votes. Depending on the strategy, the estimates of functions f_Q and f_B vary. For example imagine that p_{err}=.2, meaning here that 20% of the actual comparisons are wrong on average. If the majority strategy is applied and all false votes are countered by more true votes, the quality of the result can still be impeccable. On the other hand, if no fault-tolerant mechanism is used, the result quality will suffer because wrong comparisons will be propagated directly into the result. Obviously, this is the best case scenario for the majority strategy and in practice even with fault-tolerance comparisons will be propagated wrongly into the decision structure. But given the integrated quality assurance mechanism of the majority strategy, a smaller number of comparisons will be wrong which means higher result quality in the end. Thus, our first observation is that depending on the interpretation strategy used (for example the *direct* propagation of votes or the *majority* strategy explained in the previous example), the budget and the quality output changes.

Figure 4 depicts a simulated budget-quality trade-off for the two discussed strategies, the direct propagation of votes and the majority interpretation strategy. It shows that fault-tolerance increases the budget requirements for QuickSort but it also improves the result quality, i.e. the number of pair-to-pair comparisons that are confirmed by the ground truth. To better understand this observation take again the running example: If the crowd worker error rate is

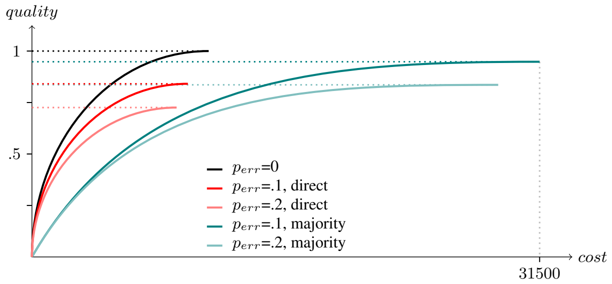


Figure 4: Fault-Tolerance Strategy Trade-Offs for QuickSort

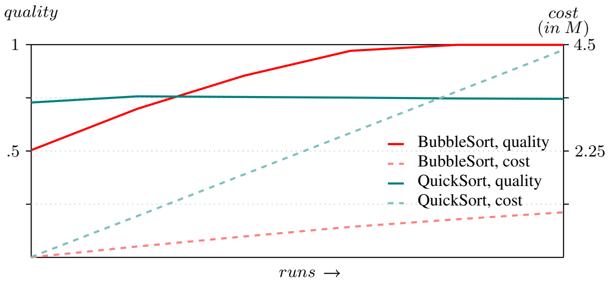


Figure 5: QuickSort vs BubbleSort, $p_{err} = .2$

p_{err}=.2, at least one edge is faulty out of the six that we need for our QuickSort solution in Figure 3. The result quality of our example is improved if the majority interpretation strategy is applied because on average, the erroneous votes are better distributed than in the direct scenario thus resulting in less erroneous comparisons as exemplified above. The figure also visualizes that applying the majority strategy leads to a higher budget as a comparison is made multiple times, thus the overall information gain is slower per unit of budget.

During the simulations a decrease in cost can be observed when the error rate increases, i.e. the cost for QuickSort with the direct strategy and without noise is approximately 11,000 while the cost decreases for $p_{err} = .1$ to 9,700. The explanation for this behavior is that in cases where the pivot element is not a central element in the final sorting, wrong answers may benefit the overall number of comparisons as the wrongly judged item is transfered into a smaller bucket and thus used less often overall for comparisons.

Overall, the main observation for QuickSort here is that if the error increases, fault-tolerance measurements can help to improve the quality of the result set even if they also mean an increase in budget in order to reach the break-even point. On the other hand, it is essential to observe that even with fault-tolerant techniques, there is no guarantee that QuickSort will provide a correct result to the problem of sorting an input dataset. Thus, if the problem statement is to find a complete ranking of the input set, QuickSort fails as a sorting algorithm in this example setting.

3.2 BubbleSort

In contrast to QuickSort, BubbleSort is considered to be suboptimal in terms of the number of comparisons it needs to find a solution as it does local comparisons rather than global comparisons and its average comparison complexity is $O(n^2)$. In a noisy environment predicates shift and the number of pure item comparisons is not the decisive factor anymore as we will show in the following. BubbleSort specifically has one characteristic that makes this algorithm suitable for noisy environments: It benefits from the **input order** of the records. Thus, if the input is already sorted, the algorithm will run in $O(n)$. This characteristic can be leveraged by running BubbleSort multiple times over the same input dataset. Due to the input sensitivity of BubbleSort, subsequent *runs* will have a decreased number of comparisons. BubbleSort in a noisy environment will further benefit from a very intuitive modification to the algorithm that is made due to the cost sensitivity of the environment: Every comparison that is requested is stored. As a result, comparing the same item twice will not result in additional cost but if a crowd worker returns the wrong answer this answer will be propagated through the dataset. Even though this kind of propagation reduces the quality of the output, the cost of iteratively improving the sorting solution is less than running BubbleSort without storage mechanisms as verified through simulations.

These modifications are minor and do not change the character of the algorithm which proceeds by doing local comparisons and 'bubbling' items to their correct spot in the ordering.

3.3 QuickSort vs. BubbleSort

When comparing QuickSort and BubbleSort and observing their behavior in a noisy setting, we can see that traditional algorithmic assumptions shift. Consider Figure 5 which shows an example sketch for this shift: Here, we draw the behavior of the direct QuickSort approach and the BubbleSort variation described above in terms of cost and quality simulated in a noisy environment with $p_{err} = .2$ and 1000 items to be sorted. As we want to obtain the optimal result quality eventually, we execute both algorithms multiple times, the number of runs in this simulation is 500. There are two main observations that can be drawn from these simulations.

First, they show that BubbleSort can improve its output result over time due to its input sensitivity while the output quality of QuickSort remains constant. Additionally, the average number of crowd comparisons requested in BubbleSort decreases over time (in our simulations up to 50%). Thus, the output quality of BubbleSort increases because if the algorithm asks for fewer input from the crowd, fewer noise is introduced into the sorting. In contrast, the level of noise that the QuickSort algorithm has to handle remains the same over multiple runs. The second observation that can be drawn from these simulations is that even if cost reduction techniques are used, BubbleSort is in this setup 26 times more expensive than QuickSort when achieving the same quality (i.e. at the quality break-even point). Thus, even though the increase in cost is lower, the necessity of repeating the algorithm multiple times increases the overall cost for BubbleSort in the end.

The example above shows that some algorithms like BubbleSort have a natural robustness that makes them more suitable in noisy environments. Thus, we think that established algorithms that outperform others according to traditional complexity theory need to be looked upon from a new angle in noisy environments as an alternative or as a component in hybrid setups.

4. QUALITY & BUDGET ESTIMATES

In order to realize an estimation framework, we propose as first step to establish a modified complexity theory that is noise-aware. That is, in traditional complexity theory the number of comparisons per algorithm determines the usefulness of that specific algorithm. As we have shown previously, traditional assumptions may not hold in the crowdsourcing environment due to noise in the information set. Thus, algorithms need to be evaluated taking into consideration their behavior in noisy environments. This evaluation can then be used to determine which algorithm is most suitable to solve a certain problem. As a result, we propose to define a new model for the complexity of an algorithm that is dependent on a) the number of records in the input set (analogous to traditional complexity theory) b) and the amount of noise that this algorithm will encounter.

If the complexity of an algorithm is known, we can then use it to form estimates of the required budget given certain quality constraints or estimates that approximate the result quality given certain budget constraints for an algorithm as shown initially in Figure 1. Note that this notion of complexity is not restricted to sorting algorithms but it can be used for evaluating a variety of algorithms such as entity resolution, finding the maximum etc. To be able to take input parameters such as a required quality or available budget, traditional algorithms need to be adjusted. Modifying these algorithms is necessary because current algorithms take an input dataset after which they are executed to return a result with a certain quality for a certain budget independent of the crowd worker error rate. How budget and quality are distributed is subject to the specific execution run and resemble points on a curve similar to those depicted in Figure 5 where the curve varies according to the level of noise in the information set and the chosen algorithm.

In our opinion, algorithms need to be conscious of input constraints to be able to compare two algorithms in a noisy environment. Going back to our comparison of QuickSort and BubbleSort, we can determine that according to the characteristics of the algorithms, QuickSort with fault-tolerance can result in good quality results in low noise environments while BubbleSort can be more adequate in environments where the crowd answers wrongly more often. But in order to be able to define the actual trade-off, we need to compare how the algorithms react in scenarios with fixed error rate and budget or quality. Thus, we want to be able to define the cost-quality function curves instead of points on the curve that can be derived from current algorithm variations that are not constraint-aware as mentioned previously.

If these curves are known, we can use them for our estimation framework in that they define the complexity of an algorithm dependent on the crowd worker error rate where we can vary budget and quality constraints in order to find a suitable estimated output. A simple two-step estimation process may then look as follows:

1. **Initial Estimate** - Given the properties of the chosen algorithm, i.e. its complexity in this kind of noisy environment, decide upon an initial estimate.
2. **Adjust Estimate** - Using the knowledge of the input quality respectively budget parameter, adjust the estimate as to meet/exhaust the parameter. An example measure is to allocate more budget for multiple runs of the same algorithm or for fault-tolerant mechanisms.

Obviously, this process is only a rough sketch of the functionality of such an estimation framework. More specific implementation and design details of these two steps are subject to future work as is the specification of the complexity theory that enables us to give accurate estimates in noisy environments.

5. CONCLUSION

In this work, we have presented several observations made for quality and cost trade-offs in the still novel crowdsourcing environment. Using QuickSort and BubbleSort as example algorithms, we have shown that any estimation function that wants to determine the expected output of that algorithm has to take into consideration that different algorithms behave differently when confronted with noise. Their behavior not only changes in terms of budget requirements but they also vary in their robustness in noisy environments. We have presented techniques such as fault-tolerant interpretation strategies and increasing the number of executions of the algorithms that influence the result estimation and choice of algorithm when taken into consideration. Last, we have shown that traditional beliefs such that QuickSort is better than BubbleSort due to its lower number of comparisons are questioned in a noisy environment because even though QuickSort is requires a smaller budget, BubbleSort may provide better quality results due to its ingrained second chance mechanism and its ability to leverage information over multiple runs.

In the future, we plan to specify our sketched estimation framework and to define exactly how noise can be evaluated in such an environment. Additionally, we want to determine whether it is possible to formally analyze the trade-off between cost and quality for algorithms that are traditionally integrated into relational database management systems. Knowing the complexity of these algorithms will help us to determine which of them are viable options for a database system that incorporates crowdsourcing as a way to obtain information.

6. REFERENCES

[1] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. Crowddb: answering queries with crowdsourcing. In *SIGMOD Conference*, pages 61–72, 2011.

[2] A. Gruenheid, D. Kossmann, S. Ramesh, and F. Widmer. Crowdsourcing entity resolution: When is a=b? Technical report, ETH Zurich, 2012.

[3] S. Guo, A. Parameswaran, and H. Garcia-Molina. So Who Won? Dynamic Max Discovery with the Crowd. In *Proceedings SIGMOD*, 2012. to appear.

[4] A. Kittur, J. V. Nickerson, M. Bernstein, E. Gerber, A. Shaw, J. Zimmerman, M. Lease, and J. Horton. The future of crowd work. In *Proceedings of the 2013 conference on Computer supported cooperative work*, CSCW '13, pages 1301–1318, New York, NY, USA, 2013. ACM.

[5] M. Lease. On quality control and machine learning in crowdsourcing. In *Human Computation*, 2011.

[6] A. Marcus, D. Karger, S. Madden, R. Miller, and S. Oh. Counting with the crowd. In *Proceedings of the 39th international conference on Very Large Data Bases*, PVLDB'13, pages 109–120. VLDB Endowment, 2013.

[7] A. Marcus, E. Wu, D. R. Karger, S. Madden, and R. C. Miller. Demonstration of qurk: a query processor for humanoperators. In *SIGMOD Conference*, pages 1315–1318, 2011.

[8] A. Marcus, E. Wu, D. R. Karger, S. Madden, and R. C. Miller. Human-powered sorts and joins. *PVLDB*, 5(1):13–24, 2011.

[9] D. Oleson, A. Sorokin, G. P. Laughlin, V. Hester, J. Le, and L. Biewald. Programmatic gold: Targeted and scalable quality assurance in crowdsourcing. In *Human Computation*, 2011.

[10] A. Parameswaran, H. Park, H. Garcia-Molina, N. Polyzotis, and J. Widom. Deco: Declarative crowdsourcing. Infolab Technical Report, Stanford University, November 2011.

[11] R. Sedgewick and K. Wayne. *Algorithms, 4th Edition*. Addison-Wesley, 2011.

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

[13] J. Wang, T. Kraska, M. J. Franklin, and J. Feng. Crowder: Crowdsourcing entity resolution. *PVLDB*, 5(11):1483–1494, 2012.

[14] S. E. Whang, P. Lofgren, and H. Garcia-Molina. Question selection for crowd entity resolution. Technical report, Stanford University.

[15] X. Wu, W. Fan, and Y. Yu. Sembler: Ensembling crowd sequential labeling for improved quality. In *AAAI*, 2012.

Data In Context: Aiding News Consumers while Taming Dataspaces

Adam Marcus[*] , Eugene Wu, Sam Madden
MIT CSAIL
{marcua, sirrice, madden}@csail.mit.edu

...were it left to me to decide whether we should have a government without newspapers, or newspapers without a government, I should not hesitate a moment to prefer the latter.
— Thomas Jefferson

ABSTRACT

We present MuckRaker, a tool that provides news consumers with datasets and visualizations that contextualize facts and figures in the articles they read. MuckRaker takes advantage of data integration techniques to identify matching datasets, and makes use of data and schema extraction algorithms to identify data points of interest in articles. It presents the output of these algorithms to users requesting additional context, and allows users to further refine these outputs. In doing so, MuckRaker creates a synergistic relationship between news consumers and the database research community, providing training data to improve existing algorithms, and a grand challenge for the next generation of dataspace management research.

1. INTRODUCTION

One of the basic mechanisms through which end-users consume and interact with data is by reading a news source. Many articles are based on one or a few data points, be it the earnings of a company, new unemployments numbers for a country, or the number of people at a political rally. Infographics compactly present a larger set of points, typically through aggregate statistics grouped temporally or by category. Investigative efforts often uncover and join several datasets to deliver a story. In business reporting, the data is even more apparent: companies like Reuters and Bloomberg are successful in part because they generate and serve enormous amounts of data. In each of these examples, the end product—an article or a graphic—is conceptually a view over a dataset. For example, when reporting the earnings of a specific company, the article presents a view of a dataset of all company earnings in the same quarter, or earnings of the company during previous quarters.

In this light, an article's background includes tuples outside of the existing view; access to extra information would allow the reader to better understand the article's context. However, articles may miss key contextual cues for many reasons: 1) A lack of space or time, as is common in minute-by-minute reporting 2) The article is a segment in a multi-part series, 3) The reader doesn't have the assumed background knowledge, 4) A newsroom is resources-limited and can not do additional analysis in-house, 5) The writer's agenda is better served through the lack of context, or 6) The context is not materialized in a convenient place (e.g., there is no readily accessible table of historical earnings). In some cases, the missing data is often accessible (e.g, on Wikipedia), and with enough effort, an enterprising reader can usually analyze or visualize it themselves. Ideally, all news consumers would have tools to simplify this task.

Many database research results could aid readers, particularly those related to *dataspace management*. Matching records from articles with those in a relation is an entity resolution problem; aggregating contextual information from multiple sources is a schema matching and data integration problem; searching for the missing data is a deep web problem; extracting relations from web-pages and text is solved by projects like TextRunner [9] and Webtables [3]. While these efforts have pushed the limits of automated solutions, recent human-assisted approaches present new opportunities. Existing automated algorithms can be semi-automated by asking humans to vet algorithmic results and iteratively improve the algorithms over time. We believe news is an ideal match for this problem.

We believe that, given the promise of contextualized articles, readers would be willing to answer a small number of simple questions. For example, we might ask a user to highlight the data in question, identify related datasets, ensure the data in an article properly maps to a dataset, or to select a visualization. Fortuitously, each small task generates inputs for semi-automated dataspace management algorithms. Readers can also benefit from previous users' answers and view previously generated context without additional effort.

We view data in context as a grand challenge in dataspace management. If we design algorithms that are good enough to be guided by an average reader and provide the context they lack, then both society and the database community benefit. In addition, journalists can use the same tools to proactively identify relevant datasets or visualizations.

We envision a proof-of-concept user interface for an article contextualization service called MuckRaker. It serves as a front-end to the numerous database problems mentioned above. We have developed it as a bookmarklet for web browsers that allows users to select a data point on a webpage, answer a few questions about its context and origin, and see a visualization of a dataset that contextualizes their

---

[*] Eugene and Adam contributed to this paper equally.

reading experience. In this paper we outline:

1. A user interface for interacting with data items on the web that contextualizes them,
2. A study of the difficult dataspace management problems average news consumers can help solve, and
3. A collection of the challenges posed by data embedded in articles that span the web.

2. TALE OF TWO COMMUNITIES

The problem of contextualizing data lies at the intersection of two communities—news consumers and the database community—and can benefit both. As a social endeavor, we believe it encourages the general population to interact with data. MuckRaker not only helps curious readers better understand the news, but can help users spot biased attempts to generalize a single outlier data value as the norm. It serves the data management systems by helping clean, normalize, and aggregate data that consumers care about.

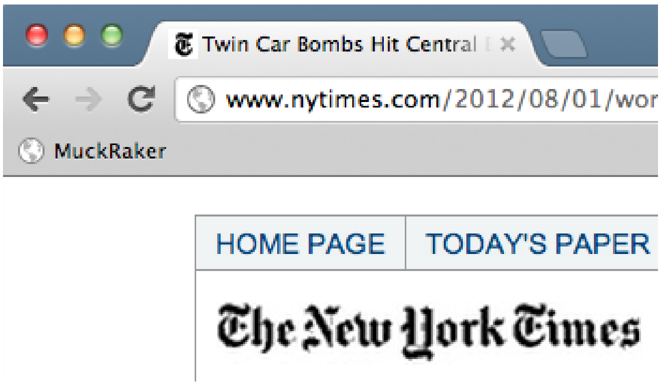
The unfettered MuckRaker vision encompasses several open problems facing the database community. We believe variants of the problem are tractable and can be solved with careful application of existing approaches. In the rest of this section, we illustrate how a constrained instance of this problem can be solved for an example article that centers on a car bombing in Iraq[1]. We first describe MuckRaker from a user's perspective, then explore a viable system implementation, and finally explore extensions to increase MuckRaker's utility and introduce interesting research problems.

2.1 A Tool for News Consumers

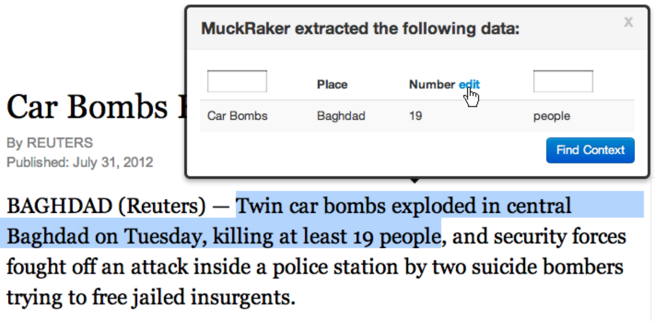
Suppose a user reads an article about a car bomb in Iraq and wants to know the scale of similar attacks. She clicks the MuckRaker bookmarklet (Figure 1a), which asks her to select the region of text that she would like contextualized. The user highlights the sentence "Twin car bombs exploded in central Baghdad on Tuesday, killing at least 19 people." MuckRaker extracts key entities and values around the highlighted region, and presents the data to the user (Figure 1b). MuckRaker could not identify attribute names for the first and last column, and prompts the user to fill them in. The attribute name in the third column is not precise enough, so the user can click it to edit the name. When the user is satisfied, she clicks "Find Context."

MuckRaker receives a request containing the highlights and extracted data. MuckRaker finds contextual information about the article using article type classification, fact, entity and numerical extraction, and other natural language processing techniques. This information, weighted by its distance to the selected text, is fed to algorithms described in Section 2.2. The information and the tables ranked most relevant by the algorithms are presented to the user (Figure 1c). The user can select the desired table, or update this information and re-execute the matching algorithms.

In this example, the first table is a list of mass bombings[2] and the second is of civilian casualties during the first two weeks of the Iraq war[3]. If a matching row exists, MuckRaker will attempt to populate empty fields with new data values.



(a) MuckRaker bookmarklet.



MuckRaker extracted the following data:

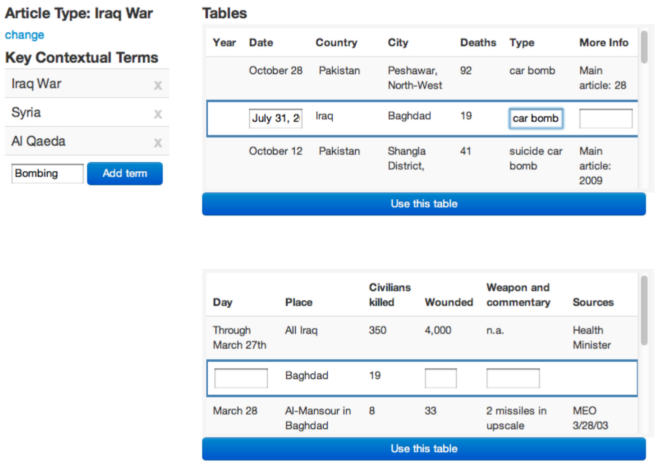
	Place	Number edit	
Car Bombs	Baghdad	19	people

Car Bombs

By REUTERS
Published: July 31, 2012

BAGHDAD (Reuters) — Twin car bombs exploded in central Baghdad on Tuesday, killing at least 19 people, and security forces fought off an attack inside a police station by two suicide bombers trying to free jailed insurgents.

(b) User selects data in text.



Article Type: Iraq War
change

Key Contextual Terms

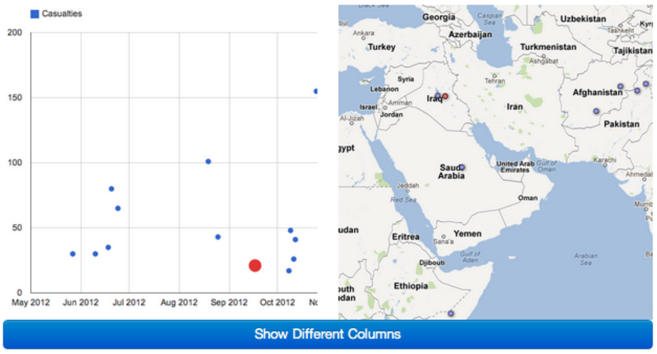
- Iraq War
- Syria
- Al Qaeda

Tables

Year	Date	Country	City	Deaths	Type	More Info
	October 28	Pakistan	Peshawar, North-West	92	car bomb	Main article: 28
	July 31, 2	Iraq	Baghdad	19	car bomb	
	October 12	Pakistan	Shangla District,	41	suicide car bomb	Main article: 2009

Day	Place	Civilians killed	Wounded	Weapon and commentary	Sources
Through March 27th	All Iraq	350	4,000	n.a.	Health Minister
	Baghdad	19			
March 28	Al-Mansour in Baghdad	8	33	2 missiles in upscale	MEO 3/28/03

(c) Tables and article context.



(d) Visualizations.

1: The MuckRaker user interface.

---

[1] http://www.nytimes.com/2012/08/01/world/middleeast/twin-car-bombs-hit-central-baghdad.html

[2] http://cursor.org/stories/iraq.html

[3] http://en.wikipedia.org/wiki/List_of_mass_car_bombings

In this case, neither table contains the user-selected data point, so MuckRaker inserts the data into both tables, and fills as many of the matched fields as possible. The row is highlighted for the user so that she can optionally fill in missing fields. She is interested in mass bombings, so selects the first table, corrects the date field, and fills in the **type** column with "car bomb," which is autocompleted. When she clicks "Use this table," the updated row is sent to MuckRaker, and she is shown condidate visualizations.

MuckRaker selects an initial visualization using the selected dataset. Previous users commonly created timeseries and maps, and Figure 1d shows these top plots: deaths by date and deaths by location. It highlights the new data point to identify contextual data (the red points). The user can specify that she is interested in different columns and construct her own chart through a chart builder, where she can specify chart types, axes, filters, and facets. MuckRaker stores the visualization configuration that the user picked, and uses it to bootstrap future requests.

2.2 Database Community

We believe that data in context can be implemented at a basic but useful level using existing technology. We hope the database community can use the core system as a starting point for numerous research problems. In the rest of this section, we sketch a basic implementation, and then describe how user interactions can both improve existing algorithms with training data, and introduce new challenges that must be addressed by future data contextualizing systems.

2.2.1 Core Implementation

The MuckRaker interface is implemented as a browser bookmarklet (Javascript that operates on the current web page). We assume that we start with a reasonable collection of tables that are clean, deduplicated, and complete. In addition, table metadata includes the surrounding text (e.g., Wikipedia article text). This data can be bootstrapped from sites like Wikipedia and groups such as the world bank, or through techniques like those in work by Cafarella et al [3].

User-selected text is sent to a backend server, which extracts numbers and their units (e.g., $, miles), dates, and known entities. Entity extraction algorithms such as KnowItAll [6] can identify the key nouns and topics. We can ask the user to clean and highlight the extracted values.

The set of possible tables is filtered by clustering the article text with table metadata. For example, an article related to the Iraq war will match tables extracted from Wikipedia articles about Iraq. We can further reduce and rank tables by canonicallizing user-specified attribute names using techniques similar to those used by Das Sarma et al [5] to perform schema ranking. A final ranking comes from comparing the table values with those in the user-extracted record.

2.2.2 Research Problems

The user interaction provides a number of strong signals that make for interesting research problems. We describe some problems in the context of interactions in the extraction, selection, and visualization phases of the user workflow.

Data Extraction and Integration. The user-selected text explicitly defines record boundaries. The collection of all user highlights can be used to train classifiers to detect strings that contain records, and focus the analysis that data record extractors like TextRunner [9] need to perform.

The user can facilitate record extraction, but values may be named, formatted, or scaled inconsistently with existing tables. With a tighter human-in-the-loop training cycle, we have more hope for improving such extraction anomalies.

Another classification challenge lies in identifying the type of context that the user interested in. She may want to see an IBM earnings report in the context of historical earnings instead of similar technology companies (select the appropriate attributes). Alternatively, a European reader may prefer to see European companies rather than American companies (select the best records). Subdividing context automatically according to user preferences is a key challenge.

Structured Search. Das Sarma et al. recently studied related table search [5], where a user specifies a table and the search engine identifies others that either extend the table horizontally or vertically. MuckRaker requires a similar search, but uses partial examples extracted from article text.

In addition, identifying the table is not enough. To be useful, tables must be transformed (e.g., currency conversion), projected, filtered (e.g., identify small number of representative rows), and aggregated (e.g., aggregate county statistics to report state granularity statistics). Learning these steps is another research challenge.

Visualization. Automated Visualization selection is difficult because it is both a dimensionality reduction problem and a design problem. Historical earnings are best plotted as a time series, while the comparative earnings of similar companies is better represented with a bar chart. A human in the loop would better assist and train these decisions.

MuckRaker can gather large volumes of training data of user-perferred columns based on the final visualizations that the user selects. One project that has facilitated user-driven visualization construction is the ManyEyes project [8], and we can use its findings as a basis for our design.

Data Management. The projects that have most closely integrated these individual research problems into a larger data integration and search system are TextRunner [9] and WebTables [3]. To the extent that these projects have been evaluated by how much deep web data can be added to web search indices, we think that the grand challenge raised by contextualizing data serves as a higher bar. While indexing websites by the data they store is useful, being able to retrieve datasets that are relevant to a user's current context would be even more powerful. The WebTables authors realize this as well: Fusion Tables [7] surfaces the data found in web tables and other datasets directly in search results, suggesting that search-based structured data retrieval is a meaningful measure of the effectiveness of these techniques.

3. GENERALIZING MUCKRAKER

In the interface described above, all user actions succeeded: the user found the correct dataset, the new record was reasonably extracted, there were no duplicate records in the table, and roughly one relevant record was extracted from the article. We now consider more challenging cases, and how the user interface can be augmented to handle them.

No matching datasets. Consider a situation where the user highlights some text and clicks "Search," no useful datasets are returned. In these situations, the user can utilize the search bar in Figure 1c to enter her own keywords, and potentially find the dataset. Failing that, she can click on "I can't find a dataset," and be prompted to either: 1)

Point at a webpage containing the dataset, or 2) Specify column headings (the schema) and row that can be extracted from this document in a spreadsheet interface. In scenario 1, an automated extractor can present the user with the newly extracted dataset, and in scenario 2, MuckRaker can search again for a matching dataset with the given schema and data point. Should this final search fail, the user will be invited to add more entries to the spreadsheet.

Incorrect extraction. In situations where a dataset is correctly identified but records are extracted incorrectly, the user can edit the row in the familiar spreadsheet interface of Figure 1c. It is possible that a user will incorrectly add field values, but MuckRaker aggregates multiple user corrections before trusting any one user's input.

Duplicate data. Duplicate rows within a table can arise if multiple users submit different articles about the same event. We can handle these by calculating a similarity measure between rows. For any newly added row that is above some threshold similarity to an existing row, we can ask a user to verify that the user indeed means to add a new data point. Data duplicated across tables requires more care. We wish to know when a table should be merged with another table, which might happen when enough rows between two tables are similar. In this situation, we can ask a user during dataset search (Figure 1c) whether the table they selected is the same as another one. If the user indicates that it is, they are then presented with the columns of both tables aligned by a schema mapping algorithm, and invited to re-arrange the mapping as they see fit. The system can merge two tables if enough users mark them as merge candidates.

Article-embedded datasets. So far, our user has highlighted a sentence that roughly translates to a single record in a table. It is often the case that an article discusses more than one data point. For example, an article that describes a trend essentially embeds multiple points from a timeseries into a dataset. Alternatively, an article summarizing a study that compares multiple groups of people would embed data about each group. Summarizing all of the extracted points in a table might be cumbersome for the user. It might be simpler to summarize the extacted data in a visualization, allowing the user to drag the points of a timeseries to match a trend, or move the bars in a bar graph to represent the relative differences between groups.

Uncertain facts. It is often the case that the news covers facts that contradict one-other (e.g., "Prior link between cancer and fruit juice challenged in latest research"). Other facts might simply expire over time. For example, records that refer to "The President, aged 51" refer to a different president or an incorrect age depending on the date of the article. A user overseeing the data extraction that knew good schema design practices (e.g., storing The President's date of birth rather than an age) could have avoided some of these issues, but MuckRaker does not leave schema design to expert database designers. To handle these types of expiration and uncertainty, attaching a source and date to extracted data may help, as would periodically asking users whether certain records are still valid in a table.

4. CONCLUSION

The core contribution of MuckRaker is to utilize a mixed-initiative interface to improve dataspace management operations as a byproduct of contextualizing the news. It would be interesting to see where else the insertion of a lightweight user interface can act as a boon to database research while benefitting another community.

There have been other calls to arms in the database community to assist the journalism process. Most prominently, Cohen et al. outlined many ways in which computational journalism can be aided by database research in areas such as fact checking and hypothesis finding [4]. The PANDA project [1] aims to provide centralized dataset storage and search functionality within newsrooms. DataPress makes it easier to embed structured data and visualizations into blog posts [2]. MuckRaker approaches the journalism-data interface from a different perspective: it seeks to aid news consumers in situations where the Journalism process has left them with an incomplete picture of the world. It can also help journalists and editors preempt this problem by helping them find contextualizing datasets and visualizations.

A key question in designing the MuckRaker experience is whether the interface we are designing is lightweight enough, or whether we are asking for too much from any one user. In exchange for context behind an article, we believe users are willing to answer a few small questions, mostly through point-and-click interfaces. If it turns out that we are asking too much from each user, however, we can design interfaces that load-balance the data integration, extraction, and visualization tasks across users, especially in scenarios where multiple users are reading the same article.

In presenting MuckRaker, we hope to bridge the gap between end-users and deep data exploration. We hope that the database community is excited to improve on its algorithms with help from the average news consumer.

5. REFERENCES

[1] The PANDA project, August 2012. `http://pandaproject.net/`.

[2] E. Benson, A. Marcus, F. Howahl, and D. Karger. Talking about data: Sharing richly structured information through blogs and wikis. In *ISWC*. 2010.

[3] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. WebTables: exploring the power of tables on the web. *Proc. VLDB Endow.*, 2008.

[4] S. Cohen, C. Li, J. Yang, and C. Yu. Computational journalism: A call to arms to database researchers. In *CIDR*, 2011.

[5] A. Das Sarma, L. Fang, N. Gupta, A. Halevy, H. Lee, F. Wu, R. Xin, and C. Yu. Finding related tables. In *SIGMOD*, 2012.

[6] O. Etzioni, M. Cafarella, D. Downey, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Unsupervised named-entity extraction from the web: an experimental study. *Artif. Intell.*, 165(1):91–134, June 2005.

[7] H. Gonzalez, A. Y. Halevy, C. S. Jensen, A. Langen, J. Madhavan, R. Shapley, W. Shen, and J. Goldberg-Kidon. Google fusion tables: web-centered data management and collaboration. In *SIGMOD*, 2010.

[8] F. B. Viégas, M. Wattenberg, F. van Ham, J. Kriss, and M. M. McKeon. ManyEyes: a site for visualization at internet scale. *IEEE Trans. Vis. Comput. Graph.*, 2007.

[9] A. Yates, M. Cafarella, M. Banko, O. Etzioni, M. Broadhead, and S. Soderland. TextRunner: open information extraction on the web. In *ACL*, 2007.