Relay: A Workflow for Efficient Crowdsourced List-Making

ABSTRACT

Tasks like planning a trip, generating sales leads, and doing product research often involve creating detailed and lengthy lists from the web — called *enumeration queries*. However, these tasks are often tedious, as they usually involve browsing multiple sources (e.g., websites) for items that match a specific criteria for inclusion on the list. Crowdsourcing enables the delegation of such work to an on-demand workforce able to scale depending on the size of the task, but ensuring response diversity and reducing duplication of worker effort remains an open problem. Traditional microtask workflows for dividing labor among crowd workers fail to account for the diversity and non-uniformity of website structures that are often encountered while searching for list items. To address these problems, we propose a workflow for enumeration tasks —called *Source-Oriented Enumeration (SOE)*— that mitigates duplicative effort by ensuring that workers visit different sources, coordinates the hand-off of longer sources, and leverages auto-suggest to connect new entries with differently-entered versions of existing items. SOE yields twice as many unique results using the same amount of total worker time as is needed by existing approaches.

Author Keywords

Crowdsourcing, task decomposition, list enumeration

INTRODUCTION

The ability to generate lists of items (e.g., of products, references, events) from natural language queries would allow end users to hand off a time-consuming information retrieval task, freeing them to focus on their use for the information itself, e.g., comparing and contrasting items, or getting a broad overview of the set. Unfortunately, understanding the nuance behind a natural language description and grounding that understanding in individual items or entries available from open-ended knowledgebases like the Web is beyond the capability of automated approaches. Human computation systems have so far been predominantly focused on transforming, summarizing, or augmenting existing data sets [2, 11, 23, 24]. An emerging frontier concerns the problem of enumerating latent datasets of yet-to-be-determined size [19]. This may include gathering existing facts [5, 26] or systematically generating new ideas [3, 10]. Consider the following scenarios:

- Susan is planning a work trip to New York City and would like to know what tourist attractions she can see in her free time. She describes the types of attractions she's interested in to Relay, and a moment later, gets a list of options back.
- Ethan wants to start a new diet he recently heard about, but is unsure of its safety. He turns to Relay to gather reputable articles discussing the safety of the diet to help him decide.

CHI'16, May 07–12, 2016, San Jose, CA, USA

© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 123-4567-24-567/08/06...$15.00

DOI: http://dx.doi.org/10.475/123_4

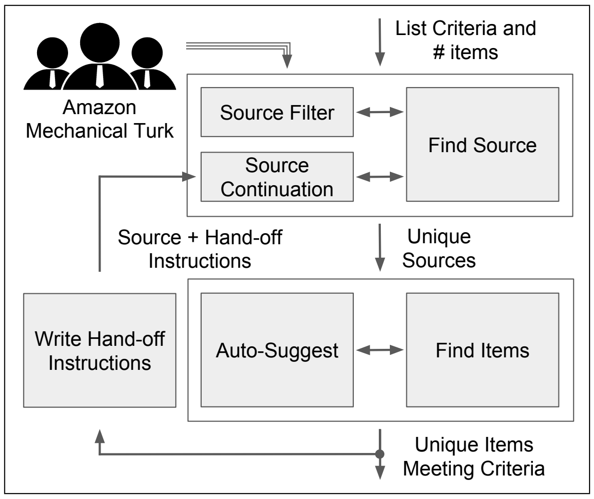
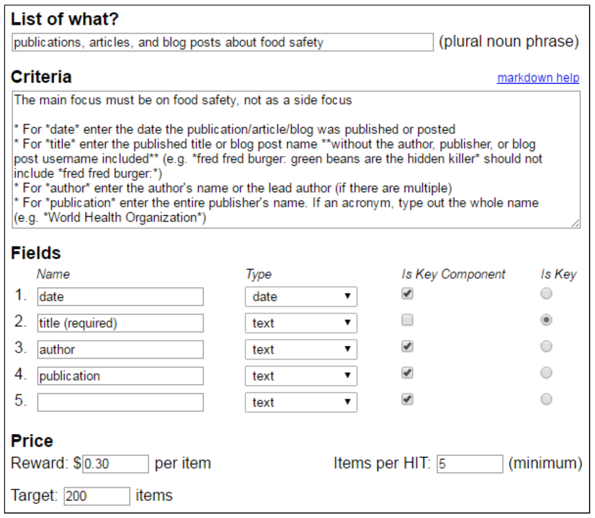

List Criteria and # items
Amazon Mechanical Turk
Source Filter
Find Source
Source Continuation
Source + Hand-off Instructions
Unique Sources
Write Hand-off Instructions
Auto-Suggest
Find Items
Unique Items Meeting Criteria

Figure 1. Relay demonstrates how uniqueness and comprehensiveness can be achieved proactively through a workflow called *Source-Oriented Enumeration (SOE)*. SOE keeps track of the sources workers use to find list items, ensures workers visit different sources, and coordinates the hand-off of longer sources to allow for their full extraction. Additionally, SOE leverages auto-suggest to connect new entries with differently spelled versions of existing items.

In each of these cases, the total number of possible list items is not known in advance. Furthermore, creativity and knowledge are needed to find a good source of list items. This may include checking various web sites, using different search terms, applying multiple filters, etc. in order to have the best chance of finding items.

The on-demand nature of the crowd is, in some respects, especially amenable to the task because workers can be recruited to match the expected list length. However, the use of crowds introduces new constraints, such as uncertainty in terms of when workers are available and how long they will participate in a task. A naïve solution to the problem would be to post a task asking each worker to enter a certain number of list items. Then, once the list is complete, the items could be deduped using crowd-based techniques for identifying matching entries [8, 30]. The limitations of this approach are straight forward. Since each worker would be approaching the problem from the beginning, they would likely start by doing many of the same searches and checking many of the same sites – leading to many duplicate results and thus wasted effort.

Instead of leaving deduplication as a post hoc process, we propose proactively deduplicating by coordinating workers to visit a variety of list-item sources. We accomplish this task


List of what?

publications, articles, and blog posts about food safety (plural noun phrase)

Criteria markdown help

```
The main focus must be on food safety, not as a side focus

* For *date* enter the date the publication/article/blog was published or posted
* For *title* enter the published title or blog post name **without the author, publisher, or blog
post username included** (e.g. *fred fred burger: green beans are the hidden killer* should not
include *fred fred burger:*)
* For *author* enter the author's name or the lead author (if there are multiple)
* For *publication* enter the entire publisher's name. If an acronym, type out the whole name
(e.g. *World Health Organization*)
```

Fields

	Name	Type	Is Key Component	Is Key
1.	date	date	✓	○
2.	title (required)	text	☐	●
3.	author	text	✓	○
4.	publication	text	✓	○
5.		text	✓	○

Price

Reward: $0.30 per item Items per HIT: 5 (minimum)

Target: 200 items

Figure 2. Relay job specification interface

by orienting HITs around sources of list items first, then by the list items themselves. Our approach draws boundaries between sources when they would otherwise be unclear (due to varying website structures) by enabling workers to define them according to where items are found —whether they are the different parts of a website, different websites, or even within search results. This decomposition strategy enables workers to complete as much or little work as they wish, be paid per item that they were able to contribute, and allow for future workers to pick up where they left off.

Even if every worker checked different sources there would still be duplication due to the same item being listed among multiple of them. Variation in spelling and synonyms makes eliminating the duplication after the items are entered a computational challenge in itself. Rather than resolving them after the fact, we propose making use of worker's knowledge of their items to deduplicate at the time of entry.

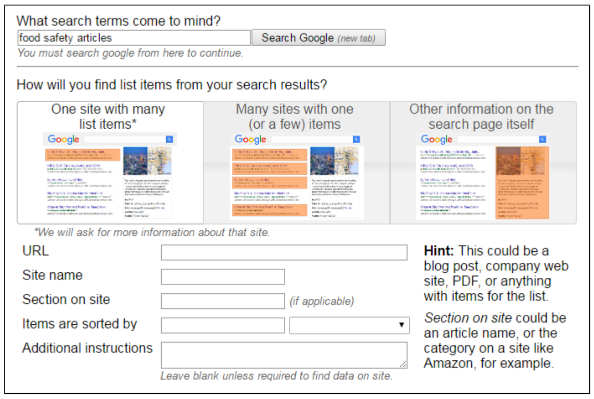
We created Relay to embody these ideas into an interface for coordinating paid crowd workers on Amazon Mechanical Turk to create lists. We are interested in building lists of items that are of factual nature. Creative generation of subjective lists (e.g., "things you can do with a brick") is out of the scope of this paper.

The key contributions of this paper are:

- A workflow for the list enumeration task called *Source-Oriented Enumeration (SOE)* that controls the sources workers visit, leverages auto-suggest clicks to reduce duplication in the final output (while ensuring fair pay to workers), and enables fruitful (long) information sources to be fully leveraged via iterative extraction.
- Relay, a system that implements SOE in order to evaluate and illustrate the design details needed to make it work.
- An evaluation measuring Relay's effectiveness at ensuring comprehensiveness and uniqueness of list items.

RELATED WORK

Our work builds on prior work in crowdsourced enumeration queries and other crowdsourced web searching tasks. Enumeration queries were first established as a means to overcome the closed-world assumption that exists in relational databases (i.e., the assumption that information not in the database does not exist) [5]. CrowdDB was the first system to use the crowd for this purpose, using worker's web searching skills to find items fitting a criteria to be added to the database. At the same time, the system Qurk was introduced to efficiently use crowd workers for other database operations like filters and joins [15]. Later work by Trushkowsky et al. further explored challenges associated with not knowing the size of queries ahead of time by creating algorithms to predict them based on the rate at which crowd workers find unique items [26]. These works have found success in using crowd workers to overcome the challenges associated with collecting data from external sources, but the workflows needed to coordinate workers across these sources remain non-existent.


What search terms come to mind?

food safety articles Search Google *(new tab)*

You must search google from here to continue.

How will you find list items from your search results?

One site with many list items*
Google
Many sites with one (or a few) items
Google
Other information on the search page itself
Google

**We will ask for more information about that site.*

URL	
Site name	
Section on site	*(if applicable)*
Items are sorted by	
Additional instructions	*Leave blank unless required to find data on site.*

Hint: This could be a blog post, company web site, PDF, or anything with items for the list.

Section on site could be an article name, or the category on a site like Amazon, for example.

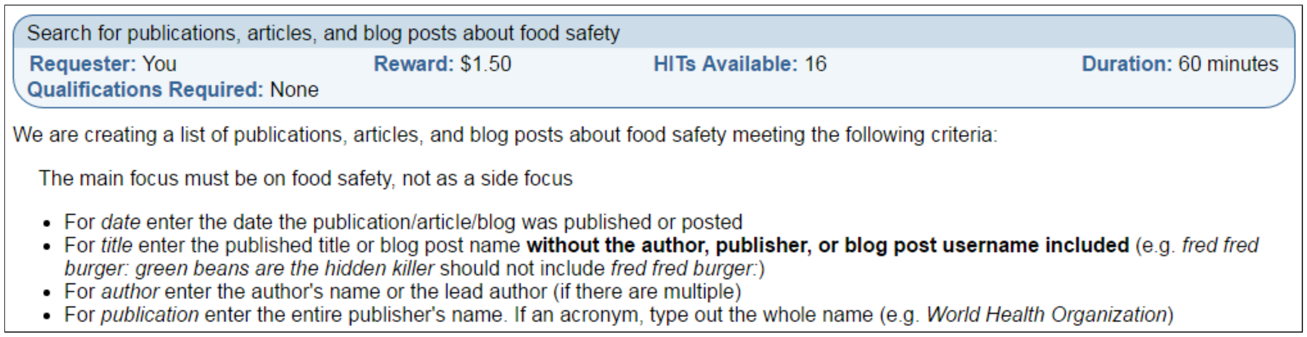
Figure 3. Relay prompt for search result information

Entity Resolution

Going to external sources for data opens up the possibility of entering duplicate entries, motivating much work to prevent or connect these items. In the context of relational databases, many have proposed using computational or crowd-based techniques as a deduplication post-processing step to data entry [4, 6, 30, 31]. These methods, however, can be costly and the use of people to collect data introduces the possibility of preventing duplication in the first place. Motivated by this idea, Trushkowsky et al. proposed a "negative suggest" that prevents crowd workers from entering list items that match items already collected [27]. While this method leads to far less duplication, no method exists to assist workers in finding a website that has not already been exhausted and the question of how to incentivize workers to continue searching after the obvious set of list items has been added remains unanswered.

Crowdsourced Filtering and Data Completion

In parallel to the enumeration task, the crowd has been used to overcome limitations in other data manipulation tasks such as filtering and completion [18, 21]. For example, Parameswaran et al.'s DataSift system was able to complete queries that contain non-textual fragments to filter image, video, and product search results [20]. They proposed a workflow that divides work according to API-related tasks such as formulating search queries and reviewing specific search results. When an API


Search for publications, articles, and blog posts about food safety

Requester: You **Reward:** $1.50 **HITs Available:** 16 **Duration:** 60 minutes
Qualifications Required: None

We are creating a list of publications, articles, and blog posts about food safety meeting the following criteria:

The main focus must be on food safety, not as a side focus

- For *date* enter the date the publication/article/blog was published or posted
- For *title* enter the published title or blog post name **without the author, publisher, or blog post username included** (e.g. *fred fred burger: green beans are the hidden killer* should not include *fred fred burger:*)
- For *author* enter the author's name or the lead author (if there are multiple)
- For *publication* enter the entire publisher's name. If an acronym, type out the whole name (e.g. *World Health Organization*)

Figure 4. AMT HIT job criteria for Relay

is not available and relevant items can only be found in websites from a general web search, no workflow exists to draw meaningful divisions of labor.

Additionally, the crowd has been used to add columns or fill-in missing cells of existing datasets [22]. Park and Wisdom's CrowdFill system completed this task with a table interface that allows workers to fill-in missing cells [32]. They explored quality control methods that allowed workers to up/down vote responses from other workers. When list items are present, they can serve as a natural task decomposition boundary, but efficient decomposition without these items remains hard.

Collaborative Web Search and Question Answering

The task of list-making can be thought of as a form of collaborative search. Prior work has focused primarily on collocated searchers [1, 16, 17], although some of the problems associated with crowd-based search have also been explored. For example, Chorus is a crowd-based conversational agent that successfully used real-time crowds to answer simple questions [13]. Later work by Gouravajhala et al. explored how to maintain context of prior dialog by keeping a memory of important points in conversation [7]. These agents, however, are still limited in their ability to answer questions to what workers can find independently. The task of coordinating multiple workers to complete different parts of these queries is still unexplored.

RELAY

In this section we introduce Relay, a system that implements SOE to complete the list enumeration task. At a high level, Relay enables individuals and organizations alike to describe the format, criteria, topic, and per-item reward of a list where it then employs workers from the crowdsourcing platform Amazon Mechanical Turk (AMT) to create it.

Enumeration tasks can be broken down into two steps: 1) find a source, then 2) find list items, both completed by a single crowd worker (as outlined by the two large boxes in Figure 1). As in the baseline method, our workflow also uses a single crowd worker to complete theses steps, but we build upon it by adding three interactive components and an additional step. Two of these components, the source filter and source continuation, interact with the worker to ensure they find a unique source and fully extract existing sources. The auto-suggest component interacts with the worker to increase system-level prediction of the number of unique output. We add an additional task, to write hand-off instructions, to ensure a high-quality transition of sources between workers.

Design Considerations

Relay is designed with the knowledge that crowd worker's incentives can play a large role in shaping the quality and efficiency of their work [9, 14, 29]. For this reason, we curated our HITs so that incentives would align with efficient acquisition of list items by doing the following:

1. *Workers find list items from their own sources.* Contrary to a divide-and-conquer style workflow that might ask some workers to find sources and others to find list items, Relay asks workers to complete a portion of each subtask (find a source and find list items). This design choice encourages workers to be creative, find proliferate sources, and draw distinctions between sources on meaningful boundaries (between search terms, websites, and parts of websites).

2. *Workers are paid per item.* Paying workers a fixed rate per list item encourages them to optimize the rate at which they find items, aligning with our goal of maximized efficiency.

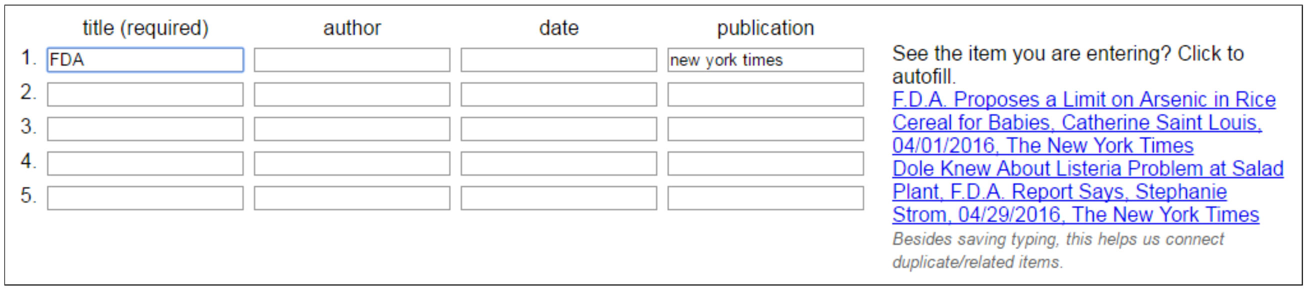
End User Interaction

When a requester uses Relay to build a list, she first makes her way to Relay's job specification form (Figure 2). On the form, she specifies her list topic, more specific criteria, list item format, the number of items she wants, and how much she is willing to pay per item. In this way, she knows the most money that will be spent to make the list, making the challenge to maximize the value of the money she invested.

Once the requester is finished, the system posts several Human Intelligence Tasks (HITs) to AMT. Relay posts up to five HITs in parallel per list, re-posting HITs as workers submit them. While our system limits concurrent work, future systems could allow for more or less parallel HITs depending on the number of desired items and according to the time and cost constraints.

Worker Interaction

Workers that accept the HIT will see instructions explaining both the list topic and the more specific criteria as described by the requester on the job specification form (Figure 4). For example, the list topic "US coffee brands" had criteria that list items must be sold in whole bean or ground bean form and can be purchased in US grocery or retail stores. The criteria also explained the various fields that workers were expected to



	title (required)	author	date	publication
1.	FDA			new york times
2.				
3.				
4.				
5.				

See the item you are entering? Click to autofill.
F.D.A. Proposes a Limit on Arsenic in Rice Cereal for Babies, Catherine Saint Louis, 04/01/2016, The New York Times
Dole Knew About Listeria Problem at Salad Plant, F.D.A. Report Says, Stephanie Strom, 04/29/2016, The New York Times
Besides saving typing, this helps us connect duplicate/related items.

Figure 5. Prior list items are shown beside the editing interface in Relay. Clicking these saves the worker keystrokes and enables us to connect duplicate or related entries since they will have the same spelling.

gather about each item. For example, workers were expected to gather the name and the URL of the coffee brand's website.

Controlling worker's sources
To coordinate workers to use a *diverse* set of sources, Relay retains memory about how workers find list items. Relay records information about the search made, how list items were gathered from the search results page, and the specific website that was used to find list items (Figure 3). It does this by orienting its interface around sources of information (instead of entries to the list) and requires workers to record how they make their search. A source can be anything where list items are found including a section of a website, an entire website, or even search results. To find a source, workers are required to use a search engine. They can move back and forth between a separate browser tab and Relay's interface. Our goal was to maximize the variety of sources that workers can find on the Internet yet maintain the capability of restricting which sources they are allowed to use.

With a source chosen, the interface allows workers to enter list items. List items are made up of two parts: fields marked required by the requester called "keys" and other, less critical fields called "key components". For example, the list topic "C and C++ course websites" would likely have a key value specified as the URL of the course website and key components specified as the university, the course name, the course number, the instructor, etc. The purpose of identifying the key and key components of list items is to aid the process of identifying duplicates. Items which have either identical keys or an identical combination of key components are automatically designated as duplicates. A more robust definition of duplicate list items will be described later in this paper.

To allow for workers to hand incomplete sources off to one another, Relay records information about a source's progress at the end of the HIT. When the worker is finished finding entries, the worker is posed with three options before completing the HIT: "All done!", "Not done yet...", and "Can't continue" (Figure 7). If a worker chooses the option "Not done yet..." they are prompted for the number of entries they estimate are left in the source and instructions for how another worker could pick up the source and continue where the previous worker left off. A worker may continue using a source for as many list items as they would like.

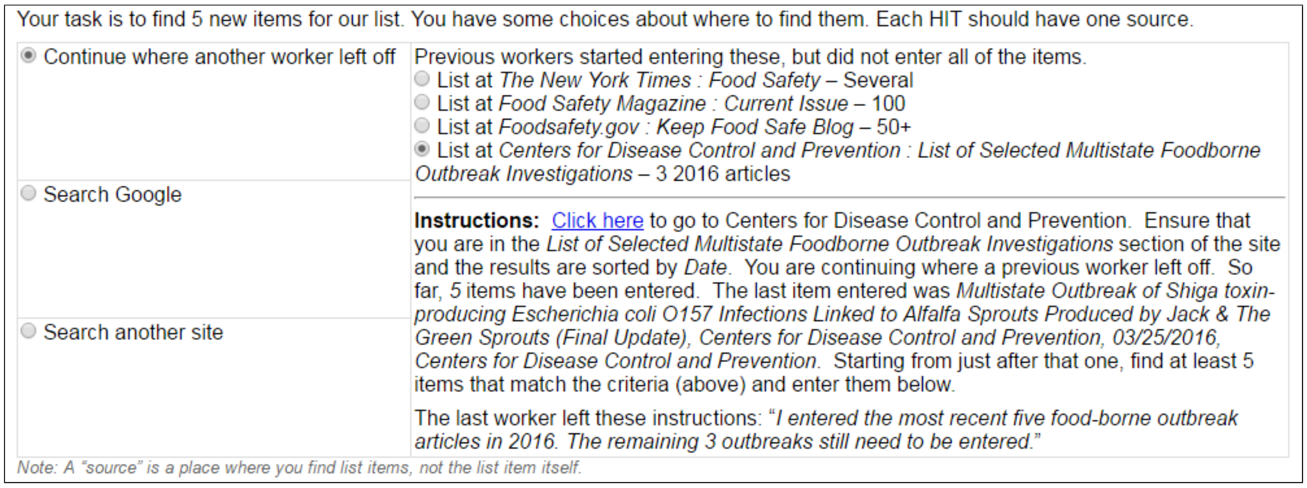
Components
Now that we have described Relay's implementation, we will describe some of the components that make up its workflow. These components:

- Prevents workers from accidentally using the same source as someone else. Called the **source filter**.
- Catch duplicate list items by suggesting possible matches during data entry. Called the **auto-suggest**.
- Allow workers to continue using a source where another worker left off with it. Called **source continuation**.

Source filter
The purpose of the source filter component is to control whether or not workers are forced to use a variety of sources. It engages worker's creativity and knowledge during the process of finding a source, but is flexible enough to allow for the serendipitous discovery of new sources from old ones.

When a worker finds a source they would like to use, Relay checks to see if their source is reserved or exhausted. If this is the case, then the original worker is presented with an error message and is required to find a different one. Relay relies on workers coming into the problem blind (without knowing what other sources are being used by other workers) for two main reasons. First, allowing workers to see a list of all sources that are currently or have been previously used could become tedious when checking to make sure a source is not in the list. Secondly, we aim to maintain the creativity of a "first look" approach at the problem. Showing examples of what other workers have done would constrain their creativity [25] producing more trivial query modifications and result in more overlap of search results. When a worker manages to find an available source and begins entering list items, the source filter synchronously reserves it.

Auto-suggest
The auto-suggest component reduces the effort necessary during postprocessing by catching duplicates at the time of entry. The component enables workers to identify duplicate list items with the incentive of less typing. As workers find and enter list items into the HIT, similar items are suggested next to the worker's current items (Figure 5). Relay suggests items based on a simple auto-complete, fuzzy matching algorithm. When a worker marks an entry as duplicate, Relay ensures it is entered in the same way as the prior item and can be easily identified using computational methods. The auto-suggest updates synchronously enabling parallel work.


Your task is to find 5 new items for our list. You have some choices about where to find them. Each HIT should have one source.

● Continue where another worker left off

Previous workers started entering these, but did not enter all of the items.
- ○ List at *The New York Times : Food Safety* – Several
- ○ List at *Food Safety Magazine : Current Issue* – 100
- ○ List at *Foodsafety.gov : Keep Food Safe Blog* – 50+
- ● List at *Centers for Disease Control and Prevention : List of Selected Multistate Foodborne Outbreak Investigations* – 3 2016 articles

Instructions: Click here to go to Centers for Disease Control and Prevention. Ensure that you are in the *List of Selected Multistate Foodborne Outbreak Investigations* section of the site and the results are sorted by *Date*. You are continuing where a previous worker left off. So far, *5* items have been entered. The last item entered was *Multistate Outbreak of Shiga toxin-producing Escherichia coli O157 Infections Linked to Alfalfa Sprouts Produced by Jack & The Green Sprouts (Final Update), Centers for Disease Control and Prevention, 03/25/2016, Centers for Disease Control and Prevention*. Starting from just after that one, find at least 5 items that match the criteria (above) and enter them below.

The last worker left these instructions: "*I entered the most recent five food-borne outbreak articles in 2016. The remaining 3 outbreaks still need to be entered.*"

○ Search Google

○ Search another site

Note: A "source" is a place where you find list items, not the list item itself.

Figure 6. Relay handing off sources between workers with source continuation enabled

The auto-suggest represents a step toward the goal of paying workers proportionally to the effort they expend. Achieving this goal is difficult in the task of list generation because an increasing list size leads to increased effort to find new list items. A potential solution to this problem is to escalate the price per item as the list becomes longer. This solution relies on a good estimation of the time it will take workers to find a unique item, an estimation that would have to take into consideration many factors and would likely have large fluctuations from HIT to HIT. The auto-suggest advocates that the requester pay a flat rate per list item (unique or not), rely on an estimator to find such a rate, and take on the fluctuations in item-to-item price knowing that the aggregate of such fluctuations will balance out. This approach incentives the use of good sources and efficient acquisition of list items, an incentive that aligns both the requester's and workers' objectives.

Source continuation

Relay reuses sources that are effective by allowing workers to hand them off to one another. The hand off is performed in two steps: first a worker describes where they are in their source and second a worker picks up from where the previous worker left off. Step one is completed after finding list items. As seen in Figure 7 workers describe how many items are left and how the next worker can pick up from where they left off. The second step is done when the next worker is choosing a source (see Figure 6). When source continuation is active, workers are given the option to "Continue where another worker left off." instead of finding a new source. Clicking this option reveals a list of sources that can be continued along with the remaining item predictions made by previous workers. After picking a source, workers are informed about how to pick up from where the last worker left off.

STUDY

We designed our study to evaluate the following hypotheses:

H1: The source filter component will lead to more efficient list item acquisition due to improved diversity of sources.

H2: The use of the auto-suggest component will lead to a better system prediction of the number of unique items and will save workers typing time for suggested items that they would otherwise enter manually.

H3: The addition of the source continuation component with the source filter component will lead to more efficient list item acquisition due to more comprehensive coverage of sources.

H4: The combination of these components into the SOE workflow will lead to better efficiency than any one component alone, without compromising the system's ability to predict unique items.

We tested these hypotheses by running Relay on four topics and two sizes as shown in Table 1. We chose these list topics because they meet the following criteria:

1. They are of factual nature (e.g., verifiable events).
2. Searching for list items requires more than trivial web searching skills (i.e. no existing API will return a comprehensive list of results).
3. They represent a realistic list of interest to requesters.

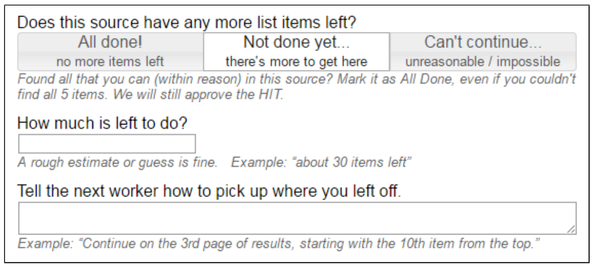
For each of the chosen list topics, anonymous workers from the crowd were recruited off of AMT's marketplace and randomly assigned a treatment. Workers were limited to five entries per HIT, but were allowed to complete multiple HITs (retaining the treatment they were assigned). In this way, workers were allowed to contribute as much or little to our list as they desired. They were paid 30¢ per list item that they added.

Metrics

Relay was designed to let workers contribute as much or little as they want. To neutralize the effects of an unbalanced distribution of work between workers, we created four metrics to measure various aspects of list quality that can be calculated per HIT submission (only including HITs with five entries). The first three metrics were used to evaluate all hypotheses while the last metric was used to evaluate H2 and H4.

We define worker efficiency as follows:

$$\text{worker efficiency} = \frac{\text{\# unique entries}}{\text{worker time (minutes)}}$$


Does this source have any more list items left?

All done! no more items left | Not done yet... there's more to get here | Can't continue... unreasonable / impossible

Found all that you can (within reason) in this source? Mark it as All Done, even if you couldn't find all 5 items. We will still approve the HIT.

How much is left to do?

A rough estimate or guess is fine. Example: "about 30 items left"

Tell the next worker how to pick up where you left off.

Example: "Continue on the 3rd page of results, starting with the 10th item from the top."

Figure 7. Relay instructions to the next worker

Worker efficiency is the primary metric for evaluating the unique item throughput. We calculated this metric for every HIT that was submitted and aggregated all HITs in each list for a list-wide measurement. While this metric is useful for measuring end-to-end efficiency, we were also interesting in some of the components that make up that efficiency such as:

$$\text{search efficiency} = \frac{\#\ \text{total entries}}{\text{worker time (minutes)}}$$

We used this metric to measure the rate at which workers were finding list items, regardless of uniqueness. We again calculating it for each HIT and aggregated across lists. Another component we measured was:

$$\text{uniqueness} = \frac{\#\ \text{unique entries}}{\#\ \text{total entries}}$$

Uniqueness was used to measure the fraction of unique list items in a given list. It was calculated in the same was as the prior two metrics. Finally:

$$\text{accuracy} = \frac{\#\ \text{caught duplicate entries}}{\#\ \text{duplicate entries}}$$

This final metric was used to measure the system's ability to detect duplicate list items (accuracy of its prediction). It was also calculated at the HIT level, but only included HITs that had at least one duplicate.

We measured worker time by aggregating the time it takes workers to find a source and five list items. HITs that contained less than five list items were not included in the analysis because the time it takes workers to complete the interface components related to entering a source could skew the results. Also, HITs that took a substantially greater amount of time compared to that of other HITs were treated as outliers and were not included in the analysis.

Duplicates were counted by the time-order of HIT submission. In other words, if a worker submitted a HIT with five unique list items and another worker submits a HIT containing four new items and one overlapping list item, we would mark the second HIT as containing a duplicate and the first HIT as containing no duplicates (instead of marking both HITs as containing a duplicate).

Treatments

We created a set of five treatments to test the necessity of each component in our workflow and to test our hypotheses of how they behave when singled out. Each treatment was independent of the others, having no components populate or interact with list items found by workers in other treatments. All treatments gathered approximately the same number of list items. The treatments were configured as follows:

List Topic	Target # entries
United States demonstrations that were violent or lead to violence	100
tourist attractions in Austin, Texas	100
publications, articles, and blog posts about food safety	200
research talks in university computer science departments	200

Table 1. List topics and the number of list items we requested. Note: the above numbers represent the total number of list items we gathered from the crowd for each treatment. The gathered lists contained duplicates, so the number of unique entries was less than requested and varied depending on the treatment.

1. Control (all hypotheses): All components were disabled to resemble a non-SOE baseline. Workers were asked to find five list items and the source they used. Note: Duplicates can still be caught, but only if they have the same spelling.
2. Source filter enabled (H1): The source filter was enabled by itself to test the case where sources are controlled and workers find a unique one before finding items.
3. Auto-suggest enabled (H2): Auto-suggest was enabled alone to test the case where potential duplicates are suggested while workers enter their items into the form.
4. Source filter and source continuation enabled together (H3): Both the source filter and source continuation components were enabled to single out the source continuation strategy. Note: Source continuation cannot be tested without the source filter for practical reasons —sources must be controlled in order to hand them off between workers.
5. All components enabled (all hypotheses): All components were enabled to demonstrate the full potential of SOE.

The baseline

A requester who wishes to use the crowd for list generation might simply ask workers to add an item to their list (as Trushkowsky et al. did). We ask workers to also provide the source that they use. This was done for two reasons:

- For most applications, the integrity of data acquired by workers matters to the requester. Asking workers to provide their source serves as the easiest way to check for errors.
- Including overhead time (the time it takes workers to type their source into the interface) makes our evaluation implementation independent. Since all treatments have equivalent overhead, the only variables manipulated are the components, which are generalizable to other types of crowd tasks.

Rater

To evaluate Relay, we hired a third party rater to avoid any bias. The third party was instructed to mark items as irrelevant if they do not meet the specific criteria of our list or if it is obvious to the third party that the list item is not a true instance or event. We instructed the third party to mark items as duplicate if they meet any of the following criteria:

Treatment	# workers	# HITs
Control	53	105
Source filter enabled	84	111
All features enabled	36	98
Auto-suggest enabled	70	110
Source filter and source continuation enabled	62	118

Table 2. Number of workers recruited and number of HITs completed over the course of our experiments for each treatment. Note: workers were allowed to complete multiple HITs and each HIT was used as a single data point in our analysis.

1. The list item has an identical key field as another list item (the field that uniquely identifies the item).
2. All completed fields of the list item (even if misspelled, shortened, or referred to by a different name) point to the same instance or event as another list item.
3. The list item matches an instance or event but has an incorrect field (and if taken verbatim would not point to any instance or event), it is up to the third party to correct the error before assessing criteria (2).

RESULTS

We found that the auto-suggest works independently to improve accuracy, but that all components must be enabled to see an improvement in worker efficiency, demonstrating full support for H2 and H4 partial support for H3, and no support for H1 (number of workers recruited per treatment is shown in Table 2). Additionally, we looked into some of the reasons that might have caused components to fail individually, but synergize when enabled together such as whether workers were incentivized correctly with the auto-suggest enabled or if the mistakes of some workers compounded into mistakes by others. Finally, we report some of the more practical benefits of our strategies, such as improvements in end-to-end time and cost. For the rest of this section, we will discuss how our system components can be used to improve accuracy and worker efficiency, how they effected the data quality, and the practical benefits that can be expected when they are enabled.

Auto-suggest improves accuracy

After enabling the auto-suggest, we saw improvements in accuracy over the control demonstrating support for H2. We performed a robust one-way ANOVA test across all five conditions and saw statistical significance ($F(4, 84.73) = 16.43$, $p < 0.001$). The presence of the source filter and source continuation had no significant effect on accuracy (Figure 8). We found that the auto-suggest performed the best on list topics that have the most fluid of list items (i.e. multiple names/spellings of the same list item).

The auto-suggest was the only component that lead to an increase in the accuracy. Enabling the source filter alone and enabling the source filter and source continuation together lead to a non-significant change. Additionally, there was no significant difference between enabling all features and enabling only the auto-suggest. The increase in accuracy when the auto-suggest was enabled alone (mean = 0.74, std = 0.38) was significant over the control (mean = 0.33, std = 0.42) with a p-value < 0.001. The increase when all features were enabled (mean = 0.61, std = 0.14) was also significant (p = 0.003).

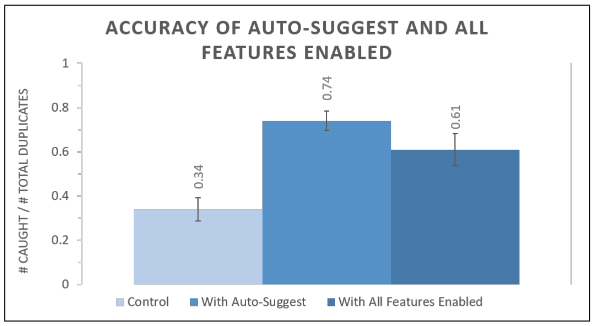


Figure 8. Enabling the auto-suggest leads to a substantial increase in accuracy. Enabling the source filter and/or source continuation had no significant effect on accuracy.

List topics with list items that have the largest room for variability saw the greatest improvements in accuracy. For example, the research talks list topic contained many fields that, without the auto-suggest, were likely to be entered into the form the same way by workers. These field types, such as the "URL", "date", and "speaker" stayed mostly consistent across sources. The auto-suggest was the least beneficial in this case because the control had a high accuracy already. By comparison, the United States demonstrations list topic may have different spellings, orderings, or aliases for each demonstration. Sources contained different versions of the same demonstrations, thereby leading to poor accuracy in the control. The auto-suggest was most effective in these scenarios showing that workers actively resolved entities.

Improving worker efficiency

We found that enabling features by themselves leads to a decrease in performance; however, enabling all features together lead to a large increase in performance (see Figure 9 and Figure 10) demonstrating no support for H1, partial support for H3, and full support for H4. A robust one-way ANOVA test on uniqueness, search efficiency, and worker efficiency reveals statistical significance between the conditions ($F(4, 159.01) = 4.89$, $p < 0.001$, $F(4, 153.6) = 12.81$, $p < 0.001$, and $F(4, 114.94) = 12.33$, $p < 0.001$ respectively). We will now discuss how each feature affected efficiency metrics.

Source continuation improves uniqueness

When the source filter was evaluated by itself we saw a decrease in search efficiency and worker efficiency and a non-significant change in uniqueness over the control. By itself the source filter had a mean search efficiency of 0.62 and standard deviation of 0.47 while the control had mean 0.93 and standard deviation of 0.47 (p = 0.003). Worker efficiency showed the same relationship (mean = 0.45, std = 0.44 vs. mean = 0.70, std = 0.72), p = 0.039.

While enabling the source filter by itself lead to no change in uniqueness, we found that combining it with source continuation leads to an increase in uniqueness (mean = 0.76, std = 0.30), p = 0.043. This indicates that the source filter and source continuation together achieve part of their intended purpose: more unique entries.

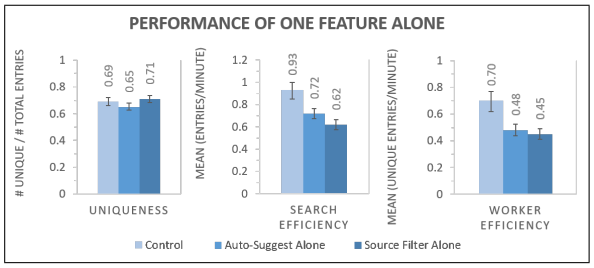


Figure 9. Enabling features alone leads to a decrease in performance. Note: The "Auto-Suggest Alone" uniqueness and search efficiency metrics and the "Source Filter Alone" uniqueness metric were not significantly different from the control.

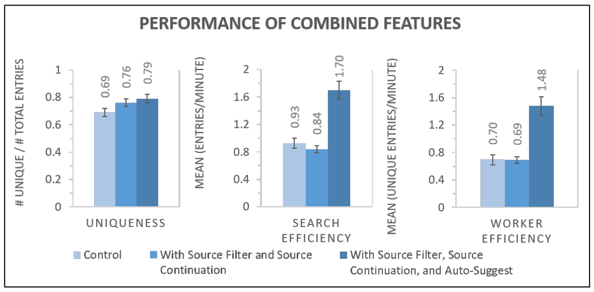


Figure 10. Enabling multiple features at once leads to an increase in uniqueness while enabling all features leads to an increase in search efficiency and worker efficiency.

Contrary to H1 we found that enabling the source filter alone leads to worse efficiency. This demonstrates that using the component adds overhead and is not beneficial by itself, but (as we will discuss shortly) at the benefit of added organization for other components to take advantage of. The added overhead manifests itself when workers are forced to find a unique source that is possibly further down the search results and less prolific than a more intuitive one at the top of the search results. We expected that this organization would provide us with a diversity of sources, lead to a diversity of list items, and thereby outweigh the cost. While we did see a diversity of sources and list items, the cost was too great for compensation.

Organization alone was not enough to see a performance benefit; however, it provides us with a means to apply other techniques that do see a performance benefit. Source continuation is an example of a technique that requires organization (How do you hand off work if you do not have a place to hand it off from?). When we added source continuation in addition to the source filter, we saw a statistically significant performance improvement in uniqueness, but not in search efficiency or worker efficiency. This result partially supports our hypothesis (H3) that we would see greater coverage of sources (as can be seen by the increase in uniqueness), but the added overhead was still too great to see an improvement in efficiency. We will later discuss how using the auto-suggest in combination with this strategy lead to faster list item acquisition while maintaining this improved uniqueness.

To ensure that the task hand-off present in the source continuation component was working as intended, we split the source continuation treatment into two conditions: list items contained within sources that were handed-off at least one time and list items contained within sources that were not handed-off. We performed a Wilcoxon rank sum test against these two conditions and found that they were statistically significant ($W = 916$, $p < 0.001$). Items from sources that were handed-off at least one time averaged a uniqueness of 0.86 compared to the average of 0.62 of items from sources that were not handed-off. This indicates that workers were successful in picking up work from previous workers.

Looking closer at the instructions workers left for one another reveals that their instructions were helpful (see Table 3). Most workers provided instructions with some sort of useful information for the next worker. For example:

"Last date was 6-14-2016, so go forward from there."

represents a typical hand-off instruction from workers. It mentions the item the worker left off with (which is not useful because it is automatically recorded) and how to proceed to find new items. The fewest number of workers provided the highest quality of instructions: information about the source's structure. For example, one worker left the instructions:

"When you search 'Tourist attractions Austin Texas', there is a list that appears on the top on the page. I left off at the bottom of the 4th column (I did not include parks because there isn't a phone number to list)."

This demonstrates a successful hand-off of work on a non-trivially formatted webpage. In contrast, a few workers provided information that could be misleading to the next worker: a search engine results page entry. For example:

"Continue from result number 6 on the first page."

could be misleading because the next worker might have a different result number six than the previous worker. Yet other workers provided no relevant information for the next worker. Commonly, workers in this category provided only their current website in their instructions:

"http://foodbabe.com/investigations/"

Worker's sources are recorded at the time of reservation, so this information provides no usefulness to the next worker.

Looking at the estimations workers made (of the number of list items remaining in the source they were using) reveals that approximately half of the estimates were reasonable and helpful, but another half were unreasonable or unhelpful (see Table 4). We found that workers were good at making estimates when there were few items remaining in the source, but as the source becomes larger and more complex, workers tend to make unreasonably large or non-specific estimates. The larger the source, the more factors workers must take into

Criteria	% Instructions
Mentions website on results page	27
Mentions item on website	65
Mentions detail specific to the structure of the website	15
Mentions none of the above	15

Table 3. Hand-off instructions labelled by their contribution. Note: Many instructions fit multiple categories, so the right sums to more than 100%.

Estimate Range	% Estimates Provided
< 50	31
[50, 100)	14
[100, 200)	15
>= 200	29
Non-Specific	11

Table 4. Workers estimated how many items were left in their source during the hand-off. We categorized them according to the range their estimates fell into.

consideration when making an estimate, many that they would not naturally take into consideration (such as the decreasing relevance of a search results page). A purely computational estimator (such as the estimator Trushkowsky et. al. introduced for the combination of all sources) might replace crowd estimates in these settings.

Putting it all together

Our final condition enables all components to demonstrate the full potential of our workflow. Under this condition we saw an increase in search efficiency and worker efficiency while maintaining no significant change in uniqueness from the source filter and source continuation case (mean = 1.70, std = 1.30 and mean = 1.48, std = 1.36 respectively) demonstrating full support for H4. Unexpectedly, we found that enabling the auto-suggest in combination with the source filter and source continuation components provided the additional benefit of saved typing time. Since the auto-suggest fills in items when identified as duplicate, workers were more easily able to move onto new items within their source. In contrast, enabling the auto-suggest by itself did not benefit from saved typing time because, without the source filter, workers had no requirement to find unique sources and thereby duplicated them. Auto-filling with a duplicate source just leads to faster acquisition of duplicate list items and no improved efficiency. Most importantly, this finding motivates the need for the complexity SOE introduces and that its components synergize to lead to an overall improvement.

Data quality

In addition to the previous analyses, we evaluated the quality of data we received from workers. We looked into two aspects of data quality: intentional gaming of the system and whether our components caused workers to stray from the criteria, providing us with more irrelevant entries.

Were workers incentivized correctly?

The auto-suggest component enabled workers to see duplicate list items and to click on them to automatically fill in the item. It is possible that workers abused this feature to quickly enumerate items, thereby making more money. We performed analysis and verification to answer this question. Out of 3000 entries, 253 exactly matched an an entry by a prior worker. We consider abuse to be when a worker lists an item in their form that cannot be found at the source they specified. We could not check 120 entries because, most commonly, they were news/calendar sites that no longer cover the time period of the given list items. We checked the remaining 133 items manually. With auto-suggest enabled, 4/109 (3.7%) were not found at the stated source. Without auto-suggest, 1/24 (4.2%) were not found. A chi-squared test found no significant difference (i.e., no significant evidence of abuse).

List topic	% Time Improvement	% Cost Improvement
Protests	32.76	22.64
Austin	167.90	23.33
Articles	21.66	10.42
Talks	-28.42	11.32

Table 5. Relay saw improvements in both end-to-end time and cost in virtually every list topic.

The low rate of cheating is by design. One reason to require workers to provide us with a source at the front of the HIT is because of the potential audit trail it leaves behind. To notify workers of this potential, we included the following line in the HIT: "We will be spot-checking to verify that list entries are at the selected sources".

Did workers follow the list criteria?

Another important aspect of data quality is the potential for our components to compound errors made by previous workers. This could manifest itself in compounding deviations from the list criteria when workers see incorrect entries made by other workers when suggested by the auto-suggest component or when they read bad instructions given by the source continuation component.

Our 3rd-party rater reviewed all 3066 list entries and flagged 66 (2.1%) that violated our original list criteria. These were excluded from our analysis. Since the criteria were presented in the same way regardless of treatment, we would not expect any effect on relevance, and thus we assume the 66 (2.1%) irrelevant items were due to inattentive workers.

To verify this assumption, we used a chi-square significance test to compare the number of irrelevant entries between the different conditions. Since workers contributed varying numbers of items, and in some cases repeated similar mistakes, we compared the number of workers who made at least one mistake. On this metric, there was no significant difference between any treatment over the control showing that workers who saw the errors of other workers did not repeat the mistake.

What practical benefits can be expected?

Relay saw improvements in end-to-end time and cost over the control. A summary of the benefits can be seen in Table 5. Benefits can be tailored to the needs of the requester. If a requester wishes to optimize for end-to-end time, they can allow more workers to work in parallel; however, they will have to pay more per unique list item. If a requester wishes to optimize for cost, they can limit the number of workers that can work in parallel but will experience a longer wait time.

DISCUSSION AND FUTURE WORK

Our findings have several implications for the future of research into crowdsourced enumeration queries and to research into crowdsourcing workflows. We will first discuss the costs versus benefits trade-off that our workflow introduces, then discuss the parallels and contrasts of Relay's workflow to the workflows present in other forms of labor, and conclude with a way to improve upon our technique in future work.

Costs and benefits

SOE leads to a significant improvement in the efficiency of crowd workers and accuracy of system duplicate prediction. Contrary to our hypothesis, we found that using just a part of our workflow, the source filter, came at the cost of significantly increased overhead. Singling out a second piece of our workflow, source continuation, lead to yet more increased overhead due to the time it takes workers to fill out hand-off instructions, but it did lead to the expected improvement in coverage of sources and overall uniqueness. Singling out a third piece of the workflow, the auto-suggest, did not introduce any costs, but by itself we only saw the benefit of improved accuracy. It was only until all components were enabled that we saw the overhead of the source filter and source continuation pay dividends (i.e. increased uniqueness due to better extraction of sources and saved typing time from the auto-suggest). These findings demonstrate the robustness of our workflow: each of its components play a vital role in its success and that they should not be used by themselves. This is consistent with prior findings in other types of crowd workflows. That is, that some components introduce an up-front cost, but later benefit from the supplementary organization [12, 28]. Future work might make additional use of the organization manifested in our workflow.

Parallels and contrasts to Relay's workflow

A core benefit of our workflow is that it allows for the division of labor along fluid boundaries. This is done by allowing workers to define what their source is (whether it be search terms, an entire website, or part of a website) enabling both the serendipitous discovery of new sources from old ones and incentives for the workers to draw distinctions between sources on meaningful boundaries. This contrasts with the typical organizational structures of crowdsourcing workflows that enforce rigid boundaries through divide-and-conquer techniques. We demonstrated that our style works well for the list enumeration task, but it might apply to other kinds of work.

Additionally, our workflow parallels the way many offline organizations manage work, where a fluid division of labor is critical to success. For example, software engineering tasks are usually divided along boundaries that change over the course of completing the task and as the nuances of the original task are revealed. Relay's workflow brings crowdsourcing one step closer to this kind of fluidity.

Mechanisms for quality control

While the vast majority of workers adhered to the constraints we imposed on them in our four list topics, list topics that require more expertise (such as a list of related works for a paper) would likely see more errors. Imposing mechanisms to check results (such as an extra filtration step) or mechanisms to identify workers with expertise prior to starting the task could be fruitful directions for future improvement.

CONCLUSION

Relay represents a step toward crowd-powered systems that enumerate new datasets rather than operating on existing ones. We advance the work of Trushkowsky et al. by introducing a workflow that greatly improves the efficiency of workers through intelligent source management. We demonstrated the necessary complexity of our workflow through a thorough examination of each component, showing that singled them out leads to worse or marginal improvement in the quality of the output. We believe the task of list-making is one that will have practical benefits for consumers, businesses, researchers, and other users and organizations. It might even be applied as a first step in a literature review in an unfamiliar area.

Furthermore, Relay takes steps toward preserving the traditional independence of crowd workers. Microtask workers expect to be paid a constant amount per task, but also that the pay be proportional to the effort. Relay has made some effort toward this goal by associating pay with both the task of finding a source and a per-item price. Additionally, our approach represents a step toward drawing divisions of labor along fluid boundaries by allowing workers to define their source according to where items are found. We demonstrate the feasibility of our approach through an evaluation and open up future areas for improvement.

REFERENCES

1. Saleema Amershi and Meredith Ringel Morris. 2008. CoSearch: a system for co-located collaborative web search. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, New York, NY, USA, 1647–1656.

2. Michael S Bernstein, Greg Little, Robert C Miller, Björn Hartmann, Mark S Ackerman, David R Karger, David Crowell, and Katrina Panovich. 2015. Soylent: a word processor with a crowd inside. *Commun. ACM* 58, 8 (2015), 85–94.

3. Joel Chan, Steven Dang, Peter Kremer, Lucy Guo, and Steven Dow. 2014. Ideagens: A social ideation system for guided crowd brainstorming. In *Second AAAI Conference on Human Computation and Crowdsourcing*. AAAI, Palo Alto CA.

4. Gianluca Demartini, Djellel Eddine Difallah, and Philippe Cudré-Mauroux. 2012. ZenCrowd: leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking. In *Proceedings of the 21st international conference on World Wide Web*. ACM, New York, NY, USA, 469–478.

5. Michael J Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. 2011. CrowdDB: answering queries with crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. ACM, New York, NY, USA, 61–72.

6. Chaitanya Gokhale, Sanjib Das, AnHai Doan, Jeffrey F Naughton, Narasimhan Rampalli, Jude Shavlik, and Xiaojin Zhu. 2014. Corleone: Hands-off crowdsourcing for entity matching. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, New York, NY, USA, 601–612.

7. Sai R. Gouravajhala, Youxuan Jiang, Preetraj Kaur, Jarir Chaar, and Walter S Lasecki. 2018. Finding Mnemo: Hybrid Intelligence Memory in a Crowd-Powered Dialog

System. *Proceedings of Collective Intelligence (CI 2018)* (2018).

8. Benoit Groz, Ezra Levin, Isaac Meilijson, and Tova Milo. 2016. Filtering with the crowd: Crowdscreen revisited. In *LIPIcs-Leibniz International Proceedings in Informatics*, Vol. 48. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 12:1–12:18.

9. Gary Hsieh and Rafał Kocielnik. 2016. You get who you pay for: The impact of incentives on participation bias. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*. ACM, 823–835.

10. Gaoping Huang and Alexander J Quinn. 2017. BlueSky: Crowd-powered Uniform Sampling of Idea Spaces. In *Proceedings of the 11th ACM Creativity & Cognition*. ACM, New York, NY, USA.

11. Panagiotis G Ipeirotis, Foster Provost, and Jing Wang. 2010. Quality management on amazon mechanical turk. In *Proceedings of the ACM SIGKDD workshop on human computation*. ACM, New York, NY, USA, 64–67.

12. Anand Kulkarni, Matthew Can, and Björn Hartmann. 2012. Collaboratively Crowdsourcing Workflows with Turkomatic. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work (CSCW '12)*. ACM, New York, NY, USA, 1003–1012. DOI: http://dx.doi.org/10.1145/2145204.2145354

13. Walter S Lasecki, Rachel Wesley, Jeffrey Nichols, Anand Kulkarni, James F Allen, and Jeffrey P Bigham. 2013. Chorus: a crowd-powered conversational assistant. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*. ACM, 151–162.

14. Yang Liu and Yiling Chen. 2016. Learning to incentivize: Eliciting effort via output agreement. *arXiv preprint arXiv:1604.04928* (2016).

15. Adam Marcus, Eugene Wu, David Karger, Samuel Madden, and Robert Miller. 2011. Human-powered Sorts and Joins. *Proc. VLDB Endow.* 5, 1 (Sept. 2011), 13–24. DOI:http://dx.doi.org/10.14778/2047485.2047487

16. Meredith Ringel Morris and Eric Horvitz. 2007. SearchTogether: an interface for collaborative web search. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*. ACM, New York, NY, USA, 3–12.

17. Meredith Ringel Morris, Jarrod Lombardo, and Daniel Wigdor. 2010. WeSearch: supporting collaborative search and sensemaking on a tabletop display. In *Proceedings of the 2010 ACM conference on Computer supported cooperative work*. ACM, New York, NY, USA, 401–410.

18. Aditya Parameswaran, Stephen Boyd, Hector Garcia-Molina, Ashish Gupta, Neoklis Polyzotis, and Jennifer Widom. 2014. Optimal crowd-powered rating and filtering algorithms. *Proceedings of the VLDB Endowment* 7, 9 (2014), 685–696.

19. Aditya Parameswaran, Akash Das Sarma, and Vipul Venkataraman. 2016. *Optimizing Open-Ended Crowdsourcing: The Next Frontier in Crowdsourced Data Management*. Technical Report arXiv:1610.05377. arXiv preprint.

20. Aditya Parameswaran, Ming Han Teh, Hector Garcia-Molina, and Jennifer Widom. 2013. Datasift: An expressive and accurate crowd-powered search toolkit. In *First AAAI Conference on Human Computation and Crowdsourcing*. AAAI, Palo Alto CA, 112–120.

21. Aditya G Parameswaran, Hector Garcia-Molina, Hyunjung Park, Neoklis Polyzotis, Aditya Ramesh, and Jennifer Widom. 2012. Crowdscreen: Algorithms for filtering data with humans. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM, New York, NY, USA, 361–372.

22. Hyunjung Park, Hector Garcia-Molina, Richard Pang, Neoklis Polyzotis, Aditya Parameswaran, and Jennifer Widom. 2012. Deco: A system for declarative crowdsourcing. *Proceedings of the VLDB Endowment* 5, 12 (2012), 1990–1993.

23. Alexander J Quinn and Benjamin B Bederson. 2011. Human computation: a survey and taxonomy of a growing field. In *Proceedings of the SIGCHI conference on human factors in computing systems*. ACM, New York, NY, USA, 1403–1412.

24. Victor S Sheng, Foster Provost, and Panagiotis G Ipeirotis. 2008. Get another label? improving data quality and data mining using multiple, noisy labelers. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, New York, NY, USA, 614–622.

25. Steven M Smith, Thomas B Ward, and Jay S Schumacher. 1993. Constraining effects of examples in a creative generation task. *Memory & Cognition* 21, 6 (1993), 837–845.

26. Beth Trushkowsky, Tim Kraska, Michael J Franklin, and Pradyut Sarkar. 2013. Crowdsourced enumeration queries. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*. IEEE, Piscataway NJ, 673–684.

27. Beth Trushkowsky, Tim Kraska, Michael J Franklin, Purnamrita Sarkar, and Venketaram Ramachandran. 2015. Crowdsourcing enumeration queries: Estimators and interfaces. *IEEE Transactions on Knowledge and Data Engineering* 27, 7 (2015), 1796–1809.

28. Melissa A. Valentine, Daniela Retelny, Alexandra To, Negar Rahmati, Tulsee Doshi, and Michael S. Bernstein. 2017. Flash Organizations: Crowdsourcing Complex Work by Structuring Crowds As Organizations. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 3523–3537. DOI: http://dx.doi.org/10.1145/3025453.3025811

29. Luis Von Ahn. 2006. Games with a purpose. *Computer* 39, 6 (2006), 92–94.

30. Jiannan Wang, Tim Kraska, Michael J Franklin, and Jianhua Feng. 2012. Crowder: Crowdsourcing entity resolution. *Proceedings of the VLDB Endowment* 5, 11 (2012), 1483–1494.

31. Steven Euijong Whang, Peter Lofgren, and Hector Garcia-Molina. 2013. Question selection for crowd entity resolution. *Proceedings of the VLDB Endowment* 6, 6 (2013), 349–360.

32. Haoqi Zhang, Edith Law, Rob Miller, Krzysztof Gajos, David Parkes, and Eric Horvitz. 2012. Human computation tasks with global constraints. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, New York, NY, USA, 217–226.