Leveraging Transitive Relations for Crowdsourced Joins*

Jiannan Wang[#], Guoliang Li[#], Tim Kraska[†], Michael J. Franklin[‡], Jianhua Feng[#]
[#]Department of Computer Science, Tsinghua University, [†]Brown University, [‡]AMPLab, UC Berkeley
wjn08@mails.tsinghua.edu.cn, ligl@tsinghua.edu.cn, tim_kraska@brown.edu
franklin@cs.berkeley.edu, fengjh@tsinghua.edu.cn

ABSTRACT

The development of crowdsourced query processing systems has recently attracted a significant attention in the database community. A variety of crowdsourced queries have been investigated. In this paper, we focus on the crowdsourced join query which aims to utilize humans to find all pairs of matching objects from two collections. As a human-only solution is expensive, we adopt a hybrid human-machine approach which first uses machines to generate a candidate set of matching pairs, and then asks humans to label the pairs in the candidate set as either matching or non-matching. Given the candidate pairs, existing approaches will publish all pairs for verification to a crowdsourcing platform. However, they neglect the fact that the pairs satisfy transitive relations. As an example, if o_1 matches with o_2, and o_2 matches with o_3, then we can deduce that o_1 matches with o_3 without needing to crowdsource (o_1, o_3). To this end, we study how to leverage transitive relations for crowdsourced joins. We present a hybrid transitive-relations and crowdsourcing labeling framework which aims to crowdsource the minimum number of pairs to label all the candidate pairs. We propose a heuristic labeling order and devise a parallel labeling algorithm to efficiently crowdsource the pairs following the order. We evaluate our approaches in both simulated environment and a real crowdsourcing platform. Experimental results show that our approaches with transitive relations can save much more money and time than existing methods, with a little loss in the result quality.

Categories and Subject Descriptors: H.2.4 [Database Management]: Systems—*Query processing*;

*Revised September 2014. This is a revised and corrected version of a paper that appeared in the ACM SIGMOD 2013 Conference. The original version contained a claim that the algorithm for ordering pairs of records for presentation to the crowd (Section 4.2) was optimal. A subsequent paper in VLDB 2014 by Vesdapunt et al. [23] showed that the ordering problem is, in fact, NP-hard. Thus, in this version we have removed the claim of optimality and have updated the discussion and example in Section 4.2 accordingly. A more detailed explanation of this can be found in [26]. The copyright notice on the SIGMOD 2013 paper is as follows:

SIGMOD'13, June 22–27, 2013, New York, New York, USA.

Keywords: Crowdsourcing, Join Operator, Transitive Relations, Entity Resolution

1. INTRODUCTION

The development of crowdsourced query processing systems has recently attracted a significant attention in the database community [4,14,16]. A variety of crowdsourced queries have been investigated, such as crowdsourced `MAX` query [6,22], crowdsourced `SELECT` query [19,21], and crowdsourced `JOIN` query [3,14,25,27]. In this paper, we focus on the crowdsourced join query for entity resolution which aims to identify all pairs of *matching* objects between two collections of objects, where humans are utilized to decide whether a pair of objects is matching, i.e. referring to the same real-world entity. The crowdsourced join query can help to solve many real problems that are hard for computers. For example, given two collections of product records from two online retailers, there may be different records that refer to the same product, e.g. "iPad 2nd Gen" and "iPad Two". In order to integrate them, we can perform a crowdsourced join query to find all pairs of matching products.

A human-only implementation of crowdsourced joins is to ask humans to label every pair of objects from the two collections as either matching or non-matching [14]. Since the human-only solution is wasteful, prior works [3,25,27] showed how to build hybrid human-machine approaches. In our paper, we adopt a hybrid approach which first uses machines to generate a candidate set of matching pairs, and only then asks humans to label the pairs in the candidate set. Given the candidate pairs, existing hybrid solutions will publish all of them to a crowdsourcing platform, e.g., Amazon Mechanical Turk (AMT). However, existing hybrid solutions neglect the fact that the pairs satisfy transitive relations. By applying transitive relations, we can deduce some pairs' labels without asking humans to label them, thus reducing the number of crowdsourced pairs. For example, if o_1 and o_2 are matching, and o_2 and o_3 are matching, we do not need to crowdsource the label for (o_1, o_3) since they can be deduced as matching based on transitive relations.

Based on this idea, in our paper, we study the problem of combining transitive relations and crowdsourcing to label the candidate pairs generated by machines. We formulate this problem, and propose a hybrid labeling framework which aims to crowdsource the minimum number of pairs for labeling all the candidate pairs. We find that the labeling order, i.e. which pairs should be labeled first, has a significant effect on the total number of crowdsourced pairs. We prove that labeling first all matching pairs, and then the

other non-matching pairs leads to the optimal labeling order. However, the optimal order requires to know the real matching pairs upfront which cannot be achieved in reality. Therefore, we propose a heuristic labeling order which labels the pairs in the decreasing order of the likelihood that they are a matching pair. The likelihood could be given by some machine-learning methods [25]. In order to label the pairs in this order, one simple way is to label them from the first pair to the last pair one by one. However, this method prohibits workers from doing tasks in parallel and leads to a long completion time. To address this problem, we devise a parallel labeling algorithm which can identify the pairs that must need to be crowdsourced, and ask crowd workers to label them in parallel. To summarize, we make the following contributions in the paper:

- We formulate the problem of utilizing transitive relations to label the candidate pairs in crowdsourcing, and propose a hybrid transitive-relations and crowdsourcing labeling framework to address this problem.
- We find the labeling order has a significant effect on the number of crowdsourced pairs, and respectively propose an optimal labeling order and a heuristic labeling order.
- We devise a parallel labeling algorithm to reduce the labeling time, and propose two optimization techniques to further enhance the performance.
- We present our evaluations using both simulation and AMT. The experimental results show that our approaches with transitive relations can save much more money and time than existing methods, with a little loss in the result quality.

Organization. We formulate our problem in Section 2 and propose a hybrid labeling framewo1rk in Section 3. Section 4 discusses the optimal and expected optimal labeling orders. We devise a parallel labeling algorithm and two optimization techniques in Section 5. Experimental study is presented in Section 6. We cover related work in Section 7, and present our conclusion and future work in Section 8.

2. PROBLEM FORMULATION

In this section, we first introduce crowdsourcing (Section 2.1) and then define transitive relations (Section 2.2). Finally, we formulate our problem of utilizing transitive relations to label a set of pairs in crowdsourcing (Section 2.3).

2.1 Crowdsourcing

There are many crowdsourcing platforms, such as AMT and MobileWorks, which provide APIs for easily calling large numbers of workers to complete micro-tasks (called Human Intelligent Tasks (HITs)). To label whether two objects in a pair are identical through crowdsourcing, we create an HIT for the pair, and publish it to a crowdsourcing platform. Figure 1 shows an example HIT for a pair ("iPad 2", "iPad two"). In the HIT, the workers are required to submit "YES" if they think "iPad 2" and "iPad two" are the same or submit "NO" if they think "iPad 2" and "iPad two" are different. After the workers have completed the HIT, we obtain the crowdsourced label of the pair.

Assumption: Of course, workers might return wrong results and ambiguities in the question might exist. Techniques to address this problem were, for example, proposed in [2,7,13,24]. However, this problem can be treated as an orthogonal issue, as shown by [18,27], and we assume only correct answers for the remainder of the paper.

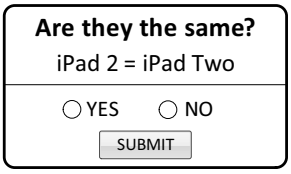


Figure 1: An example HIT for an object pair.

2.2 Transitive Relations

In this section, we discuss how to utilize transitive relations to deduce the label of a pair. We use the following notations. Let $p = (o, o')$ denote an object pair. The label of $p = (o, o')$ could be either "matching" or "non-matching", which respectively means that o and o' refer to the same real-world entity, and o and o' refer to different real-world entities. If o and o' are matching (non-matching), they are denoted by $o = o'$ ($o \neq o'$).

There are two types of transitive relations.

Positive Transitive Relation: Given three objects, o_1, o_2 and o_3, if $o_1 = o_2$, and $o_2 = o_3$, then we have $o_1 = o_3$. For example, consider the three objects "iPad 2nd Gen", "iPad Two" and "iPad 2". As "iPad 2nd Gen" and "iPad Two" are matching, and "iPad Two" and "iPad 2" are matching, then "iPad 2nd Gen" and "iPad 2" can be deduced as a matching pair based on positive transitive relation.

Negative Transitive Relation: Given three objects, o_1, o_2 and o_3, if $o_1 = o_2$, and $o_2 \neq o_3$, then we have $o_1 \neq o_3$. For example, consider the three objects "iPad Two", "iPad 2" and "iPad 3". As "iPad Two" and "iPad 2" are matching, and "iPad 2" and "iPad 3" are non-matching, then "iPad Two" and "iPad 3" can be deduced as a non-matching pair based on negative transitive relation.

By applying positive and negative transitive relations to n objects, we have the following lemma.

LEMMA 1. *Given a set of objects, $o_1, o_2, \cdots, o_n$, (1) if $o_i = o_{i+1}$ $(1 \leq i < n)$, then we have $o_1 = o_n$; (2) if $o_i = o_{i+1}$ $(1 \leq i < n, i \neq k)$, and $o_k \neq o_{k+1}$, then we have $o_1 \neq o_n$.*

Given a set of labeled pairs, to check if a new pair (o, o') can be deduced from them, we build a graph for the labeled pairs for ease of presentation. In the graph, each vertex represents an object, and each edge denotes a labeled pair. (For simplicity, an object and a pair of objects are respectively mentioned interchangeably with its corresponding vertex and edge in later text.) From Lemma 1, we can easily deduce the following conditions:

1. If there exists a path from o to o' which only consists of matching pairs, then (o, o') can be deduced as a matching pair;

2. If there exists a path from o to o' which contains a single non-matching pair, then (o, o') can be deduced as a non-matching pair;

3. If any path from o to o' contains more than one non-matching pair, (o, o') cannot be deduced.

EXAMPLE 1. *Consider seven labeled pairs: three matching pairs* $(o_1, o_2), (o_3, o_4), (o_4, o_5)$ *and four non-matching pairs* $(o_1, o_6), (o_2, o_3), (o_3, o_7), (o_5, o_6)$. *To check whether the unlabeled pairs* $(o_3, o_5), (o_5, o_7), (o_1, o_7)$ *can be deduced from them, we first build a graph as shown in Figure 2.*

For the unlabeled pair (o_3, o_5), *there is a path* $o_3 \rightarrow o_4 \rightarrow o_5$ *from* o_3 *to* o_5 *which only consists of matching pairs, i.e.* $o_3 = o_4$, $o_4 = o_5$, *thus* (o_3, o_5) *can be deduced as a matching pair.*

For the unlabeled pair (o_5, o_7), *there is a path* $o_5 \rightarrow o_4 \rightarrow o_3 \rightarrow o_7$ *from* o_5 *to* o_7 *which contains a single non-matching pair, i.e.* $o_5 = o_4$, $o_4 = o_3$, $o_3 \neq o_7$, *thus* (o_5, o_7) *can be deduced as a non-matching pair.*

For the unlabeled pair (o_1, o_7), *there are two paths* $o_1 \rightarrow o_2 \rightarrow o_3 \rightarrow o_7$ *and* $o_1 \rightarrow o_6 \rightarrow o_5 \rightarrow o_4 \rightarrow o_3 \rightarrow o_7$ *from* o_1 *to* o_7. *As both of them contain more than one non-matching pair,* (o_1, o_7) *cannot be deduced.*

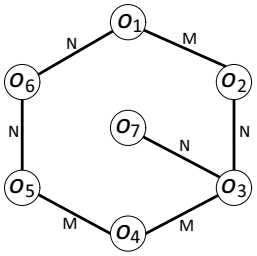


Figure 2: Graph illustrating Example 1 ("M" denotes "matching", and "N" denotes "non-matching").

2.3 Problem Description

To process crowdsourced joins, we first use machine-based techniques to generate a candidate set of matching pairs. This has already been studied by previous work [25]. The goal of our work is to study how to label the candidate pairs. Since in our setting, some pairs will be labeled by crowd workers, and others will be deduced using transitive relations. We call the former *crowdsourced (labeled) pairs*, and the latter *deduced (labeled) pairs*. Typically, on a crowdsourcing platform, we need to pay for crowdsourced pairs, thus there is a financial incentive to minimize the number of crowdsourced pairs. Based on this idea, we define our problem as below.

DEFINITION 1. *Given a set of pairs that need to be labeled, our goal is to crowdsource the minimum number of pairs such that for the other pairs, their labels can be deduced from the crowdsourced pairs based on transitive relations.*

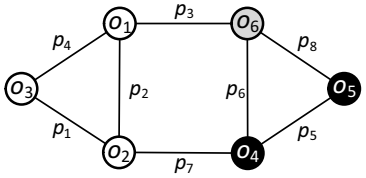
EXAMPLE 2. *Figure 3 shows eight pairs, i.e.,* $p_1, p_2 \cdots, p_8$, *generated by machine-based methods for labeling (Please ignore the Likelihood column for now). We build a graph for these pairs, where the vertices with the same grey level represent the matching objects. One possible way to label them is to crowdsource seven pairs* $p_1, p_2, p_3, p_5, p_6, p_7, p_8$. *For the other pair* p_4, *as shown in the graph, it can be deduced from* p_1 *and* p_2 *based on transitive relations. A better way to label them only needs to crowdsource six pairs* $p_1, p_2, p_3, p_5, p_7, p_8$. *For the other pairs, as shown in the graph,* p_4 *can be deduced from* p_1 *and* p_2, *and* p_6 *can be deduced from* p_5 *and* p_8. *It is not possible to further reduce the amount of crowdsourced pairs. Thus, six is the optimal amount.*

ID	Object
o_1	iPhone 2nd Gen
o_2	iPhone Two
o_3	iPhone 2
o_4	iPad Two
o_5	iPad 2
o_6	iPad 3rd Gen

(a) A set of objects

ID	Object Pairs	Likelihood
p_1	(o_2, o_3)	0.85
p_2	(o_1, o_2)	0.75
p_3	(o_1, o_6)	0.72
p_4	(o_1, o_3)	0.65
p_5	(o_4, o_5)	0.55
p_6	(o_4, o_6)	0.48
p_7	(o_2, o_4)	0.45
p_8	(o_5, o_6)	0.42

(b) A set of object pairs



(c) A graph built for the set of object pairs

Figure 3: A running example.

3. LABELING FRAMEWORK

We propose a hybrid transitive-relations and crowdsourcing labeling framework in this section. Our framework takes as input a set of unlabeled pairs generated by machine-based techniques, and identifies these pairs' labels either through crowdsourcing or by using transitive relations. As shown in Figure 4, our framework mainly consists of two components, `Sorting` and `Labeling`. Their details will be described in Sections 3.1 and 3.2, respectively.

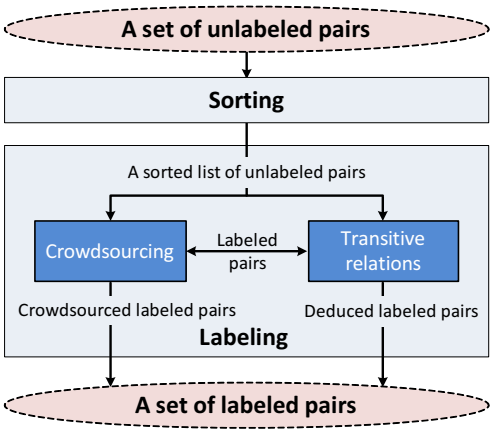


Figure 4: Hybrid transitive-relations and crowdsourcing labeling framework.

3.1 Sorting Component

Given a set of unlabeled pairs, we have an interesting finding that the labeling order of the pairs will affect the number of crowdsourced pairs. A labeling order can be taken as a sorted list of pairs, denoted by $\omega = \langle p_1, p_2, \cdots, p_n \rangle$, where p_i $(2 \leq i \leq n)$ will be labeled after $p_1, p_2, \cdots, p_{i-1}$. For example, suppose we need to label three pairs, (o_1, o_2), (o_2, o_3), (o_1, o_3), where $o_1 = o_2$ and $o_2 \neq o_3$ and $o_1 \neq o_3$. If the la-

Algorithm 1: DEDUCELABEL(p, $\mathcal{L}$)

```
   Input: p = (o, o') : an object pair; L: a set of labeled pairs
   Output: ℓ: the deduced label
1  begin
2      Build a ClusterGraph for L;
3      Let cluster(o) and cluster(o') denote the cluster of
       objects o and o' respectively;
4      if cluster(o) = cluster(o') then
5          ℓ = "matching";
6      else
7          if there is an edge between cluster(o) and cluster(o')
           then
8              ℓ = "non-matching";
9          else
10             ℓ = "undeduced";
11     return ℓ;
12 end
```

Figure 5: DeduceLabel algorithm.

beling order is $\omega = \langle (o_1, o_2), (o_2, o_3), (o_1, o_3) \rangle$, after labeling the first two pairs through crowdsourcing, we obtain $o_1 = o_2$ and $o_2 \neq o_3$. For the third pair, we can deduce $o_1 \neq o_3$ from $o_1 = o_2$ and $o_2 \neq o_3$ based on transitive relations, thus ω requires crowdsourcing *two* pairs. However, if we use a different labeling order $\omega' = \langle (o_2, o_3), (o_1, o_3), (o_1, o_2) \rangle$, after labeling the first two pairs through crowdsourcing, we obtain $o_2 \neq o_3$ and $o_1 \neq o_3$. We are unable to deduce $o_1 = o_2$ from $o_2 \neq o_3$ and $o_1 \neq o_3$ based on transitive relations, thus ω' requires crowdsourcing *three* pairs which is more than that required by ω.

Based on this observation, in our framework, the sorting component attempts to identify the optimal labeling order to minimize the number of crowdsourced pairs. Thus, it takes as input a set of unlabeled pairs and outputs a sorted list of unlabeled pairs. The details of identifying the optimal labeling order will be presented in Section 4.

3.2 Labeling Component

Given a sorted list of unlabeled pairs, the labeling component labels the pairs in the sorted order. In this section, we present a very simple, one-pair-at-a-time, labeling algorithm to achieve this goal. Consider a sorted list of pairs $\omega = \langle p_1, p_2, \cdots, p_n \rangle$. The algorithm will start with labeling from the first pair, and then label each pair one by one. When labeling the i-th pair p_i, if its label cannot be deduced from the already labeled pairs (i.e., $\{p_1, p_2, \cdots, p_{i-1}\}$) based on transitive relations, we publish p_i to a crowdsourcing platform and obtain its crowdsourced label; otherwise, we deduce its label from $p_1, p_2, \cdots, p_{i-1}$, and output the deduced label. After obtaining the label of p_i, we begin to process the next pair p_{i+1}, and use the same method to get its label. The algorithm stops until all the pairs are labeled.

Next, we discuss how to check whether an unlabeled pair $p = (o, o')$ can be deduced from a set of labeled pairs based on transitive relations. As mentioned in Section 2.2, we can build a graph for the labeled pairs, and check the graph whether there is a path from o to o' which contains no more than one non-matching pair. If there exists such a path, p can be deduced from the labeled pairs; otherwise, p cannot be deduced. One naive solution to do this checking is enumerating every path from o to o', and counting the number of non-matching pairs in each path. However, the number of enumerated paths may increase exponentially with the number of vertices in the graph, thus we propose an efficient graph-clustering-based method to solve this problem.

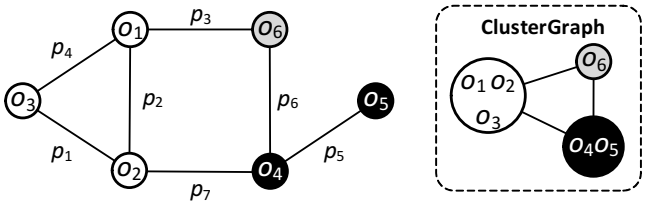


Figure 6: A ClusterGraph built for the first seven labeled pairs $\{p_1, p_2, \cdots, p_7\}$ in Figure 3.

When enumerating each path, we find that only non-matching pairs in the path can affect the checking result. In other words, the matching pairs have no effect on the checking result. This observation inspires us to merge the matching objects into the same cluster, and then, for each pair of non-matching objects, we add an edge between their corresponding clusters. We call the new graph a CLUSTERGRAPH. By using the CLUSTERGRAPH, we can efficiently check whether an unlabeled pair $p = (o, o')$ can be deduced from the already labeled pairs. Figure 5 shows the pseudocode of the algorithm. Given a set of labeled pairs $\mathcal{L}$, we first build a CLUSTERGRAPH for $\mathcal{L}$ using Union-Find algorithm [20] (Line 2). Then for the unlabeled pair $p = (o, o')$,

(1) If o and o' are in the same cluster, then there is a path from o to o' which only consists of matching pairs. Thus, p can be deduced as a matching pair (Lines 4-5);

(2) If o and o' are in two different clusters,

(2.1) If there is an edge between the two clusters, there exists a path from o to o' with a single non-matching pair. Thus, p can be deduced as a non-matching pair (Lines 7-8);

(2.2) If there is no edge between the two clusters, there does not exist a path from o to o' with no more than one non-matching pair. Thus, p cannot be deduced (Lines 9-10).

EXAMPLE 3. *Suppose we have already labeled seven pairs $\{p_1, p_2, \cdots, p_7\}$ in Figure 6. To check whether $p_8 = (o_5, o_6)$ can be deduced from them, we first build a* CLUSTERGRAPH *as follows. Since o_1, o_2, o_3 are matching, we merge them into one cluster. As o_4 and o_5 are matching, we merge them into another cluster. Since o_6 does not match with any other object, we take itself as one cluster. There are three non-matching pairs, i.e., $p_3 = (o_1, o_6)$, $p_6 = (o_4, o_6)$ and $p_7 = (o_2, o_4)$. We respectively add three non-matching edges between cluster(o_1) and cluster(o_6), cluster(o_4) and cluster(o_6), cluster(o_2) and cluster(o_4).*

Consider the unlabeled pair $p_8 = (o_5, o_6)$. In the CLUSTERGRAPH*, since o_5 and o_6 are in different clusters, and cluster(o_5) and cluster(o_6) have an edge, then there must exist a path from o_5 to o_6 which contains a single non-matching pair (e.g., $o_5 \to o_4 \to o_6$), thus $p_8 = (o_5, o_6)$ can be deduced as a non-matching pair from $\{p_1, p_2, \cdots, p_7\}$.*

The labeling algorithm needs to enumerate each pair one by one and it can only publish a single pair to the crowdsourcing platform. Hence, every time there is only one available HIT in the crowdsourcing platform. This constraint makes the workers unable to do HIT simultaneously and results in long latency. Notice that batching strategies [14,25], which place multiple pairs into a single HIT, have been proved useful in reducing the money cost. However, the simple approach is unable to support the batching techniques since only one pair is allowed to publish every time, and thus

results in more money cost. To overcome these drawbacks, we propose a parallel labeling algorithm in Section 5, which can crowdsource multiple pairs every time without increasing the total number of crowdsourced pairs.

4. SORTING

As observed in Section 3.1, different labeling orders result in different numbers of crowdsourced pairs. In this section, we explore the optimal labeling order to minimize the number of crowdsourced pairs. We begin with the formulation of this problem and discuss how to find the optimal labeling order in Section 4.1. However, the optimal order cannot be achieved in reality. Thus, in Section 4.2, we propose a heuristic labeling order.

4.1 Optimal Labeling Order

Given a labeling order $\omega = \langle p_1, p_2, \cdots, p_n \rangle$, let $\mathcal{C}(\omega)$ denote the number of crowdsourced pairs required by ω. Our goal is to identify the optimal labeling order which results in the minimum number of crowdsourced pairs. The formal definition of this problem is as follows.

Definition 2 (Optimal Labeling Order). *Given a set of object pairs, the problem of identifying the optimal labeling order is to get a sorted list of pairs, ω_{op}, such that the number of crowdsourced pairs is minimal using the order, i.e. $\mathcal{C}(\omega_{op}) \leq \mathcal{C}(\omega)$ holds for any other order ω.*

For example, assume three object pairs, $p_1 = (o_1, o_2)$, $p_2 = (o_2, o_3)$, $p_3 = (o_1, o_3)$, where p_1 is a matching pair, and p_2, p_3 are two non-matching pairs. We can label them in six different labeling orders, $\omega_1 = \langle p_1, p_2, p_3 \rangle$, $\omega_2 = \langle p_1, p_3, p_2 \rangle$, $\omega_3 = \langle p_2, p_3, p_1 \rangle$, $\omega_4 = \langle p_2, p_1, p_3 \rangle$, $\omega_5 = \langle p_3, p_1, p_2 \rangle$, and $\omega_6 = \langle p_3, p_2, p_1 \rangle$. The numbers of crowdsourced pairs of the six orders are respectively $\mathcal{C}(\omega_1) = 2$, $\mathcal{C}(\omega_2) = 2$, $\mathcal{C}(\omega_3) = 3$, $\mathcal{C}(\omega_4) = 2$, $\mathcal{C}(\omega_5) = 2$, and $\mathcal{C}(\omega_6) = 3$. As $\omega_1, \omega_2, \omega_4, \omega_5$ lead to the minimum number of crowdsourced pairs, any one of them can be seen as the optimal labeling order.

Notice that any labeling order can be changed to another labeling order by swapping adjacent pairs. We first study how swapping two adjacent pairs affects the number of crowdsourced pairs. We have an observation that it is always *better* to first label a matching pair and then a non-matching pair, i.e., it will lead to fewer or equal number of crowdsourced pairs. Recall $\omega_3 = \langle p_2, p_3, p_1 \rangle$ and $\omega_4 = \langle p_2, p_1, p_3 \rangle$ in the above example. ω_3 labels a non-matching pair p_3 before a matching pair p_1 while ω_4 swaps the positions of p_1 and p_3, and labels a matching pair p_1 before a non-matching pair p_3. As $\mathcal{C}(\omega_3) = 3$ and $\mathcal{C}(\omega_4) = 2$, the example shows that ω_4, which first labels a matching pair, needs to crowdsource fewer pairs than ω_3. Lemma 2 formulates this idea.

Lemma 2. *Consider two labeling orders,*

$$\omega = \langle p_1, \cdots, p_{i-1}, \boldsymbol{p_i}, \boldsymbol{p_{i+1}}, p_{i+2}, \cdots p_n \rangle,$$

$$\omega' = \langle p_1, \cdots, p_{i-1}, \boldsymbol{p_{i+1}}, \boldsymbol{p_i}, p_{i+2}, \cdots p_n \rangle,$$

where ω' is obtained by swapping p_i and p_{i+1} in ω. If p_i is a non-matching pair, and p_{i+1} is a matching pair, then we have $\mathcal{C}(\omega') \leq \mathcal{C}(\omega)$.

Proof. For a pair p_j, if $j \notin \{i, i+1\}$, it is easy to see that $\{p_1, p_2, \cdots, p_{j-1}\}$ in ω is the same as that in ω'. Therefore, if p_j is a crowdsourced pair in ω, it must be a crowdsourced pair in ω', and vice versa. Hence, we only need to check whether p_i, p_{i+1} are crowdsourced pairs in ω and ω'.

There are four possible cases for p_i, p_{i+1} in ω: deduced and crowdsourced pairs, crowdsourced and crowdsourced pairs, deduced and deduced pairs, or crowdsourced and deduced pairs. We prove that in any case, $\{p_i, p_{i+1}\}$ in ω' would contain fewer or equal number of crowdsourced pairs than $\{p_i, p_{i+1}\}$ in ω, thus $\mathcal{C}(\omega') \leq \mathcal{C}(\omega)$ holds.

Case 1: *p_i is deduced and p_{i+1} is crowdsourced.* As p_i in ω can be deduced from $\{p_1, p_2, \cdots, p_{i-1}\}$, in the graph built for them, there is a path from one object of p_i to the other with no more than one non-matching pairs. Such path still exists in the graph built for more pairs $\{p_1, p_2, \cdots, p_{i-1}, p_{i+1}\}$, thus p_i in ω' is also a deduced pair. Similarly, as p_{i+1} in ω cannot be deduced from $\{p_1, p_2, \cdots, p_i\}$, it cannot be deduced from fewer pairs $\{p_1, p_2, \cdots, p_{i-1}\}$, thus p_{i+1} in ω' is also a crowdsourced pair. Therefore, $\mathcal{C}(\omega') \leq \mathcal{C}(\omega)$ holds in Case 1.

Case 2: *p_i is crowdsourced and p_{i+1} is crowdsourced.* In the worst case, both p_i and p_{i+1} in ω' still need to be crowdsourced, so $\mathcal{C}(\omega') \leq \mathcal{C}(\omega)$ holds in Case 2.

Case 3: *p_i is deduced and p_{i+1} is deduced.* As p_i in ω can be deduced from $\{p_1, p_2, \cdots, p_{i-1}\}$, it can also be deduced from more pairs $\{p_1, p_2, \cdots, p_{i-1}, p_{i+1}\}$, thus p_i in ω' is also a deduced pair. As p_{i+1} in ω can be deduced from $\{p_1, p_2, \cdots, p_i\}$, it is easy to derive that p_{i+1} can also be deduced from the *crowdsourced* pairs in $\{p_1, p_2, \cdots, p_i\}$. Since p_i is not a *crowdsourced* pair based on the given condition, p_{i+1} can be deduced from the *crowdsourced* pairs in $\{p_1, p_2, \cdots, p_{i-1}\}$, thus p_{i+1} in ω' is a deduced pair. As p_i and p_{i+1} in ω' are both deduced pairs, $\mathcal{C}(\omega') \leq \mathcal{C}(\omega)$ holds in Case 3.

Case 4: *p_i is crowdsourced and p_{i+1} is deduced.* As the matching pair p_{i+1} in ω can be deduced from $\{p_1, p_2, \cdots, p_i\}$, in the graph built for $\{p_1, p_2, \cdots, p_i\}$, there exists a path from one object of p_{i+1} to the other which only consists of matching pairs. As p_i is a non-matching pair, after removing p_i from the graph, the path still exists in the graph built for $\{p_1, p_2, \cdots, p_{i-1}\}$, thus p_{i+1} can be deduced from $\{p_1, p_2, \cdots, p_{i-1}\}$. Hence, p_{i+1} in ω' is also a deduced pair. In the worst case, p_i in ω' still need to be crowdsourced, so $\mathcal{C}(\omega') \leq \mathcal{C}(\omega)$ holds in Case 4. □

For a given labeling order, by using the above method to swap adjacent object pairs, we can put all the matching pairs prior to the non-matching pairs. As it is better to label a matching pair first and then a non-matching pair, the new order will require fewer or equal number of crowdsourced pairs than the original labeling order.

Next we prove that swapping adjacent matching pairs or non-matching pairs will not change the number of crowdsourced pairs.

Lemma 3. *Consider two labeling orders,*

$$\omega = \langle p_1, \cdots, p_{i-1}, \boldsymbol{p_i}, \boldsymbol{p_{i+1}}, p_{i+2}, \cdots p_n \rangle,$$

$$\omega' = \langle p_1, \cdots, p_{i-1}, \boldsymbol{p_{i+1}}, \boldsymbol{p_i}, p_{i+2}, \cdots p_n \rangle,$$

where ω' is obtained by swapping p_i and p_{i+1} in ω. If p_i and p_{i+1} are both matching pairs or both non-matching pairs, then we have $\mathcal{C}(\omega') = \mathcal{C}(\omega)$.

Proof. Since swapping p_i and p_{i+1} will not affect the other pairs except p_i and p_{i+1} (see the proof in Lemma 2),

we only need to prove that $\{p_i, p_{i+1}\}$ in ω' requires the same number of crowdsourced pairs as $\{p_i, p_{i+1}\}$ in ω, thus $\mathcal{C}(\omega') = \mathcal{C}(\omega)$ holds. We still consider the four cases.

Case 1: *p_i is deduced and p_{i+1} is crowdsourced.* As in the proof of Case 1 in Lemma 2, we have p_i in ω' is also a deduced pair, and p_{i+1} in ω' is also a crowdsourced pair. Therefore, $\mathcal{C}(\omega') = \mathcal{C}(\omega)$ holds in Case 1.

Case 2: *p_i is crowdsourced and p_{i+1} is crowdsourced.* As p_{i+1} in ω cannot be deduced from $\{p_1, p_2, \cdots, p_i\}$, it cannot be deduced from fewer pairs $\{p_1, p_2, \cdots, p_{i-1}\}$, thus p_{i+1} in ω' is also a crowdsourced pair.

Next, we prove by contradiction p_i in ω' is also a crowdsourced pair. Assume p_i in ω' is not a crowdsourced pair. Then p_i can be deduced from $\{p_1, p_2, \cdots, p_{i-1}, p_{i+1}\}$. In the graph built for $\{p_1, p_2, \cdots, p_{i-1}, p_{i+1}\}$, there exists a path from one object of p_i to the other which contains no more than one non-matching pair. And since p_i in ω is crowdsourced, p_i cannot be deduced from $\{p_1, p_2, \cdots, p_{i-1}\}$, thus p_{i+1} must be in the path. By removing p_{i+1} from the path and adding p_i to the path, we obtain a new path from one object of p_{i+1} to the other. Since p_{i+1} is removed and p_i is added, the path must be in the graph built for $\{p_1, p_2, \cdots, p_i\}$. As p_i, p_{i+1} are either both matching pairs or both non-matching pairs, the path contains no more than one non-matching pair. Therefore, p_{i+1} can be deduced from $\{p_1, p_2, \cdots, p_i\}$ which contradicts p_{i+1} in ω is a crowdsourced pair. Hence, the assumption does not hold, and p_i in ω' is a crowdsourced pair. Since p_i, p_{i+1} in ω' are both crowdsourced pairs, $\mathcal{C}(\omega') = \mathcal{C}(\omega)$ holds in Case 2.

Case 3: *p_i is deduced and p_{i+1} is deduced.* As in the proof of Case 3 in Lemma 2, we have p_i in ω' is also a deduced pair, and p_{i+1} in ω' is also a deduced pair. Therefore, $\mathcal{C}(\omega') = \mathcal{C}(\omega)$ holds in Case 3.

Case 4: *p_i is crowdsourced and p_{i+1} is deduced.* As p_{i+1} in ω can be deduced from $\{p_1, p_2, \cdots, p_i\}$, in the graph built for $\{p_1, p_2, \cdots, p_i\}$, there exist some paths from one object of p_{i+1} to the other which contains no more than one non-matching pair. There are three cases about these paths:

(a) If none of these paths contains p_i, that is, p_i and p_{i+1} will not affect each other, then p_i in ω' is also a crowdsourced pair and p_{i+1} in ω' is also a deduced pair. Therefore, $\mathcal{C}(\omega') = \mathcal{C}(\omega)$ holds in Case 4(a).

(b) If some of these paths contain p_i but others do not, then we can infer that in the graph built for $\{p_1, p_2, \cdots, p_{i-1}\}$, there exists a path from one object of p_i to the other with no more than one non-matching pair, thus p_i can be deduced from $\{p_1, p_2, \cdots, p_{i-1}\}$ which contradicts p_i in ω is a crowdsourced pair. Hence, Case 4(b) is impossible.

(c) If all of the paths contain p_i, after removing p_i from these paths, in the graph built for $\{p_1, p_2, \cdots, p_{i-1}\}$, there will be no path from one object of p_{i+1} to the other which contains no more than one non-matching pair, thus p_{i+1} cannot be deduced from $\{p_1, p_2, \cdots, p_{i-1}\}$. That is, p_{i+1} in ω' is a crowdsourced pair. Next we prove that p_i in ω' is a deduced pair. Consider one of these paths that contain p_i. After removing p_i from the path and adding p_{i+1} to the path, we obtain a new path from one object of p_i to the other. Since p_i is removed and p_{i+1} is added, the path must be in the graph built for $\{p_1, p_2, \cdots, p_{i-1}, p_{i+1}\}$. As p_i, p_{i+1} are either both matching pairs or both non-matching pairs, the path contains no more than one non-matching pair. Therefore, p_i can be deduced from $\{p_1, p_2, \cdots, p_{i-1}, p_{i+1}\}$, thus it is a deduced pair in ω'. As p_i in ω' is a deduced pair, and p_{i+1} in ω' is a crowdsourced pair, $\mathcal{C}(\omega') = \mathcal{C}(\omega)$ holds in Case 4(c). □

For two different labeling orders, if they both first label all the matching pairs and then label the other non-matching pairs, we can change one labeling order to the other by swapping adjacent matching pairs and adjacent non-matching pairs. Based on Lemma 3, the two labeling orders require the same number of crowdsourced pairs. Therefore, any labeling order, which puts all the matching pairs to the front of the other non-matching pairs, is the optimal.

THEOREM 1. *Given a set of object pairs, the optimal labeling order is to first label all the matching pairs, and then label the other non-matching pairs.*

For example, consider the pairs in Figure 3. $\omega_{op} = \langle p_1, p_2, p_4, p_5, p_3, p_6, p_7, p_8 \rangle$ is the optimal labeling order since all the matching pairs, i.e., p_1, p_2, p_4, p_5, are labeled before the other non-matching pairs, i.e., p_3, p_6, p_7, p_8.

Now we have proved that the optimal labeling order is to first label all the matching pairs, and then label the other non-matching pairs. However, when identifying the labeling order, we have no idea about whether a pair is matching or non-matching, therefore, the optimal labeling order cannot be achieved in reality. To address this problem, we investigate an expected optimal labeling order in the next section.

4.2 Expected Optimal Labeling Order

In this section, we aim to identify a labeling order that requires as few crowdsourced pairs as possible. Recall the optimal labeling order which first labels the matching pairs and then labels the non-matching pairs. Although we do not know the real matching pairs upfront, machine-based methods can be applied to compute for each pair the likelihood that they are matching. For example, the likelihood can be the similarity computed by a given similarity function [25].

Consider a labeling order $\omega = \langle p_1, p_2, \cdots, p_n \rangle$. Suppose each pair in ω is assigned with a probability that they are matching. Then the number of crowdsourced pairs required by ω becomes a random variable. Its expected value is computed as the sum of the probability that p_i is a crowdsourced pair ($1 \le i \le n$), i.e.,

$$\mathrm{E}\big[\mathcal{C}(\omega)\big] = \sum_{i=1}^{n} \mathbb{P}(p_i = \mathsf{crowdsourced}).$$

To compute $\mathbb{P}(p_i = \mathsf{crowdsourced})$, we enumerate the possible labels of $\{p_1, p_2, \cdots, p_n\}$, and for each possibility, since the labels of $\{p_1, p_2, \cdots, p_{i-1}\}$ are known, we can derive whether p_i is a crowdsourced pair or not. Hence, $\mathbb{P}(p_i = \mathsf{crowdsourced})$ is the sum of the probability of each possibility that p_i is a crowdsourced pair.

We aim to identify a labeling order that can minimize the expected number of crowdsourced pairs since the order is expected to require the minimum number of crowdsourced pairs. We call such an order an *expected optimal labeling order*. The following definition formulates this problem.

DEFINITION 3 (EXPECTED OPTIMAL LABELING ORDER). *Given a set of object pairs, and each object pair is assigned with a probability that they are matching, the problem of identifying the expected optimal labeling order ω_{eop} is to*

compute a sorted list of pairs such that the expected number of crowdsourced pairs is minimal using the order, i.e. $\mathrm{E}[\mathcal{C}(\omega_{eop})] \leq \mathrm{E}[\mathcal{C}(\omega)]$ *holds for any other order* ω.

EXAMPLE 4. *Consider three pairs,* $p_1 = (o_1, o_2)$, $p_2 = (o_2, o_3)$ *and* $p_3 = (o_1, o_3)$. *Suppose the probabilities that* p_1, p_2 *and* p_3 *are matching pairs are respectively 0.9, 0.5 and 0.1. There are six different labeling orders,* $\omega_1 = \langle p_1, p_2, p_3\rangle$, $\omega_2 = \langle p_1, p_3, p_2\rangle$, $\omega_3 = \langle p_2, p_3, p_1\rangle$, $\omega_4 = \langle p_2, p_1, p_3\rangle$, $\omega_5 = \langle p_3, p_1, p_2\rangle$, *and* $\omega_6 = \langle p_3, p_2, p_1\rangle$. *We first compute the excepted number of crowdsourced pairs for* $\omega_1 = \langle p_1, p_2, p_3\rangle$. *For the first pair* p_1, *as there are no labeled pairs, it must need crowdsourcing, thus* $\mathbb{P}(p_1 =$ *crowdsourced*$) = 1$. *For the second pair* p_2, *as* $p_2 = (o_2, o_3)$ *cannot be deduced from* $p_1 = (o_1, o_2)$, *it must need crowdsourcing, thus* $\mathbb{P}(p_2 =$ *crowdsourced*$) = 1$. *For the third pair* p_3, *we enumerate the possible labels of* $\{p_1, p_2, p_3\}$, *i.e.,* {*matching, matching, matching*}, {*non-matching, matching, non-matching*}, {*matching, non-matching, non-matching*}, {*non-matching, non-matching, matching*}, {*non-matching, non-matching, non-matching*}. *Among the five possibilities,* p_3 *needs to be crowdsourced only when both* p_1 *and* p_2 *are non-matching pairs (i.e., the last two possibilities). Hence, the probability that* p_3 *is a crowdsourced pair is* $\frac{0.1*0.5*0.1+0.1*0.5*0.9}{0.9*0.5*0.1+0.1*0.5*0.9+0.9*0.5*0.9+0.1*0.5*0.1+0.1*0.5*0.9} = 0.09$. *By summing up the probabilities that* p_1, p_2, p_3 *are crowdsourced pairs, we have* $\mathrm{E}[\mathcal{C}(\omega_1)] = 1+1+0.09 = 2.09$. *Similarly, we can compute* $\mathrm{E}[\mathcal{C}(\omega_2)] = 2.17$, $\mathrm{E}[\mathcal{C}(\omega_3)] = 2.83$, $\mathrm{E}[\mathcal{C}(\omega_4)] = 2.09$, $\mathrm{E}[\mathcal{C}(\omega_5)] = 2.17$, *and* $\mathrm{E}[\mathcal{C}(\omega_6)] = 2.83$. *As* ω_1 *and* ω_4 *require the minimum expected number of crowdsourced pairs, either one of them can be taken as the expected optimal labeling order.*

A recent VLDB paper has proved that the problem of identifying the expected optimal labeling order is NP-hard [23]. In our paper, we propose a heuristic method to solve this problem. Recall the analysis of Section 4.1, we have proved that it is better to label a matching pair before a non-matching pair (Lemma 2). This idea inspires us to label the object pairs in the decreasing order of the likelihood that they are matching. For example, consider the unlabeled pairs $p_1, p_2, \cdots, p_8$ in Figure 3. To identify their labeling order, we first use a machine-based method to compute a likelihood for each pair that it is a matching pair, and then label the pairs in the decreasing order of the likelihood, i.e., $\omega_{eop} = \langle p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\rangle$.

5. PARALLEL LABELING

After identifying a labeling order, our labeling framework will label the unlabeled pairs in this order. In Section 3.2, we present a simple approach to achieve this goal. However, the approach only allows to publish a single pair to the crowdsourcing platform, which is unable to label the pairs simultaneously and results in long latency. To alleviate this problem, we propose a parallel labeling algorithm in Section 5.1, which can crowdsource multiple pairs every time without increasing the total number of required crowdsourced pairs. To further improve the parallelism, we present two optimization techniques in Section 5.2.

5.1 Parallel Labeling Algorithm

We first use an example to show our basic idea. Consider the labeling order $\omega = \langle (o_1, o_2), (o_2, o_3), (o_3, o_4)\rangle$. The simple labeling approach will first crowdsource the first pair (o_1, o_2), and cannot crowdsource the second pair until the first pair is labeled. However, for the second pair (o_2, o_3), we observe that no matter which label the first pair gets, we must need to crowdsource it since the second pair (o_2, o_3) cannot be deduced from the first pair (o_1, o_2). For the third pair (o_3, o_4), we have a similar observation that no matter which labels the first two pairs get, we must crowdsource it since the third pair (o_3, o_4) cannot be deduced from the first two pairs (o_1, o_2) and (o_2, o_3). Therefore, all the pairs in ω can be crowdsourced together instead of individually. Based on this idea, we propose a parallel labeling algorithm as shown in Figure 7.

Algorithm 2: PARALLELLABELING(ω)

```
   Input: ω = ⟨p_1, p_2, ··· , p_n⟩ : a sorted list of unlabeled pairs
   Output: L = {(p_i, ℓ) | 1 ≤ i ≤ n}: a set of labeled pairs
1  begin
2      L = {};
3      while there is an unlabeled pair in ω do
4          P = ParallelCrowdsourcedPairs(ω);
5          L ∪= CrowdsourceLabels(P);
6          for each unlabeled pair p ∈ ω do
7              if DeducedLabel(p, L) then
8                  Add (p, ℓ) into L;
9      return L;
10 end
```

Figure 7: Parallel labeling algorithm.

Algorithm Overview: Our parallel labeling algorithm employs an iterative strategy. In each iteration, the algorithm first identifies a set of pairs that can be crowdsourced in parallel (Line 4). Then the algorithm publishes the pairs simultaneously to the crowdsourcing platform, and obtains their crowdsourced labels (Line 5). After that, the algorithm utilizes the already labeled pairs to deduce subsequent unlabeled pairs (Lines 6-8). The algorithm repeats the iterative process until all the pairs are labeled.

A big challenge in the algorithm is to identify a set of pairs that can be crowdsourced in parallel. We know that a pair $p_i = (o, o')$ needs to be crowdsourced if and only if p_i cannot be deduced from $\{p_1, p_2, \cdots, p_{i-1}\}$.

In the case that $\{p_1, p_2, \cdots, p_{i-1}\}$ are labeled, we need to check the graph built for $\{p_1, p_2, \cdots, p_{i-1}\}$. If every path from o to o' contains more than one non-matching pair, then p_i cannot be deduced from $\{p_1, p_2, \cdots, p_{i-1}\}$. However, some pairs in $\{p_1, p_2, \cdots, p_{i-1}\}$ may have not been labeled. In this case, we have no idea about the exact number of non-matching pairs in some paths. In order to see if every path must contain more than one non-matching pair, we compute the minimum number of non-matching pairs in each path by supposing all the unlabeled pairs are matching pairs. If the minimum number of non-matching pairs in each path is larger than one, then whatever the unlabeled pairs in $\{p_1, p_2, \cdots, p_{i-1}\}$ are labeled, the number of non-matching pairs in each path must be larger than one, thus p_i cannot be deduced from $\{p_1, p_2, \cdots, p_{i-1}\}$, and needs to be crowdsourced.

Based on this idea, given a sorted list of object pairs, $\omega = \langle p_1, p_2, \cdots, p_n\rangle$, where some pairs have not been labeled, to identify which pairs can be crowdsourced in parallel, we first suppose all the unlabeled pairs are matching pairs, and then for each pair p_i $(1 \leq i \leq n)$, we out-

Algorithm 3: ParallelCrowdsourcedPairs(ω, $\mathcal{L}$)

Input: $\omega = \langle p_1, p_2, \cdots, p_n \rangle$: a sorted list of unlabeled pairs
$\mathcal{L}$: a set of labeled pairs
Output: $\mathcal{P}$: a set of pairs that can be crowdsourced in parallel

```
1  begin
2      P = {};
3      Initialize an empty ClusterGraph;
4      for i = 1 to n do
5          if (p_i, ℓ) ∈ L then
6              Insert (p_i, ℓ) into ClusterGraph;
7          else
8              p_i = (o, o');
9              if cluster(o) ≠ cluster(o') and there is no edge
               between cluster(o) and cluster(o') then
10                 Add p_i into P;
11             Insert (p_i, "matching") into ClusterGraph;
12     return P;
13 end
```

Figure 8: ParallelCrowdsourcedPairs algorithm.

put p_i as a crowdsourced pair if it cannot be deduced from $\{p_1, p_2, \cdots, p_{i-1}\}$. As discussed in Section 3.2, ClusterGraph can be utilized to efficiently decide whether p_i cannot be deduced from $\{p_1, p_2, \cdots, p_{i-1}\}$. Note that to make a decision for each p_i $(1 \leq i \leq n)$, we do not need to build the ClusterGraph from scratch since the ClusterGraph can be easily obtained by inserting p_{i-1} into the ClusterGraph built for $\{p_1, p_2, \cdots, p_{i-2}\}$.

Figure 8 shows the algorithm for identifying the pairs that can be crowdsourced in parallel in each iteration. The algorithm first initializes an empty ClusterGraph, and then checks each pair p_i $(1 \leq i \leq n)$. If p_i has already been labeled, it does not need to be crowdsourced any more, thus we update the ClusterGraph by inserting p_i, and go to the next pair p_{i+1} (Lines 5-6); otherwise, we check whether p_i can be deduced from $\{p_1, p_2, \cdots, p_{i-1}\}$. In the ClusterGraph, if cluster(o) $\neq$ cluster(o') and there is no edge between cluster(o) and cluster(o'), p_i cannot be deduced, and thus needs to be crowdsourced (Lines 9-10). In this case, since p_i is unlabeled, we suppose it is a matching pair, and insert p_i into the ClusterGraph, and go to the next pair p_{i+1} (Line 11). After checking all the pairs, we return the obtained crowdsourced pairs.

Example 5. *Consider the running example in Figure 3. Given the labeling order $\langle p_1, p_2, \cdots, p_8 \rangle$, Figure 9 shows how to use the parallel labeling algorithm to label the pairs (The solid edges represent labeled pairs and the dotted edges represent unlabeled pairs.).*

In the first iteration, since all the given pairs are unlabeled, we first suppose $\{p_1, p_2, \cdots, p_8\}$ are matching pairs, and then identify five pairs (i.e., p_1, p_2, p_3, p_5, p_6) that can be crowdsourced in parallel (the bold solid edges in Figure 9(a)). For example, p_5 is identified since it cannot be deduced from $\{p_1, p_2, p_3, p_4\}$ while p_7 is not identified since it can be deduced from $\{p_1, p_2, p_3, p_4, p_5, p_6\}$ (Note that they are supposed as matching pairs). We are able to publish $\{p_1$, p_2, p_3, p_5, $p_6\}$ simultaneously to the crowdsourcing platform. After obtaining their labels, based on transitive relations, we can deduce p_4 from p_1 and p_2, and deduce p_8 from p_5 and p_6 (the bold solid edges in Figure 9(b)). Since there still exists an unlabeled pair (i.e. p_7), we repeat the iteration process.

In the second iteration, since only p_7 is unlabeled, we first suppose p_7 is a matching pair, and then identify one pair (i.e., p_7) for crowdsourcing (the bold-solid edges in Figure 9(c)). We publish p_7 to the crowdsourcing platform. After it is labeled, we find all the pairs have been labeled, thus the algorithm is terminated, and the labeled pairs are returned.

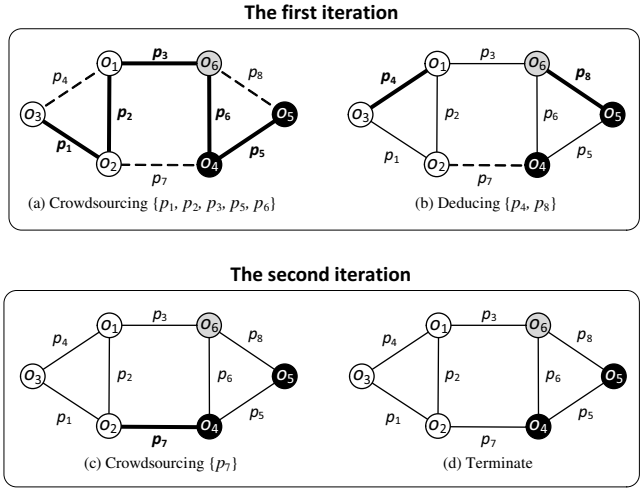


Figure 9: An illustration of parallel labeling algorithm.

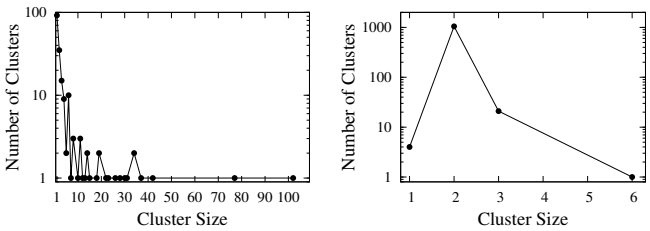
5.2 Optimization Techniques

In this section, we propose two optimization techniques, instant decision and non-matching first, to further enhance our parallel labeling algorithm.

Instant Decision: Recall our parallel labeling algorithm. The algorithm will first publish some pairs to the crowdsourcing platform, and after all the published pairs have been labeled, decide which pairs can be crowdsourced next. Notice that we do not need to wait until all the published pairs have been labeled to decide the next-round crowdsourced pairs. Instead when some of the published pairs are labeled, we can utilize them instantly to crowdsource the remaining pairs. For example, in Figure 9, we first publish $\{p_1$, p_2, p_3, p_5, $p_6\}$ together to the crowdsourcing platform. If p_3 and p_6 are labeled, we can deduce that $p_7 = (o_2, o_4)$ must be a crowdsourced pair, and can be published instantly instead of waiting for the other pairs. This is because, in the graph built for $\{p_1, p_2, p_3, p_4, p_5, p_6\}$, there are two paths from o_2 to o_4, i.e., $o_2 \to o_1 \to o_6 \to o_4$ and $o_2 \to o_3 \to o_1 \to o_6 \to o_4$. Both paths contain at least two non-matching pairs (i.e., p_3 and p_6), thus p_7 cannot be deduced from $\{p_1$, p_2, p_3, p_4, p_5, $p_6\}$ based on transitive relations.

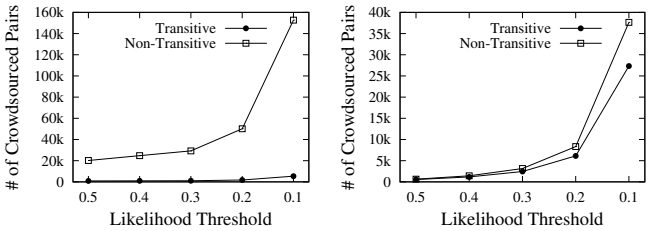
Based on this idea, we propose an optimization technique, called *instant decision*, which will make an instant decision on which pairs can be published next whenever a single published pair (instead of all the published pairs) is labeled. Achieving this goal requires a minor change to the algorithm in Figure 8 by excluding the already published pairs from $\mathcal{P}$ in Line 12. By applying the optimization technique to our parallel labeling algorithm, we are able to increase the number of the available pairs in the crowdsourcing platform to enhance the effect of parallelism.

Non-matching First: If we utilize the instant-decision optimization technique, when a published pair is labeled, we need to decide which pairs can be crowdsourced next. We



(a) Paper (b) Product

Figure 10: Cluster-size distribution.



(a) Paper (b) Product

Figure 11: Effectiveness of transitive relations.

find if the labeled pair is a matching pair, that will not lead to publishing any other pair. This is because when deciding which pairs can be crowdsourced in Figure 8, we have assumed that all the unlabeled pairs are matching pairs. Hence, knowing an unlabeled pair is a matching pair will have no effect on the algorithm. Based on this idea, we propose an optimization technique, called *non-matching first*. Consider the published pairs in the crowdsourcing platform. If we could ask the crowd workers to label the potentially non-matching pairs first, i.e., label the published pairs in the increasing order of the probability that they are a matching pair, that would increase the number of the available pairs in the crowdsourcing platform so as to enhance the effect of parallelism. It is worth noting that this order is for the published pairs in the parallel labeling algorithm, which is different from the order for labeling all pairs in Section 4.

6. EXPERIMENT

In this section, we evaluate our method. The goals of the experiments are to (1) examine the effectiveness of transitive relations in reducing the number of crowdsourced pairs, (2) compare the number of crowdsourced pairs required by different labeling orders, (3) validate the advantage of our parallel labeling algorithm over the non-parallel labeling algorithm, and (4) illustrate the performance of our method in a real crowdsourcing platform.

We used two public real-world datasets to evaluate our approaches which were widely adopted by prior works. (a) Paper (a.k.a Cora)[1] is a dataset of research publications. Each object in the dataset is a record with five attributes, *Author*, *Title*, *Venue*, *Date* and *Pages*. There are 997 distinct records, leading to $\frac{997*996}{2} = 496,506$ pairs. (b) Product (a.k.a Abt-Buy)[2] is a product dataset containing information on 1081 products from *abt.com* and 1092 products from *buy.com*. Each object is a product record with two attributes, *name* and *price*. The dataset contains a total of $1081 * 1092 = 1,180,452$ pairs.

We chose Paper and Product datasets in the experiment due to their different characteristics in the number of matching objects. To visualize the difference, we clustered the true matching objects in each dataset, and plotted the cluster-size distribution in Figure 10. We see that compared to Product, Paper has far larger clusters and should thus benefit more from using transitive relations. For example, there is a cluster consisting of 102 matching objects on the Paper dataset. For such a large cluster, using transitive relations can reduce the number of crowdsourced pairs from $\frac{102*101}{2} = 5151$ to 101. However, for smaller clusters, e.g. cluster size = 3, using transitive relations can only reduce the number of crowdsourced pairs from $\frac{3*2}{2} = 3$ to 2.

It was found that most of the pairs in the datasets look very dissimilar, and can easily be weeded out by algorithmic methods [25]. We followed this method to compute for each pair a likelihood that they are matching, and only asked the crowd workers to label the most likely matching pairs, i.e. those pairs whose likelihood is above a specified threshold.

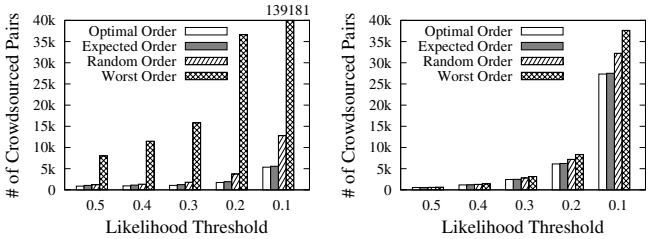
6.1 Effectiveness of Transitive Relations

In this section, we evaluate the effectiveness of transitive relations on the Paper and Product datasets. If the labeling method did not apply transitive relations, denoted by Non-Transitive, all of the pairs need to be crowdsourced. On the contrary, if the labeling method utilized transitive relations, denoted by Transitive, many pairs can be deduced based on transitive relations, and only the remaining pairs need to be crowdsourced. We varied the likelihood threshold from 0.5 to 0.1 on both the Paper and Product datasets, and respectively used Non-Transitive and Transitive to label the pairs whose likelihood is above the threshold. Figure 11 compares the number of crowdsourced pairs required by Non-Transitive and Transitive (with the optimal labeling order). On the Paper dataset, we can see Transitive reduced the number of crowdsourced pairs by 95%. For example, when the likelihood threshold was 0.3, Transitive only needed to crowdsource 1065 pairs while Non-Transitive had to crowdsource 29,281 pairs. On the Product dataset, even if there are not so many matching objects in the dataset (Figure 10), Transitive can still save about 20% crowdsourced pairs compared to Non-Transitive. For example, when the threshold is 0.2, 6134 pairs needed to be crowdsourced by Transitive while Non-Transitive required to crowdsource 8315 pairs.

6.2 Evaluating Different Labeling Orders

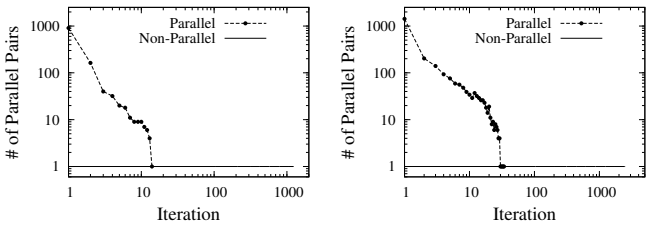
Having shown the benefits of transitive relations in reducing the crowdsourced pairs, we now turn to examining how different labeling orders affect the effectiveness of transitive relations. We compare the number of crowdsourced pairs required by different labeling orders in Figure 12. Optimal Order, Expect Order, Random Order, and Worst Order respectively denote the labeling orders which label first all matching pairs then the other non-matching pairs, label the pairs in the decreasing order of likelihood, label the pairs randomly, and label first all non-matching pairs then the other matching pairs. By comparing Worst Order with Optimal Order, we can see the selection of labeling orders has a significant effect on the number of required crowdsourced pairs. For example, on the Paper dataset, if labeling the pairs whose likelihood is above 0.1 in the worst order, we needed to crowdsource 139,181 pairs, which was about 26 times more than the crowdsourced pairs required by the op-

[1] http://www.cs.umass.edu/∼mccallum/data/cora-refs.tar.gz
[2] http://dbs.uni-leipzig.de/file/Abt-Buy.zip



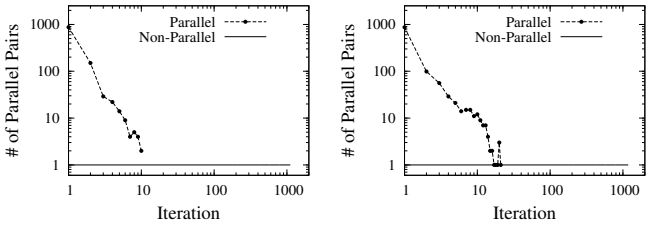
(a) Paper (b) Product

Figure 12: The number of crowdsourced pairs required by different labeling orders.



(a) Paper (b) Product

Figure 13: Parallel v.s. non-parallel labeling algorithm (likelihood threshold = 0.3).



(a) Paper (b) Product

Figure 14: Parallel v.s. non-parallel labeling algorithm (likelihood threshold = 0.4).

timal order. By comparing Expect Order and Random Order with Optimal Order, we can see that the Expect Order needed to crowdsource a few more pairs than the Optimal Order but the Random Order involved much more crowdsourced pairs, which validated that our heuristic labeling order has a very good performance in practice. Unless otherwise stated, we will use the Expect Order to label the pairs in later experiments.

6.3 Evaluating Parallel Labeling Algorithm

In this section, we evaluate our parallel labeling algorithm as well as the corresponding optimization techniques.

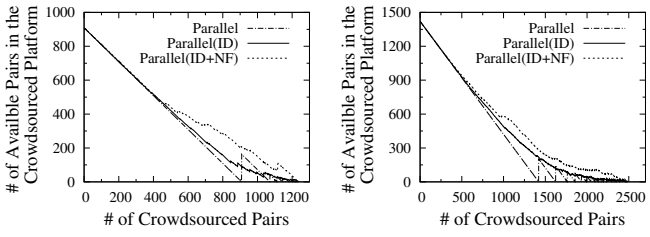
We first compare the parallel labeling algorithm (referred to as Parallel) with the non-parallel labeling algorithm (referred to as Non-Parallel). We respectively used Parallel and Non-Parallel to label the pairs whose likelihood was above 0.3. Figure 13 illustrates their number of parallel pairs in each iteration. Compared to Non-Parallel, Parallel significantly reduced the total number of iterations. For example, on the Paper dataset, there were a total of 1237 crowdsourced pairs. For this, Non-Parallel required 1237 iterations, i.e., in each iteration only a single pair could be crowdsourced. But Parallel reduced the number of iterations to 14, where in each iteration, 908, 163, 40, 32, 20, 18, 11, 9, 9, 9, 7, 6, 4, and 1 pair(s) respectively have been crowdsourced in parallel. We also evaluated Parallel for other likelihood thresholds, and found that a better performance can be achieved for higher likelihood thresholds. For example, Figure 14 shows the result for a threshold of 0.4. Comparing to the result in Figure 13 (threshold=0.3), Parallel involved fewer iterations on both datasets. This is because for a larger threshold, there were fewer number of pairs whose likelihood was above the threshold. Thus the graph built for the pairs became more sparse, and allowed to crowdsource more pairs per iteration.

Next we evaluate optimization techniques for parallel labeling algorithm. In Figure 15, Parallel, Parallel(ID), and Parallel(ID+NF) respectively denote the parallel algorithm without any optimization technique, the parallel algorithm with the instant-decision optimization technique, and the parallel algorithm with both instant-decision and non-matching-first optimization techniques. (Note that the parallel algorithm with only the non-matching-first optimization technique is the same as Parallel.) At the beginning, all of the three algorithms published a set of pairs to the crowdsourcing platform, and then waited for the crowd workers to label them. Parallel and Parallel(ID) were supposed to label the pairs randomly while Parallel(ID+NF) was supposed to first label the most unlikely matching pairs. When a pair was labeled, Parallel would not publish any new pairs until all the pairs in the crowdsourcing platform had been labeled, whereas both Parallel(ID) and Parallel(ID+NF) would instantly decide which pair to publish next. Figure 15 illustrates that the number of available pairs in the crowdsourcing platform changed with the increasing number of pairs labeled by the crowd. Unlike Parallel, Parallel(ID) and Parallel(ID+NF) ensured that at any time, there were sufficient pairs available in the crowdsourcing platform, which kept the crowd doing our work continuously. For example, on the Product dataset, after 1420 pairs were crowdsourced, Parallel only had one available pair in the crowdsourcing platform while Parallel(ID) and Parallel(ID+NF) respectively had 219 pairs and 281 pairs in the crowdsourcing platform. In addition, we can also see from the figure that Parallel(ID+NF) lead to more available pairs than Parallel(ID), which validated the effectiveness of the non-matching-first optimization technique.

6.4 Evaluating our approaches in a real crowdsourcing platform

Finally we evaluate our approaches with AMT. We paid workers 2 cents for completing each HIT. In order to reduce the cost, we adopted a batching strategy [14,25] by placing 20 pairs into one HIT. To control the result quality, each HIT was replicated into three assignments. That is, each pair would be labeled by three different workers. The final decision for each pair was made by majority vote.

We first compare Parallel(ID) with Non-Parallel in AMT using a threshold of 0.3. (We were unable to evaluate Parallel(ID+NF) in AMT since the current AMT can only randomly assign HITs to workers.) As we only focused on the difference between their completion time, we simulated that the crowd in AMT always gave us correct labels. In this way, the two algorithms would crowdsource the same number of pairs, thus requiring the same amount of money. As discussed in Section 3.2, the batching strategy is not applicable to Non-Parallel. To make a fair comparison, Non-Parallel used the same HITs as Parallel(ID), but published a single



(a) Paper (b) Product

Figure 15: Optimization techniques for the parallel labeling algorithm (likelihood threshold = 0.3).

one per iteration. Table 1 compares their completion time. We can see Parallel(ID) significantly improved the labeling performance over Non-Parallel. For example, on the Paper dataset, if we used Non-Parallel to publish 68 HITs in a non-parallel way, we had to wait for 78 hours. However, if we published them in parallel, the waiting time reduced by almost one order of magnitude.

Table 1: Comparing Parallel(ID) with Non-Parallel in AMT (likelihood threshold = 0.3).

Dataset	# of HITs	Non-Parallel	Parallel(ID)
Paper	68	78 hours	8 hours
Product	144	97 hours	14 hours

In order to evaluate the effectiveness of transitive relations with AMT, we respectively used Transitive and Non-Transitive to label the pairs whose likelihood was above 0.3, where Transitive adopted Parallel(ID) to label the pairs in the Expect Order, and Non-Transitive simply published all the pairs simultaneously to the crowdsourcing platform. We compared Transitive with Non-Transitive in terms of completion time, number of HITs, and result quality. Table 2 shows the respective results on the Paper and Product datasets. We employed *Precision*, *Recall*, and *F-measure* to evaluate the result quality. Let tp denote the number of correctly labeled matching pairs, fp the number of wrongly labeled matching pairs, and fn the number of falsely labeled non-matching pairs. Precision and recall are respectively defined as $\frac{tp}{tp+fp}$ and $\frac{tp}{tp+fn}$. F-measure is defined as the harmonic mean of precision and recall, i.e. $\frac{2 \cdot precison \cdot recall}{precison+recall}$.

We used qualification tests to improve the result quality. A qualification test consisted of three specified pairs. Only the workers who correctly labeled the pairs were allowed to do our HITs. In Table 2(a), to label 29,281 pairs on Paper dataset, Non-Transitive published $\frac{29281}{20} = 1465$ HITs, and waited for 755 hours until all the HITs were completed, whereas Transitive can reduce the HITs by 96.5% and the time by 95.8% with about 5% loss in the result quality. This experimental result indicates that for the dataset with a lot of matching objects, Transitive can save a large amount of cost and time with a little loss in result quality. The reason for the loss of quality is that some pairs' labels were falsely deduced from incorrectly labeled pairs based on transitive relations.

Next we turn to the experimental result on Product dataset. In Table 2(b), 3154 pairs needed to be labeled, and Non-Transitive published $\frac{3154}{20} = 158$ HITs, and waited for 22 hours until all the HITs were completed. Since there are not so many matching objects in the dataset, Transitive can only save about 10% of the HITs. Due to the iterative process of publishing HITs, Transitive lead to a little longer completion time. But in terms of quality, Transitive was almost the same as Non-Transitive. This experimental result indicates that for the dataset with not so many matching objects, transitive relations can help to save some money with almost no loss in result quality but may lead to longer completion time.

Table 2: Comparing Transitive with Non-Transitive in AMT (likelihood threshold = 0.3)

(a) Paper

	# of HITs	Time	Quality		
			Precision	Recall	F-measure
Non-Transitive	1465	755 hours	68.82%	95.03%	79.83%
Transitive	52	32 hours	62.96%	90.47%	74.25%

(b) Product

	# of HITs	Time	Quality		
			Precision	Recall	F-measure
Non-Transitive	158	22 hours	95.69%	68.94%	80.14%
Transitive	144	30 hours	94.70%	68.82%	79.71%

7. RELATED WORK

Recently, several projects on crowd-enabled query processing system [4,14,16] and hybrid crowd-machine data integration system [9] were proposed in the database community. To implement such systems, there are many studies in processing a variety of crowdsourced queries [3,6,14,17,18, 21,22,25,27]. As one of the most important queries, crowdsourced joins have been widely investigated in [3,14,25,27]. Marcus et al. [14] proposed a human-only technique with some batching and feature filtering optimizations for crowdsourced joins. Wang et al. [25] developed a hybrid human-machine workflow which first utilized machine-based techniques to weed out a large number of obvious non-matching pairs, and only asked the crowd workers to label the remaining pairs. Demartini et al. [3] also employed a hybrid human-machine technique, and in addition, they developed a probabilistic framework for deriving the final join result. Whang et al. [27] proposed a budget-based method for crowdsourced joins which assumed there was not enough money to label all the pairs, and explored how to make a good use of limited money to label a certain number of pairs. When using the crowd workers to label a set of pairs, the prior works neglected the fact that transitive relations hold among the pairs. Therefore, our work complements them by leveraging transitive relations to reduce the number of crowdsourced pairs. In a recent technical report, Gruenheid et al. [5] also explored how to leverage transitive relations for crowdsourced joins, but they mainly studied how to decide whether two objects refer to the same entity when crowd workers give inconsistent answers, which has a different focus than our work.

Some real applications such as entity resolution [1,15,28] also seek to benefit from transitive relations. Essentially, these works first label pairs using some sophisticated algorithms, and then utilize transitive relations to obtain the final result. They mainly focused on how to resolve the conflicts introduced by transitive relations rather than reduce the precious human effort. In a pay-as-you-go data integration system, Jeffrey et al. [8] studied the problem of minimizing the human work to achieve the best data-integration quality. But their approach aimed to identify the most uncertain pairs for verification, without considering the benefits of transitive relations.

There are also some studies on parallel crowdsourcing. Little et al. [11] compared iterative crowdsourcing model with parallel crowdsourcing model in a variety of problem domains, and provided some advice about the selection of models. TurKit [12] is a toolkit based on the crash-and-rerun programming model which makes it easier to write parallel crowdsourcing algorithms. CrowdForge [10] is a map-reduce style framework which partitions a complex task into subtasks that can be done in parallel. These tools can help us to easily implement the parallel labeling algorithm in a real crowdsourcing platform.

8. CONCLUSION AND FUTURE WORK

We studied the problem of leveraging transitive relations for crowdsourced joins (e.g., for entity resolution), to minimize the number of crowdsourced pair verifications. Our approach consists of two components: (1) The sorting component takes the pre-matched pairs from a machine-based method and determines the best order for verification. We found that the labeling order has a significant effect on the number of crowdsourced pairs. We proved that the optimal labeling order, which minimizes the number of crowdsourced pairs, has to first label all the matching pairs, and then label the other non-matching pairs. As this is impossible to achieve (we do not know the real matching pairs upfront), we proposed a heuristic labeling order that labels the pairs in the decreasing order of the probability that they are a matching pair. (2) For the labeling component, we found that a simple labeling method lead to longer latency and increased cost. We devised a novel parallel labeling algorithm to overcome these drawbacks. We have evaluated our approaches, both using simulation and with AMT, and showed, that transitive relations can lead to significant cost savings with no or little loss in result quality.

Various future directions exist for this work including to detect non-transitive relations, automate money/time/quality trade-offs for joins, explore other kinds of relations (e.g. one-to-one relationship) or extend to non-equality joins (i.e., general theta-joins). In this work, we showed that transitive relations can lead to significant cost savings in crowdsourced joins for entity resolution.

Acknowledgements. This work was partly supported by the National Natural Science Foundation of China under Grant No. 61003004 and 61272090, National Grand Fundamental Research 973 Program of China under Grant No. 2011CB302206, and a project of Tsinghua University under Grant No. 20111081073, and the "NExT Research Center" funded by MDA, Singapore, under Grant No. WBS:R-252-300-001-490, and NSF CISE Expeditions award CCF-1139158 and DARPA XData Award FA8750-12-2-0331, and gifts from Amazon Web Services, Google, SAP, Blue Goji, Cisco, Clearstory Data, Cloudera, Ericsson, Facebook, General Electric, Hortonworks, Huawei, Intel, Microsoft, NetApp, Oracle, Quanta, Samsung, Splunk, VMware and Yahoo!.

9. REFERENCES

[1] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine Learning*, 56(1-3):89–113, 2004.

[2] O. Dekel and O. Shamir. Vox populi: Collecting high-quality labels from a crowd. In *COLT*, 2009.

[3] 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.

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

[5] A. Gruenheid, D. Kossmann, S. Ramesh, and F. Widmer. Crowdsourcing entity resolution: When is A=B? Technical report, ETH Zürich.

[6] S. Guo, A. G. Parameswaran, and H. Garcia-Molina. So who won?: dynamic max discovery with the crowd. In *SIGMOD Conference*, pages 385–396, 2012.

[7] P. G. Ipeirotis, F. Provost, and J. Wang. Quality management on amazon mechanical turk. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, pages 64–67, 2010.

[8] S. R. Jeffery, M. J. Franklin, and A. Y. Halevy. Pay-as-you-go user feedback for dataspace systems. In *SIGMOD Conference*, pages 847–860, 2008.

[9] S. R. Jeffery, L. Sun, M. DeLand, N. Pendar, R. Barber, and A. Galdi. Arnold: Declarative crowd-machine data integration. *CIDR*, 2013.

[10] A. Kittur, B. Smus, S. Khamkar, and R. E. Kraut. CrowdForge: crowdsourcing complex work. In *UIST*, pages 43–52, 2011.

[11] G. Little, L. B. Chilton, M. Goldman, and R. C. Miller. Exploring iterative and parallel human computation processes. In *Proceedings of the ACM SIGKDD workshop on human computation*, pages 68–76. ACM, 2010.

[12] G. Little, L. B. Chilton, M. Goldman, and R. C. Miller. TurKit: human computation algorithms on mechanical turk. In *UIST*, pages 57–66, 2010.

[13] X. Liu, M. Lu, B. C. Ooi, Y. Shen, S. Wu, and M. Zhang. CDAS: A crowdsourcing data analytics system. *PVLDB*, 5(10):1040–1051, 2012.

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

[15] A. E. Monge and C. Elkan. An efficient domain-independent algorithm for detecting approximately duplicate database records. In *DMKD*, pages 23–29, 1997.

[16] A. Parameswaran, H. Park, H. Garcia-Molina, N. Polyzotis, and J. Widom. Deco: Declarative crowdsourcing. Technical report, Stanford University. http://ilpubs.stanford.edu:8090/1015/.

[17] A. G. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom. CrowdScreen: algorithms for filtering data with humans. In *SIGMOD Conference*, pages 361–372, 2012.

[18] A. G. Parameswaran, A. D. Sarma, H. Garcia-Molina, N. Polyzotis, and J. Widom. Human-assisted graph search: it's okay to ask questions. *PVLDB*, 4(5):267–278, 2011.

[19] A. D. Sarma, A. Parameswaran, H. Garcia-Molina, and A. Halevy. Finding with the crowd. Technical report, Stanford University.

[20] R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2):215–225, 1975.

[21] B. Trushkowsky, T. Kraska, M. J. Franklin, and P. Sarkar. Crowdsourced enumeration queries. In *ICDE*, pages 673–684, 2013.

[22] P. Venetis, H. Garcia-Molina, K. Huang, and N. Polyzotis. Max algorithms in crowdsourcing environments. In *WWW*, pages 989–998, 2012.

[23] N. Vesdapunt, K. Bellare, and N. Dalvi. Crowdsourcing algorithms for entity resolution. *PVLDB*, 7(12):1071 – 1082, 2014.

[24] P. Wais, S. Lingamneni, D. Cook, J. Fennell, B. Goldenberg, D. Lubarov, D. Marin, and H. Simons. Towards building a high-quality workforce with mechanical turk. *Proceedings of Computational Social Science and the Wisdom of Crowds (NIPS)*, pages 1–5, 2010.

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

[26] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng. The Expected Optimal Labeling Order Problem for Crowdsourced Joins and Entity Resolution. *ArXiv e-prints*, Sept. 2014. `http://tiny.cc/eolo`.

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

[28] S. E. Whang, D. Menestrina, G. Koutrika, M. Theobald, and H. Garcia-Molina. Entity resolution with iterative blocking. In *SIGMOD Conference*, pages 219–232, 2009.