Crowdsourcing Algorithms for Entity Resolution

Norases Vesdapunt
Stanford University
norases@cs.stanford.edu

Kedar Bellare
Facebook
kedar.bellare@gmail.com

Nilesh Dalvi
Facebook
nileshdalvi@gmail.com

ABSTRACT

In this paper, we study a hybrid human-machine approach for solving the problem of Entity Resolution (ER). The goal of ER is to identify all records in a database that refer to the same underlying entity, and are therefore duplicates of each other. Our input is a graph over all the records in a database, where each edge has a probability denoting our prior belief (based on Machine Learning models) that the pair of records represented by the given edge are duplicates. Our objective is to resolve all the duplicates by asking humans to verify the equality of a subset of edges, leveraging the transitivity of the equality relation to infer the remaining edges (e.g. $a = c$ can be inferred given $a = b$ and $b = c$). We consider the problem of designing optimal strategies for asking questions to humans that minimize the expected number of questions asked. Using our theoretical framework, we analyze several strategies, and show that a strategy, claimed as "*optimal*" for this problem in a recent work, can perform arbitrarily bad in theory. We propose alternate strategies with theoretical guarantees. Using both public datasets as well as the production system at Facebook, we show that our techniques are effective in practice.

1. INTRODUCTION

Entity Resolution (ER) is the problem of identifying records in a database that refer to the same underlying real-world entity. ER is a challenging problem since the same entity can be represented in a database in multiple ambiguous and error-prone ways. Recently, the availability of crowdsourcing resources such as Amazon Mechanical Turk (AMT) has provided new opportunities for improved ER, since humans are usually better than machines at performing ER tasks. This is because humans can exploit domain knowledge and other expertise to identify duplicates more easily. For example, given four places (a) `Stanford University`, (b) `Standford Univ.`, (c) `Stanford` and (d) `Stanford University Church`, it is easy for a human to tell that (a), (b) and (c) are about the same entity and (d) refers to a different entity, whereas it would be very challenging for an automated system to do so.

In this paper, we study a hybrid human-machine approach for ER where a machine learned model first assigns candidate pairs of records a probability about how likely they are to be duplicates, and then we ask humans questions about record pairs until we have completely resolved all records in our database. Asking humans about all possible record pairs can be prohibitively expensive and time-consuming. If our database contains n records we may need to ask $O(n^2)$ pairwise questions for ER. In general, we can exploit the transitive nature of the equality relation to reduce the number of questions we ask the crowd [16]. For example, consider the `Stanford` example described above. If the crowd first answers $a = b$ and then $a \neq d$, we can infer that $b \neq d$, therefore saving one question. On the other hand, if the crowd first answers $a \neq d$ and then $b \neq d$, we still need to ask the crowd about whether $a \overset{?}{=} b$. The example shows that the order in which we ask questions can affect the number of questions required to completely resolve all duplicates. In this paper, we consider the problem of devising optimal strategies for asking questions to the crowd, based on the pairwise matching probabilities, that minimize the expected numbers of questions required.

Recently, Wang et al. [16] posited that the strategy of asking questions in decreasing order of pairwise probability is the *optimal* strategy for this problem. However, we show in this paper that the claim is not correct. In fact, we show that computing the optimal strategy is NP-hard and furthermore, we prove that Wang et al.'s strategy can be $\Omega(n)$ worse than optimal, where n is the number of records in the database.

We analyze two alternate strategies and provide theoretical bounds on the cost of each strategy. The first strategy simply asks pair of records in random order. Surprisingly, although this approach ignores probabilities altogether, we find that it performs well both in theory and practice. Although this random strategy has been used in several other papers as baseline (e.g., [10, 16, 18]), all the work that we know of does not prove the theoretical bounds and this is non-trivial. Our second strategy fixes a priority over records based on the expected number of duplicates and then for each record asks whether it is a duplicate of a higher-priority record in decreasing order of matching probability until a match is found. We theoretically prove that both strategies are at most $O(k)$ worse than optimal, where k is the expected number of clusters. We compare various strategies on

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/3.0/. Obtain permission prior to any use beyond those covered by the license. Contact copyright holder by emailing info@vldb.org. Articles from this volume were invited to present their results at the 40th International Conference on Very Large Data Bases, September 1st - 5th 2014, Hangzhou, China.
Proceedings of the VLDB Endowment, Vol. 7, No. 12
Copyright 2014 VLDB Endowment 2150-8097/14/08.

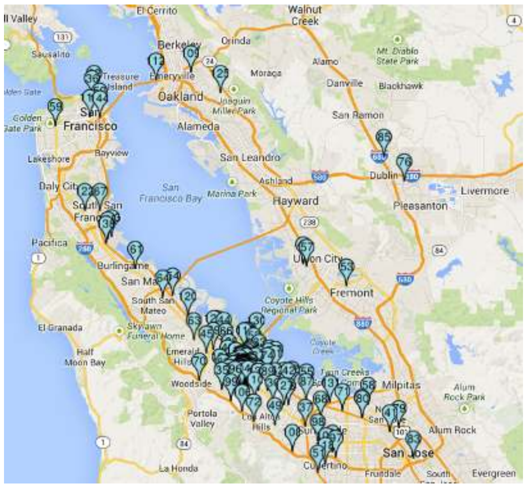
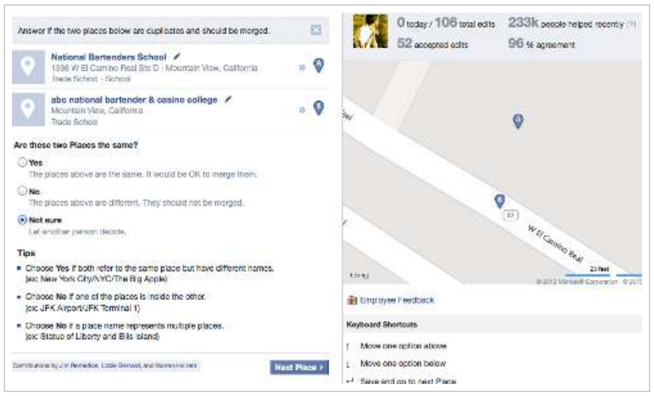


Figure 1: Duplicates of `Stanford University`


Answer if the two places below are duplicates and should be merged.

National Bartenders School
1336 W El Camino Real Ste D · Mountain View, California
Trade School · School

abc national bartender & casino college
Mountain View, California
Trade School

Are these two Places the same?

○ **Yes**
The places above are the same. It would be OK to merge them.

○ **No**
The places above are different. They should not be merged.

[✓ **Not sure**]
Let another person decide.

Tips
- Choose **Yes** if both refer to the same place but have different names. (ex: New York City/NYC/The Big Apple)
- Choose **No** if one of the places is inside the other. (ex: JFK Airport/JFK Terminal 1)
- Choose **No** if a place name represents multiple places. (ex: Statue of Liberty and Ellis Island)

Contributions by Jim Remedios, Lizzie Bennett, and Warren Helmers

Next Place ›

0 today / 106 total edits 233k people helped recently
52 accepted edits 96 % agreement

Employee Feedback

Keyboard Shortcuts

↑ Move one option above
↓ Move one option below
↵ Save and go to next Place

Figure 2: Crowdsourcing Flow

two real-world datasets and show that our node-prioritized strategy can save an order of magnitude more questions over Wang et al. [16]'s in practice. We also demonstrate that both our strategies are robust to adversarial probabilities. Although this scenario is not common, it can indeed arise when there are systematic biases in a machine learning system for entity resolution.

The summary of our contributions is as follows:

- We formulate the problem of minimizing the expected number of crowdsourcing questions for deduplication (Section 3).
- We analyze worst case bounds and prove that Wang et al. [16]'s strategy can be $\Omega(n)$ worse than optimal. We also show that finding the optimal strategy is NP-hard (Section 4).
- We present the randomized and node-prioritized strategies and prove that they are at most $O(k)$ worse than optimal (Section 5).
- We evaluate our algorithms against optimal, and Wang et al. [16]'s using real-world datasets (Section 6).

2. DEDUPLICATION AND THE CROWD

In this section, we describe our deduplication problem, and give an overview of the Facebook Crowdsourcing system.

2.1 Deduplication

Our motivation for this work was deduplicating the Facebook places database. Our places database contains hundreds of millions of places across the world and predominantly consists of user-generated content (UGC). Facebook lets users search for places and check-in to them. When users cannot find a place they are trying to check-in to, they can create a new place. The place then gets added to the database and is available for subsequent users to search and check-in to.

There are various reasons why duplicate records arise in our database. The place that the user is trying to check-in to might exist in the database, but not in the results of the search query. Search failures can happen for two reasons: (1) the user search query might have a typo, alternate spelling, abbreviation, etc., and, (2) the user might try to check-in to a place from a physical location that is far away from the actual place location, e.g., trying to check-in to a restaurant on the way back home. These failures might result in the place not being in the search results or not being ranked at the top in the search results. In these situations the user might simply choose to create a new place instead of refining or altering the search query. Given the extremely high volume of Facebook check-ins, even a very small fraction of search failures, for the reasons outlined above, can result in a large number of duplicates.

The fact that our database predominantly consists of UGC makes it an extremely challenging deduplication problem. When users create places, they do not necessarily add detailed attributes. E.g., a user trying to check-in to a restaurant might not know the full street address, the business phone or the website for the restaurant. This leads to severe attribute sparsity, reducing the effectiveness of automated deduplication techniques. User generated content also implies that there is no standardization in terms of naming places. Examples of this includes descriptions like "`GocHi Japanese Fusion Tapas`" vs. "`Gochi Asian Restaurant`", abbreviations like "`JF Kennedy Intl. Arpt.`", misspellings, alternate spellings, and so on.

Further, deduplication often requires domain knowledge that is hard to completely capture algorithmically. For example, the pairs "`Newpark Mall Gap Outlet`" and "`Newpark Mall Sears Outlet`" have a very high string similarity, but separating them requires the domain knowledge that they represent two different store names in the same mall. Deduplication also frequently requires geographical knowledge. For example, Fig. 1 shows all the places called "`Stanford University`" that users have created in the San Francisco Bay Area, along with the locations where they were created. Merging them requires the geographical knowledge that there is a single Stanford University campus in the entire region.

We train a machine learning model that predicts record pairs that are potential duplicates, along with confidences. However, in light of the challenges described in this section, we use our model in conjunction with crowdsourcing. Crowdsourcing allows us to discover false positives and false negatives that arise from imperfect machine learned probabilities.

2.2 Facebook Crowdsourcing System

Facebook lets any user suggest edits to the places database through its crowdsourcing system called *Places Editor* [1]. Users can suggest attributes like `address`, `phone`, `website` and `category` for places with missing attributes, review suggestions by other users, as well as review potential duplicates. Figure 2 shows a snapshot of the interface for reviewing duplicates. The interface shows a pair of places, the attributes of each place, and a map containing the locations where they were created. We use this interface to review the candidate pairs generated by our machine learned model as described in the previous section. We currently ignore when users answer "Not Sure" but it can be useful for estimating the difficulty of the candidate pairs. The crowdsourcing interface gets hundreds of thousands of suggestions each day from the users.

Note that, as with any crowdsourcing system, we cannot assume that the users are always correct. Hence, we have to ask the same question to multiple users to improve the confidence of the answers. Several techniques have been proposed in the literature for reconciling the answers from multiple users, which range from simple majority voting to sophisticated matrix factorization techniques [6, 9, 12]. Facebook crowdsourcing system maintains a reliability score for each user based on all the previous suggestions. Given any question that we want to resolve, the system repeatedly asks users the question until "*enough*" users with "*enough*" reliability "*agree*" on the answer. The exact details are not important. For the purpose of this paper, we assume that we have an abstraction that, given any question, reliably returns the truth value of the question using the crowd.

2.3 Crowdsourcing Entity Resolution

When resolving duplicate candidate pairs, the order in which they are resolved matters. We illustrate with an example.

Example 1. *Consider the following three records:*

a	:	`SFO San Francisco International Airport`
b	:	`Starbucks San Francisco International Airport`
c	:	`Starbucks SFO`

Records b and c are duplicates, while a refers to a distinct entity. Suppose all the three pairs (a,b), (b,c) and (a,c) are generated as duplicate candidates that need to be reviewed by the crowd. If the pair (b,c) is asked first, followed by (a,c), we gain the knowledge that b is same as c and a is distinct from c, which allows us to infer that a is distinct from b. On the other hand, if we ask (a,b) first and (a,c) next, we don't learn anything about the third edge, so we have to review the third pair as well. Thus, different orderings require different number of crowd labels.

The focus of this paper is to design strategies for reviewing duplicate pairs, based on the confidences given by the machine learned models, that minimize the crowdsourcing effort in expectation. The next section formally defines the problem.

3. PROBLEM FORMULATION

Let $\mathbb{X} = \{x_1, \cdots, x_n\}$ be a set of records. Let $G(\mathbb{X}, E)$ be an undirected graph on $\mathbb{X}$. Let p be a function that assigns, to each edge $e \in E$ a number $p(e) \in [0,1]$. Let $\mathbf{P}$ denote the probability distribution on undirected graphs over $\mathbb{X}$, where each edge (x_i, x_j) is chosen independently with probability $p(x_i, x_j)$. In other words, $p(x_i, x_j)$ denotes the probability that x_i and x_j are *matching*. x_i matches x_j when they refer to the same real-world entity. Therefore, an edge (x_i, x_j) exists iff x_i and x_j are matching.

We say that a graph C over $\mathbb{X}$ is a *clustering* of $\mathbb{X}$ if it is transitively closed. Thus C is a partition of $\mathbb{X}$ into cliques where all records in each clique represents a distinct real-world entity. We call each clique a *cluster* of C. Let $L(G)$ denote the set of subgraphs of G which are clusterings of $\mathbb{X}$.

Let Δ denote the event that a graph chosen from the distribution $\mathbf{P}$ is a clustering. Let $\mathbf{Q} = \mathbf{P}(. \mid \Delta)$ denote the probability distribution $\mathbf{P}$ conditioned on the event Δ. Thus, $\mathbf{Q}$ defines a distribution over $L(G)$.

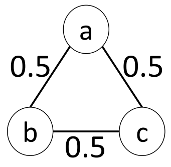


Figure 3: Problability Distribution $\mathbf{P}$ for $\mathbb{X} = \{a, b, c\}$

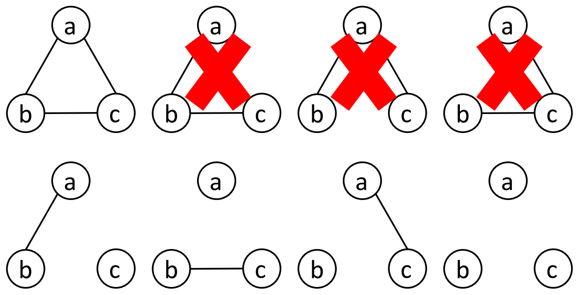


Figure 4: $L(G)$ for Figure 3

Example 2. *Consider $\mathbf{P}$ and $\mathbb{X}$ in Figure 3. Each record pair has the match probability of 0.5, i.e., $p(a,b) = p(b,c) = p(a,c) = 0.5$. Out of all 8 possible worlds, three of them (marked with red crosses in Figure 4) are not transitively closed. Thus, there are 5 clusterings $L(G)$. For any clustering C in Figure 4, $\mathbf{Q}(C) = 0.2$.*

Querying Humans We can ask a human to verify whether an edge exists or not. Recall that an edge connects two records that are matching (duplicates). We can ask a human *questions* of the form "Does x_1 represent the same real-world entity as x_2?" Let us denote this question as $\mathsf{query}(x_1, x_2)$. Let us assume that humans are perfect (i.e., asking humans is equivalent to asking an oracle). Humans make mistakes but we can use simple techniques such as majority vote to reduce human errors (or sophisticated techniques discussed in [6, 9, 12]). There are two possible responses for $\mathsf{query}(x_1, x_2)$: YES and NO. A human answers YES (NO) when x_1 and x_2 are matching (non-matching).

Evaluation Strategies Consider an unknown clustering C along with an oracle access to C (abstraction for querying perfect crowds) that, given a pair of nodes x_1, x_2, tells whether C contains the edge (x_1, x_2). We want to find a strategy that finds out C with as few oracle calls as possible. We can represent a strategy π by a decision tree whose internal nodes are pairs of records in $\mathbb{X}$ with two outgoing edges corresponding to the response of the oracle. Each leaf is labeled with a clustering C such that C is the unique clustering of $\mathbb{X}$ that is consistent with all oracle responses. The cost of the strategy, denoted $\texttt{cost}(\pi)$, is the expected number of oracle calls that π makes on a clustering drawn from the distribution $\mathbf{Q}$. We define the crowdsourcing problem as:

PROBLEM 1 (CROWD-CC). *Given an problem instance $(G(\mathbb{X}, E), p)$, find the strategy π with the smallest $\texttt{cost}(\pi)$.*

In the rest of the paper, we assume that $p(e) \in (0, 1)$ for all edges e, as $p(e) = 0$ is equivalent to removing the edge and $p(e) = 1$ is equivalent to merging the edge in the original problem definition. Further, we assume that G is connected, since otherwise, we can independently work on each of the connected component.

4. COMPLEXITY ANALYSIS

Let $(G(\mathbb{X}, E), p)$ be a problem instance, where G is a connected graphs with $|\mathbb{X}| = n$, and $p(e) \in (0, 1)$ for all edges $e \in E$. Let us denote the optimal strategy by π_{opt}. We want to analyze the complexity of computing π_{opt}, and compare the performance of various heuristics w.r.t. the optimal.

Given a set of edges T along with the response of the oracle on the edges, we say the edges positively (negatively) infer an edge e, if for all $C \in L(G)$ consistent with T, we have $e \in C$ ($e \notin C$).

LEMMA 3. *Let $T = T^+ \cup T^-$ be a set of edges along with the oracle responses, where T^+ are the edges with YES response, and T^- with NO response.*

1. *An edge (x_1, x_2) can be positively inferred from T iff there is a path from x_1 to x_2 using T^+ edges.*
2. *An edge (x_1, x_2) can be negatively inferred from T iff there exists nodes x_1', x_2', s.t. (1) there is a path from x_1 to x_1' using T^+ (2) there is a path from x_2 to x_2' using T^+, and (3) the edge (x_1', x_2') is either absent in G or belongs to T^-.*

Lemma 3 is the prior result from Wang et al. [16]. See their Lemma 1 for more details.

We assume that a strategy never makes redundant queries, i.e., it never queries edges which can be inferred, positively or negatively, from the previous query responses.

4.1 Worst case bounds

Given a strategy π, let $\texttt{cost}(\pi)/\texttt{cost}(\pi_{opt})$ denote the approximation ratio of π. The main result of this section is an upper bound on the approximation ratio of any arbitrary strategy. In particular, we show that every strategy is at most $O(n)$ worse than the optimal.

Let $C \in L(G)$ be any clustering of $\mathbb{X}$. For a strategy π, let $\texttt{cost}(\pi, C)$ denote the number of queries that π makes on the clustering C. Thus, $\texttt{cost}(\pi, C)$ lies in the range $[1, n^2]$. Given a cluster $c \in C$, let $\texttt{edges}(c)$ denote all the edges in c. For pairs of clusters $c_1, c_2 \in C$, let $\texttt{edges}(c_1, c_2)$ denote all edges (n_1, n_2) in G such that $n_1 \in c_1, n_2 \in c_2$.

LEMMA 4. *For any cluster $c \in C$, π asks exactly $|c| - 1$ edges in $\texttt{edges}(c)$.*

PROOF. We will show that the set of edges that π queries in $\texttt{edges}(c)$ must form a spanning tree of c. It is easy to see that the set cannot contain a cycle, because the last edge in the cycle can be positively inferred from the previous edges, and hence, will be redundant.

Next, we show that the queries in $\texttt{edges}(c)$ must connect c. If not, then we can divide c into subgraphs c_1 and c_2 such that π does not query any edges across them. Let C' be the clustering obtained from C by breaking c into c_1 and c_2. Then, using the given queries, π cannot distinguish between C and C', which is a contradiction.

Thus, queries in *edges*(π, c) form a spanning tree of c, and thus have size exactly $|c| - 1$. □

LEMMA 5. *For any cluster $c_1, c_2 \in C$, if $\texttt{edges}(c_1, c_2) = c_1 \times c_2$, then π queries at least one edge in $\texttt{edges}(c_1, c_2)$.*

PROOF. Let C' be the clustering obtained from C by merging c_1 and c_2. Since G contains all possible edges between c_1 and c_2, $C' \in L(G)$. If π does not query any edge in $\texttt{edges}(c_1, c_2)$, it cannot distinguish between C and C'. □

LEMMA 6. *For any strategy π and clustering C,*

$$\texttt{cost}(\pi, C) \leq 2(n + 1).\texttt{cost}(\pi_{opt}, C)$$

PROOF. Recall that C is a set of clusters (cliques). We divide the clusters in C into two groups, C_1 and C_2, where C_1 is the set of all the singleton clusters and C_2 contains the rest. Let $|C_1| = k_1$ and $|C_2| = k_2$. The total number of nodes in C_1 is exactly k_1 (since all are singleton), while the total number of nodes in C_2 is $n - k_1$. Since each cluster in C_2 is at least of size 2, we have $k_2 \leq (n - k_1)/2$.

The total number of intra-cluster edges queried by any strategy, using Lemma 4, is given by

$$\sum_{c \in C} |c| - 1 = n - |C| = n - k_1 - k_2$$

The total number of inter-cluster edges can be divided into two groups: (1) edges between two clusters in C_1, and (2) all other inter-cluster edges. Let G contain l edges between clusters in C_1. Since all clusters in C_1 are singleton, by Lemma 5, any strategy must query all these edges. Thus,

$$\texttt{cost}(\pi_{opt}, C) \geq (n - k_1 - k_2) + l \geq (n - k_1)/2 + l$$

Further, there can be at most $n.(n - k_1)$ edges in the second group. Thus,

$$\begin{aligned}\texttt{cost}(\pi, C) &\leq (n - k_1 - k_2) + l + n.(n - k_1)\\ &\leq (n + 1)(n - k_1) + l\end{aligned}$$

Thus, the Lemma follows. □

The following is immediate.

THEOREM 7. *For any strategy π,*

$$\texttt{cost}(\pi) \leq 2(n + 1).\texttt{cost}(\pi_{opt})$$

Thus, any strategy is at most $O(n)$ worse than optimal.

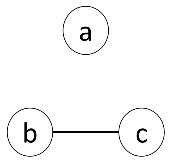


Figure 5: Example of Two Clusters: $\{a\}$ $\{b,c\}$

4.2 Analysis of Wang et al. [16] strategy

Wang et al. [16] observe that it is beneficial to first query the edges which are more likely to be true, as the following example demonstrates.

EXAMPLE 8. *Suppose we have $\mathbb{X} = \{a, b, c\}$. Consider the clustering C shown in Fig. 5. If we query in the order (a, b), (a, c), (b, c), we will have to query all the three edges in order to infer C. On the other hand, if we query in the order (b, c), (a, b), (a, c), then, after evaluating the first two edges, we can infer the value of the third edge without querying it. Thus, it is beneficial to first query edges that are likely to be in the clustering.*

Let π_{dec} denote the strategy where edges are queried in the decreasing order of their probability $p(e)$, skipping the edges whose values can be inferred from the previous answers. Based on the intuition of the above example, Wang et al. [16] posit, without a formal proof, that π_{dec} is the "*optimal*" strategy for the problem. However, in this paper, we show that the problem of finding the optimal strategy is NP-hard. Further, we show that there exist instances of the problem where π_{dec} performs $\Omega(n)$ worse than the optimal. In light of Theorem 7, this shows that π_{dec} can be as bad as an arbitrary strategy can be.

We start by showing a relationship between our CROWD-CC problem and the correlation clustering problem. In correlation clustering [4], we are given a complete graph $H(V, E^+, E^-)$ on vertices V, where edges are divided into two sets, positive edges E^+ and negative edges E^-. The problem is to find a clustering of V that minimizes the total number of *disagreements*, defined as positive edges going across clustering or negative edges within clusters.

Before we present a proof, we describe an example when π_{dec} is suboptimal while correlation clustering finds an optimal order.

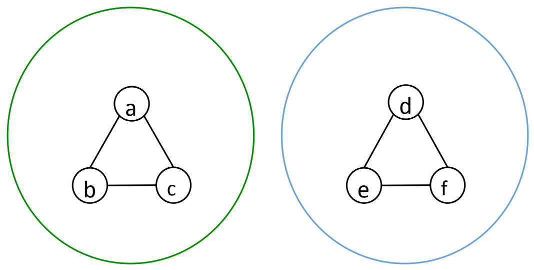


Figure 6: Counter Example for π_{dec}

EXAMPLE 9. *In figure 6, we have the ground truth with two cliques (clusters) of size 3. Each clique represents a real-world entity, i.e., all nodes in the clique are duplicates. Nodes in different cliques are not duplicates. For example, a and b are duplicates while a and d are not duplicates.*

Optimal Order π_{opt}: *The optimal strategy asks two questions for pairs in each clique and one question for a pair between the two cliques. For instance, one possible optimal strategy queries (a, b) and (b, c) (pairs within the left clique) and merges them using positive transitive relations. Then it queries pairs (d, e) and (e, f) (pairs within the right clique) and merges them similarly. Finally, it asks a pair (a, d) which will answer NO, thus, completely resolving the graph.*

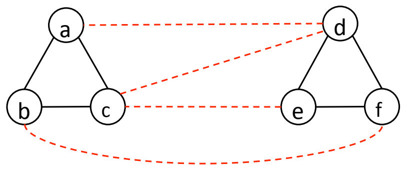


Figure 7: Probability Distribution **P** for Figure 6

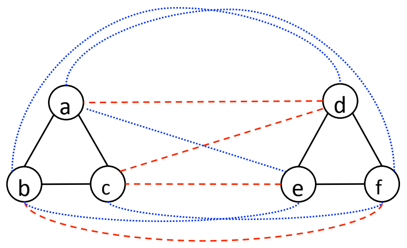


Figure 8: Correlation Clustering Evaluation

In reality, we do not know whether an edge exists between nodes or not and we use probability estimates. These estimates can be noisy because the machine learning model may make erroneous predictions. Let Figure 7 be our probability graph. Let the probability estimates for edges between nodes inside a clique (solid edges) be $q = 1 - \epsilon$. Let the probability estimates for edges between nodes across two cliques (dashed edges) be $p = 1 - \epsilon'$ for some $\epsilon' < \epsilon$, i.e., p is slightly greater than q. Therefore, E^+ consists of all of these solid and dashed edges. All other edges are $\in E^-$.

Decreasing Edge Probablity Order π_{dec}: *In this case, π_{dec} will ask cross edges first: (a, d), (c, d), (c, e), and (b, f). This order will receive NO for all questions above. Then it proceeds to ask edges inside the two cliques and will receive YES for all questions. The algorithm can apply positive relations here and ask two questions for each clique. This order asks three questions more than optimal in this case because it asks all cross edges first.*

Correlation Clustering: *On the other hand, if we use correlation clustering, we can achieve the optimal order in this case. In correlation clustering, the objective function is trying to minimize the disagreements (cost), namely the number of edges across clusters and the number of non-edges within clusters. For example, if we split the graph into two clusters $\{a, b, c\}\{d, e, f\}$, the disagreements are dashed edges shown in Figure 8: (a, d), (c, d), (c, e), and (b, f). Thus, the cost is 4. On the other hand, if we put all nodes into one cluster $\{a, b, c, d, e, f\}$, the disagreements are dotted edges (these are non-edges) shown in Figure 8: (a, e), (a, f), (c, f), (b, d), and (b, e). Thus, the cost is 5. Since the cost* *of splitting the graph into two clusters is lower than putting every node in one cluster, correlation clustering will choose to split. We can easily see that other splits are more costly. Therefore, correlation clustering will find the true clusters. Asking questions for edges within the cluster first and then ask one question for an edge across clusters will yield the optimal order.*

Given an instance $H(V, E^+, E^-)$ of correlation clustering problem, we create an instance $(G(\mathbb{X}, E), p)$ of CROWD-CC as follows. Let $\epsilon > 0$ be a real number. Let $\mathbb{X} = V$, $E = E^+ \cup E^-$, and

$$p(e) = \begin{cases} 1-\epsilon & \text{for } e \in E^+, \\ \epsilon & \text{for } e \in E^-. \end{cases}$$

We now show that π_{dec} can be $\Omega(n)$ worse than π_{opt}. Let C_0 denote the clustering of $\mathbb{X}$ that minimizes the disagreements in H. Assume for now that C_0 is unique. Let k_0 denote the number of disagreements of C_0, and let $\epsilon_0 = \frac{\epsilon}{1-\epsilon}$.

LEMMA 10. *Let H, G, C_0 be as defined above, and let $\epsilon_0 = n^{-n-2}$. Then, for any strategy π,*

$$\texttt{cost}(\pi) = \texttt{cost}(\pi, C_0) + O(1)$$

PROOF. Let $m = |E|$. For any clustering C with k disagreements, it is straightforward to see that

$$\mathbf{Q}(C) = \frac{\epsilon^k(1-\epsilon)^{m-k}}{\mathbf{P}(\Delta)} = \frac{(1-\epsilon)^m}{\mathbf{P}(\Delta)}\epsilon_0^k$$

Thus, for any clustering $C \neq C_0$,

$$\mathbf{Q}(C) \leq \epsilon_0 \mathbf{Q}(C_0)$$

Since there are less than n^n possible clusterings (each clustering can be represented as a function from node to a canonical node, and there are at most n^n functions from $[1, n]$ to $[1, n]$),

$$\mathbf{Q}(C_0) = 1 - \sum_{C \neq C_0} \mathbf{Q}(C) > 1 - n^n \epsilon_0 \mathbf{Q}(C_0) = 1 - \frac{1}{n^2}$$

Thus,

$$\begin{aligned} \texttt{cost}(\pi) &\geq Q(C_0)\texttt{cost}(\pi, C_0) \\ &> \texttt{cost}(\pi, C_0) - \frac{\texttt{cost}(\pi, C_0)}{n^2} \\ &\geq \texttt{cost}(\pi, C_0) - 1 \end{aligned}$$

Similarly,

$$\begin{aligned} \texttt{cost}(\pi) &= Q(C_0)\texttt{cost}(\pi, C_0) + \sum_{C \neq C_0} Q(C)\texttt{cost}(\pi, C) \\ &\leq \texttt{cost}(\pi, C_0) + \sum_{C \neq C_0} Q(C)\texttt{cost}(\pi, C) \\ &\leq \texttt{cost}(\pi, C_0) + n^2 \sum_{C \neq C_0} Q(C) \\ &\leq \texttt{cost}(\pi, C_0) + n^2(1 - \mathbf{Q}(C_0)) \\ &< \texttt{cost}(\pi, C_0) + 1 \end{aligned}$$

Thus, the lemma follows. □

The lemma tells us that the cost of any strategy is dominated by the cost it incurs on the clustering C_0. Now, to show that strategy π_{dec} can be $\Omega(n)$ worse than the optimal, we need the following lemma.

LEMMA 11. *There exists an instance $H(V, E^+, E^-)$ of the correlation clustering problem such that $|V| = n$, and the clustering C_0 which minimizes the total disagreements has the following properties:*

1. *C_0 has two clusters.*
2. *There are $\Omega(n^2)$ positive edges crossing the two clusters in C_0.*

Before we prove the Lemma, let's see how this gives us the desired result. Let $\{c_1, c_2\}$ denote the two clusters of C_0. Consider the corresponding $(G(\mathbb{X}, E), p)$ crowdsourcing problem. Consider a strategy π_0 that queries (a) some spanning tree of c_1, then (b) some spanning tree of c_2, then (c) an edge across, and (d) all remaining edges in some order. For the clustering C_0, π_0 will be done after step (c), and hence, $\texttt{cost}(\pi_0, C_0) = |c_1| - 1 + |c_2| - 1 + 1 = n - 2$. Hence, by Lemma 10, $\texttt{cost}(\pi_0) = \texttt{cost}(\pi_0, C_0) + O(1) = O(n)$. The strategy π_{dec} queries in decreasing order of p. Since all positive edges (edges in E^+) have $p(e) = 1 - \epsilon$, π_{dec} can choose to query them in any order. In particular, π_{dec} may choose to first query all positive edges crossing c_1 and c_2. We can also choose $p(e) = 1 - \epsilon'$ for all positive edges crossing the clusters, for some $\epsilon' < \epsilon$, to guarantee that π_{dec} chooses them first. For the clustering C_0, by Lemma 3, none of the edges crossing the clusters can be inferred from the rest. Hence, for C_0, π_{dec} ends up querying all $\Omega(n^2)$ positive edges crossing two clusters. Thus, $\texttt{cost}(\pi_{dec}, C_0) = \Omega(n^2)$, and hence, $\texttt{cost}(\pi_{dec}) = \Omega(n^2)$. Since there exists a strategy π_0 with cost $O(n)$, π_{dec} is $\Omega(n)$ worse than the optimal. It now remains to prove Lemma 11, which we show next.

PROOF. (Lemma 11) We will construct such an instance $H(V, E^+, E^-)$. Let $V = L \cup R$, where $L = \{l_1, \cdots, l_n\}$ and $R = \{r_1, \cdots, r_n\}$ are two sets of vertices, E^+ consists of all edges within L, all edges within R, and all edges (l_i, r_j) such that $i + j \equiv 0\ (mod\ 4)$, and E^- consists of all remaining edges. We will show that C_0 consists of two clusters, L and R.

Assume that $C_0 = \{c_1, c_2, \cdots, c_k\}$, where $c_1, \cdots, c_k$ partition the vertices V. Let $L(c_i)$ denote $L \cap c_i$ and $R(c_i)$ denote $R \cap c_i$. Call a cluster c_i *left-heavy* if $|L(c_i)| > |R(c_i)|$, *right-heavy* if $|L(c_i)| < |R(c_i)|$, and *balanced* otherwise. Given two subsets V_1, V_2 of V, let $pos(V_1, V_2)$ denote the number of positive edges between the two sets, $neg(V_1, V_2)$ denote the number of negative edges, and let $d(V_1, V_2) = pos(V_1, V_2) - neg(V_1, V_2)$. We will show that there can be at most one left-heavy and one right-heavy clusters. Consider any two clusters c_i, c_j. If we merge the two clusters in C_0, we save $pos(c_i, c_j)$ disagreements, but introduce $neg(c_i, c_j)$ new disagreements. Total savings is given by

$$\begin{aligned} S &= d(c_i, c_j) \\ &= d(L_i, L_j) + d(R_i, R_j) + d(L_i, R_j) + d(R_i, L_j) \\ &= |L(c_i)||L(c_j)| + |R(c_i)||R(c_j)| + d(L_i, R_j) + d(R_i, L_j) \\ &= (|L(c_i)| - |R(c_i)|)(|L(c_j)| - |R(c_j)|) + \\ &\quad 2pos(L_i, R_j) + 2pos(R_i, L_j) \end{aligned}$$

If c_i and c_j are both left-heavy or both right-heavy, the first product term in the above equation will be strictly positive, and hence, S will be strictly positive, contradicting the optimality of C_0. Hence, there are at most one left-heavy and one-right heavy clusters. Further, if c_i is a balanced cluster, the first term is 0, and hence $S \geq 0$. For C_0 to be optimal, S must be exactly 0. So for any balanced cluster in C_0, we can merge it with any other cluster without affecting the number of disagreements. Let $C_0 = \{c_l, c_r, b_1, \cdots, b_k\}$, where c_l is left-heavy, c_r is right-heavy, and each b_i is balanced. Merge each balanced cluster with one of the heavy clusters to obtain $C'_0 = \{c'_l, c'_r\}$, such that C'_0 is another optimal solution that has same number of disagreements as C_0. W.l.o.g., assume that $|c'_l| \geq |c'_r|$. Thus, $|L(c'_l)| \geq n/2$. Suppose $L(c'_r)$ is non-empty. Then, if we move all of $L(c'_r)$ from c'_r to c'_l, we save at least $pos(L(c'_l), L(c'_r)) \geq |L(c'_r)|n/2$ disagreements, since each vertex in $L(c'_r)$ is connected to every vertex in $L(c'_l)$. Also, we introduce $pos(L(c'_r), R(c'_r)) \leq |L(c'_r)|n/4$ disagreements, because each L vertex is connected to at most $n/4$ vertices in R. Hence, we have a strictly better solution, which is a contradiction. Thus, $L(c'_r)$ must be empty. Similarly, $R(c'_r)$ must be empty. Hence, $C'_0 = \{L, R\}$. Hence, $C_0 = \{L, R\}$, since there cannot be a non-trivial balanced cluster that is a subset of L or R. □

Thus, we get the following result.

Corollary 12. *There exists instances of* crowd-cc *such that π_{dec} is $\Omega(n)$ worse than π_{opt}.*

4.3 Complexity of Computing π_{opt}

Lemma 10 shows that when all probabilities are close to 0 or 1, then the problem of finding the optimal strategy becomes closely related to the problem of finding the clustering in a graph that minimizes the number of disagreements, which is a known NP-hard problem. By extending the ideas in the proof of the Lemma, we can show the following.

Theorem 13. *The* crowd-cc *problem is* NP*-hard.*

We omit the proof for lack of space.

5. ALGORITHMS

In the previous section, we analyzed the π_{dec} algorithm, and showed that it can perform $\Omega(n)$ worse than the optimal. In this section, we propose some alternate algorithms with better theoretical guarantees.

We use query(x_1, x_2) to denote a function that asks the pair of records to humans, and returns the Boolean response, where `true` represents that the records are duplicates, and `false` otherwise.

All of our algorithms use the following subroutine, called query-cache(x_1, x_2), which is defined as follows:

Subroutine 1 query-cache(x_1, x_2)

```
1: if $(x_1, x_2)$ can be inferred from the set of human responses according to Lemma 3 then
2:     return the inferred value
3: else
4:     $r \leftarrow$ query$(x_1, x_2)$
5:     add $r$ to the set of responses
6:     return $r$
7: end if
```

We can easily implement query-cache to run in time $O(n)$. In prior work, Wang et al. [16] presented a version of query-cache called the DeduceLabel algorithm. We maintain a set of connected components connected by the positive edges. For each node, we store the mapping to its component. We also maintain, for each pair of components, where there is at least one negative edge between them. Given a pair of nodes, we can check whether they can be inferred from the current set of responses in constant time. They can be positively inferred iff their components are same, and can be negatively inferred iff their components have at least one negative edge. Also, given a new response, we can update our data structures in constant time. If it is a negative response, we simply add a negative edge between the corresponding components, if it does not already exist. If it is a positive response, we merge the two components, and union their negative edges.

Using this subroutine, the π_{dec} algorithm that we described in the previous section is formally presented in Algorithm 2.

Algorithm 2 $\pi_{dec}(G(\mathbb{X}, E), p)$

```
1: for $(x_i, x_j) \in$ decreasing $p(x_i, x_j)$ do
2:     query-cache$(x_i, x_j)$
3: end for
```

5.1 Randomized Querying

Since π_{dec} can perform badly in the worst case, we present some alternatives. The first alternative, which we call π_{rand}, is extremely simple. It queries edges in random order, as presented in Algorithm 3. Every time an edge is labeled as positive, it contracts the edge.

Algorithm 3 $\pi_{rand}(G(\mathbb{X}, E), p)$

```
1: while there are unresolved edges in $G$ do
2:     $(x_i, x_j) \leftarrow$ random edge in $G$
3:     $r \leftarrow$ query-cache$(x_i, x_j)$
4:     if $r \equiv$ true then
5:         $G \leftarrow G/(x_i, x_j)$ // contract edge $(x_i, x_j)$
6:     end if
7: end while
```

While π_{rand} seems to be inferior to π_{dec} since it completely ignores the probabilities, we show that, in theory, it has better guarantees than π_{dec} owing to randomization.

We will show that π_{rand} is at most $O(k)$ worse than optimal, where k is the expected number of clusters in the graph. Since, in general, the number of clusters is much smaller than the number of nodes, we expect π_{rand} to perform well in practice. Recall that, in contrast, π_{dec} can be $\Omega(n)$ worse than the optimal, even when there are only 2 clusters.

Given the problem instance $(G(\mathbb{X}, E), p)$, consider any valid clustering C of G that has k clusters. We will show that, on this world C, π_{rand} makes $O(k)$ times the number of queries made by π_{opt}.

Suppose $k \geq n/2$. Then, by Lemma 6, the assertion is true for any strategy. So let us assume than $k < n/2$. Let $a_1, a_2, \cdots, a_k$ be the sizes of the clusters. By Lemma 4, any solution has to ask at least $(a_1 - 1) + \cdots + (a_k - 1) = n - k =$

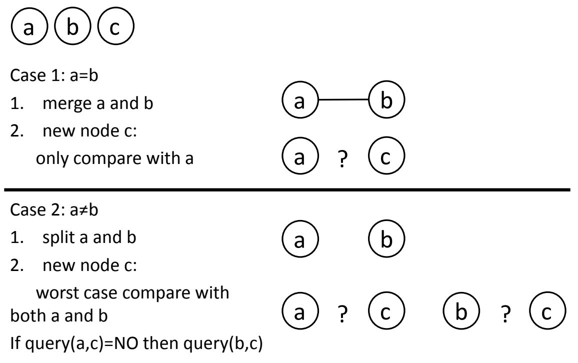


Figure 9: Illustration for π_{node}

$\Omega(n)$ edges. We will show that π_{rand} asks at most $O(kn)$ edges.

Lemma 14. *Let $(G(\mathbb{X}, E)$,$p)$ and C be as defined above. Then, if a random edge is chosen from E, with probability $\Omega(1/k)$, the edge is a positive edge in C.*

Proof. The number of positive edges in C is exactly

$$\begin{aligned}&\binom{a_1}{2} + \cdots + \binom{a_k}{2}\\ = \quad &1/2(\sum a_i^2) - n/2\end{aligned}$$

By a classic inequality for non-negative numbers,

$$(\sum a_i^2)/k \geq (\sum a_i/k)^2$$

Applying the inequality, the number of positive edges in C is at least $k/2(\sum a_i/k)^2 - n/2 = n^2/2k - n/2 = \Omega(n^2/k)$, since $k < n/2$. Since there are $O(n^2)$ total edges in G, with probability $\Omega(1/k)$, a random edge is positive. □

It follows from the Lemma that if we randomly pick edges from G, then in at most $O(k)$ steps, we will get a positive edge. When we contract the edge, the new graph has the same number of clusters, with one less node. Inductively applying the Lemma on the contracted graph, we can show that in at most $O(nk)$ steps, we can reduce the graph G to at most $O(k)$ nodes. At this point, with $O(k^2)$ more queries, we can resolve all edges. The total number of queries is $O(nk + k^2) = O(nk)$. Thus, we have the following:

Lemma 15. *Let $(G(\mathbb{X}, E)$,$p)$ be a problem instance and C be a clustering of G with k clusters. Then,*

$$\texttt{cost}(\pi_{rand}, C) = O(k)\texttt{cost}(\pi_{opt}, C)$$

By taking expectation of all possible clusterings, we have the following result.

Theorem 16. *Let $(G(\mathbb{X}, E)$,$p)$ be a problem instance and let k be the expected number of clusters. Then,*

$$\texttt{cost}(\pi_{rand}) = O(k)\texttt{cost}(\pi_{opt})$$

5.2 Node Priority Querying

While the randomized algorithm has nice theoretical guarantees, it does not use the probabilities at all. In this section, we propose an algorithm that still has similar guarantees, but can also leverage the input probabilities. The algorithm is again extremely simple: it fixes a global priority of nodes. For each node, it queries it with all nodes with a higher priority, until we get a match. We call this algorithm π_{node}, and is described in Algorithm 4. This version does not yet use probabilities, but we will show later how the algorithm can be modified to make use of them.

Algorithm 4 $\pi_{node}(G(\mathbb{X}, E)$,$p)$

```
1: for $i = 1 \to n$ do
2:     for $j = 1 \to i - 1$ do
3:         $r \leftarrow$query-cache($x_i$,$x_j$)
4:         if $r \equiv$ true then
5:             break
6:         end if
7:     end for
8: end for
```

First, we explain the intuition for π_{node} using the illustration in Figure 9.

Example 17. *Let $\mathbb{X} = \{a, b, c\}$. Let the node ordering be a,b, and then c. We will describe different cases in Figure 9 in detail. π_{node} first compares a and b. If query(a,b) answers YES (Case 1), then it merges a and b into a cluster. Then it only has to compare c with a because the answer to query(b,c) can be inferred from the answer to query(a,c) using transitivity. On the other hand, if query(a,b) answers NO (Case 2), it splits a and b into two clusters. In this case, the worst case has to compare c with both a and b. This case happens when query(a,c) answers NO because the relationship between b and c cannot be inferred using transitivity. If query(a,c) answers YES, it merges a and c into a cluster and exits.*

Next we show that, irrespective of the node ordering, the algorithm satisfies the following property.

Theorem 18. *Let $(G(\mathbb{X}, E)$,$p)$ be a problem instance and let k be the expected number of clusters. Then,*

$$\texttt{cost}(\pi_{node}) = O(k)\texttt{cost}(\pi_{opt})$$

Proof. Let C be any clustering of G with k clusters. We will show that $\texttt{cost}(\pi_{node}, C) = O(k)\texttt{cost}(\pi_{opt}, C)$.

Suppose after t iterations of the outer loop, the algorithm has processed $X_t = \{x_1, \cdots, x_t\}$ nodes. Let b_t denote the number of connected components of X_t formed by the set of positive edges (edges where query returned true) after iteration t. We will show, by induction, that $b_t \leq k$. Clearly $b_0 = 0 \leq k$, since X_t is empty. Suppose $b_t \leq k$. Since X_{t+1} contains one new node, the number of components can increase by at most 1. If $b_t < k$, this implies that $b_{t+1} \leq k$. So assume that $b_t = k$. This implies that X_t contains at least one node from each cluster of C. Since x_{t+1} is queried with all the previous nodes until a match, it will return a positive match with some node x' in X_t that belongs to the same cluster as x_{t+1}. Thus, x_{t+1} gets merged with the component of x', and hence, b_{t+1} remains k.

It follows that for each i in the outer loop, the query-cache calls query at most k times. This is because, for each of the connected components, query is called at most once. For every other member of the component, the response can be inferred from one member. Thus, the total number of queries

Algorithm 5 $\pi_{node}(G(\mathbb{X}, E),p)$, with priorities

```
1: Sort $\mathbb{X}$ in decreasing order of expected #neighbors
2: for $i = 1 \to n$ do
3:     for $j \in$ decreasing $p(x_i, x_j)$   $j = 1 \to i - 1$ do
4:         $r \leftarrow$query-cache($x_i$,$x_j$)
5:         if $r \equiv$ true then
6:             break
7:         end if
8:     end for
9: end for
```

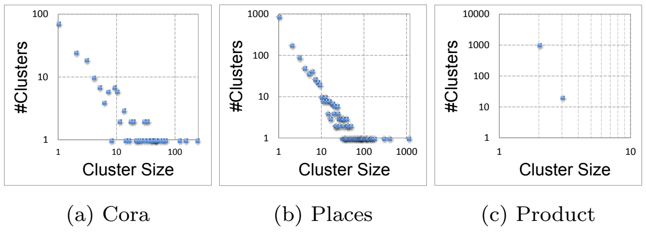


Figure 10: Cluster-Size Distribution

asked by π_{node} on this clustering is at most nk. On the other hand, the optimal algorithm requires at least $n - k$ queries. For $k < n/2$, we get $\texttt{cost}(\pi_{node}, C) = O(k)\texttt{cost}(\pi_{opt}, C)$. For $k \geq n/2$, this follows from Lemma 6.

The theorem follows by taking the expectation over all possible clusterings. □

So far, we have not made use of the input probabilities. However, they can be easily incorporated in the algorithm to give heuristics that perform really well in practice. Note that Theorem 18 holds irrespective of the ordering chosen for the inner and the outer loop. For the inner loop, we sort $\{x_1, \cdots, x_{i-1}\}$ based on decreasing probability of matches with x_i. When the probabilities are reliable, we can match x_i to the correct cluster early and break. For the outer loop, we sort by decreasing order of expected number of neighbours (sums of probabilities of outgoing edges), so that several nodes in the inner loop can be matched to it. The final algorithm is presented as Algorithm 5.

6. EXPERIMENTS

We compare all algorithms on three real datasets: Cora, Places, and Product. We know the ground truth for all of the record pairs and use all records for the testing set for all datasets. We reimplement Wang et al. [16]'s approach (π_{dec}) as described in Section 5. See Algorithm 2 and the subroutine `query-cache` for more details. For each data point of π_{rand}, we take the average recall from 100 runs for all of our experiments. We implemented our algorithms in Python and ran experiments in memory on a 1.7GHz Intel(R) Core i7 processor with 8GB of RAM.

6.1 Datasets

6.1.1 Cora

Cora [14] is a public publication dataset of 1,838 records. These records come from 190 real-world entities (clusters). The cluster size distribution is shown in Figure 10. Each record contains the following attributes: title, author, venue, date, and pages.

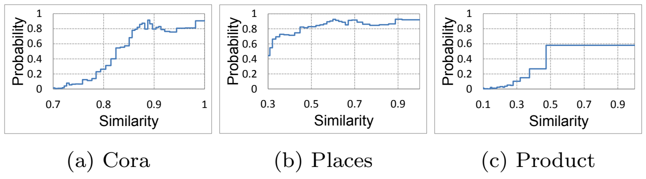


Figure 11: Similarity-Probability Mappings

We follow the similarity function as in [18] for our Cora experiments. We use the Jaro measure [19] string similarity function (ranging from 0 to 1) to compute title and author similarities between record pairs and return the average similarity. Figure 11 shows similarity-probability mappings for Cora record pairs with similarity of at least 0.7. Record pairs with similarity below 0.7 match with probability 0.15%.

We map similarities to probabilities using buckets because it is simple and effective (see Section 3.1 in [18] for more discussion). We use the all record pairs with similarity of at least 0.7 to compute these mappings so that our probabilities are computed from as many samples as possible. We first evenly divide the record pairs with similarity at least 0.7 into 80 buckets (each bucket contains about 3100 pairs). Then we use ground truth to compute the probability for each bucket.

6.1.2 Places

For our second dataset, we use data from our production system at Facebook. We constructed a dataset of 9000 entities from Facebook Places [2]. This group of entities is produced by running our machine learning model on the entire dataset, and choosing a subset of the output. There are 1,493 real-world entities (clusters) in this dataset. The cluster size distribution is shown in Figure 10.

Our similarity function is a machine learning model that predicts scores for given record pairs ranging from 0 to 1. The higher the score, the more similar the record pair is predicted by machines. Figure 11 shows similarity-probability mappings for Places record pairs with similarity of at least 0.3. Record pairs with similarity below 0.3 match with probability 1.41%.

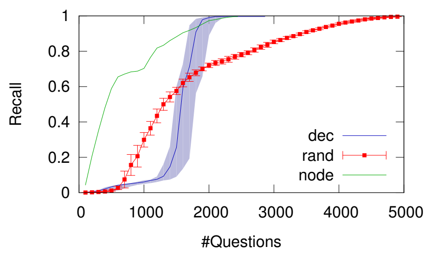
6.1.3 Product

Product [3] is a public product dataset of mappings from 1,081 *abt.com* products to 1,092 *buy.com* products. These records come from 576 real-world entities. The cluster size distribution is shown in Figure 10. Each record contains the following attributes: name and price.

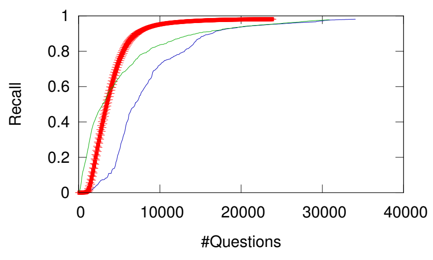
We follow the similarity function as in [15] for our Product experiments. For each record pair, we compute the Jaccard Similarity between the token sets generated from record attributes. Figure 11 shows similarity-probability mappings for Product record pairs with similarity of at least 0.1. Record pairs with similarity below 0.1 match with probability 0.0027%.

6.2 Evaluation

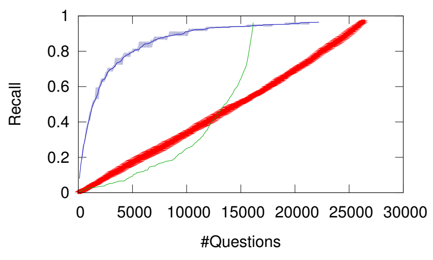
We use two different studies to evaluate our strategies. The first is the *progressive recall* study. Here, we study the pairwise recall of different strategies as a function of number of questions asked, where the pairwise recall is defined as the number of duplicate pairs that can be inferred based



(a) Cora

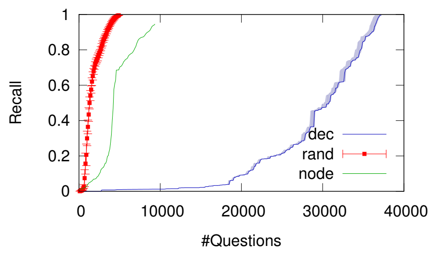


(b) Places

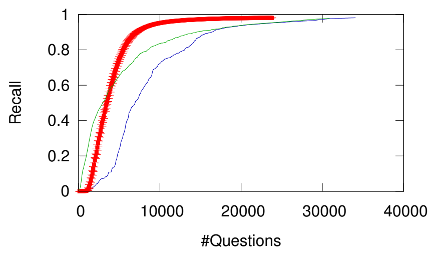


(c) Product

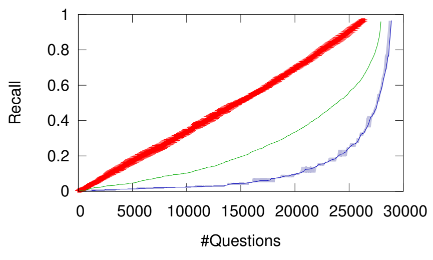
Figure 12: #Questions vs Recall



(a) Cora



(b) Places



(c) Product

Figure 13: Adversarial #Questions vs Recall

on the questions asked, as a fraction of the total number of true duplicate pairs. The second is the *complete resolution* study, where we study the number of questions required by each strategy to completely resolve the database. All of our algorithms have perfect precision, since we assume that our crowd abstraction is perfect. Thus, we will only focus on the recall.

6.2.1 Progressive Recall

Figure 12 plots the recall of various strategies as a function of the number of questions asked. We see that π_{node} is the best strategy overall. For Cora, π_{node} achieves 0.7 recall at 1,000 questions while π_{dec} and π_{rand} achieve only 0.05 and 0.3 recall respectively. To achieve 0.7 recall, π_{dec} and π_{rand} require 1,700 and 2,300 questions respectively. Similarly, π_{node} supersedes both π_{dec} and π_{rand} for Places. It is worth noting that for Places, π_{rand} requires significantly fewer questions than π_{dec} to achieve the same recall for recall levels up to 0.95. This is because π_{rand} does not use probabilities and in expectation it queries only $O(kn)$ questions where n=number of nodes and k=number of true clusters. On the other hand, π_{dec} relies on imperfect probabilities predicted by machines. For Product, π_{dec} is the best strategy for recall levels < 0.9. This is because π_{dec} asks the most likely pairs first while π_{node} finds the correct cluster for each record. π_{node} catches up at recall level 0.9 because after k clusters are discovered, it only needs to ask k questions in the worst case for each node. The relationships between x and all other nodes in different clusters can be inferred using negative transitivity. For recall levels greater than 0.9, π_{node} requires fewer questions than π_{dec}. To achieve 0.96 recall, π_{node} only requires 16,200 questions while π_{dec} requires 22,200 questions.

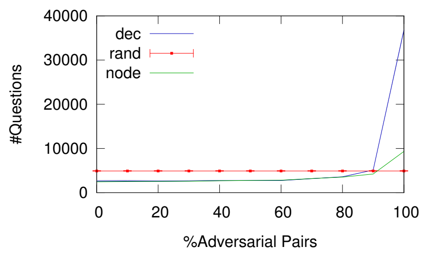
π_{dec} has a wide range (big difference between the worst case and best case) of 600 questions for Cora because there are many indistinguishable pairs. For example, there are 900 pairs with the same similarity score of 0.89.

Figure 13 shows the results for the adversarial case where the match probabilities are reversed. For example, if $p(a, b) = 0.2$, the reversal is $1 - 0.2 = 0.8$. This is the worst case because the probabilities are the opposite so π_{dec} will do the worst in expectation. This experiment shows that π_{node} is more robust to noisy probabilities than π_{dec} is. π_{rand} is the best strategy overall while π_{dec} suffers the most. For Cora and Product, π_{node} is much closer to π_{rand} than π_{dec} is. For Places, π_{node} supersedes π_{rand} for recall values below 0.5. For higher recall values, π_{node} is still close to π_{rand}. π_{dec} is the worst strategy for all recall values. Overall, π_{dec} is much worse than other strategies in the adversarial case.

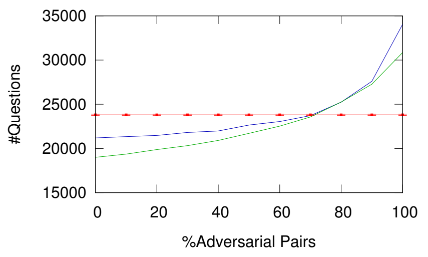
6.2.2 Complete Resolution

In these experiments, we look at the number of questions required by each strategy to completely resolve all the duplicates. We study this as a function of **1)** the percentage of adversarial pairs, and **2)** the granularity levels of probabilities. These experiments allow us to measure the sensitivity of our three strategies to match probabilities (correctness and granularity respectively).

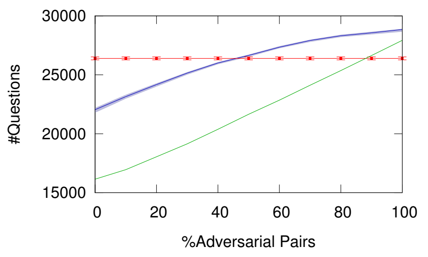
%Adversarial Pairs: Here we evaluate strategies based on their sensitivity to the correctness of match probabilities. Recall that an adversarial pair is a record pair with the match probability reversed. For example, %*Adversarial Pairs* = 10 means that 10% of the candidate pairs have their match probabilities reversed. We randomly choose the pairs to reverse the probabilities. Figure 14 plots the number of questions asked by each strategy at completion as a function of %Adversarial Pairs. It shows that π_{node} is more robust than π_{dec}. π_{node} requires fewer questions than π_{rand} for most cases except for when most pairs (80-100%) are adversarial. π_{dec} is the most robust for Cora compared



(a) Cora

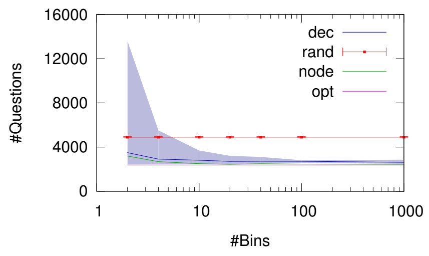


(b) Places

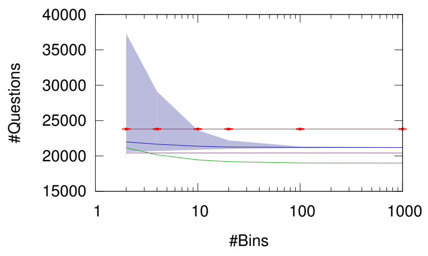


(c) Product

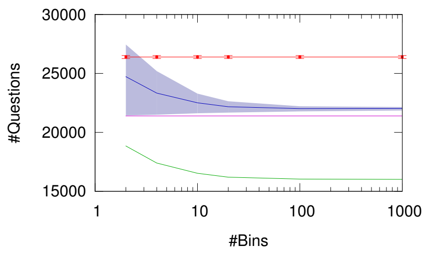
Figure 14: %Adversarial Pairs vs #Questions



(a) Cora



(b) Places



(c) Product

Figure 15: #Bins vs #Questions

to other datasets. This is because Cora has a lot of large clusters relative to its number of records. In the best case scenario, a cluster of size k remains a connected component when $k-1$ pairs are present while there are $k(k-1)/2$ pairs in the cluster. For example, let us consider a cluster of size 100 that has $100(100-1)/2 = 4,950$ candidate pairs. It is possible to delete all but 99 pairs without disconnecting the cluster. In our case, if we reverse 98% of the pairs, the remaining 2% may still connect a 100-record cluster. Therefore, both strategies π_{dec} and π_{node} are not affected when fewer than 70% of the pairs are adversarial. On the other hand, Product has clusters of size 2 and 3. 10% adversarial pairs result in both strategies requiring 5% more questions to complete the resolution. π_{dec} can tolerate only 40% and 60% adversarial pairs for Product and Places respectively.

#Bins: Let the number of bins (#Bins) denote the number of granularity levels of probability, capturing the amount of information available to the algorithm. The higher the #Bins is, the more fine-grained the probability. For example, $\#Bins = 10$ means we classify candidate pairs into 10 different bins and within the same bin we cannot distinguish which pair is more likely to match. Figure 15 plots the number of questions asked by each strategy at completion as a function of #Bins. It shows that π_{node} is the best strategy overall. For small #Bins, π_{dec} varies a lot and the worst case of π_{dec} requires many orders of magnitude more questions than π_{opt} (optimal). For Cora, we stop asking questions when all the remaining pairs are below a similarity threshold of 0.7. All algorithms achieve a recall of 0.9963 on completion. For Places, we use the similarity threshold of 0.3 and π_{dec} and π_{rand} achieve 0.981 recall while π_{node} achieves 0.978 recall. For Product, we use the similarity threshold of 0.1 and π_{dec} and π_{rand} achieve 0.9642 recall while π_{node} achieves 0.9606 recall. π_{node} saves 5,000 questions over π_{opt} but the recall loss is only 4 pairs. For Product, π_{node} saves about 28% of questions over π_{dec} consistently across different #Bins. For the other datasets, π_{node} saves about 5% of questions over π_{dec} consistently across different #Bins. Furthermore, as #Bins increases, π_{node} requires fewer questions (very close to π_{opt} at the largest #Bins).

Overall, we observe that π_{node} consistently supersedes π_{rand} and π_{dec} in our experiments on all datasets. π_{node} is also close to π_{opt}. Furthermore, π_{rand} supersedes π_{dec} in some cases.

7. RELATED WORK

Entity Resolution (ER) has been well studied (see [17] for a recent survey). Recently, many frameworks [7, 11, 13, 15, 18] have been developed to leverage humans for performing ER tasks. Marcus et. al. [13] rely solely on humans to perform joins in their Qurk system. CrowdER [15] uses a hybrid human-machine framework by first automatically detecting matching pairs or clusters that are then verified by humans. Machines are purely used as a way to filter non-duplicates rather than a way of providing an ordering over pairwise questions that the crowd should be asked. Demartini et. al. [7] dynamically generates crowdsourcing questions for record linking based on a probabilistic framework. Jeffery et. al. [11] describe Arnold, a declarative data cleaning and integration system using machines and humans. Georgescu et al. [8] proposed using crowdsourced examples for active learning in ER. None of these works exploit transitivity to reduce the human querying cost.

Recently, there have been several works that exploit transitive relations in hybrid human-machine ER. Gruenheid et. al. [10] focus on using strong and weak transitivity for tolerating errors from conflicting human votes. Their algorithm is similar to π_{rand} but they ask each question to multiple humans, and use transitivity to determine number of humans to ask a question. There is no guarantee of optimality. In contrast, we give theoretical guarantees when the crowd is perfect, assuming the existence approaches [6, 9, 12] to make crowd reliable. Whang et. al. [18] describe an approach to find the best next question to ask the crowd that maximizes the information gain. However, there are two shortcomings. First, it uses a brute force approach with $O(n^4)$ complexity, which makes it infeasible, even for very small datasets. Second, it assumes a different probabilistic model where transitive closures are performed after picking a random world, rather than randomly picking a transitively closed world. This model has undesirable properties, where small number of edges with high probabilities can create large clusters, even if there are several edges with extremely low probabilities. The closest related work to ours is Wang et. al. [16], which incorrectly claim the optimality of their techniques.

8. EXTENSIONS

Our strategies are simple and powerful that we can easily extend them. For example if we have constraints that some edges have to be (or cannot be) in the graph, we can just preprocess these edges (keeping our state transitively closed) and run our algorithms. Another possible extension is updating our probabilities after each query but proving any bound is non-trivial. For example, we can use Reflect and Correct [5] from active learning to update the probabilities after acquiring each answer from the crowd, reorder pairs based on the new probabilities, and run our strategy iteratively (i.e., each iteration gets new probabilities).

9. CONCLUSION

We studied the problem of completely resolving an entity graph using crowdsourcing where edges are annotated with probability of matching from a machine learned model. We analyzed the worst-case and optimal strategies for ordering the questions based on machine learned probabilities. In addition, it was shown that computing the optimal strategy is NP-hard. We showed that a previously described "optimal" algorithm can in fact be $O(n)$ worse than optimal. We presented two alternate algorithms and proved that both algorithms are at most $O(k)$ worse than optimal for expected number of clusters k. Finally, we experimentally demonstrated that our node-wise strategy performs very well in practice on three real-world data sets. We also showed that our algorithms are quite robust to adversarial probabilities whereas the previous approach performs very poorly.

10. ACKNOWLEDGEMENTS

We would like to thank Christopher Re, Steven Euijong Whang, and Hector Garcia-Molina for their invaluable feedback.

11. REFERENCES

[1] `http://www.facebook.com/places/editor`.

[2] `http://www.facebook.com/about/location`.

[3] `http://dbs.uni-leipzig.de/file/Abt-Buy.zip`.

[4] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. In *MACHINE LEARNING*, pages 238–247, 2002.

[5] M. Bilgic and L. Getoor. Active inference for collective classification. In *Twenty-Fourth Conference on Artificial Intelligence (AAAI NECTAR Track)*, pages 1652–1655, 2010.

[6] N. N. Dalvi, A. Dasgupta, R. Kumar, and V. Rastogi. Aggregating crowdsourced binary ratings. In D. Schwabe, V. A. F. Almeida, H. Glaser, R. A. Baeza-Yates, and S. B. Moon, editors, *WWW*, pages 285–294. International World Wide Web Conferences Steering Committee / ACM, 2013.

[7] G. Demartini, D. E. Difallah, and P. Cudré-Mauroux. Zencrowd: leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking. In *Proceedings of the 21st international conference on World Wide Web*, WWW '12, pages 469–478, New York, NY, USA, 2012. ACM.

[8] M. Georgescu, D. D. Pham, C. S. Firan, W. Nejdl, and J. Gaugaz. Map to humans and reduce error: crowdsourcing for deduplication applied to digital libraries. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, CIKM '12, pages 1970–1974, New York, NY, USA, 2012. ACM.

[9] A. Ghosh, S. Kale, and P. McAfee. Who moderates the moderators?: crowdsourcing abuse detection in user-generated content. In *Proceedings of the 12th ACM conference on Electronic commerce*, EC '11, pages 167–176, New York, NY, USA, 2011. ACM.

[10] A. Gruenheid, D. Kossmann, R. Sukriti, and F. Widmer. Crowdsourcing entity resolution: When is a=b? Technical Report 785, ETH Zurich, Sept. 2012.

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

[12] D. R. Karger, S. Oh, and D. Shah. Iterative learning for reliable crowdsourcing systems. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. C. N. Pereira, and K. Q. Weinberger, editors, *NIPS*, pages 1953–1961, 2011.

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

[14] A. McCallum. Cora dataset. `http://www.cs.umass.edu/~mcallum/data/cora-refs.tar.gz`, 2004.

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

[16] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng. Leveraging transitive relations for crowdsourced joins. In K. A. Ross, D. Srivastava, and D. Papadias, editors, *SIGMOD Conference*, pages 229–240. ACM, 2013.

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

[18] S. E. Whang, P. Lofgren, and H. Garcia-Molina. Question selection for crowd entity resolution. In *PVLDB*. Stanford InfoLab, August 2013.

[19] W. E. Winkler, W. E. Winkler, and N. P. Overview of record linkage and current research directions. Technical report, Bureau of the Census, 2006.