Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Technical Report Nr. 785

Systems Group, Department of Computer Science, ETH Zurich

Crowdsourcing Entity Resolution: When is A=B?

by
Anja Gruenheid, Donald Kossmann, Sukriti Ramesh, Florian Widmer

September 2012

inf | Informatik
Computer Science

Abstract

There are several computational tasks for which the help of people is useful. One such task is entity resolution. For this task, human experts can help to identify whether two customers are identical given their profile. Since crowdsourcing is expensive, the goal is to ask as few questions as possible. At the same time, high quality results can only be achieved if several experts are asked for their opinion and for confirmation. This paper shows how to address this cost / quality trade-off and how to tolerate and resolve errors from the crowd. Specifically, this paper shows how to exploit mathematical properties such as symmetry, transitivity, and anti-transitivity of the *is-same-entity-as* relation to improve both cost and quality. The results of extensive experiments provide surprising insights on how best to crowd-source for entity resolution and other classification problems.

1 Introduction

Entity resolution is one of the oldest open problems in the database community. It has been studied since the Nineties (e.g., [11]) and is still an active area of research (e.g., [10]). The reason for this interest is that entity resolution is both important and hard. It is important because there are numerous applications that depend on efficient and effective entity resolution. Traditional applications include customer relationship management in the telecommunication and financial industry. A bank, for instance, wants to have a consolidated view of all the accounts and portfolios of a customer in order to offer the most appropriate financial products and avoid embarrassing situations such as charging fees from a wealthy customer whose checking account incidentally runs into a negative balance. Recently, entity resolution has gained revived interest as part of *Open Data* initiatives. Freebase is the most prominent example. Freebase tries to establish a comprehensive knowledge base that maintains the relationships of (virtually) all entities of the world; the usefulness of the Freebase knowledge base depends heavily on the ability to identify whether two nodes in the Freebase graph refer to the same real-world entity.

Despite of more than twenty years of efforts to solve the entity resolution problem automatically, it still remains an open problem. Problems like entity resolution are often referred to as *AI-complete* because they involve human-like intelligence. It is, for instance, easy for a human to detect that "IBM" and "Big Blue" refer to the same real-world entity whereas this task is almost impossible for a machine without prior knowledge and context. To systematically address such AI-complete problems, crowd-sourcing has recently become popular. The idea of crowd-sourcing is to integrate human input into the execution and to let machines do what they are best at (e.g., number crunching) and humans do what they are best at (e.g., doing comparisons such as "IBM" = "Big Blue"). In the database community, such *hybrid systems* that involve humans and machines have been studied as part of the CrowdDB [7], Deco [16], and Qurk [14] projects. The goal of this paper is to lay the foundations for carrying out entity resolution in such hybrid systems. In fact, Freebase is exactly such a hybrid system which relies heavily on crowdsourcing in order to carry out entity resolution.

1.1 Running Example

To illustrate the issues related to crowdsourcing for entity resolution with the help of crowdsourced information, we would like to give the following example which we will use throughout this paper. The director of the Uffizi, the famous museum in Florence, would like to rearrange the exhibition of paintings and group all the paintings by painter. That is, she would like to have all the paintings of Botticelli in one room, all the paintings of Lippi in another room, etc. To complicate the situation, the archives of the Uffizi are big and contain thousands of old paintings whose painters are unknown. Fortunately, there are a number of talented students who are eager to help. Given two paintings, a student can guess whether the two paintings were created by the same painter based on an analysis of the painting styles. Since the students are human, they make mistakes so it is wise to ask several students for confirmation.

At first glance, traditional entity resolution algorithms that try to automate every-

thing are directly applicable to the classification problem of the director of the Uffizi. Unfortunately, these algorithms make a number of assumptions that do not if most of the information needs to be crowdsourced as in the Uffizi example. The first assumption made by traditional algorithms is that all comparisons are deterministic. This assumption is reasonable in a fully automated scenario in which comparisons are implemented using, e.g., machine learning techniques, but this assumption obviously does not hold if humans carry out comparisons; for instance, two students may come to a different conclusion given a pair of paintings. A second assumption made in most traditional algorithm is that comparisons are cheap or a complete graph that materializes the probabilities that two paintings are from the same artist is known in advance. Again, this assumption is natural for a fully automated entity resolution system, but does not hold for crowdsourced entity resolution: At the beginning, the director of the Uffizi may know the artists of some paintings, but she will need to pay students to compare a great deal of the paintings. As a result, in the context of crowd-sourcing, entity resolution needs to be completely rethought. (We will revisit this aspect when we discuss related work in Section 2.)

A naïve approach to classify all paintings is to ask the students to compare each pair of paintings thereby carrying out a total of $\mathcal{O}(n^2)$ student comparisons, with n the number of paintings. This approach is wasteful. If several students have determined that 'Primavera' and 'Annunciation' have been painted by the same painter (i.e., Botticelli) and that 'Primavera' and 'Calumny' also have been painted by the same painter, then there is no need to ask whether 'Calumny' and 'Annunciation' have been painted by the same painter. This information can be inferred by exploiting the *transitivity* of the *is-painted-by-the-same-painter* relationship. Analogously, the director of the Uffizi can exploit *anti-transitivity*: If it is known that 'Primavera' and 'Annunciation' were painted by the same painter and 'Primavera' and 'Allegory' were painted by different painters, then the director can infer that 'Annunciation' and 'Allegory' were painted by different painters without asking her students. In general, all classification problems including entity resolution involve an equivalence relation which is reflexive, symmetric, transitive, and anti-transitive. Given these mathematical properties, it is possible to run the following simple, generic algorithm to group all paintings by painter:

- **Step 1**: Randomly select two paintings that have not been compared yet.
- **Step 2**: If it can be inferred that these two paintings have been painted by the same or different painters, then exploit that fact. Otherwise, ask students to compare the two paintings and remember the result.
- **Step 3**: Goto Step 1 until all pairs of paintings have been compared.

Step 2 is the heart of this algorithm. This step determines the cost and quality of doing crowd-based entity resolution and is the focus of this paper. We will discuss variants of how to implement this step and thoroughly evaluate these variants. We will also discuss more sophisticated variants of the entire algorithm (Section 7), but as will be shown in the experiments, good results can be achieved even with this simple three step algorithm if Step 2 is implemented efficiently. Studying sophisticated variants of the entire algorithm in detail is left for future work.

1.2 Crowdsourced Comparisons

There are two main issues in Step 2 of the simple algorithm above. First, inference is difficult if the judgement of the students is error-prone. In such situations, the votes of the students can be contradictory and the system must make the best given the information it gets from the students. We will illustrate in this work how to make decisions when such conflicts arise.

A second challenge for entity resolution based on information collected through the crowd is to decide which paintings to compare in Step 2 of the algorithm above. We call this problem the *next-crowdsource* problem. We will show that it is *not* always best to crowd-source the two paintings selected in Step 1 directly. Instead, the choice depends on information that has been gained from previous crowdsourcing.

The main result of this paper is that clever inference and selection of pairs of paintings to crowd-source can result in low cost and high result quality even if the students give input with a high error rate. This result is somewhat surprising and a major difference to the behavior of traditional database systems. Traditional database systems exhibit a 'garbage-in / garbage-out' behavior; the higher the error rate, the lower the quality of results. To guarantee this behavior, the fundamental assumption made is that the students do not err in a systematic way. If the students are malicious and deliberately give wrong answers or have all attended lectures by the same professor who miseducated them, then no high quality results can be guaranteed.

1.3 Contributions and Overview

Overall, this paper makes three main contributions:

1. **Inference**: We show how to exploit mathematical properties such as transitivity and anti-transitivity in order to minimize cost (i.e., minimize the number of times that the crowd needs to be asked for input). One particularly important feature is that our technique is able to tolerate errors.

2. **Next-crowdsource Problem**: We present an algorithm that determines the em most promising pair of paintings to compare next with the help of crowd-sourcing. Most promising here means the pair of paintings that are likely to result in overall least cost and highest quality.

3. **Evaluations**: We discuss the results of comprehensive experiments that assess the cost and result quality of the proposed Inference and Next-crowdsource techniques thereby varying the error rate of the crowd input. The surprising result is that the proposed techniques produce high quality results (less classification errors) than a high-cost variant that always asks the crowd and uses a high quorum for crowdsourcing.

The remainder of this paper is structured as follows: Section 2 discusses related work. Section 3 lays the foundations for our algorithms by providing definitions and explaining the data structures used to infer comparisons. Section 4 shows how to tolerate error-prone crowd input. Section 5 addresses the *next-crowdsource* problem and when to stop crowd-sourcing. Section 6 presents the results of experiments that study the cost

and quality of alternative approaches to carry out crowd-based entity resolution. Section 7 sketches avenues for planned future work. Section 8 contains conclusions.

2 Related Work

Entity resolution is a well-established research topic in the database community and has been studied extensively in the past. A number of sophisticated algorithms have been proposed in order to automatically de-duplicate entities based on various features, for example [4, 5, 11]. Furthermore, the machine learning community has devised several techniques to address classification problems. In particular, tools like GATE [8] or Minorthird [2] can be used for classifying text and information extraction from text. In addition, the SERF project at Stanford has devised a number of different techniques to support entity resolution tasks. A good overview of the SERF project is given in [23].

Among others and most relevant to our work, the SERF project studied the Swoosh algorithm for *generic entity resolution* [1]. The goal of Swoosh is to minimize the number of comparisons which means to reduce the cost of entity resolution and increase the performance. Swoosh exploits mathematical properties of entity resolution operations such as the associativity of the *merge* operation in order to reduce the number of comparisons needed to carry out an entity resolution task. What differentiates our work from Swoosh and any existing work on entity resolution that we are aware of is that we explicitly model and handle errors and inconsistencies in the comparison function and try to resolve them using a decision function. In contrast, all comparisons and entity resolution operations in Swoosh are assumed to be deterministic which means that calling the comparison function twice on the same pair of paintings is assumed to always return the same result in Swoosh. This assumption is obviously not true if the comparison is crowdsourced where different human experts may return different results. Additionally, this approach assumes that the knowledge about the relationship between different records is *complete*. This implicates that if an edge exists resp. not exists between any two records, this (non-) existence is associated with a specific meaning. In contrast, in a crowd environment the non-existence of an edge signifies that no information is known about the relationship between those two records due to cost-efficiency reasons as explained in the introduction. Thus, entity resolution has to function on an *incomplete* graph.

Recently, there have been several proposals to enhance database systems with crowdsourcing. Examples are CrowdDB [7], Deco [16], and Qurk [14]; an overview of these systems is given in [3]. Furthermore, there has been work on implementing certain database operators using crowdsourcing, most notably sorts and joins [13]. [9] shows how the *maximum* function (or Top 1 function) can be crowdsourced using tournament algorithms. There are some similarities between the approach proposed in [9] and our work. In particular, [9] also models crowd input as a graph and it devises several decision functions and heuristics to find the maximum with as high precision as possible.

Initial work on crowd-based entity resolution was published in [22, 24]. The goal of that work is to prune the search space of entity resolution (i.e., eliminate candidates) before querying the crowd in order to reduce the cost. In CrowdER [22] an initial filter

in form of an automatic similarity evaluation of the records is applied. The amount of questions that the crowd is asked is thus reduced to questions that refer to similar records measured by syntactic closeness. In contrast to our approach, this technique relies on the correct identification of similarity between records by the computer which is highly dependent on the applied similarity measurements. Thus, the disadvantage of this technique is that if the measurements are susceptible to erroneous data such as missing information or false values, applying them leads to a loss in result quality.
Pay-As-You-Go entity resolution [24] in contrast aims to exploit *hints* such as an ordered list of likelihood matches to construct entities on the go. Thus, the main constraint is to construct a good entity resolution given a limited number of resources. Note that the entity resolution algorithms are not in the focus of this work but the authors explore how differently structured hints benefit crowd-based entity resolution. These hints are again based on automatic processing of the data which is in contrast to our approach which retrieves all the information used for the entity resolution purely from the crowd and then applies an adapted entity resolution algorithm that takes the changed requirements of the crowd environment into consideration.

Other related areas of work include probabilistic databases and fuzzy logic [20, 12]. In the last decade, probabilistic databases have been proposed to model uncertainty. Again, this work is directly applicable to our problem. As probabilistic databases they can present the underlying structure for the entity resolution efficiently and using possible worlds semantics the most likely classification or deduplication could be determined. To the best of our knowledge, nobody has ever studied algorithms to carry out entity resolution in a probabilistic database with possible worlds semantics. It is likely that doing so is $\mathcal{NP}$-hard and computationally intractable even for small data sets. In some sense, our approach of inferring equivalences with a specified decision function can be seen as one possible approach,the first that we are aware of, to compute the most likely classification from a probabilistic structure. Fuzzy sets in contrast model that a glass is half full (or half empty). Even though fuzzy logic provides several important tools including a theory of decision functions, fuzzy logic is not applicable to our specific scenario. Applied to our running example, fuzzy logic could model that two paintings are partially painted by the same painter. This kind of modeling does not really make sense for this kind of classification problem because the ground truth is binary and not fuzzy.

3 Error-Free Crowdsourcing

This section lays the foundations for inferring new results from crowdsourced input. For ease of presentation, we assume throughout this section that all the input from the crowd is correct and consistent. That is, crowd workers are perfect in their judgment whether two records belong to the same entity. Section 4 then explains how we propose to tolerate errors and inconsistencies.

Student	*Painting 1*	*Painting 2*	*Same Painter?*
Mary	Calumny	Primavera	Yes
John	Allegory	Calumny	No
Mary	Allegory	Calumny	No
John	Annunciation	Primavera	Yes
John	Allegory	Saint Gerome	Yes
Mary	Allegory	Saint Gerome	Yes
Mary	Flora	Saint Gerome	No
John	Annunciation	Flora	No

Table 1: Example Votes Table

3.1 Voting Scheme

Entity resolution refers to the task of finding the real-world entities pointed to by a set of records in a data set. In our running example, an entity contains the set of paintings that were created by the same painter. If we cannot determine algorithmically whether two paintings have the same painter, crowd workers are asked for their opinion on the relationship of the paintings. The basic data structure to capture all the input of the crowd is the `Votes` table as shown for the running example in Table 1. It captures a snapshot D_t of all the input provided by students at a certain point t in time. As the algorithm collects more input from the crowd, the size of the `Votes` table increases, therefore $|D_{t_1}| \leq |D_{t_2}|$ if $t_1 < t_2$. More information enables the algorithm to return a result with a higher quality. To support more efficient look-ups in the `Votes` table, the lexicographically smaller painting is stored in the *Painting 1* column, thereby exploiting the symmetry of the *is-painted-by-the-same-painter* relation.

A natural way to visualize votes is to represent them as edges in a graph. Figure 1 shows the graph that corresponds to Table 1.

Definition 1 *Votes Graph. A votes graph is the representation of a set of records as nodes in a graph. The nodes are connected by weighted edges that represent the votes of the crowd.*

There are two kinds of edges: Edges with a positive weight represent votes indicating that two records belong to the same entity. In our running example, a positive weight would thus represent the number of students who believe that the paintings were created by the same painter. Edges with a negative weight represent votes that indicate that two records are not the same. Again, the weight indicates the number of students that have voted in this way. For instance, the weight of the 'Allegory-Calumny' edge is -2 in Figure 1 indicating that two students believe that these two paintings are from different artists.

3.2 Inference

If all students are perfect, inference is simple and straight-forward. In a nutshell, two records are equal (e.g., two paintings have the same painter) if there is at least one path

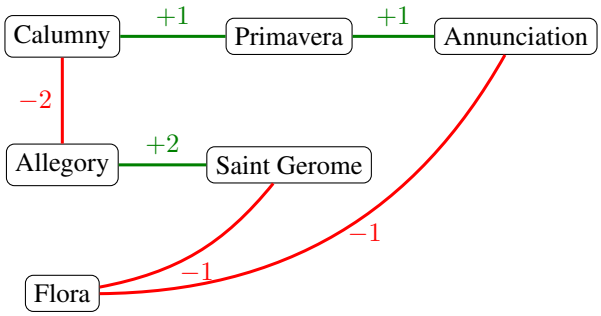

Figure 1: Example Votes Graph (Table 1)

that connects the two records in the graph and this path has only edges with a positive weight. Correspondingly, it can be inferred that two records are not the same if there is a path that connects the corresponding two nodes and this path has exactly one negative edge. Paths with two or more negative edges are ignored for inference: If A is different from B and B is different from C, no conclusion can be drawn with regard to A and C (they may or may not be the same).

Definition 2 *Positive Path. A path is categorized as a positive path if it connects two records and all edges on the path have a positive weight.*

Definition 3 *Negative Path. A path is categorized as a negative path if it connects two records and exactly one of the edges on the path has a negative weight.*

Example 1 *From the votes graph of Figure 1, it can be concluded that 'Calumny' and 'Annunciation' were created by the same painter (i.e., Botticelli). There is one positive path (i.e., 'Calumn-Primavera-Annunciation') and no negative path. On the other hand, 'Saint Gerome' and 'Calumny' were created by different painters according to the votes graph of Figure 1 because there is one negative path (i.e., 'Saint Gerome-Allegory-Calumny') and no positive path.*

Fortunately, it is also straightforward to implement this inference scheme by connectivity with positive and negative paths using a relational database system. We first show how to represent and detect positive paths efficiently. The key idea is to define a *representative* for each group of paintings created by the same painter and connect each painting of that group to that representative [18]. Logically, a connected sub-graph (with positive edges) is represented by a *star* with the representative being in the center of the star. Table 2 illustrates this idea for the running example. 'Annunciation', 'Calumny', and 'Primavera' have all been created by the same painter (i.e., Botticelli). They form one entity with representative 'Calumny'. 'Allegory' and 'Saint Gerome' form another entity (painted by Lippi). A third entity in this example is 'Flora' (Titan is the painter of that group). Checking whether two records are the same can be carried out by looking up the representatives of the two records. If the two representatives are the same, then there exists a positive path between the two nodes and it can be inferred that the two

Painting	`Representative`
Annunciation	Calumny
Calumny	Calumny
Allegory	Allegory
Primavera	Calumny
Saint Gerome	Allegory
Flora	Flora

Table 2: Representative Table (Figure 1)

Entity1	*Entity2*
Allegory	Calumny
Allegory	Flora
Calumny	Flora

Table 3: Unequal Table (Figure 1)

paintings were created by the same painter. For instance, it can be inferred with two look-ups on the Table 2 that 'Calumny' and 'Annunciation' were created by the same painter and without exploring any paths between these two nodes in the votes graph. In this approach, the choice of the representative is arbitrary; it could be any painting of a group of paintings to guarantee that positive paths can be detected with two look-ups on the `Representative` table.

If two entities have different representatives in the `Representative` table, they may or may not be the same. To find out whether a negative path exists to conclude that they are *not* the same, we propose to maintain an `Unequal` table. Table 3 shows the Unequal table for our running example. An `Unequal` table records pairs of representatives for which it is known that they are not the same. This way, the `Unequal` table provides a fast way to represent negative paths.

Example 2 *To compare 'Saint Gerome' and 'Annunciation', the representatives of these two paintings are looked up using the* `Representative` *table (Table 2). The two representatives are not the same (i.e., 'Allegory' and 'Calumny') so that the* `Unequal` *table (Table 3) is consulted. Since the pair* $\langle$ *Allegory, Calumny* $\rangle$ *is recorded in Table 3, it can be inferred that different artists created 'Saint Gerome' and 'Annunciation'.*

If both, the look-up in the `Representative` and the look-up in the `Unequal` are unsuccessful, additional crowdsourcing is required. In this case, there is neither a positive nor a negative path in the votes graph that connects the two nodes that represent the two entities.

At the beginning when the votes graph is empty, the `Representative` table only contains trivial equivalence; e.g., the tuple $\langle$Primavera, Primavera$\rangle$. The `Unequal` table is empty at the beginning. Given this state, no information can be inferred and the crowd has to be asked to resolve all comparisons. As more information becomes available due to crowdsourcing, the `Representative` and `Unequal` tables are updated and

inference becomes effective. Whenever a human worker states that two records are not the same, the representatives of these two records are recorded in the `Unequal` table. If the crowd judges that two records are the same, then the respective entities are merged. This merge operation is carried out by updating all records in one entity with the representative of the other entity. The selection of the representative can be done randomly because the effectiveness and performance do not depend on the right choice of the representative (as described above). Merging two groups of entities involves updating both the `Representative` and `Unequal` table because the representatives of the previously separate and now merged records need to be adapted to the changes.

The basic idea of maintaining representatives is well known and has been described in textbooks such as [18]. In particular, [18] gives algorithms to merge two groups in logarithmic time. In our implementation, we chose to implement the merge in linear time and have constant time for probes. Going into the details of these algorithms and trade-offs of alternative implementations is beyond the scope of this paper. In any way, it is straight-forward to implement this approach using a relational database system. The only conceptual novelty of our approach is the `Unequal` table in order to infer inequality using anti-transitivity.

4 Error-Tolerant Crowdsourcing

Unfortunately, humans make mistakes when evaluating comparisons or other crowd tasks. This section shows how to tolerate errors by introducing a decision function that resolves inconsistencies between conflicting votes.

4.1 Conflicts and Decision Functions

A conflict for two records r_1 and r_2 can be defined as follows:

- There is at least one positive path from r_1 to r_2 in the votes graph.
- There is at least one negative path from r_1 to r_2 in the votes graph.

Using the principles described in Section 3, both *equality* and *inequality* can be inferred in such a conflict situation. Therefore, we need additional techniques to resolve such conflicts.

Example 3 *Figure 2 gives an example for a votes graph with a conflict. Compared to the graph of Figure 1, the graph of Figure 2 has an additional positive edge between 'Allegory' and 'Flora' which is in conflict with the negative path 'Allegory-Saint Gerome-Flora'.*

Such inconsistencies are a fact of life in a system that is based on crowdsourcing: people make errors. An effective crowdsourcing system should tolerate conflict and errors and should make decisions even at the risk of making wrong decisions. In Figure 2, the natural decision when comparing 'Flora' and 'Allegory' is to decide that these two paintings were painted by different painters and should, therefore, be placed in different rooms of the Uffizi. The weights of all the edges along the 'Allegory-Saint

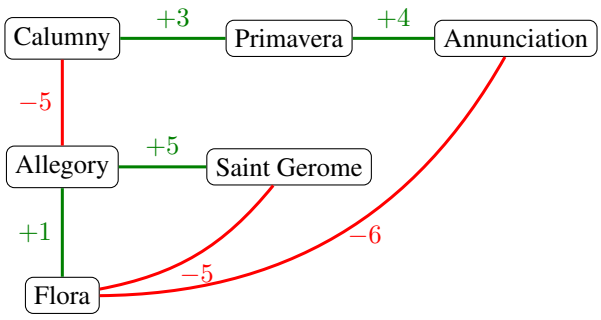

Figure 2: Votes Graph with Conflicts

Gerome-Flora' path are high which signals high confidence. In contrast, the weight of the 'Allegory-Flora' edge is low which signals low confidence. More formally, given a votes graph and two nodes of the graph, a decision function f makes the following decision:

- *Yes*: The two nodes (probably) represent the same entity.
- *No*: The two nodes (probably) represent different entities.
- *Do-not-know*: Additional crowdsourcing is needed in order to make a decision.

In other words, the decision function implements the condition of Step 2 of the algorithm presented in the introduction.

What should such a decision function look like? Obviously, it should answer *Do-not-know* if there is no path between the two nodes (e.g., at the beginning when no information is available). Furthermore, it should return *Do-not-know* if there is a conflict and the weights of the conflicting paths are similar. However, it should make a decision (either *Yes* or *No*) if the weights of the conflicting paths vary significantly. Furthermore, the decision function should obey all mathematical properties of an equivalence relation. More formally, we expect the following properties from a decision function, f:

- **Consistency:** If the decision function decides that two nodes are the same, then there has to be a positive path between the two nodes in the votes graph. Likewise, there must exist a negative path for a *No* decision.

- **Convergence:** For every pair of nodes and a given votes graph, there has to exist a sequence of votes so that the decision function decides *Yes*, for example three successive *Yes* votes. Equally, there has to exist a sequence of votes so that the decision function decides *No*.

- **Reflexivity:** For any node r_1 and any votes graph:
  $f(r_1, r_1) = \text{Yes}$

- **Symmetry:** For two nodes r_1, r_2 and any votes graph:
  $f(r_1, r_2) = \text{Yes} \implies f(r_2, r_1) = \text{Yes}$
  $f(r_1, r_2) = \text{No} \implies f(r_2, r_1) = \text{No}$
  $f(r_1, r_2) = \text{Do-not-know} \implies f(r_2, r_1) = \text{Do-not-know}$

- **(Strong) Transitivity:** For nodes r_1, e_2, r_3 and any votes graph:
  $f(r_1, r_2) = \text{Yes} \wedge f(r_2, r_3) = \text{Yes} \implies f(r_1, r_3) = \text{Yes}$

- **(Strong) Anti-transitivity:** For nodes r_1, r_2, e_3 and any votes graph:
  $f(r_1, r_2) = \text{Yes} \wedge f(r_2, r_3) = \text{No} \implies f(r_1, r_3) = \text{No}$
  $f(r_1, r_2) = \text{No} \wedge f(r_2, r_3) = \text{Yes} \implies f(r_1, r_3) = \text{No}$

Consistency guarantees that the decision function forms appropriate decisions while convergence is important in order to rule out a decision function that always decides *Do-not-know*. The remaining properties formalize the mathematical properties of an equivalence relation such as the 'same-entity-as' relation.

Finding a decision function that meets all these properties is difficult. The next subsection describes our proposal, the MinMax Decision function. The MinMax decision function is inspired by earlier work on preference functions by Fagin and Wimmers [6]. It is also related to the procedure proposed by Schulze for ordering [17]. Section 4.3 shows with the help of examples why other intuitive decision functions that are based on, e.g., average weights or other aggregate functions do now work well.

4.2 The MinMax Decision Function

The MinMax function considers all positive and negative paths between two nodes. Paths with more than one negative edge are ignored, as described in Section 3. For each path, the MinMax function computes a score: For a positive path, the score is the *minimum* of the weights of the edges of the path. For a negative path, the score is defined as the *minimum* of the *absolute* weights of the edges (i.e., the weight of the only negative edge is multiplied by -1 for this purpose). The intuition behind this scoring function is that a path is as strong as its weakest link. Another way to interpret the *minimum* is that it implements a conjunction (i.e., $\wedge$) along the path, thereby interpreting each edge as a predicate.

Example 4 *The score of the path 'Calumny-Allegory-Flora' in Figure 2 is 1 while the score for the positive path 'Calumny-Primavera-Annunciation' is 3.*

After computing the scores for all positive and negative paths, the MinMax decision function aggregates these scores into a single positive score, *pScore*, and a single negative score, *nScore*. *pScore* is the *maximum* of the scores of all positive paths. If there is no positive path, then *pScore* = 0. Analogously, *nScore* is the *maximum* of the scores of all negative paths. If there is no negative path, then *nScore* = 0. These values represent the maximum impact that a positive respectively negative path can have within an entity.

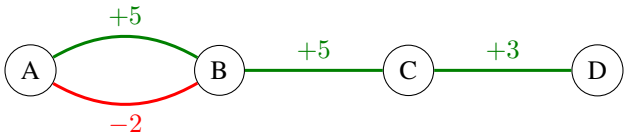

Figure 3: MinMax is Weakly Transitive

Finally, the MinMax function uses a quorum q in order to form a final decision based on the positive and negative scores. In our example we will chose $q = 3$ which practically means that three people have to agree to the notion of equality resp. inequality more than for the opposite option. That is, if the positive score is 3 or more higher than the negative score then the MinMax function decides that the two nodes are the same. More formally, the decision part of MinMax is defined as follows.

$$f(r_1, r_2) = \begin{cases} \text{Yes}, & pScore(r_1, r_2) - nScore(r_1, r_2) \geq q \\ \text{No}, & nScore(r_1, r_2) - pScore(r_1, r_2) \geq q \\ \text{Do-not-know}, & otherwise \end{cases}$$

The notion of *nScore* and *pScore* values can also be translated into the classic, complete, entity resolution context. Here, entities are formed for dense parts of the graph while sparsity, few connections between entities, means that the respective entities should be kept apart. In the *Yes* scenario, a large difference between the *pScore* of the records and its *nScore* signifies a high connectedness between r_1 and r_2 while a large value for the difference between the *nScore* and *pScore* symbolizes sparsity.

The MinMax function meets the *Consistency*, *Convergence*, *Reflexivity*, and *Symmetry* criteria defined in the previous subsection. Furthermore, the MinMax function meets the *Transitivity* and *Anti-transitivity* criteria in most cases. Unfortunately, it is possible to construct votes graphs in which the MinMax function does not meet these criteria. Figure 3 gives such an example. With $q = 3$, the MinMax function would judge that nodes A and C are the same: *pScore* = 5 because of the positive path A-B-C; *nScore* = 2 because of the negative path A-B-C. Furthermore, the MinMax function would decide that Nodes C and D are the same: There is only one (positive) path from C to D and the score of that path meets the quorum. Comparing A and D, however, the MinMax function would judge *Do-not-know* because *pScore*(A, D) = 3 and *nScore*(A, D) = 2 so that there is no quorum.

MinMax nevertheless fulfills a weaker variant of transitivity and anti-transitivity which can be defined as follows:

Weak Transitivity: For three records r_1, r_2, r_3 and any votes graph holds:

$$\begin{aligned} f(r_1, r_2) = \text{Yes} \ \wedge \ f(r_2, r_3) = \text{Yes} \\ \implies f(r_1, r_3) \in \{\text{Yes, Do-not-know}\} \end{aligned}$$

To understand the difference between weak and strong transitivity and why it is more desirable to have a strongly transitive decision function, we would like to re-iterate

the main goal of this paper: The goal is to evaluate the relationship between records, thereby making as few mistakes as possible (i.e., maximizing correctness) and asking the crowd as little as possible (i.e., minimizing cost). Inferring comparisons with a strongly transitive decision function makes it possible to achieve both goals. In comparison, inferring relationships with a weakly transitive decision function guarantees correctness, but it cannot guarantee optimality in terms of cost. In other words, the MinMax function is *conservative*. If it is in doubt, it will ask the crowd. In the example of Figure 3, the MinMax function would trigger additional crowdsourcing to compare A and D even though the answer seems obvious. This way, the MinMax function favors correctness over the minimization of cost.

4.3 Alternative Decision Functions

There is a huge space of possible decision functions for entity resolution. Decision functions used in *classic entity resolution* (i.e., fully automated entity resolution) aim to maximize intra-cluster density and at the same time maximize inter-cluster sparsity. As mentioned in Section 2, common techniques to do so in the context of non-Euclidean spaces such as for entity resolution range from simple partitioning algorithms to more complex adjacency-based measurements such as correlation clustering and Markov chain clustering [10]. Again, as discussed in Section 2, all of these clustering methods make a critical assumption, namely complete information to determine the graph structure. This assumption is not supported in a crowd-based environment and this fact makes these classic techniques not appropriate for crowd-based entity resolution. The big advantage of the MinMax function is that it is incremental and, thus, works well for incomplete graphs that improve over time as more and more information becomes available.

We have experimented with several other *incremental* decision functions; e.g., functions that are based on the `Average` or the `Sum` of weighted edges. None of these functions could compete with the MinMax function so we do not include the results of our experiments in this paper. One problem of these functions is complexity. Unlike the MinMax, functions that are based on `Average Weight` or `Total Weight` require to look at *all* paths. Unfortunately, finding all paths in an undirected cyclic graph such as a votes graph is $\mathcal{NP}$-hard [21] while the next subsection shows how the MinMax function can be implemented efficiently. Second and more important, these decision functions also violate transitivity and have higher cost than the MinMax function without improving on correctness. In fact, these decision functions even violate *weak transitivity* so that it is possible to have situations in which $A = B$, $B = C$, and $A \neq C$. Furthermore, these decision functions are less robust if the graph is incomplete or has cycles. For instance, the following two examples demonstrates how weights can be watered down or amplified if *average* or *sum* aggregate functions rather than *max* are used.

Example 5 *Figure 4 presents a slight variation of the example of Figure 3 where the weight of the edge connecting nodes C and D is increased to 5. If MinMax is applied to determine the relationship of A and D, it would reach quorum with $pScore = 5$ and $nScore = 2$. On the other hand, a decision function that is based on the* `Average Weight` *of all edges of a path would result in 15/3 = 5 as a score for the positive path*

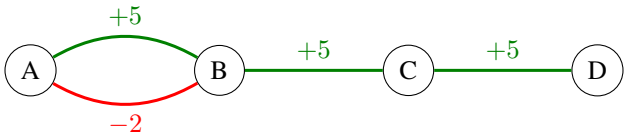

Figure 4: Average Decision Function Example

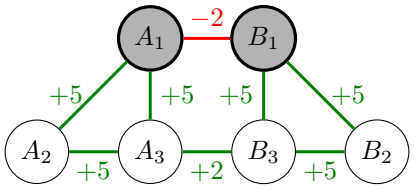

Figure 5: Sum Decision Function Example

and 8/3 = 2 2/3 as a score for the negative path which would result in a Do-not-know *decision with a quorum of 3.*

Example 6 *Figure 5 gives another example of a votes graph.* $\{A_1, A_2, A_3\}$ *form one group of records and* $\{B_1, B_2, B_3\}$ *a second one. Within these groups, there is strong affinity between the records. However, it is not yet clear whether the two groups can be merged: The* $A_1 - B_1$ *edge indicates that they cannot be merged; the* $A_3 - B_3$ *edge indicates that they can be merged. Both edges have a weight of 2 so that additional crowdsourcing is needed in order to resolve the conflict.*

Applying the MinMax function to the graph of Figure 5, the comparison $A_1 = B_1$ *indeed returns* Do-not-know *which is the right and expected behavior. A decision function that sums up the scores of all positive and negative paths (i.e., the minimum weight of all edges of a path), however, would conclude that* $A_1 = B_1$*; it would detect four positive paths from* A_1 *to* B_1*, each with score 2, and only one negative path, also with vote 2. The problem is that the results of such a* sum *(or* count*) aggregate function depend on the connectivity of nodes within a group and, thus, the completeness of the votes graph, whereas the MinMax decision function is robust independently whether the votes graph is complete or incomplete.*

Finding and experimenting with other decision functions further is an important avenue for future work. So far, the MinMax function is the best function that we could find and it dominates all other functions that we could come up with in terms of correctness, cost, and computation time. In the next section, we will show how the MinMax decision function can be implemented efficiently in an incremental way as the votes graph is expanded with human input.

Node	*Representative*	*pScore*
A	A	∞
B	A	6
C	A	5
D	D	∞
E	D	8
F	F	∞

Table 4: Representative Table (Figure 6)

Rep1	*Rep2*	e^+	e^-
A	D	1	9
A	F	1	5

Table 5: Unequal Table (Figure 6)

4.4 Implementing Min/Max

In order to implement the MinMax function, we extended the data structures presented in Section 3.2. As described in Section 3.2, we maintain a `Representative` table and an `Unequal` table. The `Representative` table materializes groups of records for which the MinMax function decides that they are the same by designating an arbitrary representative for each group of records. Likewise, the `Unequal` table materializes pairs of representatives for which the MinMax function decides that they are not the same. This way, it is possible to infer whether two entities efficiently; in the same way as described in Section 3.2.

With error-prone and conflicting votes, the maintenance of these tables becomes more complicated. Conceptually, these two tables are materialized views on the `Votes` table that need to be maintained with every new vote which is recorded in the `Votes` table. It is possible that at some point in time two records are believed to be the same which means that they have the same representative in the `Representative` table whereas further crowdsourcing reveals that they are in fact not the same. Likewise, it is possible that two records are initially voted to be dissimilar because their representatives have been recorded in the `Unequal` table and further crowdsourcing leads to a different evaluation of the relationship between these two records. This way, there is churn in the `Representative` and `Unequal` tables: Entities as defined through the `Representative` table may be split again and records in the `Unequal` table may be deleted. Furthermore, it is important to keep track and detect when a quorum for a pair of records is reached, indicating a merge of the groups that they belong to.

Obviously, it is computationally intractable to recompute the `Representative` and `Unequal` tables with every new vote that is recorded. Fortunately, the `Representative` and `Unequal` tables only need to be updated rarely, and it is possible to find criteria that indicate under which circumstances a new vote may lead to an update of these two materialized views. Equations 1 and 2 derived below provide such criteria. By checking these equations, the maintenance becomes affordable.

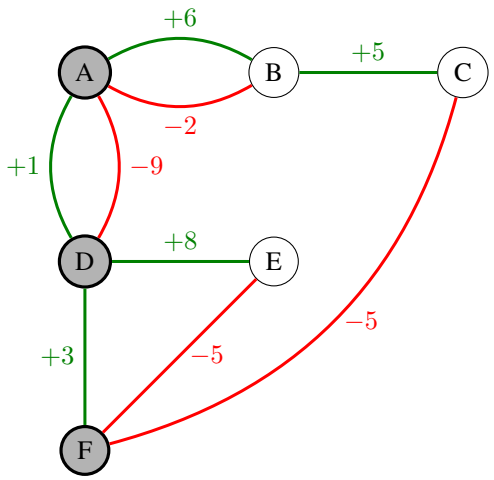

Figure 6: Example Votes Graph

Example 7 *For illustration, take Figure 6. Here, a quorum of* $q = 3$ *is assumed and using the MinMax function, the paintings within each entity reach that quorum. Using the MinMax function,* A *and* D *are judged to be not the same (*nScore = *9;* pScore = *1). Likewise, the MinMax function decides that* A *and* F *are not the same (*nScore = *5;* pScore = *1). Tables 4 and 5 show the* `Representative` *resp.* `Unequal` *table for the running example.*

We then extend the *Representative* table with an additional column which materializes the *pScore* of every node to its representative; i.e., the score of the best positive path between a node and its representative. The *pScore* is set to ∞ if the node is the representative. The use of this *pScore* value will become clear below. Table `Unequal` in contrast is extended to contain two values, e^+ and e^-, that capture the relationship between the two entities represented in a row of the `Unequal` table. The first value, e^+, represents the weight of the highest positive edge between the two groups while e^- is the weight of the largest value of any negative edge between the two entities.

In a fully connected votes graph such as the one in Figure 6, a vote can impact the relation between any two nodes in the graph. For instance, positive votes for B and E can impact the *nScore* of D and F because they establish (or impact) a negative path between D and F. Furthermore, positive votes for B and E can impact the decisions of the MinMax function within entity $\{A, B, C\}$ and may even result in a split of that entity because they create an additional negative path within the entity. The question addressed in the remainder of this section is how to quickly detect that new votes do or do not impact any existing relationships.

To answer this question, we will introduce two additional data structures and explain the importance of the additional columns added to the `Representative` and `Unequal` tables. The first data structure is the `Candidates` table. This table materializes all pairs of representatives for which positive or negative paths exist and for

Rep1	*Rep2*	e^+	e^-
D	F	3	5

Table 6: Candidates Table (Figure 6)

Rep	*pGroup*	*nGroup*
A	5	2
D	8	0
F	∞	0

Table 7: Group Coherence Table (Figure 6)

which the MinMax function cannot (yet) make a decision because no quorum is reached. The `Candidates` table also maintains the weight of the best positive edge, e^+, and best negative edge, e^-, between two nodes of the two candidate entities.

Example 8 *With regard to D and F in Figure 6, the MinMax function is undecided. As* pScore = *3 and* nScore = *5, $q \leq +3$ with $q = 5 - 3 = 2$ holds. The stored* `Candidates` *table is visualized in Table 6.*

The second data structure that helps to quickly infer whether the `Representative` or `Unequal` tables need to be updated is the `Coherence` table. This table materializes the *minimum pScore* value between two records of an entity denoted as *pGroup* and the maximum *nScore* value between two records of an entity denoted as *nGroup*. Table 7 shows the *Coherence* table for the example votes graph of Figure 6.

Within an entity, the following condition must hold at all times: *pGroup - quorum* $\geq$ *nGroup*. If this condition holds, we can always infer that two entities of the group (i.e., to nodes represented by the same representative in the *Representative* table) are considered to be the same according to the MinMax function. If this condition does not hold, we *split* the group by computing the subgroups for which this condition holds.

Using a similar equation, we can decide when an entry in the *Candidates* table should be promoted to an equality. Two entities represented by *Rep1* and *Rep2* in the *Candidates* table are *merged* if the following condition holds:

$$\begin{aligned} min(pGroup(Rep1), pGroup(Rep2), e^+) - quorum \\ \geq max(nGroup(Rep1), nGroup(Rep2), e^-) \end{aligned} \tag{1}$$

In other words, if the positive relationships between the entities exceed the negative scores by a difference bigger or equal than the quorum, the two entities can be merged. Analogously, two entities from the `Candidates` table can be added to the `Unequal` table if the following condition is met:

$$\begin{aligned} min(pGroup(Rep1), pGroup(Rep2), e^-) - quorum \\ \geq min(pGroup(Rep1), pGroup(Rep2), e^+) \end{aligned} \tag{2}$$

There are many technical details in our implementation of the MinMax function to make it more efficient. Describing all these details is not possible in this paper because

of space constraints. A complete description can be found in [25]. One important point, however, is that our implementation of the MinMax function is conservative. That is, our data structures may indicate *Do-not-know* even though a precise implementation of the MinMax function may indicate *Yes* or *No*. Here, we are making a response time / cost trade-off. Implementing the MinMax function precisely would have prohibitively high computational cost in order to explore paths every time we cannot infer an answer from our data structures. As a result, we crowdsource too often. Fortunately, those situations are rare as shown in Section 6.

5 Entity Resolution Algorithms

This section revisits the entity resolution algorithm sketched in the introduction. It specifically addresses two questions concerning Step 2 of the algorithm: Which two records have to be crowdsourced, the *next-crowdsource problem*, and when to stop crowdsourcing.

Algorithm 1 shows a more formal version of the algorithm presented in the introduction. It receives a set of n records and a number of crowd workers w as input and determines for every pair of records the relationship between these records. As a result, this algorithm returns a two dimensional matrix.

Example 9 *Given our running example, the entity resolution algorithm is called with n paintings representing the records as input while w students take up the positions as crowd workers. The algorithm then returns a matrix that describes which paintings were created by the same painter.*

Steps 1 and 3 of the entity resolution algorithm are identical to the corresponding steps of the algorithm sketched in the introduction. In contrast, we now specify the exact procedure to determine the relationship of two records through inference. The first task thus is to determine whether two records have the same representative respectively are unequal by leveraging the knowledge stored in the `Representative` and `Unequal` tables. If these operations are unsuccessful, a function *crowdsource()* is triggered that evaluates the relationship of the records through the crowd. If it returns 'false', the crowdsourcing was unsuccessful in that it could not determine the relationship of the records. If that happens, Algorithm 1 returns *Do-no-know* for that specific pair of records. This problem might occur in cases where the crowd is unsure about the decision and MinMax cannot reach the specified quorum. If *crowdsource()* returns 'true', a decision has been made and tables `Representative` and `Unequal` are updated accordingly. The results can thus be retrieved during the next iteration of the loop. We will describe the *crowdsource()* function and how it exactly decides to return **false** in the following subsections.

5.1 Asking the Right Questions

To understand the *crowdsource()* function, let us go back to the example of Figure 3. As discussed in Section 4.2, the MinMax function cannot make a decision when comparing A and D given the votes graph of Figure 3. The question then is whether it is

Algorithm 1: Entity Resolution Algorithm

Input : Set of records to be compared $\{r_1, \ldots, r_n\}$;
Number of crowd workers w

Output: Result matrix:
result(r_i, r_j) $\in$ {*Yes*, *No*, *Do-not-know*}

Step 1: Randomly select two records, (r_1, r_2) that have not been compared yet.
Step 2:
while *true* **do**
  $a_1 \leftarrow$ representative(r_1);
  $a_2 \leftarrow$ representative(r_2);
  if $a_1 = a_2$ **then**
    result(r_1, r_2) $\leftarrow$ *Yes*;
    goto Step 3;
  end
  if *unequal*(a_1, a_2) **then**
    result(r_1, r_2) $\leftarrow$ *No*;
    goto Step 3;
  end
  if *crowdsource()*(r_1, r_2, w) = *false* **then**
    result(r_1, r_2) $\leftarrow$ *Do-not-know*;
    goto Step 3;
  end
end
Step 3: goto Step 1 until all pairs of records have been compared.

possible to decrease the crowdsourcing effort and thus the cost by not asking the crowd about the direct relationship between A and D but rather by leveraging the knowledge about the indirect relationship of those two records.

Looking at Figure 3, the most promising edge to invest in for deciding whether A and D are the same is $C - D$. We suspect that C and D are the same and with extra support, more information that increases resp. decreases the weight of an edge, on that edge we could infer that A and D are the same. More concretely, in the best case scenario for this example only two additional positive votes on $C - D$ are required to infer that A and D are the same. In contrast, if we crowdsource A and D directly, then at least five positive votes are required to come to the same conclusion.

In general, our data structures and Equations 1 and 2 indicate into which edges to invest. Continuing the example of Figure 3, there are two entities at the status quo, $\{A, B\}$ and $\{C, D\}$. The two entities cannot be merged because the criterion of Equation 1 is not met with *pGroup(A)* $= 5$, *pGroup(C)* $= 3$, $e^+ = 5$ so that the *minimum* on the left side of Equation 1 is 3. Likewise, *nGroup(A)* $= 2$, *nGroup(C)* $= 0$, $e^- = 0$ so that the *maximum* on the right side of Equation 1 is 2 and quorum is not reached. In order to reach quorum, we need to invest into the left side of Equation 1 and increase the *minimum*. That is, we need to increase *pGroup(C)* in this example and we can do this by increasing the weight of the $C - D$ edge. An increase in weight is

achieved by asking for additional support. Obviously, there is no way to *decrease* the *maximum* on the right side of Equation 1 because additional votes never decrease any *nGroup* or e^- values thus increasing the *pGroup(C)* is the most efficient way to process the evaluation of the relationship between A and D.

For brevity, we will not fully discuss the other case in which we suspect that two entities are *not* the same. In essence, this case works in an analogous way where we use the criterion of Equation 2 and, again, crowdsource in order to increase the *minimum* on the left side of Equation 2. The decision of whether to crowdsource for *equality* or *inequality* meaning whether to target Equation 1 or 2 is done by inspecting the e^+ and e^- values in the `Candidates` table. If $e^+ > e^-$, the *crowdsource()* function will try to increase the minimum on the left side of Equation 1. Otherwise, it will crowdsource in order to increase the minimum on the left side of Equation 2. If the representatives of two records r_1 and r_2 are not part of the `Candidates` table which is the case if neither a positive nor a negative path exists between r_1 and r_2, then the *crowdsource()* function will directly crowdsource r_1 and r_2. This situation arises for instance at the beginning of the execution of Algorithm 1.

Once the *crowdsource()* function has selected two records to compare, it will ask a random crowd worker who has not yet compared the two records to cast his/her vote. As part of this process, all the data structures presented in Section 4.4 are updated. In particular, the `Representative` and `Unequal` tables may be updated so that a decision can be made in the next iteration of the *while* loop in Step 2 of Algorithm 1. If the updates are not decisive, crowdsourcing continues until a decision is made or we run out of crowdsourcing budget. The idea of the budget for a crowdsourcing task is described in the next subsection of this paper.

The discussion of this subsection and the design of the *crowdsource()* function represent a key insight of this paper: When comparing two records, for example two paintings, it is critical to consider the *entire* votes graph that has been constructed up to this point. Given the incompleteness of the underlying graph structure, it is not sufficient to look at the two records and their direct relationships only but in order to decrease the cost for asking the crowd, information can be inferred through edges connecting the records indirectly. This method is in contrast to existing entity resolution algorithms which aim to create a consistent votes graph from the start and which determine the relationship of records by evaluating their direct environment in the graph. As the votes graph conveys partial information, this information has to be leveraged smartly but can be used to efficiently construct which records belong to the same or distinct entities. This observation and insight will be studied in more detail as part of the experiments because it has significant impact on both the cost and the quality of crowdsourced comparisons.

5.2 Convergence

Throughout this work, we assumed that there is a ground truth: Two paintings are either created by the same painter or not. There are situations in which no ground truth exists or it may be impossible to get to it even with crowdsourcing. In such situations, it is best to ask the crowd reasonably often but then return `Do-not-know`, rather than increasing the cost unnecessarily by asking the crowd for more input. That is what the

w parameter of Algorithm 1 is used for.

More concretely, the w parameter specifies the maximum numbers of crowd workers we ask to compare a given pair of paintings. For instance, if there are only w students available, then it does not make sense to ask more than w times to compare two paintings because asking the same student twice does not reveal any new information. If the budget of w questions has been exhausted and still no decision can be made with the MinMax function (because the students cannot agree), then the *crowdsource()* function returns false and Algorithm 1 returns `Do-not-know`. A `Do-not-know` decision is neither correct nor incorrect; it simply indicates an incomplete result. We studied *completeness* in addition to *correctness* and *cost* as part of our experiments (Section 6).

6 Experiments and Results

This section presents the results of experiments that study the cost (i.e., number of questions asked to the crowd) and result quality (i.e., correctness and completeness) of the method to carry out comparisons in Step 2 of Algorithm 1 proposed in Sections 4 and 5. We call this method *Inference* because it exploits the mathematical properties of an equivalence relation in order to *infer* comparisons between records. As baselines, we used two naïve approaches: (a) an approach that *Always* asks the crowd and (b) an approach that *Caches* the results of crowdsourcing without any inference.

6.1 Methodology

The main goal of the experiments was to study cost / quality trade-offs of comparing two records with a varying error rate of human workers. Furthermore, we were interested in the robustness of the *Inference* approach and to find out whether errors would propagate by inference (leading to poor quality) or whether the *Inference* approach would improve quality by asking for more input from the crowd in the event of conflicts.

The experiments were carried out in the following way. In the first step, a *ground truth* was generated. That is, we generated a (synthetic) mini-world with 1000 paintings and 100 painters (Table 8). We experimented with two different distributions that associate painters to paintings: with a Uniform distribution, each painting was associated to each painter with the same probability; with a Zipf distribution, some painters were more prolific than others according to a Zipf distribution. We only report the results with the Zipf distribution in this paper because the results with the Uniform distribution were almost identical. We also experimented with bigger databases (more paintings and more painters), but do not show the results because, again, the effects were almost identical. With a larger database, it just takes longer before anything can be inferred from the similarity graph.

After generating the *ground truth*, we ran a sequence of queries (up to 500,000). Each query involves a pair of paintings and represents one iteration of Algorithm 1. We used a variation of Algorithm 1. Rather than selecting *different* pairs of paintings in each iteration of Step 1, we randomly selected pairs of paintings without memory, thereby possibly comparing pairs of paintings twice or even more often as part of the benchmark workload. This way, we could compare the cost of our proposed *Inference* approach

Mini-world	
Paintings	1000
Painters	100
Distribution	Zipf
Workload	
Comparisons	vary up to 500,000
Distribution	Uniform
Error rate	vary 0% to 90%
Fault-tolerance	
Quorum q	1, 3, 5
# Students s	30

Table 8: Benchmark Parameters

with an approach that *caches* the results of all comparisons. Caching is a natural baseline and has been integrated into systems such as CrowdDB, but it would not make sense if each pair of paintings is studied only once. In all experiments, the benchmark selects the two paintings to compare next in Step 1 using a Uniform distribution

In all our experiments, we simulated crowdsourcing as follows. Whenever we needed to ask the crowd for a pair of paintings in Step 2 of Algorithm 1, we probed the ground truth for that pair of paintings in order to get the correct result. Then, we used a random generator in order to decide whether the right or a wrong result was returned. For instance, if the error rate was set to 5%, then we returned the correct result with probability 95%. In most experiments, we varied the error-rate between 0% and 20%. We also studied some extreme cases with higher error rates. The parameter, s, that determined how many people (i.e., students) were available was set to 30 in all experiments reported in this paper. It turned out that the results presented in this paper are not particularly sensitive to this parameter. We varied the quorum for making a decision with the MinMax function from 1 to 5.

An important assumption of our way to simulate crowdsourcing is that students make errors independently. In reality, errors are likely to be correlated. That is, students who have attended classes by the same professor tend to make the same kind of errors. To model such correlated errors, we carried out experiments with very high error rates (i.e., 90%); these experiments model malicious students who deliberately give wrong answers. In general, modeling correlated errors is an art in its own right and studying its effects in a more precise way is left for future work. In the extreme, if all students err in the same way, then the system will accept the wrong result as "truth" and there is nothing that can be done. For this reason, we do not report on results with a 100% error rate.

While running the experiments, we studied the following three metrics:

- *Cost:* The number of comparisons carried out by students. We assume that each crowdsourced comparison has a fixed, constant cost so that the total cost of crowdsourcing is proportional to the number of comparisons carried out by students. In practice, it may be cheaper to compare a Picasso with a Botticelli

(even lay-people can do that in an ad-hoc way) than to compare a Lippi with a Botticelli or two Botticellis (even experts will need to carry out a detailed analysis).

- *Correctness:* How often does the system misclassify two paintings. That is, how often is the wrong branch taken in Step 2 of Algorithm 1.

- *Completeness:* How often does the system not classify a painting at all after asking all s students; i.e., how often could no quorum be achieved and the *Do-not-know* branch is taken in Step 2 of Algorithm 1.

We study all these questions as a function of the error rate and over time. We expect that all three metrics get worse with an increasing error rate. Furthermore, we expect that all three metrics improve over time: At the beginning, no information is available and crowdsourcing is needed for every pair of paintings; over time, however, more information is available and the *Inference* approach can infer new equalities and inequalities without asking the crowd (thereby reducing cost) and it can detect conflicts in the similarity graph (thereby possibly improving correctness).

We also studied the computational cost and size of the database. These results are presented at the end of this section and show the effectiveness of the data structures presented in Section 4.4.

Throughout this study, we used the following two baselines in order to put the results of the *Inference* approach into perspective:

- *Always:* Do not exploit symmetry and transitivity and ask students whenever two paintings need to be compared in Step 2 of Algorithm 1. In this approach, no `Votes` table or any other data structure discussed in Sections 3 and 4 need to be maintained.

- *Caching:* Only exploit symmetry, but not transitivity. That is, the system maintains a `Votes` table, but does not maintain any of the other data structures described in Sections 3 and 4. If the same two paintings are compared twice, then the same result is given each time: the first time with crowdsourcing, the second time from the cache.

In terms of cost, it can be expected that *Inference* does better than *Always*. In terms of robustness and quality (i.e., correctness and completeness), the winner is less clear. Also, it is not clear how well *Inference* fares as compared to *Caching*.

6.2 Software and Hardware Used

All experiments were carried out on a quad-core Intel Xeon X3360 machine with 8 GB of main memory running Ubuntu 10.04. The database system was PostgreSQL 9.1. We implemented the benchmark and the three variants, *Always, Caching,* and *Inference*, in Java using JDBC and stored procedures in PostgreSQL. The *Inference* approach was implemented as described in Sections 4 and 5. The *Always* and *Caching* approaches were implemented in a straight-forward way as described in the previous sub-section.

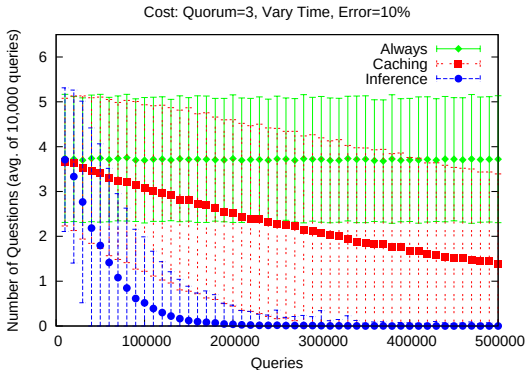

Figure 7: Cost: q=3, Vary Time, Error=10%

6.3 Cost

6.3.1 Vary Time

In the first set of experiments, we studied the cost (i.e., number of questions asked to students) as a function of time. The results are shown in Figure 7. Each data point in this figure represents the average cost per comparison over 10,000 pairs of paintings. In this experiment, we set the error rate to 10% (i.e., a student gave a wrong answers with probability 0.1 when asked to compare two paintings) and the quorum was set to 3 for all three approaches.

Obviously, all three approaches behave similarly at the beginning. At the beginning, the `Votes` table is empty and no comparisons can be inferred so that all three approaches need to crowdsource for each pair of paintings. As time progresses and more and more information has been crowdsourced, the *Inference* and *Caching* approaches improve significantly in terms of cost. Both of these approaches can use the information from previous comparisons to reduce cost. Obviously, the *Inference* approach can make much better use of previous comparisons by exploiting transitivity and anti-transitivity. As a result, *Inference* has lower cost than *Caching* throughout this experiment. The cost of *Always* is constant because *Always* does not remember any crowdsourced input and carries out crowdsourcing for each pair of paintings.

The error bars in Figures 7 indicate that the amount of crowdsourcing per comparison varied significantly in all three approaches. For *Caching* and *Inference*, the high variance is no surprise: For some queries no crowdsourcing was needed because the result was cached (*Caching*) or could be inferred by transitivity or anti-transitivity (*Inference*) whereas for other queries crowdsourcing was needed because no suitable information was available. For *Always*, the standard deviation in cost per pair of painting was lower because crowdsourcing was required all the time. For the *Always* approach, the variance in cost is caused by mis-judgments from students; depending on errors, a varying number of students had to be asked before reaching quorum.

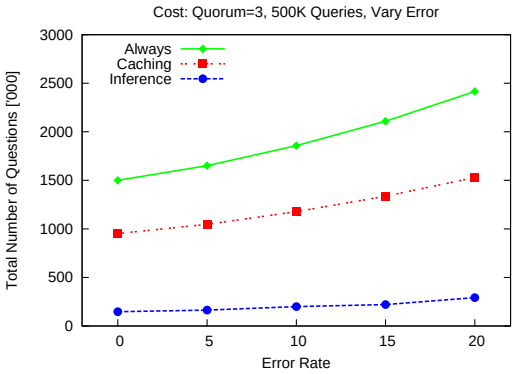

Figure 8: Cost: q=3, 500K Queries, Vary Error

6.3.2 Vary Error Rate

In the second set of experiments, we studied the cost of the three approaches as a function of the error rate. In this experiment, we computed the total cost for comparing 500,000 pairs of paintings. Figure 8 shows the results. Figure 8 shows that the cost of all three approaches increases with the error rate. This observation is no surprise as it gets increasingly difficult to reach quorum as the level of "noise" from the crowd increases. Furthermore, Figure 8 shows that *Inference* has lower cost than *Always* and *Caching* independent of the error rate. Again, taking advantage of transitivity and anti-transitivity explains this effect.

6.4 Quality

6.4.1 Correctness

Figure 9 shows the number of pairs of paintings that were mis-classified by the three approaches, thereby varying the error rate. Overall, all three approaches produce good results with a quorum of 3. Of the 500,000 queries, 2% (i.e., 10,000) queries were answered incorrectly, even for a high error rate of 20%. Comparing the three approaches, *Inference* is the clear winner. Even with an error rate of 20%, it makes only about 1000 (0.2%) errors. For an error rate of 10% and less, it is (almost) perfect (only a few errors with an error rate of 10%).

Always and *Caching* perform almost identically in terms of correctness. With an error rate of 10%, they misclassify several hundred queries in this experiment, with 15% almost 3000 queries, and with 20% almost 8,000 queries are answered incorrectly. So, the quality of results produced by these approaches is about one order of magnitude lower than for the *Inference* approach. The conclusion that can be drawn from this experiment is that the *Inference* approach is much more robust to the error rate in terms of correctness. In other words, carrying out *Inference* and trying to keep the entire similarity graph consistent does not only reduce cost; it also improves result quality significantly.

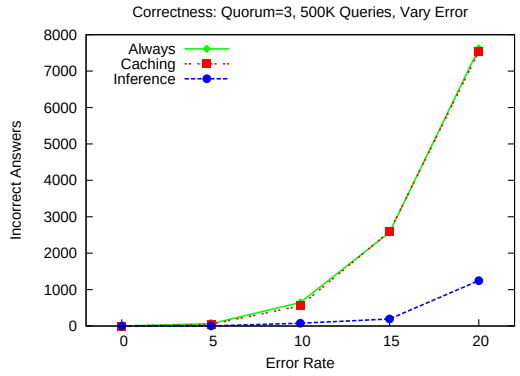

Figure 9: Correctness: q=3, 500K Queries, Vary Error

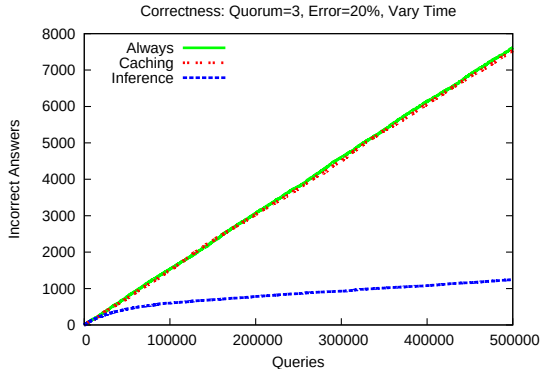

Figure 10: Correctness: q=3, Error=20%, Vary Time

Figure 10 shows the total number of errors made by the three approaches over time. This figure shows that neither *Always* nor *Caching* improve over time; their graphs overlap completely in Figure 10. Both *Always* and *Caching* make errors independent of how much information has already been gained as part of crowdsourcing in the past: *Caching* benefits from crowdsourcing over time in terms of cost (as shown in Figure 7), but not in terms of correctness. In contrast, *Inference* does improve. (Almost) all its mistakes are made at the beginning when it behaves like *Always* and *Caching*. As it learns more about the graph, *Inference* detects and resolves conflicts, resulting in a flattening of the error curve. If we would run Algorithm 1 completely, *Inference* would resolve almost all its mistakes made at the beginning and ultimately produce a very high quality entity resolution at the end.

6.4.2 Completeness

Unfortunately, the high correctness of the *Inference* approach comes at an expense: *completeness*. Figure 11 shows the number of times each approach responded *Do-not-know* in Step 2 of Algorithm 1 after exhausting its crowdsourcing budget of 30 questions

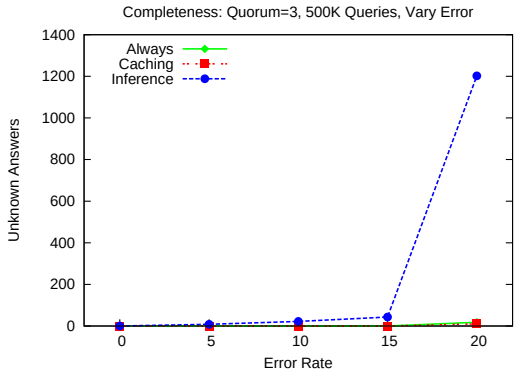

Figure 11: Completeness: q=3, 500K Queries, Vary Error

per pair of paintings, again using a quorum of 3.

Comparing Figures 11 and 9 the results are almost inverted. For high error rates, *Inference* shows poor completeness and high correctness; in contrast, *Always* and *Caching* show poor correctness and high completeness for high error rates. In other words, the *Inference* approach can be characterized as being *conservative*, thereby preferring to give no answer than a wrong answer. The choice of a conservative decision function (Section 4.2) may have added to this effect. The *Always* and *Caching* approaches can be characterized as *aggressive*, thereby (almost) always making a decision even if wrong. Of course, it depends on the application whether a conservative or aggressive approach is preferable.

We also studied completeness over time. Again, the results are almost inverted as compared to the correctness results. *Always* and *Caching* are constant and do not change their behavior over time. In contrast for high error rates (e.g., 20%), the completeness of *Inference* gets worse over time. Over time, *Inference* detects more and more conflicts in the similarity graph and cannot resolve them with such high error rates and, thus, ultimately returns *Do-not-know* for many queries. Arguably, this is the right behavior. We do not show the graphs for brevity.

6.5 Vary Quorum

In the next set of experiments, we studied the cost / quality trade-offs with a varying quorum for all three approaches. The expectation is that both cost and quality increase with an increasing quorum. Table 9 shows the results for a 20% error rate. We also ran this experiment with a smaller error rate (not shown): With a smaller error rate, having a high quorum is less important so that the results are less pronounced.

Table 9 confirms the expectation that a higher quorum results in higher cost, higher correctness, and lower completeness, regardless of which approach is used. With a quorum of 1, all approaches generated a large number of incorrect answers: For *Inference*, 6% of all the 500,000 queries were answered incorrectly and for *Always* and *Caching* almost 20% of the queries were answered incorrectly. So, errors of students directly translate into wrong results with $q = 1$ for *Always* and *Caching* whereas

	Cost per Query			*Incorrect (total)*			*Do-not-know (total)*		
	q=1	*q=3*	*q=5*	*q=1*	*q=3*	*q=5*	*q=1*	*q=3*	*q=5*
A	1.0	4.8	8.3	99,043	7,619	485	0	18	500
C	0.6	3.1	5.3	99,730	7,543	478	0	10	539
I	<0.1	0.6	0.9	31,856	1,246	79	0	1,202	6,012

Table 9: All Metrics: Vary q, 500K Queries, 20% Error

	Cost per Query			*Incorrect (%)*			*Do-not-know (%)*		
err	*20%*	*50%*	*90%*	*20%*	*50%*	*90%*	*20%*	*50%*	*90%*
A	8.3	19.3	6.1	0.1%	37%	100%	0.1%	28%	0%
C	5.3	12.2	3.8	0.1%	38%	100%	0.1%	29%	0%
I	0.9	0.3	<0.1	<0.1%	6%	97%	1%	93%	<0.1%

Table 10: All Metrics: q=5, 500K Queries, Vary Error

Inference can do better by looking at the whole graph even in this extreme case. Overall, *Inference* wins independent of the quorum for the same reasons as explained in the previous sub-sections.

6.6 Robustness: High Error Rates

In the last set of experiments, we were interested how robust the *Inference* approach is to extremely high error rates such as 50% and 90%. An error rate of 50% means that crowdsourcing is worthless; all the answers are random. An error rate of 90% models a scenario in which the crowd is malicious and deliberately gives wrong answers in almost all the cases. In other words, a 90% error rate models a scenario in which errors made by students are correlated. Table 10 shows the cost, correctness, and completeness of the three approaches for such high error rates. As a baseline, Table 10 also shows the results for a 20% error rate (repeated from previous sub-sections). In this experiment, the quorum was set to 5, the most error-tolerant setting.

Again, *Inference* is the clear winner and for the same reasons as in all previous experiments. What is interesting to see is how it reacts to the two scenarios: totally random input (i.e., 50% error rate) and malicious input (i.e., 90% error rate). For random input, *Inference* does the right thing: After a while, it gives up to make the similarity graph consistent and simply returns *Do-not-know* for all queries. At this point, it also stops crowdsourcing so that it has very low cost. In other words, the *Inference* approach is able to detect when crowdsourcing does not make sense.

Unfortunately, the *Inference* approach is not able to detect malicious input from the crowd (i.e., 90% error rate). In this case, *Inference* also answers almost all queries incorrectly and does not detect that crowdsourcing is not effective. On the positive side, the similarity graph converges extremely fast in this scenario because the crowd will almost always answer that two paintings have been painted by the same painter so that *Inference* has almost zero cost in this scenario.

	Always		*Caching*		*Inference*	
Error Rate	*avg*	*var*	*avg*	*var*	*avg*	*var*
0%	0.002	0.002	0.57	1.12	11.07	1923.66
5%	0.002	0.002	0.63	1.09	34.44	726354.00
10%	0.002	0.002	0.71	1.89	50.20	1451601.25
15%	0.002	0.002	0.78	1.44	64.02	3015557.10
20%	0.002	0.002	0.93	9.87	97.48	3523059.08

Table 11: Avg. Response Time [msecs]: q=3, Vary Error

6.7 Complexity

The main goal of all experiments so far was to study cost / quality trade-offs of the *Inference* approach. Cost was defined as the cost of crowdsourcing. The purpose of this section is to investigate the computational and storage requirements of the alternative approaches. Obviously, the *Inference* approach only makes sense if the computational complexity is affordable.

Table 11 shows the average response time in milliseconds per query and variance of the three approaches with varying error rate. In this experiment, the time to simulate crowdsourcing (Section 6.1) is *not* included in the response time because we could not simulate it. However, the purpose of this experiment was to show whether the *Inference* approach was computationally affordable.

As expected, the *Inference* approach is computationally more expensive than the other approaches. It also has the highest variance because it involves splits and merges for some queries. Furthermore, as expected, the amount of computation increases with the error rate as more conflicts need to be resolved. In almost all cases, however, the *Inference* approach could be carried out within sub-seconds. In very rare and extreme cases, it could take up to a few minutes to process a single query because a sequence of conflicting votes caused cascades of splits and re-merges. Compared to the time that it takes to crowdsource information, however, these computational overheads are negligible.

Table 12 shows the storage required to keep the `Votes` tables (for *Caching*) and the additional tables described in Sections 4.4 for the *Inference* approach. Table 12 shows the results after 500,000 queries (at the end) and for a quorum of 5 (the case with the highest footprint). Obviously, the *Always* approach has no storage requirements. Both, *Caching* and *Inference* have affordable storage overheads of only a few mega-bytes. The storage overhead grows quadratically with the number of records. Nevertheless, even for large databases with, say, hundred thousand records, all data structures fit in main-memory of a modern machine. As a result, the storage overhead should not be a consideration in selecting the right method to carry out entity resolution with the crowd.

7 Future Work

We believe that this paper lays the most important foundations for solving classification problems with the help of crowdsourcing. However, there is still a number of

Error Rate	*Always*	*Caching*	*Inference*
0%	0	54	33
5%	0	54	39
10%	0	53	47
15%	0	54	55
20%	0	54	65

Table 12: Storage [MB]: q=5, Vary Error

important directions for future work. The first issue that we want to find new entity resolution algorithms that allow to control quality and cost. For instance, we would like to find an algorithm that finds the highest possible correctness and completeness for a given total budget of say, n crowdsourced comparisons. Likewise, we would like to find an algorithm that minimizes the number of crowdsourced comparisons that need to be carried out in order to achieve a certain level of correctness and completeness (e.g., 95% precision and 90% recall). Algorithm 1 achieves a very good quality / cost ratio (the best of all algorithms that we are aware of), but it does not explicitly allow to control cost and result quality.

Second, we see potential for further research on new decision functions and variants to Algorithm 1. For instance, we would like to explore variants of Algorithm 1 that do not select random pairs of records in Step 1. Instead, this choice should be carried out with care as the order in which the records are selected in Step 1 of Algorithm 1 may impact both cost and result quality, a similar result was reported in [22]. One variant could be to automatically pre-compute weights between pairs of records using heuristics such as those proposed in [5]. Another variant of Algorithm 1 involves comparing the same pair of records twice or even more often. During the course of developing the techniques presented in this paper, we experimented with such settings and these experiments have shown that such an approach can improve result quality for high error rates, but we still need to find a systematic way that determines when to stop; i.e., defining the right termination condition for Step 3 of Algorithm 1.

An important property of Algorithm 1 is that it parallelizes well. In Step 1, it is possible to select several pairs of records simultaneously and carry out Step 2 for all these pairs concurrently. Parallelization is important for systems that are based on crowdsourcing in order to achieve an acceptable response time. Going back to the Uffizi example, it makes no sense to have only one student work at every given point in time. All of the angles for future work mentioned above thus have to be designed in a way as to suit this requirement.

A more fundamental change of Algorithm 1 involves a completely different approach of selecting pairs of paintings to compare in Step 1. Algorithm 1 (and all the variants discussed at the beginning of this section) suggest to select these pairs in a *closed crowdsourcing model*. In such a closed model, the system decides which information to crowdsource. Such a closed crowdsourcing model has been adopted by most recent crowdsourcing systems such as CrowdDB, Deco, and Qurk [3]. The main ideas of this paper, however, can also be integrated in an *open crowdsourcing model*. In such an open model, also called bottom-up model, the crowd workers make the comparisons that they

are comfortable with. Freebase is a prominent system that has adopted such an open model.

Finally, we would like to determine how our techniques can be applied to other problems such as sorting and Top N; initial work on processing Top 1 queries using crowdsourcing will be published in a forth-coming paper [9], but the techniques presented in [9] are quite different from the techniques presented in this paper.

8 Conclusion

The main observation studied in this paper is that it is important to look at the *entire* votes graph rather than at every individual edge when comparing two records as part of a classification problem. While it is intuitive that such a holistic approach is critical to achieve high quality in the presence of errors, it is less obvious that such an approach makes it also possible to reduce cost; i.e., minimize the number of comparisons that need to be crowdsourced. The main contribution of this paper is to lay the foundations for such a holistic approach to compare entities with the help of crowdsourcing. This paper presented the MinMax decision function as a practical way to decide whether two records are the same or belong to the same class. Furthermore, it sketched the data structures needed to implement the MinMax function efficiently in a dynamic environment in which new votes are recorded continuously. Based on these data structures, the paper also showed which edges of the graph to crowdsource next in a closed crowdsourcing model. Finally, the paper presented the results of extensive experiments that studied the cost and result quality as a function of the quorum and error rate. The surprising result was that both cost and result quality could be improved as compared to the baseline approaches that use crowdsourcing to compare all pairs of entities.

9 Acknowledgments

We would like to thank Ludwig Busse and Joachim Buhmann from the ETH machine learning group for useful discussions and input on the theory of decision functions and classification problems. This work was partially funded by the Swiss National Science Foundation as part of the CrowdDB project.

References

[1] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. E. Whang, and J. Widom. Swoosh: a generic approach to entity resolution. *VLDB J.*, 18(1):255–276, 2009.

[2] W. W. Cohen. Learning and discovering structure in web pages. *IEEE Data Eng. Bull.*, 26(3):3–10, 2003.

[3] A. Doan, M. J. Franklin, D. Kossmann, and T. Kraska. Crowdsourcing applications and platforms: A data management perspective. *PVLDB*, 4(12):1508–1509, 2011.

[4] X. Dong, A. Y. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In Özcan [15], pages 85–96.

[5] X. L. Dong and D. Srivastava. Large-scale copy detection. In Sellis et al. [19], pages 1205–1208.

[6] R. Fagin and E. L. Wimmers. A formula for incorporating weights into scoring rules. *Theoretical Computer Science*, 239(2):309–338, 2000.

[7] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. Crowddb: answering queries with crowdsourcing. In Sellis et al. [19], pages 61–72.

[8] GATE: Information Extraction. http://gate.ac.uk/ie/.

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

[10] O. Hassanzadeh, F. Chiang, R. J. Miller, and H. C. Lee. Framework for evaluating clustering algorithms in duplicate detection. *PVLDB*, 2(1):1282–1293, 2009.

[11] M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. In M. J. Carey and D. A. Schneider, editors, *SIGMOD Conference*, pages 127–138. ACM Press, 1995.

[12] G. J. Klir and T. A. Folger. *Fuzzy Sets, Uncertainty, and Information*. Prentice Hall, 1988.

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

[14] A. Marcus, E. Wu, S. Madden, and R. C. Miller. Crowdsourced databases: Query processing with people. In *CIDR*, pages 211–214. www.crdrdb.org, 2011.

[15] F. Özcan, editor. *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*. ACM, 2005.

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

[17] M. Schulze. A new monotonic, clone-independent, reversal symmetric, and condorcet-consistent single-winner election method. *Social Choice and Welfare*, 36:267–303, 2011. 10.1007/s00355-010-0475-4.

[18] R. Sedgewick. *Algorithms in C++ - parts 1 - 4: fundamentals, data structures, sorting, searching (3. ed.)*. Addison-Wesley-Longman, 1999.

[19] T. K. Sellis, R. J. Miller, A. Kementsietsidis, and Y. Velegrakis, editors. *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*. ACM, 2011.

[20] D. Suciu and N. N. Dalvi. Foundations of probabilistic answers to queries. In Özcan [15], page 963.

[21] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.

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

[23] S. E. Whang and H. Garcia-Molina. Developments in generic entity resolution. *IEEE Data Eng. Bull.*, 34(3):51–59, 2011.

[24] S. E. Whang, D. Marmaros, and H. Garcia-Molina. Pay-as-you-go entity resolution. *IEEE Transactions on Knowledge and Data Engineering*, 2012.

[25] F. Widmer. Memoization of crowd-sourced comparisons. February 2012.