Online Entity Resolution Using an Oracle

Donatella Firmani*
Univ. of Rome "Tor Vergata"
firmani@ing.uniroma2.it

Barna Saha†
UMass Amherst
barna@cs.umass.edu

Divesh Srivastava
AT&T Labs – Research
divesh@research.att.com

ABSTRACT

Entity resolution (ER) is the task of identifying all records in a database that refer to the same underlying entity. This is an expensive task, and can take a significant amount of money and time; the end-user may want to take decisions during the process, rather than waiting for the task to be completed. We formalize an *online* version of the entity resolution task, and use an oracle which correctly labels matching and non-matching pairs through queries. In this setting, we design algorithms that seek to maximize progressive recall, and develop a novel analysis framework for prior proposals on entity resolution with an oracle, beyond their worst case guarantees. Finally, we provide both theoretical and experimental analysis of the proposed algorithms.

1. INTRODUCTION

Entity resolution (ER, record linkage, deduplication, etc.) seeks to identify which records in a data set refer to the same underlying real-world entity [6, 8]. It is a challenging problem for many reasons, intricate because of our ability to represent and misrepresent information about real-world entities in very diverse ways. For example, collecting profiles of people and businesses, or specifications of products and services, from websites and social media sites can result in billions of records that need to be resolved; these entities are identified in a wide variety of ways that humans can match and distinguish based on domain knowledge, but would be challenging for automated strategies.

Although there is an obvious need for ER, traditional strategies (which consider it to be an *offline* task that needs to be completed before results can be used) can be extremely expensive in resolving billions of records. To address this concern, recent strategies such as pay-as-you-go ER [18] and progressive deduplication [13] propose to identify more duplicate records early in the resolution process. In particular, Whang et al. [18] compare record pairs in non-increasing match likelihood ordering in a blocking-aware way. Similarly, Papenbrock et al. [13] adapt traditional sorted neighborhood-based and blocking-based ER techniques to ensure that record pairs with a higher match likelihood are compared first. Such *online* strategies are empirically shown to enable higher recall (i.e., more complete results) in the event of early termination or if there is limited resolution time available. However, these works do not provide analytical results or any formal guarantees about the proposed online strategies.

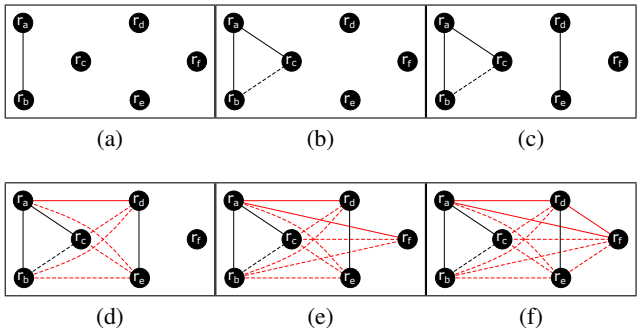


Figure 1: Optimal ordering for our John F. Kennedy example. Record pairs connected by solid black (resp. red) edges are labeled by the oracle as "matching" (resp. "non-matching"). Labels of pairs connected by dashed edges can be inferred via transitive relations.

Consider the following illustrative example, shown in Figure 1. There are a large number of places named after John F. Kennedy around the world, including airports, schools, bridges, plazas, memorials, and so on.[1] Given the six places (r_a) John F. Kennedy International Airport, (r_b) JFK Airport, (r_c) Kennedy Airport, NY (r_d) John F. Kennedy Memorial Airport, (r_e) Kennedy Memorial Airport, WI, (r_f) John F. Kennedy Memorial Plaza, humans can determine using domain knowledge that these correspond to three entities: r_a, r_b, and r_c refer to one entity, r_d and r_e refer to a second entity, and r_f refers to a third entity.

To have higher recall earlier in the online ER process, a good ordering of comparing record pairs is O_1: (r_a, r_b),(r_a, r_c), (r_b, r_c), (r_d, r_e), (r_a, r_d), (r_a, r_f), (r_d, r_f), (r_a, r_e), (r_b, r_d), (r_b, r_e), (r_b, r_f), (r_c, r_d), (r_c, r_e), (r_c, r_f), (r_e, r_f). Essentially, the matching pairs are compared before the non-matching pairs, and the recall is 1 after the fourth record pair is compared. If, instead, record pairs were to be compared in the reverse ordering O_2 of O_1, the recall would be 0 even after comparing the first 11 record pairs, making this ordering bad for online ER.

*Partially supported by MIUR, the Italian Ministry of Education, University and Research, under Project AMANDA (Algorithmics for MAssive and Networked DAta).
†Partially supported by NSF CCF 1464310 grant.

Proceedings of the VLDB Endowment, Vol. 9, No. 5

[1] http://en.wikipedia.org/wiki/Memorials_to_John_F._Kennedy

An elegant abstraction for formally comparing offline ER strategies was recently studied by Wang et al. [16] and Vesdapunt et al. [14]. They propose the notion of an *oracle* that correctly answers questions of the form "Do records u and v refer to the same entity?" in the context of crowdsourcing strategies, where such questions are answered by the crowd. ER strategies are then compared in terms of the total number of questions asked to an oracle to achieve a recall of 1, that is, completely resolving all the records into clusters, each referring to a distinct real-world entity. The strategies proposed by [16, 14] show how to make effective use of their oracle, and its consequence that ER satisfies the transitive relations (in which known match and non-match labels on some record pairs can be used to automatically infer match or non-match labels on other record pairs) to reduce the total number of questions that need to be asked of the oracle. In particular, Wang et al. [16] propose a strategy that asks oracle questions in non-increasing match probability ordering, while the strategy of Vesdapunt et al. [14] ask oracle questions based on ordering records in a non-increasing ordering of expected sizes of the records' clusters. However, minimizing the total number of oracle questions is not a suitable optimization objective for online ER, and the strategies of [16, 14] do not always perform well in the online setting.

Let us revisit our illustrative example. If we make use of the transitive relations for ordering O_1, the oracle would be asked only 6 record pairs (i.e., (r_a, r_b), (r_a, r_c), (r_d, r_e), (r_a, r_d), (r_a, r_f), (r_d, r_f)), and the labels of the other 9 record pairs can be inferred.

Note that the offline ER optimization objective of minimizing the total number of record pair queries to the oracle would also be achieved by considering the record pair queries to the oracle in the following ordering O_3: (r_a, r_d), (r_a, r_f), (r_d, r_f), (r_d, r_e), (r_a, r_b), (r_a, r_c), (r_b, r_c), (r_a, r_e), (r_b, r_d), (r_b, r_e), (r_b, r_f), (r_c, r_d), (r_c, r_e), (r_c, r_f), (r_e, r_f). Exactly the same 6 record pair queries would be issued to the oracle as in the ordering O_1. However, the recall would be 0 after labeling the first three record pairs, 0.25 after labeling the fourth record pair, 0.5 after labeling the fifth record pair, and 1.0 after labeling the sixth record pair, making O_3 less suitable than O_1 for online ER. Let us assume that we have the following probability estimates for record pairs[2]. Matching pairs: $p(r_d, r_e) = 0.80$, $p(r_b, r_c) = 0.60$, $p(r_a, r_c) = 0.54$, $p(r_a, r_b) = 0.46$. Non-matching pairs: $p(r_a, r_d) = 0.84$, $p(r_d, r_f) = 0.81$, $p(r_c, r_e) = 0.72$, $p(r_a, r_e) = 0.65$, $p(r_c, r_d) = 0.59$, $p(r_e, r_f) = 0.59$, $p(r_a, r_f) = 0.55$, $p(r_b, r_d) = 0.51$, $p(r_b, r_e) = 0.46$, $p(r_c, r_f) = 0.45$, $p(r_b, r_f) = 0.29$. In this case, the technique of Wang et al. [16] would consider pairs in the following (non-increasing ordering of probability) ordering O_4: (r_a, r_d), (r_d, r_f), (r_d, r_e), (r_c, r_e), (r_a, r_e), (r_b, r_c), (r_c, r_d), (r_e, r_f), (r_a, r_f), (r_a, r_c), (r_b, r_d), (r_a, r_b), (r_b, r_e), (r_c, r_f), (r_b, r_f). A total of 7 record pair queries would be issued to the oracle (i.e., (r_a, r_d), (r_d, r_f), (r_d, r_e), (r_c, r_e), (r_b, r_c), (r_a, r_f), (r_a, r_c)), which is more than in the ordering O_1. Since higher probability node pairs may be non-matching, this is only to be expected.

The strategies proposed by [16, 14] ask one oracle question at a time. However, some applications may benefit a lot from asking multiple questions in parallel. For instance, in crowd-sourcing, asking workers to execute one task at a time would not be viable in practice. To this end, Wang et al. [16] provide a parallel version of their strategy, such that (i) questions asked do not have transitive dependence, and (ii) if the probability estimates are not too noisy, performance is similar to sequential version. This parallel strategy has similar limitations as the sequential one in the online setting.

[2]We let some non-matching pairs have higher probability than matching pairs. Some observers, for instance, may consider r_a and r_e more likely to be the same entity, rather than r_a and r_b.

1.1 Contributions

In this paper, we formally study the problem of online ER for an input graph, where nodes correspond to records, and edges have match probabilities, using the oracle abstraction of [16, 14]. We make the following contributions.

First, we propose the use of *progressive recall* as the metric to be maximized by online ER strategies. If one plots a curve of recall (i.e., fraction of the total number of matching record pairs that have answered as such by the oracle, or inferred) as a function of the number of oracle queries, progressive recall is quantified as the area under this curve. Intuitively, progressive recall is maximized when (a) the oracle is asked about matching record pairs before being asked about non-matching record pairs, and (b) the oracle is asked about matching record pairs in larger clusters (i.e., entities with many records) in a connected manner, before being asked about matching record pairs in smaller clusters. The decision version of our problem is NP-complete, since it generalizes the traditional optimization objective of minimizing the number of oracle queries at the completion [14].

In our illustrative example, it is easy to verify that ordering O_1 indeed maximizes progressive recall (as shown in Figure 1), confirming our intuition that it is the best ordering for online ER.

Second, we propose a novel benefit metric, which is a robust estimate of the expected marginal gain in recall when processing a previously unprocessed record or record pair. We present greedy algorithms that use this benefit metric, and build on the edge-ordering strategy of Wang et al. [16] and the node-ordering strategy of Vesdapunt et al. [14]. We also develop an optimized hybrid strategy that provides a significant performance improvement in practice.

Third, we propose the *edge noise* model based on real-world data, and present approximation results for the quality of the solutions obtained by our strategies for optimizing progressive recall, and those by the strategies of [16, 14] for the traditional optimization objective and for progressive recall. The analysis of [14] showed an $O(n)$ worst-case approximation guarantee for the algorithm proposed by Wang et al. [16], and an $O(k)$ approximation guarantee for their own method, where n represents the total number of records, and k represents the number of actual entities. However, these worst-case behaviors are not observed in practice. Our analysis helps to explain this discrepancy by showing that, under the reasonable edge noise model, the algorithms of [16, 14] have much better approximation guarantees. These improved approximation guarantees are consistent with empirical results (see below), which show that each of these techniques does well in some cases, but does poorly in other cases.

Fourth, we do a thorough empirical comparison of our strategies with the strategies of Wang et al. [16], Vesdapunt et al. [14], and Papenbrock et al. [13] (which was shown to dominate the techniques of Whang et al. [18]) on real and synthetic datasets. We also implement parallel versions of our strategies, based on the same principles described in [16]. Our evaluation demonstrates the superiority of our hybrid strategy over the alternatives. In particular, our hybrid strategy shows a high progressive recall on data sets with a skewed distribution of entity cluster sizes (e.g., Cora bibliography data, where the strategy of Wang et al. [16] performs poorly), and on sparse data sets (e.g., ABT-BUY Products, where the technique of Vesdapunt et al. [14] performs poorly). Our strategy also shows much higher progressive recall over large real data with blocking (DBLP data) compared to the strategy of Papenbrock et al. [13].

Our results provide the foundations for an online view of the ER task, especially in a crowdsourcing setting, which we think is both more realistic and more flexible than its traditional offline view.

1.2 Related Work

ER has a long history (see, e.g., [6, 8] for surveys), from the seminal paper by Fellegi and Sunter in 1969 [7], which proposed the use of a learning-based approach to build a classifier, to rule-based and distance-based approaches (see, e.g., [6]), to the recently proposed crowdsourcing and hybrid human-machine approaches (see, e.g., [15, 10]) for this challenging task. The closest related works to ours are the ones by Whang et al. [18] and Papenbrock et al. [13] (for online ER strategies) and by Wang et al. [16] and Vesdapunt et al. [14] (for the oracle model, and use of transitivity-aware strategies). We have discussed these earlier in this section. Here, we present other closely related works, and refer the reader to prior surveys for a detailed discussion of other strategies.

Progressive recall. Progressive recall has been used by Vesdapunt et al. [14] for experimental evaluation, and by Altowim et al. [2]. In these works, the different approaches are compared empirically by showing recall as a function of oracle questions or overall processing time, but no formal definition of progressive recall is provided. In [2], the authors also introduce a benefit function, which does not take into account transitivity. It is worth noting that the algorithms described in [2] are based on a *resolve* function (which is implemented as a binary Naive Bayes classifier, so there is no guarantee of correct answers) rather than our oracle abstraction.

Hybrid crowdsourcing. Many frameworks have been developed to leverage humans for performing ER tasks [15, 10]. Wang et al. [15] describe a hybrid human-machine framework CrowdER, that automatically detects pairs or clusters that have a high likelihood of matching, which are then verified by humans. Gokhale et al. [10] proposed a hybrid approach for the end-to-end workflow of ER (including blocking and matching), making effective use of random forests classifiers and active learning via human labeling.

Dynamic crowdsourcing. Dynamic aspects of the crowdsourcing process, different than progressive recall, are tackled in [5, 17]. Demartini et al. [5] dynamically generate crowdsourcing questions to assess the results of human workers. Whang et al. [17] propose a budget-based method and explore how to make a good use of limited resources to maximize accuracy.

Oracle errors. To deal with the possibility that the crowdsourced oracle may give wrong answers, there are simple majority voting mechanisms or more sophisticated techniques [4, 9, 11] to handle such errors. We do not deal with oracle errors in this paper.

1.3 Outline

The rest of this paper is organized as follows. We formulate our problem in Section 2, and the benefit metric in Section 3. Our oracle-based strategies for online ER are described in Section 4. The edge noise model and our approximation results are presented in Section 5, and our empirical evaluation is reported in Section 6.

2. PROBLEM FORMULATION

Let $V = \{v_1, \ldots, v_n\}$ be a set of n records. Given $u, v \in V$, we say that u *matches* v when they refer to the same real-world entity. We say that a complete, edge-labeled graph $C = (V, E = E^+ \cup E^-)$, is a *clustering* of V if $C^+ = (V, E^+)$ is transitively closed, where $(u, v) \in E^+$ represents that u matches with v, and $(u, v) \in E^-$ represents that u is non-matching with v. In other words, C^+ partitions V into cliques representing distinct real-world entities. We call each clique a *cluster* of V, and denote with $c(u)$ the cluster of u. Let k denote the number of clusters, and $c_1, \ldots, c_k$ denote the clusters in non-increasing order of size:

- the number of edges in C^+ is $\left|E^+\right| = \sum_{i=1}^{k} \binom{|c_i|}{2}$;
- the size of a spanning forest of C is $n - k$.

Consider an unknown clustering C, along with an oracle access to C. Edges in C can either be asked to the oracle, or inferred – positively or negatively – leveraging transitive relations. As an example, if u matches with v, and v matches with w, then we can deduce that u matches with w without needing to ask the oracle. Similarly, if u matches with v, and v is non-matching with w, then we can deduce that u is non-matching with w as well. In the following lemma, we report a prior result of Wang et al. [16].

LEMMA 1. *Let $T = T^+ \cup T^-$ be a set of edges along with the oracle responses, where T^+ are the edges with YES (i.e., matching) response, and T^- with NO (i.e., non-matching) response.*

a. An edge (u, v) can be positively inferred from T^+ iff there exists a path from u to v which only consists of T^+ edges.

b. An edge (u, v) can be negatively inferred from T iff there exists a path from u to v which consists of T^+ edges and one T^- edge.

c. Any other edge cannot be inferred.

Let $E_T^+ \subseteq E^+$ and $E_T^- \subseteq E^-$ be the sets of all the edges that can be inferred from T, positively (including edges in T^+) and negatively (including edges in T^-). Let $C_T = (V, E_T)$ be the subgraph of C induced by E_T. We say that C_T is a T-*clustering* of V. Analogously, we call each clique a T-*cluster* of V, and denote with $c_T(u)$ the T-cluster of u. Given an unknown clustering C, an oracle *strategy* s incrementally grows a T-clustering, by asking edges to an oracle. As soon as $E_T^+ = E^+$, all the information about matching pairs is available to the user. This requires at least $\sum_{i=1}^{k} (|c_i| - 1) = n - k$ questions (i.e., the size of a spanning forest of C^+), by Lemma 1. However, it can take much longer, when $\left|E_T^+ \cup E_T^-\right| = \binom{n}{2}$, for the user to become aware that no further questions are needed. This requires asking at least for one question (yielding a negative answer) across every pair of clusters c_i, c_j, $i \neq j$. Let t_r be the number of questions asked by s until $E_T^+ = E^+$, and t_R be the total number of questions asked by s. Then it holds: (i) $t_r \geq n - k$; (ii) $t_R \geq n - k + \binom{k}{2}$.

We refer to the above lower-bound values for t_r and t_R as t_r^* and t_R^*, respectively. The values t_r and t_R account for all the questions, irrespective of the answer (which can be either positive or negative). Let us now define t_r^+ as the total number of *positive* questions (that is, questions returning a positive answer) asked by s for $E_T^+ = E^+$, and let t_R^+ be the total number of positive questions asked by s. It follows that $t_r^+ = t_R^+$, and that both are equal to the size of a spanning forest of C^+, i.e., $n - k$.

Recall. As more of the cluster structure on C is revealed, the recall of s increases. We define two recall functions of s,

- $\texttt{recall}(t) = |E_T^+|/|E^+|$, where T is the set of the first t oracle responses, i.e., $t = |T|$.
- $\texttt{recall}^+(t) = \left|E_{T^+}^+\right|/|E^+|$, where T^+ is the set of the first t positive oracle responses, i.e., $t = \left|T^+\right|$.

For any value of t, it holds $\texttt{recall}(t) \leq \texttt{recall}^+(t) \leq M$, where M is given by the following Lemma 2.

LEMMA 2. *Let k' be the biggest index in $[1, k]$ such that $\sum_{i=1}^{k'}(|c_i| - 1) \leq t$, and let t' be the size of a spanning forest of $c_1 \cup c_2 \cup \cdots \cup c_{k'}$, that is $t' = \sum_{i=1}^{k'}(|c_i| - 1)$. Then, $M = \frac{\sum_{i=1}^{k'} \binom{|c_i|}{2} + \binom{t - t'}{2}}{\sum_{i=1}^{k} \binom{|c_i|}{2}}$.*

PROOF. Due to transitive relations, $\texttt{recall}^+(t')$ is maximized by a strategy s' which finds the top k' clusters in non-increasing order of size, yielding $\texttt{recall}^+(t') = \sum_{i=1}^{k'} \binom{|c_i|}{2} / |E^+|$. Similarly,

$\texttt{recall}^+(t)$ is maximized by a strategy s that does like s' after the first t' questions, and then asks a spanning forest of any sized $t-t'$ subgraph of $c_{k'+1}$. For s, $\texttt{recall}^+(t) = \frac{\sum_{i=1}^{k'} \binom{|c_i|}{2} + \binom{t-t'}{2}}{|E^+|}$. Since $|E^+| = \sum_{i=1}^{k} \binom{|c_i|}{2}$, the bound follows. □

2.1 Progressive Recall

The recall of s denotes the fraction of positive edges found, among those of the unknown clustering C. It is the simplest measure of the amount of the information that is available to the user at a given point. However, it cannot distinguish the dynamic behaviour of different strategies. To this end, we define two *progressive recall* functions, denoting the *area* under the recall-questions curves $\texttt{recall}(t)$ and $\texttt{recall}^+(t)$.

- $\texttt{precall}(t) = \sum_{t'=1}^{t} \texttt{recall}(t')$.
- $\texttt{precall}^+(t) = \sum_{t'=1}^{t} \texttt{recall}^+(t')$.

Let us consider a strategy s^* for which $\texttt{precall}$ is maximized. s^* first grows the largest cluster c_1 by asking adjacent edges belonging to a spanning tree of c_1. That is, every asked edge shares one of its endpoints with previously asked edges. After c_1 is grown, s^* grows in sequence $c_2, \ldots, c_k$ in a similar fashion. Finally, s^* asks edges in E^- in any order, until all the labels are known. After the smallest cluster c_k is fully grown, $|T| = t_r^*$, $E_T^+ = E^+$, $\texttt{recall}(t) = 1$ and $E_T^- = \emptyset$. The final phase requires $\binom{k}{2}$ additional questions. Therefore s^* minimizes both t_r and t_R.

Benefit metrics. Both progressive recall functions of a strategy s can be expressed as *fractions* of the corresponding functions of s^*. We refer to progressive recall of s^* as $\texttt{precall}^*$ (for s^*, $\texttt{precall}$=$\texttt{precall}^+$). We refer to such ratio functions as *normalized* progressive recall functions:

- $\texttt{nPrecall}(t) = \frac{\texttt{precall}(t)}{\texttt{precall}^*(t)}$, in particular we define $\texttt{benefit}$=$\texttt{nPrecall}(t_r^*)$.
- $\texttt{nPrecall}^+(t) = \frac{\texttt{precall}^+(t)}{\texttt{precall}^*(t)}$, in particular we define $\texttt{benefit}^+$=$\texttt{nPrecall}^+(t_r^*)$

We choose to express benefit metrics with respect to t_r^* rather than t_R^*. Normalized progressive recall functions get closer to 1 as t gets bigger, and t_R can vary up to $\binom{n}{2}$ on very sparse instances. Instead, t_r^* is bounded by n. We note that $\texttt{benefit} = 1$ if and only if $s = s^*$, while a strategy different from s^* can have $\texttt{benefit}^+ = 1$ as long as the *order* of positive responses is the same as s^*.

2.2 Oracle problem

We are now ready to define formally our problem.

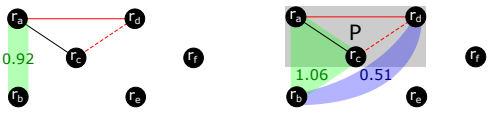
PROBLEM 1. *Given a set of records V, an oracle access to C, a subset $V' \subseteq V$, and a function $p(u, v)$ returning the probability that u and v are matching $\forall u, v \in V'$, find the strategy that maximizes* $\texttt{benefit}^+$.

We report for comparison the problem studied in [14, 16]. Problem 2 is NP-hard [14], as well as Problem 1.

PROBLEM 2. *Given a set of records V, an oracle access to C, and a function $p(u, v)$ returning the probability that u and v are matching $\forall u, v \in V$, find the strategy that minimizes t_R.*

LEMMA 3. *Problem 1 is NP-hard.*

PROOF. Problem 2 is NP-hard [14], and Problem 1 is at least as hard as Problem 2. Indeed, a strategy solving Problem 1 also solves Problem 2, but not vice versa. □



(a) $b_e(r_a, r_b)$ (b) $b_v(r_b, P)$

Figure 2: Examples of edge and node benefit values for our John F. Kennedy example. Colors and dashes have the same meaning as in Figure 1. The benefit of edge (r_a, r_b) in Figure 2a shows that we expect to find out $2 * 1 * 0.46 = 0.92$ positive edges, by asking (r_a, r_b) to the oracle. If (r_a, r_b) turns out to be positive (the probability estimate of such event is $p(r_a, r_b) = 0.46$), indeed, our current result earns two new positive edges, namely (r_a, r_b) itself, and (r_b, r_c), which can be inferred using Lemma 1. In Figure 2b we show the benefit of node r_b, by setting P to the only nodes for which we have information stored in T, $\{r_a, r_c, r_d\}$. To this end, we first compute the aggregated benefits of r_b with respect to the two T-clusters in P, namely $\{r_a, r_c\}$ and $\{r_d\}$. Specifically, the former shows that we expect to find out $0.46+0.60 = 1.06$ positive edges, by discovering that r_b belongs to $\{r_a, r_c\}$ (i.e., by asking (r_a, r_b) or (r_b, r_c) to the oracle), and the latter that we expect to find out 0.51 positive edges, by discovering that r_b belongs to $\{r_d\}$ instead. The benefit of the node r_b is finally set to the maximum aggregated benefit, i.e., 1.06.

In this paper, we consider previous and new strategies, and: (i) formally analyze their $\texttt{benefit}^+$, which is affected only by the order in which positive questions are asked; (ii) experimentally evaluate $\texttt{benefit}^+$ and $\texttt{benefit}$.

3. BENEFIT OF QUESTIONS

Let us discuss what information we have a priori, when $T = \emptyset$. As input to the considered problems, we have a (partial) function $p : V \times V \to [0, 1]$ returning the probability that u and v are matching. We can compute a function $p_s : V \to \mathbb{R}$ returning the expected cluster size of a node v (excluding v itself).

$$p_s(v) = \sum_{u \in V \setminus v} p(u, v) \tag{1}$$

p can be used as the *probability estimate* that a non-inferable edge belongs to E^+, and p_s as the *estimated cluster size* of a node v for which most incident edges are non-inferable.

Given a non-inferable edge (u, v), we use its probability estimate $p(u, v)$, for computing the expected number of edges b_e that could be positively inferred if u and v are matching (including (u, v) itself). This denotes the expected marginal gain in recall when processing the single edge (u, v).

$$b_e(u, v) = |c_T(u)| * |c_T(v)| * p(u, v) \tag{2}$$

We refer to the function b_e as the *benefit* of an edge.

Given a node v for which all its incident edges are non-inferable, that is, $c_T(v)$ consists of the singleton $\{v\}$, we use the probability estimate for computing the expected number of edges b_v that could be positively inferred (i.e., the marginal gain in recall) if any out of a given bunch of edges incident to v, belongs to E^+. Consider the bunch of edges connecting v to a set of T-clusters P.

$$b_v(v, P) = \max_{c \in C_T : c \neq \{v\}, c \subseteq P} b_{vc}(v, c) \tag{3}$$

$$b_{vc}(v, c) = p_v(v, c) * |c| \tag{4}$$

where p_v is the probability estimate that v belongs to T-cluster c. We refer to the function b_v as the *benefit* of a node, and to the

function b_{vc} as the *aggregated benefit* of a node with respect to a T-cluster. In the following, we sometimes refer to the aggregated benefit of v with respect to $c_T(u)$, as the aggregated benefit of the edge (v, u), and use both notations $b_{vc}(v, u)$ and $b_{vc}(v, c)$, where $c = c_T(u)$, equivalently.

For computing p_v we take the mean probability of edges connecting v to a given cluster c (which is distinct from $c_T(v)$) as an estimate, because that is a robust estimate (and also follows as an upper bound from the Markov inequality).

$$p_v(v, u_c) = \frac{\sum_{u \in c} p(u, v)}{|c|} \tag{5}$$

We note that while the benefit of an edge (v, u) only considers the estimated probability of (v, u), the aggregated benefit takes into account all the probabilities of other edges (v, w), such that $w \in c_T(u)$. Therefore all the edges (v, w), such that $w \in c_T(u)$ will have the same aggregated benefit.

Recall the illustrative example from Section 1, and the probability estimates of the matching and non-matching edges. Figure 2 shows the benefit of sample edges and nodes when $T^+ = (r_a, r_c)$ and $T^- = (r_a, r_d)$.

4. ORACLE STRATEGIES

A strategy s for Problem 1 needs to store new responses from the oracle, in such a way that edges that cannot be inferred as in Lemma 1 can be computed efficiently, and asked to the oracle afterwards. To this end we introduce the `query`(u, v) method, that returns the response for (u, v) from the oracle, and updates the data structures used by a strategy accordingly. We also introduce few auxiliaries methods, that will be used by the considered strategies:

- `top-e`(w) returns the top-w non-inferable edges, in non-increasing order of p.
- `top-v`(w) returns the top-w nodes with no incident inferable edges, in non-increasing order of p_s.
- `edges`$(\{v\}, P)$ returns an iterator over the non-inferable edges connecting a singleton cluster $\{v\}$ to T-clusters in $P \subseteq C_T$, in non-increasing order of p.
- `edges'`$(\{v\}, P)$ returns an iterator over the non-inferable edges connecting a singleton cluster $\{v\}$ to T-clusters in $P \subseteq C_T$, in non-increasing order of b_{vc}.

In the following, we sometimes refer to the w parameter as the *window size* – or simply *window* – of a strategy.

Complexity of update and auxiliary methods. We refer to the total number of operations that a strategy performs for resolving V as its *work*. Assuming that edges that are inferable are never asked to the oracle, the minimum work for resolving V is $t_R^* = (n - k) + \binom{k}{2}$. For efficiently computing edges that cannot be inferred as in Lemma 1, we use a graph data structure `GS`, where nodes are T-clusters and edges are non-inferable edges. Upon a positive response from the oracle we *contract* the corresponding edge, and upon a negative response we *delete* the corresponding edge. During edge contraction, only one of the maximum probability edges between two clusters survives. This requires $O(n(n - k) + t_R)$ work, as in the following lemma.

LEMMA 4. `query` *method requires* $O(n(n - k) + t_R)$ *work.*

PROOF. The fundamental operation of our algorithm is a form of edge contraction. The result of contracting the edge (u, v) is new *supernode* uv. For each node $w \notin \{u, v\}$, in `GS`, edges (w, u) and (w, v) are replaced by an edge (w, uv) having probability set to $\max\{p(w, u), p(w, v)\}$. Finally, the contracted nodes u and v with all their incident edges are removed. If the edge (w, uv) needs to be asked, we ask to the oracle if w and the lowest-id node in the supernode, e.g., u, are matching[3]. When `GS` is represented using adjacency lists or an adjacency matrix, a single edge contraction operation can be implemented with a linear number of updates. Since the total number of edge contractions and deletions is bounded by $n - k$ and t_R respectively, the claim follows. □

Algorithm 1 s_{wang}, described by Wang et al.

```
1: while V not resolved do
2:     (u, v) ← top-e(1)
3:     T ← T ∪ query(u, v)
4: end while
```

Algorithm 2 s_{vesd}, described by Vesdapunt et al.

```
1: P ← top-v(1)
2: while P ≠ V do
3:     v ← top-v(1)
4:     l ← edges({v}, P)
5:     while l.next() do
6:         (u, v) ← l.getNext()
7:         T ← T ∪ query(u, v)
8:         if (u, v) ∈ E then break end if
9:     end while
10:    P ← P ∪ {v}
11: end while
```

Using our contract/delete algorithm, `top-e`, `edges` and `edges'` can be done in $O(1)$ time. For implementing `top-e`, we assume that edges are sorted during a preprocessing phase in non-increasing order of p and are available in a list `LE`. `query` method can maintain `LE` updated at no additional work. `edges` is simply an iterator over the adjacency list of u's supernode. For implementing `edges'` we need in addition an independent execution of the contraction/delete algorithm, where probability scores of replacement edges upon contraction are set to mean probability of replaced edges (b_{vc} is computed as in Eq. 4). Analogously to `top-e`, for implementing `top-v`, we assume that nodes are sorted during a preprocessing phase in non-increasing order of p_s and are available in a list `LV`. Upon a new response from the oracle, let (u, v) be the relative edge, `LV` can be maintained updated by removing nodes u and v at no additional work than `query`.

4.1 Prior Strategies

In this section, we describe strategies by Wang et al. [16] and by Vesdapunt et al. [14] in our framework. Such strategies have been designed as a solution to Problem 2 and we use them as a frame of comparison for our strategies.

Wang et al. The strategy shown in Algorithm 1 selects the first non-inferable edge in non-increasing order of probability estimate p, and asks it to the oracle. It continues as long as there are non-inferable edges. s_{wang} does constant additional work than `query`. In our John F. Kennedy example (see Section 1 for pairwise probability estimates), the first question made by s_{wang} is (r_a, r_d), which is the edge with the highest value of p, yielding a negative answer, and then (r_d, r_f), (r_d, r_e), and so on. We notice that in real cases, negative edges may be asked before positive edges as well.

Vesdapunt et al. The strategy shown in Algorithm 2 maintains a set P of "processed" nodes, having the invariant that all the edges

[3] Any of the endpoints of the contracted edge can be used. We use the lowest-id node for sake of simplicity.

Figure 3: Example of s_{vesd}'s invariant-maintaining procedure, on the clustering of Figure 1. Colors and dashes have the same meaning as in Figure 1. In the leftmost figure we show the set P after processing 2 nodes. In the rightmost figure we show what happens when node r_c is added to P: let (r_a, r_c) be the edge selected at line 6 of Algorithm 2. When a positive response is received, the edges connecting r_c to P can be inferred and r_c is added to P.

Algorithm 3 $s_{\text{edge}}(w)$.

```
1: while V not resolved do
2:     W ← top-e(w)
3:     (u, v) ← argmax_W b_e(u, v)
4:     T ← T ∪ query(u, v)
5: end while
```

connecting nodes in P have been either asked or inferred. The strategy selects the first node with no inferable incident edges in non-increasing order of estimated cluster size p_s, and adds it to P until $P = V$. Every time a node v is selected, before adding it to P, edges connecting v and P are asked to the oracle in non-increasing order of p, until $P \cup \{v\}$ satisfies the invariant. This can be achieved in two ways: (i) a positive response is received; (ii) all the edges connecting v and P have been asked (with negative responses). If (i) is the case, all the edges connecting v and P that have not been asked can be inferred either positively or negatively (because P satisfies the invariant) as in Figure 3. s_{vesd} does constant additional work than `query`. In our John F. Kennedy example, P is set initially to $\{r_d\}$, which is the node with the highest value of p_s (although it belongs to the second largest cluster)[4]. Next selected node is r_e, and the first question made by s_{vesd} is (r_e, r_d), yielding a positive answer. P is updated to $\{r_d, r_e\}$. Next selected node is r_a, and the second question made by s_{vesd} is (r_a, r_d), which is the edge with highest probability among those connecting r_a to P, yielding a negative answer. (r_a, r_e) is not asked, as it can be negatively inferred. P is updated to $\{r_d, r_e, r_a\}$ and so on. We notice that in real cases, expected cluster sizes may lead to a different ordering than actual cluster sizes, as in our illustrative example.

Other examples. s_{wang} and s_{vesd} executions over real dataset and actual probability estimates are shown in Section 6.3.

4.2 Strategies for Progressive Recall

The goal of the strategies described in Section 4.1 is minimizing the total number of questions t_R asked to the oracle. Hence benefit is not taken into account explicitly. Our strategies instead, as more cluster structure is revealed by the oracle, select what edge to ask the oracle as to maximize the expected marginal gain in recall.

Edge ordering. The strategy shown in Algorithm 3 selects the highest benefit edge (as in Equation 2) among the top-w non-inferable edges in non-increasing order of p. Then it asks the edge to the oracle and repeat this process until $E_T^+ = E^+$ and $E_T^- = E^-$. Initially, when $T = \emptyset$, all the edges have benefit equal to their probability estimate, then the first asked edge is the same as

[4] Values of p_s are $p_s(r_d) = 3.55$, $p_s(r_e) = 3.22$, $p_s(r_a) = 3.04$, $p_s(r_c) = 2.90$, $p_s(r_f) = 2.69$, $p_s(r_b) = 2.32$

Algorithm 4 $s_{\text{hybrid}}(w, \tau, \theta)$.

```
1: P ← top-v(1)
2: while P ≠ V do
3:     W ← top-v(w)
4:     v ← argmax_W b_v(v, P)
5:     l ← edges'({v}, P)
6:     for i = 1, b = 1; l.next() ∧ i ≤ τ ∧ b > θ; i++ do
7:         u ← l.getNext()
8:         b ← b_vc(v, u)
9:         T ← T ∪ query(u, v)
10:        if (u, v) ∈ E then break end if
11:    end for
12:    P ← P ∪ {v}
13: end while
14: s_edge(w)
```

in s_{wang}. As more edges of C are revealed by the oracle, however, high probability edges may have low benefit and vice versa.

For instance, let (u, v) be the highest-probability non-inferable edge at a certain point of the execution, and let u and v be singleton T-clusters. The marginal gain in recall that we would get by asking (u, v) to the oracle is $\leq \frac{1}{|E^+|}$.

If `top-e`(w) contains a higher benefit edge (w, z), s_{edge} will ask (w, z) to the oracle rather then (u, v), even though $p(w, z) < p(u, v)$. The higher the value of w the higher the chance that lower-probability higher-benefit edges are preferred to (u, v). If $w = 1$ then $s_{\text{edge}} = s_{\text{wang}}$.

s_{edge} does $O(w\,(n-k))$ additional work than `query`. Computing the benefit of edges in the window can indeed be done during edge contraction and deletion in $O(w)$ and $O(1)$ additional time respectively. If $w = qn$ for a given constant q, s_{edge} does asymptotically the same work as previous strategies.

In our John F. Kennedy example, the first question made by $s_{\text{edge}}(2)$, is the same as s_{wang}, that is (r_a, r_d). As long as all the T-clusters have size 1, s_{edge} and s_{wang} make the same choices, since the benefit of edges is equal to their probability estimate. After (r_d, r_e) is asked, yielding a positive response, the highest benefit edge in the window $\{(r_c, r_e), (r_b, r_c)\}$ ((r_a, r_e) can be negatively inferred) is still equal to the highest probability edge, that is (r_c, r_e). After (r_b, r_c) is asked, the two strategies start making different choices. s_{wang} selects (r_a, r_f), and $s_{\text{edge}}(2)$ selects (r_a, r_c) ($b_e(r_a, r_f) = 0.55$ and $b_e(r_a, r_c) = 2 * 1 * 0.54 = 1.08$) completing cluster c_1.

Hybrid ordering. The strategy shown in Algorithm 4 maintains a set P of "processed" nodes as in Algorithm 2, but no invariant is guaranteed. That is, some edges incident to nodes in P may be non-inferable at some point. The strategy selects the highest benefit node (as in Equation 3), among the top w nodes in $V \setminus P$ in non-increasing order of p_s, and adds it to P until $P = V$. Every time a node v is selected, before adding it to P, edges connecting v and P are asked to the oracle in non-increasing order of b_{vc}, until one of the following condition is satisfied: (i) a positive response is received; (ii) all the edges connecting v and P have been asked (with negative responses); (iii) the benefit does not exceed a given threshold θ; (iv) the number of questions related to node v exceeds a given amount of trials τ. The first two conditions are the same as s_{vesd}. When $P = V$, non-inferable edges can still remain, which are eventually processed using $s_{\text{edge}}(w)$ strategy.

If $w = 1$ then two extremes behaviors are possible:

- if $\tau = n$ and $\theta = 0$ then no questions are deferred to s_{edge}, and $s_{\text{hybrid}} = s_{\text{vesd}}$, except for the order in which edges in the inner loop are processed.

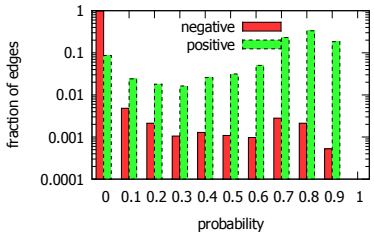


Figure 4: Probability distribution of positive (matching entities) and negative (non-matching entities) edges in cora dataset

- if $\tau = 0$ or $\theta = n$ then all the questions are deferred to $s_{\textbf{edge}}$, and $s_{\textbf{hybrid}} = s_{\textbf{wang}}$.

We set default values for θ and τ, to 0.3 and $\log n$ respectively.

Computing the benefit of w nodes takes $O(nw)$ time, and line 4 is executed n times, requiring $O(n^2 w)$ additional work than query. If w is equal to a given constant q, $s_{\textbf{hybrid}}$ does asymptotically the same work as previous strategies and $s_{\textbf{edge}}$.

In our John F. Kennedy example, P is set initially to $\{r_d\}$ as in $s_{\textbf{vesd}}$. Let us consider $s_{\textbf{hybrid}}(n,n,-1)$, i.e., window size n, and no constraints for trials and benefit. Differently from $s_{\textbf{vesd}}$, the next selected node is r_a, whose benefit $b_v(r_a, P)$ is 0.84, and the first question is (r_a, r_d). P is updated to $\{r_d, r_a\}$. Next selected node is r_f, whose benefit $b_v(r_f, P)$ is 0.81, and the second question made is (r_f, r_d). P is updated to $\{r_d, r_a, r_f\}$ and so on.[5].

Other examples. $s_{\textbf{edge}}$ and $s_{\textbf{hybrid}}$ executions over real dataset and actual probability estimates are shown in Section 6.3.

5. ANALYSIS OF STRATEGIES

In this section, we formally analyze the strategies discussed in the previous section under a reasonable noise model. Our noise model closely resembles the observed noise in estimating probability values in cora [12] dataset. cora is a widely used bibliography data set, and has been used in many prior works on ER (see, e.g., [14]). Define an indicator random variable $X_{u,v}$ for every pair of nodes u, v such that $X_{u,v} = 1$ if u and v represent the same entity, and $X_{u,v} = 0$ if u and v are different entities. Let $p(u, v)$ denote the matching probability score between two nodes u and v. If the matching function p is perfect, then one must have $p(u, v) = 1$ if u and v are the same entity, that is $X_{u,v} = 1$, and $p(u, v) = 0$ if u and v are different entities, that is $X_{u,v} = 0$. That is $\mathsf{Pr}[X_{u,v}] = p(u, v)$. However, in the real world, matching probability functions are not perfect, and are corrupted by noise. To model noise in computing probability values, we define a noise parameter $\eta_{u,v} \in [0, 1]$ for every pair of nodes u, v drawn from a probability distribution $\mathcal{D}_{u,v}$. Then, $p(u, v) = \eta_{u,v}$ if u and v represent different entities, that is $X_{u,v} = 0$. $p(u, v) = 1 - \eta_{u,v}$ if u and v represent the same entity, that is $X_{u,v} = 1$. Once again we consider $\mathsf{Pr}[X_{u,v}] = p(u, v)$. Our analysis in this paper for various algorithms is based on this *edge noise model*. Handling other noise models is part of our ongoing work.

Figure 4 represents the distribution of estimated probabilities of matching and non-matching edges in cora data set. We see for matching edges (green), more than 70% of edges are in the range of $[0.7, 1]$, and for non-matching edges almost 95% of edges have scores between $[0, 0.1]$. For the rest of the matching edges below 0.7, they follow a near uniform distribution. Similarly, for non-matching edges with score above 0.1, they follow a near uniform distribution. In the analysis, we therefore assume the following distributions. For matching edges, with probability $\left(1 - \frac{\alpha}{n}\right)$, $\alpha > 0$, the score is above 0.7, and the remaining probability mass is distributed uniformly in $[0, 0.7)$. For non-matching edges, with probability $\left(1 - \frac{\beta}{n}\right)$, $\beta > 0$, the score is below 0.1 and the remaining probability mass is distributed uniformly in $(0.1, 1]$. $Unif[x, y]$ denotes the uniform distribution in the range $(x, y]$.

5.1 Minimizing Total Number of Questions

In the following, we provide approximation results for previous strategies $s_{\textbf{wang}}$ and $s_{\textbf{vesd}}$, when solving Problem 2. As discussed in Section 2, the minimum number of questions that need to be asked to the oracle, in order to identify the k clusters, is $t_R^* = n - k + \binom{k}{2}$.

Analysis of Wang et al.'s algorithm.

THEOREM 1. *$s_{\textbf{wang}}$ gives an $O(\log^2 n)$-approximation algorithm under the edge noise model with $\alpha \leq \frac{n}{2}$, $\beta = O(\log n)$.*

PROOF. Let us consider the querying process of $s_{\textbf{wang}}$, and let us fix a cluster c_i whose size is at least $4 \log n$. At the end of the process, $s_{\textbf{wang}}$ will ask $|c_i| - 1$ edges from c_i that form a spanning tree. Let us use ST_i to denote that spanning tree. Let R_i denote the "negative" edges, i.e., belonging to E^-, incident on nodes in c_i that are queried before all the edges in ST_i are queried. Let $R_i = \{(u_1, x_1), (u_2, x_2), ..., (u_a, x_a)\}$ for some natural number $a \geq 0$, $u_j \in c_i, j = 1, 2, .., a$ and $x_j \notin c_i, j = 1, 2, ..., a$.

Consider the time when the last edge in ST_i is selected for querying. Just before that, there must be exactly 2 components of c_i when restricted to already known edges in c_i. Denote these two components by c_i^1 and c_i^2. The number of edges across c_i^1 and c_i^2 in the ground truth C is $|c_i^1||c_i^2| \geq |c_i| - 1$. For any edge $(s, t), s \in c_i^1, t \in c_i^2$, it must hold that $p(s, t) \leq \min_j p(u_j, x_j)$. It must also hold that there are exactly a non-matching edges with score higher than $\max_{(s,t) \in c_i^1 \times c_i^2} p(s, t)$.

There are overall $(n - |c_i|)|c_i|$ non-matching edges incident on some node in c_i. Among them, the expected number of edges that have values higher than 0.1 is $\frac{\beta}{n}(n - |c_i|)|c_i| \leq \beta |c_i| \leq \log n |c_i|$. Therefore, if the maximum value of matching edges across c_i^1 and c_i^2 is above 0.1, then the expected size of R_i will be less than $\log n |c_i|$.

On the other hand, the probability that the maximum of all the matching edges across c_i^1 and c_i^2 have value less than 0.1 is at most $\left(\frac{\alpha}{n}\right)^{\log n} \leq \frac{1}{n}$.

Therefore, we have

$$\mathsf{E}[|R_i|] \leq \log n |c_i| + \frac{1}{n} n |c_i| = (\log n + 1)|c_i|$$

Suppose the number of clusters of size at most $4 \log n - 1$ is r. The total number of negative edges with both end points in such small clusters is at most $16 \log^2 n$. Then, if t_R denotes the total number of questions asked to the oracle by $s_{\textbf{wang}}$ strategy, we have

$$\begin{aligned} \mathsf{E}[t_R] &\leq \sum_{i:|c_i| \geq 4 \log n} \mathsf{E}[|R_i|] + 16 \binom{r}{2} \log^2 n + \\ &\quad + \sum_{i=1}^{k} (|c_i| - 1) + \binom{k}{2} \\ &\leq O(\log^2 n) t_R^* \end{aligned}$$

□

[5] In this setting, as long as all the T-clusters have size 1, questions will be asked in non-increasing order of p as in $s_{\textbf{wang}}$. The benefit of nodes is indeed equal to the maximum probability edges incident to P. For instance when $P = \{r_d, r_a, r_f\}$, the maximum benefit node is r_e ($b_v(r_e, P) = 0.80$) and the maximum probability edge incident to P is (r_d, r_e), which has probability estimate 0.80.

This is in contrast to the result by Vesdapunt et al. [14], who showed in the worst case, Wang et al. may ask $O(nt_R^*)$ questions. They consider a very high noise level, where between two clusters of size $\frac{n}{2}$, $\frac{1}{2} * \frac{n^2}{4} - 1$ record pairs can be misclassified as matching. Under our noise model, this happens with vanishingly low probability. For example, consider pairs of clusters in the `cora` data set with the largest number of inter-cluster edges (u, v) with $p(u, v) > 0.5$: for a pair of clusters with sizes 55 and 149 respectively, $s_{\textbf{wang}}$ yields 81 negative questions more than s^* (which needs 1 negative question) on average over 10 runs, which is far from the worst case ($\frac{55*149}{2} - 1 = 4096.5$) described in [14]. This is the worst situation we found across cluster pairs in `cora`. For another pair of clusters with sizes 55 and 117 respectively, $s_{\textbf{wang}}$ asks 1 negative question, just like s^*. Consistently, we see this later behavior in most cluster pairs in `cora`. Our analysis explains why in practice $s_{\textbf{wang}}$ is much more effective than the predicted worst case analysis of Vesdapunt et al.

Analysis of Vesdapunt et al.'s algorithm. Vesdapunt et al. [14] provided a different algorithm: $s_{\textbf{vesd}}$ estimates the cluster size which contains a node v, and orders the nodes according to that order. At any time, the number of clusters maintained is at most k. When a node v is chosen, at most k questions are required to find out the cluster which contains v. Therefore, this gives an $O(k)$-approximation on the overall number of questions t_R.

THEOREM 2. *$s_{\textbf{vesd}}$ gives an $O(\log^2 n)$-approximation algorithm under the edge noise model with $\alpha \leq \frac{n}{2}$, $\beta = O(\log n)$.*

PROOF. Let C_v denote the true cluster size containing v, and $\hat{C}_v = \sum_{u \in V} p(u, v)$ its estimated cluster size. Let $c(v)$ denote the cluster that contains v.

$$\mathsf{E}\left[\hat{C}_v\right] = \sum_{u \in V} \mathsf{E}\left[p(u, v)\right] \approx 0.7|c(v)| + 0.1n$$

This shows that $\hat{C}_v$ is a highly biased estimate of C_v and unless C_v is substantially large $\omega(\sqrt{n})$, the ordering by $\hat{C}_v$ could be arbitrary. Therefore, even under the edge noise model, $s_{\textbf{vesd}}$ may pick first k nodes that belong to k different clusters. As a result $\binom{k}{2}$ "negative" questions, i.e., questions resulting in a negative answer, are issued. Onward the nodes chosen must belong to one of the selected k clusters. Suppose the algorithm chooses a node v that belongs to cluster c_i. Let R_v denote the "negative" edges, i.e., belonging to E^-, incident on node v and clusters $c_1, c_2, .., c_{i-1}, c_{i+1}, ..., c_k$ that have probability value higher than the highest probability value edge, in the ground truth C, connecting v to c_i. Let the highest probability value edge connecting v to c_i have value $1 - \eta$. Now following the same analysis as in Theorem 1, we get the desired approximation bound for $s_{\textbf{vesd}}$ □

For our application k is often $\Omega(n)$, so this gives an exponential improvement over the worst case guarantees of $s_{\textbf{vesd}}$ analysis.

5.2 Maximizing Progressive Recall

Recall from Section 2 that (i) $|c_1| \geq |c_2| \geq |c_3| \geq .... \geq |c_k|$; (ii) for maximizing progressive recall the best strategy is s^*, and (iii) the minimum number of questions that need to be asked to the oracle for $E_r^+ = E^+$ is $t_r^* = n - k$.

Let E_t^{OPT} be the maximum number of edges that can be positively inferred after $t \leq t_r^*$ questions. Let E_t denote the number of edges that can be positively inferred after t questions. If $t = \sum_{i=1}^{j} (|c_i| - 1) + l$ where $l < |c_{j+1}| - 1$, then

$$E_t^{OPT} = \sum_{i=1}^{j} \binom{|c_i|}{2} + \binom{l}{2}$$

It is tempting to use the following notion of approximation for progressive recall: $\min_t \frac{E_t^{OPT}}{E_t}$ However, such a measure is overly pessimistic. For example, if the first edge asked to the oracle by a strategy yields a negative answer, then $E_1 = 0$, whereas $E_1^{OPT} = 1$, and the approximation factor is unbounded. Similarly, considering $\frac{E_{t_r^*}^{OPT}}{E_{t_r^*}}$ is also a bad measure. Assume there are $\frac{n}{2}$ clusters each of size 2, then even with independent edge noise model, we can have $\frac{n}{2}$ non-matching record pairs with probability value higher than the probability values of the $\frac{n}{2}$ matching ones. Hence, the approximation factor will be ∞.

Since s^* asks all the "positive" questions first (i.e., resulting in a positive answer), we therefore only look at the ordering of positive questions under various strategies. That is, we only consider t questions that result in positive answers. Let F_t^{OPT} denote the maximum number of edges that can be positively inferred after t questions all of which result in positive answers, and let F^t denote the number of edges that can be positively inferred by an algorithm again when all of the t questions return positive answers. Then:

$$\texttt{benefit}^+ \geq \min_t \frac{F_t^{OPT}}{F^t}$$

is the approximation factor of the proposed algorithm.

PROPOSITION 1. *$s_{\textbf{wang}}$ has an approximation factor of $\Omega(n)$ under progressive recall.*

PROOF. $s_{\textbf{wang}}$ algorithm can have an approximation ratio as bad as $O(n)$. Suppose C contains one big cluster of size $\frac{n}{3}$ and $\frac{n}{3}$ clusters each of size 2. Consider $t = \frac{n}{3}$, $s_{\textbf{wang}}$ can pick $\frac{n}{3}$ edges from each of 2-sized clusters, whereas the optimum algorithm s^* picks $\frac{n}{3} - 1$ edges from the one big cluster and one edge from one of the small clusters. Therefore, we have $F^t = \frac{n}{3}$, whereas $F_t^{OPT} = \binom{\frac{n}{3}}{2}$. □

PROPOSITION 2. *$s_{\textbf{vesd}}$ has an approximation factor of $\Omega(\sqrt{n})$ under progressive recall.*

PROOF. Consider $\sqrt{n}$ clusters each of size $\sqrt{n}$. $s_{\textbf{vesd}}$ can pick first $\sqrt{n}$ nodes each from a different cluster. After that the next $\sqrt{n} - 1$ positive questions can correspond to a single edge from each of $\sqrt{n} - 1$ different clusters. Therefore while $F^t = \sqrt{n} - 1$, we have $F_t^{OPT} = \binom{\sqrt{n}}{2} = O(n)$. □

We now focus our attention on our *edge ordering* methodology, $s_{\textbf{edge}}$. The choice of w restricts the number of edges that the algorithm considers for computing benefit. For analysis purpose, we use an unrestricted window size w.

PROPOSITION 3. *Suppose $e_{i_1}, e_{i_2}, ..., e_{i_k}$ are the first edges picked from clusters $c_{i_1}, c_{i_2}, ..., c_{i_k}$ respectively in the order they are chosen by $s_{\textbf{edge}}$. If $i_j = j$ and edge noise is chosen from $Unif(0, \frac{1}{2})$ then $s_{\textbf{edge}}$ achieves optimum progressive recall.*

PROOF. $i_j = j$ ensures that the first edge in the ground truth C picked is from the largest size cluster c_1. Suppose $e_1 = (u, v)$ then if $|c_1| = 2$, c_1 is fully complete, and the algorithm behaves same as s^*. Otherwise, $|c_1| > 2$. Let $w \in c_1 \setminus \{u, v\}$, then the benefit of the edge (u, w) is $2(1 - \eta_{u,w})$ and the benefit of the edge (v, w) is $2(1 - \eta_{v,w})$. Let without loss of generality, benefit of edge (u, w) be higher than (v, w), now consider any other edge that does not contain either u or v, benefit of that edge, denoted (a, b) is $1 - \eta_{a,b}$. Now if $2(1 - \eta_{u,w}) \leq 1 - \eta_{a,b}$ then $1 \leq 2\eta_{u,w} - \eta_{a,b} \leq 2\eta_{u,w}$, or $\eta_{u,w} \geq \frac{1}{2}$ which is not possible. Therefore, our $s_{\textbf{edge}}$ strategy always picks an edge that grows the cluster c_1 in a connected way,

exactly like s^*. Following the argument, the algorithm completes the cluster c_1, then c_2, then c_3 and so on giving optimum value for progressive recall. □

Note that under the $Unif(0, \frac{1}{2})$ noise model, $s_{\textbf{wang}}$ asks the minimum number of overall queries, but in terms of progressive recall, its performance could be as bad as $O(n)$ from the best case.

The above proof relies on choosing the first edge from each cluster according to the non-increasing order of cluster size. However, it is possible with skewed cluster distribution that this ordering is violated. In the worst case, $s_{\textbf{edge}}$ may choose to complete the minimum size cluster first and so on, but every time it grows a cluster to completion, instead of giving a fragmented view. Our *hybrid ordering* methodology, $s_{\textbf{hybrid}}$, is designed to avoid such a scenario by providing a mechanism to select edges from clusters in non-increasing order of cluster size. We now analyze the $s_{\textbf{hybrid}}$ strategy where we ignore the conditions (iii) and (iv) for querying. The conditions (iii) and (iv) are used for the sake of optimization since if the benefit of a node falls below a certain threshold or more than t questions have already resulted in negative answers then it is likely that this node starts a new cluster.

PROPOSITION 4. *Suppose* $\mathsf{E}[|c_i|] > \mathsf{E}[|c_j|]$, $j > i$, $i, j \in [1, k]$ *then* $s_{\textbf{hybrid}}$ *achieves optimum progressive recall.*

This is easy to see since under this assumption, the algorithm always picks all the nodes from the first cluster, then the nodes from the second cluster and so on. Therefore, we now consider the worst case scenario, where the expected sizes may not reflect the true size and the first k nodes picked may correspond to k different clusters.

PROPOSITION 5. *Suppose* $v_{i_1}, v_{i_2}, ..., v_{i_k}$ *are the "second" nodes picked from clusters* $c_{i_1}, c_{i_2}, ..., c_{i_k}$ *respectively in the order they are chosen by* $s_{\textbf{hybrid}}$*. If* $i_j = j$ *and edge noise is chosen from* $Unif(0, \frac{1}{2})$ *then* $s_{\textbf{hybrid}}$ *achieves optimum progressive recall.*

The proof of the proposition is similar to Proposition 3. Essentially, once we have two representatives from the largest cluster, no matter what the edge noise is, the benefit of a third node in the same cluster is higher than any other node in a different cluster and so on. Thus the algorithm first completes the first cluster, then the second cluster and so on optimizing the progressive recall.

In the worst case, $s_{\textbf{hybrid}}$ provides an $O(\sqrt{n})$-approximation to progressive recall. Intuitively, only the expected sizes of the clusters bigger than $\Theta(\sqrt{n})$ are preserved due to the Chernoff-Hoeffding bound. Thus the algorithm performs close to optimal as long as clusters of size greater than $\Theta(\sqrt{n})$ are considered. After that, if there are r clusters of size $O(\sqrt{n})$, then $s_{\textbf{hybrid}}$ can grow these clusters from the smallest size to largest in the worst scenario. If S_{max} and S_{min} are respectively the maximum and minimum size clusters among the r clusters that is $S_{min} \leq S_{max} \leq \sqrt{n}$, then the worst case approximation factor is $\frac{S_{max}}{S_{min}} < \sqrt{n}$. For lack of space, the detailed proof can be found in the full version.

6. EXPERIMENTS

In this section we discuss the results of our experiments. We compare all the algorithms on both synthetic and real, publicly available, datasets. We implemented our and prior strategies [16, 14, 13] in Java, in a common framework. Edges and nodes are sorted during a preprocessing phase, and ties are broken randomly. For each strategy, we take the average recall from the same 5 independent runs of the random tie break process. We ran experiments on a machine with two CPU Intel Xeon E5520 units with 16 cores each, running at 2.67GHz, with 16MB of cache and 64GB RAM.

dataset	cora	skew	sqrtn	prod	dblp
n	1,878	900	900	2,173	3,057,838
k	191	93	30	1,092	2,980,737
k'	124	93	30	1,076	54,542
$\|c_1\|$	236	50	30	3	159
$\|E\|$	62,891	8,175	13,050	1,086	299,683
t_r^*	1,687	807	870	1,081	77,101
ER	D	D	D	CC	D
origin	RR	SS	SS	RR	RS

Table 1: Number of nodes n (i.e., records), number of clusters k (i.e., entities), number of non-singleton clusters k',size of the largest cluster $|c_1|$, size of the spanning forest t_r^*, type of ER (dirty or clean–clean), and origin (real or synthetic) of the data.

dataset	cora	skew	sqrtn	prod	dblp
#blocks	1	1	1	1	341,280
sim.	Jaro [19]	Ideal	Ideal	Jaccard	Jaro [19]
R	$1.8M$	$0.4M$	$0.4M$	$2.4M$	$1.8B$
n'	1,878	900	900	2,173	53,279

Table 2: Number of blocks, similarity functions, record pairs R, and n' values. Similarity in cora and prod is computed as in [17, 15]. We emphasize that the similarity function used for dblp has no connection with the one used for the silver standard. The "ideal" function returns 1 if two records are matching and 0 otherwise.

6.1 Datasets

Some of our datasets have real attribute values and come with their own real gold standard. We refer to such datasets as Real-Real (RR). Other datasets have synthetic attribute values and a synthetic gold standard (Synthetic-Synthetic, short. SS). The remaining dataset has real attribute values and a synthetic gold standard, that we refer to as "silver" standard (Real-Synthetic, short. RS). We compute the silver standard as in [13] (see Section 8.1 of [13]). The main properties of the datasets are listed in Table 1:

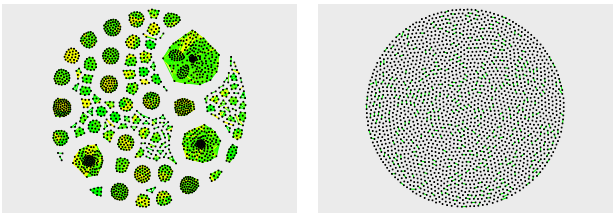
- cora [12] is a bibliography dataset. Each record contains title, author, venue, date, and pages attributes.
- prod [1] is a product dataset of mappings from 1,081 abt.com products to 1,092 buy.com products. Each record contains name and price attributes.
- skew and sqrtn contain fictitious hospital patients data, including name, phone number, birth date and address, that we produced using the data set generator of the Febrl system [3].
- dblp[6] is a bibliographic index on computer science articles.

6.1.1 Clusters

The clustering graphs of cora and prod datasets, $C_{\texttt{cora}}$ and $C_{\texttt{prod}}$, are the two extremes of a spectrum. On one side, $C_{\texttt{cora}}$ is extremely dense and few of the largest clusters account for most recall, thus, there is much gain in exploiting transitivity ($t_r^* << |E^+|$). On the other side, prod is extremely sparse and there is negligible gain in exploiting transitivity ($t_r^* \approx |E^+|$). (See Figure 5 for detailed visualizations.) In the middle of the spectrum:

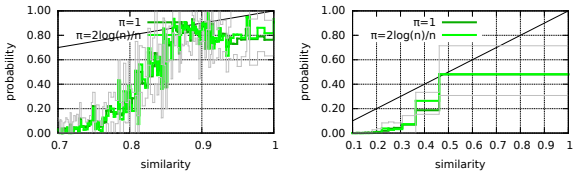
- $C_{\texttt{skew}}$ contains few ($\approx \log n$) large (size $\approx \frac{n}{\log n}$) clusters, some ($\approx \sqrt{n}$) intermediate (size $\approx \sqrt{n}$) clusters, and a long tail ($\approx \frac{n}{\log n}$) of small (size $\approx \log n$) clusters.

[6] www.informatik.uni-trier.de/ ley/db/, 13 Aug. 2015.



(a) cora (b) prod

Figure 5: Clustering graphs (i.e., C) for the RR datasets. Edge colors represent similarity scores between records (green is higher).



(a) cora (b) prod

Figure 6: Similarity-probability mappings for the RR datasets. Record pairs with similarity values below 0.7 for cora and 0.1 for prod, match with probability < 0.003. The black line shows the identity function for comparison.

- $C_{\texttt{sqrtn}}$ contains exactly $\sqrt{n}$ clusters of size $\sqrt{n}$.
- $C_{\texttt{dblp}}$ has skewed size distribution and is very sparse at the same time: only $5.4 \times 10^{-6}\%$ of record pairs are duplicates.

6.1.2 *Probabilities*

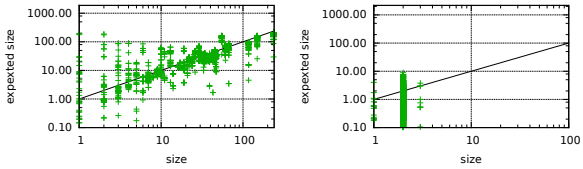
We compute edge similarities, and map them to probabilities using buckets, as in Section 3.1 of [17]. Our approach is scalable and more general than [17]: it includes blocking, a general method for selecting bucket size, and efficient probability estimation.

Blocks. Let R be the number of record pairs between which we compute similarities. When n is large, computing all $\binom{n}{2}$ pairwise similarities is not feasible. We assign records to (possibly multiple) blocks and compute similarities only between pairs in the same block. We further limit the computation to the first $O(n \operatorname{polylog} n)$ pairs (which we consider feasible), by considering blocks in non-decreasing order of size. All the remaining $\binom{n}{2} - R$ pairs are not submitted to the strategy, and are finally set as non-matching.

- We compute dblp blocks based on title and author tokens. We observed that $\log^2 n$ is the smallest $O(\operatorname{polylog} n)$ function which allows for including all the matching pairs which share at least a block, thus we set $R = n \log^2 n$.
- We consider each of other datasets as consisting of a single block (consistently with [17, 15]) and set $R = \binom{n}{2}$.

See Table 2 for details about blocking and similarity functions used.

Buckets. We evenly divide the R record pairs into buckets according to their similarity and use ground truth to compute probability for each bucket. We would like many buckets in order to get enough probability values in the mapping, and at the same time, we would like large buckets in order to get a robust probability estimate. A natural way to achieve this is by $O(\sqrt{R})$ buckets of equal size $O(\sqrt{R})$. To this end, we create similarity buckets containing $\frac{n'}{2}$ pairs[7], where n' is such that $\binom{n'}{2} = \lfloor R \rfloor$ (see Table 2).



(a) cora (b) prod

Figure 7: Expected size of clusters for the RR datasets. The black line shows the identity function.

Mapping. We extract a random sample from each bucket, by uniformly picking a fraction $\pi = 2\frac{\log n'}{n'}$ record pairs, and ask the oracle the corresponding questions for the top $Z = \frac{n'}{\log n'}$ buckets with highest similarity. The highest similarity values are mapped to the corresponding fraction of positive responses, while the remaining buckets are mapped to 0. The responses collected in this phase can be stored and re-used during the execution of the strategy. Since the expected number of questions is $O(n')$, the impact on the overall number of questions is small. In Figure 6, we compare the similarity-probability mapping of our method and the exhaustive method, which asks all the questions to the oracle ($\pi = 1$). The plots show the trimmed average, the minimum, and the maximum probability obtained in 10 runs of the experiment. In the figure, we show only the buckets with similarity above a certain threshold, such that probability is never < 0.01, which are anyway less than Z. The results suggest that our method can provide good probability estimates in practical applications. Note that in [17] and subsequent works the similarity-probability mapping is done using the exhaustive method. This is simple and effective for comparing algorithms, and we use it in the following experiments.

Expected size. We use probabilities for computing the expected cluster size of each node (as in Eq. 1) and for sorting edges and nodes during the preprocessing phase. Figure 7 plots the expected size of cluster $p_s(v)$ of each node v against its actual size $|c(v)|$, which we know from the ground truth. The results show that nodes belonging to small clusters can have large values of $p_s(v)$ and vice versa. In prod, there is no correlation between the two quantities.

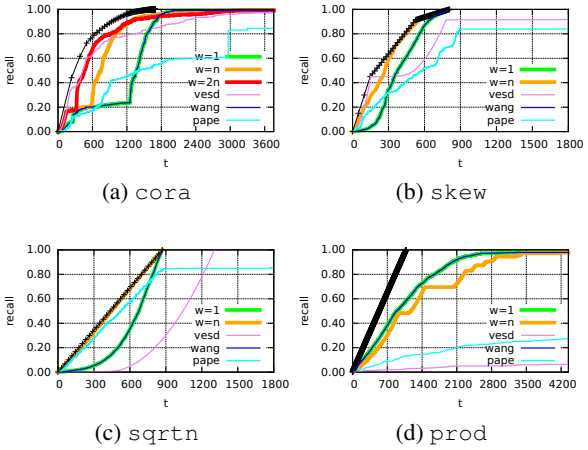
6.2 Evaluation

We study the recall functions of $s_{\textbf{edge}}$, $s_{\textbf{hybrid}}$, $s_{\textbf{wang}}$, and $s_{\textbf{vesd}}$, as the number of questions t and the number of positive questions t^+ increase[8]; the graphs for t^+ are omitted for reasons of space. Progressive recall functions are quantified as the area under these curves. In order to visually quantify normalized and benefit functions described in Section 2, we plot the recall function of s^* as frame of comparison for the different datasets as a black "+" curve.

We also include in the evaluation the approaches described in [13, 18]. Although they are designed for optimizing running time, rather than number of questions, they also solve the same basic problem of maximizing intermediate recall. Since the approach in [18] is outperformed by [13] (we refer the reader to [13] for detailed discussion and comparison), we only show the recall function

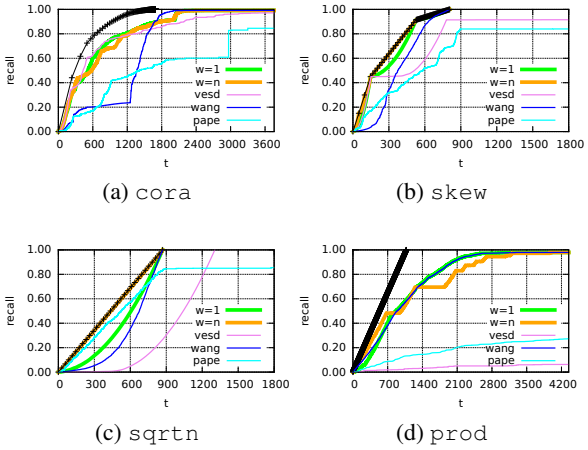
[7] Some buckets can have size sightly larger when more than $n'/2$ pairs have the same similarity score.

[8] Results for $s_{\textbf{wang}}$ and $s_{\textbf{vesd}}$ on the RR datasets are comparable to those shown in [14]. Small differences are due to small variations in the similarity probability mapping process.



(a) cora (b) skew (c) sqrtn (d) prod

Figure 8: Progressive recall of s_{edge}. s_{wang}=$s_{\text{edge}}(1)$ (see Section 4).



(a) cora (b) skew (c) sqrtn (d) prod

Figure 9: Progressive recall of s_{hybrid} with default settings. In skew and sqrtn, window size n yields optimal behaviour. In cora, questions are asked based on node benefit up to $t \approx 2000$, and after that point based on edge benefit. In prod the same happens after $t \approx 750$ questions. It is interesting to notice that for $\theta = 0.5$, in prod, all the questions are deferred to the edge benefit phase, i.e., s_{hybrid}=s_{edge} (we do not show the plots for lack of space). In both SS dataset no node/edge benefit switch occurs instead.

$s_{\text{edge}}(1)$	$s_{\text{edge}}(n)$	$s_{\text{edge}}(2n)$	$s_{\text{hybrid}}(1)$	$s_{\text{hybrid}}(n)$
0.0067	0.0022	0.0022	0.0052	0.0077

Table 3: Average loss in recall (ALR) of s_{hybrid} and s_{edge} strategies for different values of w, over all of the iterations. ALR is defined as the average difference between the recall at the end of an iteration, and the recall of the sequential strategy at the corresponding value of t. Positive values indicate higher recall of the latter.

of the latter[9], which we refer to as s_{pape}. To this end, we provide the algorithms in [13] with an oracle access to the gold standard and count the number of distinct questions asked.

- We compare our strategies with $w = \{1, n\}$ to s_{wang}, s_{vesd} and s_{pape}, on RR and SS datasets. Larger windows do not improve on $w = n$, except for cora, on which $s_{\text{edge}}(2n)$ does slightly better than any other s_{edge} setting.
- Since [16] describes a parallel version of s_{wang}, we compare our strategies to s_{wang} in a parallel setting, on cora.
- Using dblp, we show the performance of our strategies on a large, dirty dataset, and compare them to s_{pape} (which is the only strategy tested in previous works on large datasets). We do not use large values of w, because it would make the computation too slow. However, we show that even small values of $w = \log n$ have good performance.

Edge ordering. Figure 8 shows the progressive recall of s_{edge}. We first note that: s_{wang} performs poorly in cora, s_{vesd} performs poorly in prod, and s_{pape} performs poorly in cora, skew and prod. We then note that, with exception of prod, our s_{edge} strategy with largest window has better or comparable performance than any other strategy. In cora, skew and sqrtn, s_{edge} has high gain with an increasing window, because higher benefit questions have increasingly more chances to be asked first. In prod, where every cluster has size at most 3, benefit has limited impact on the computation and $s_{\text{edge}}(1)$ is the best setting. Poor performances of s_{pape} can be due to its lack of mechanisms for growing largest clusters first and for inferring negative edges. The latter has the most effect on prod. See Figure 8d for comparison.

Hybrid ordering. Figure 9 shows the progressive recall of s_{hybrid} with default settings. $s_{\text{hybrid}}(w = 1)$ has similar behaviour as s_{vesd}, because they process nodes in the same order (see Section 4 for a detailed discussion). We first note that window size has moderate effect for cora and skew, which are the two datasets showing some skew in the size distribution of clusters. This is due to the fact that in both datasets the node ordering produced by the expected size of clusters, is a good estimate of the optimal node ordering. Differently, in sqrtn and prod, which are the two datasets with constant (or nearly constant) cluster sizes, the node ordering produced by the expected size of clusters is a random permutation of V. We then note that our $s_{\text{hybrid}}(1)$ strategy has better or comparable performances than any other strategy. In non-extremely sparse datasets, progressive recall can be further increased using $w = n$, yielding optimal benefit for the SS datasets.

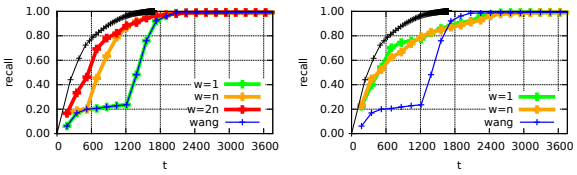
Parallel strategies. Asking questions sequentially may not be suitable in many applications. For instance, in crowd-sourcing, it prohibits workers from doing tasks in parallel and leads to long completion times. We use the same approach as [16] for selecting at most $\frac{n}{\log n}$ non-redundant questions[10] during the execution of our strategies and, at each iteration, we ask the oracle to answer all the questions in parallel. Figure 10 shows the progressive recall of the parallel versions of our strategies, compared to the parallel version of s_{wang} [16]. See Table 3 for a detailed comparison.

Large datasets. In Figure 11, we show the results of our experiments for dblp. We show the progressive recall of s_{edge} and s_{hybrid}, with respect to all the matching record pairs that share at least a block[11]. The results suggest that even when w is small with respect to n, our strategies have much better performance than s_{pape}.

[9] As suggested by the authors, we use their PSNM algorithm for the small single-block datasets, and their PB algorithm for the larger dblp dataset. Furthermore, we ran the experiments with different settings and selected for each dataset those yielding the best results.

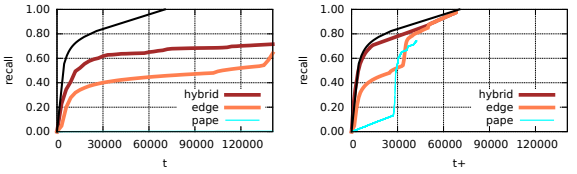
[10] That is, whose response cannot be inferred from other responses.

[11] Even though evaluating the blocking strategy is out of the scope of this paper, we mention that almost 15% of the record pairs that are in the silver standard do not share any block, that is, do not have



(a) s_{edge} (b) s_{hybrid}

Figure 10: Progressive recall of parallel s_{hybrid} and s_{edge} strategies for different values of w and default values of other parameters, using `cora` dataset. Each data point corresponds to a single iteration.



(a) all questions (b) positive questions

Figure 11: Progressive recall of s_{hybrid} and s_{edge} strategies for $w = \log n$, $\theta = 0$ and $\tau = \log n$, using `dblp` dataset.

6.3 Example Visualization

In this section we show the different T-clusterings grown by the considered strategies, with their best settings, on `cora`. In Figure 12 we show the T-clusterings after $|c_1| - 1 = 235$ questions. Ideally, the largest cluster can be grown fully. In practice, s_{wang} has grown the tighter connected "subclusters" (corresponding to highest probabilities edges), s_{vesd} has grown part of the largest clusters (corresponding to highest expected-size nodes, which are not those with the highest actual sizes), and s_{edge} has fully grown c_2. s_{hybrid} is the only strategy which has fully grown the largest cluster.

7. CONCLUSION

We formally address the problem of maximizing progressive recall in an online setting for the ER task. In this framework, we present a novel benefit metric and two greedy algorithms that use this benefit metric. We show that our problem is NP-hard, and formally analyze the quality of the solutions obtained by previous strategies and our greedy algorithms. We evaluate all the considered strategies over real and synthetic datasets. The results suggest that our greedy algorithms are robust across all sparse and dense datasets, showing higher progressive recall than previous strategies.

In our future work, we plan to extend our results to include errors in the oracle answers. Our goal is to model mistakes that a *real* oracle, such a crowd or an expert worker, could make in the labeling process, and design robust strategies in an online setting.

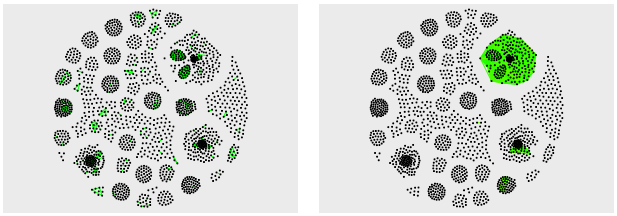
8. ACKNOWLEDGMENTS

We thank the authors of [13] for their code and dataset.

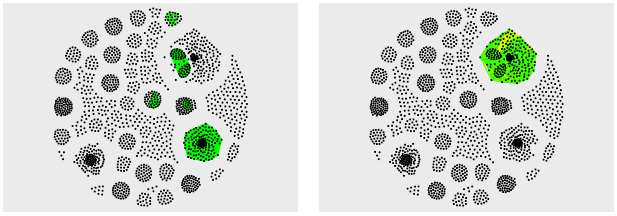
9. REFERENCES

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

any token in common both in author and title. This is mainly be due to the fact that the silver standard is synthetic and subject to errors.



(a) s_{wang} (b) s_{vesd}



(c) $s_{\text{edge}}(2n)$ (d) $s_{\text{hybrid}}(n, \log n, 0.3)$

Figure 12: Edges found for `cora` after $t = |c_1| - 1$ questions.

[2] Y. Altowim, D. V. Kalashnikov, and S. Mehrotra. Progressive approach to relational entity resolution. *PVLDB*, 7(11):999–1010, 2014.

[3] P. Christen. Febrl-: an open source data cleaning, deduplication and record linkage system with a graphical user interface. In *KDD*, pages 1065–1068, 2008.

[4] N. Dalvi, A. Dasgupta, R. Kumar, and V. Rastogi. Aggregating crowdsourced binary ratings. In *WWW*, pages 285–294, 2013.

[5] G. Demartini, D. E. Difallah, and P. Cudré-Mauroux. Zencrowd: leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking. In *WWW*, pages 469–478, 2012.

[6] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowl. Data Eng.*, 19(1):1–16, 2007.

[7] I. P. Fellegi and A. B. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.

[8] L. Getoor and A. Machanavajjhala. Entity resolution: theory, practice & open challenges. *PVLDB*, 5(12):2018–2019, 2012.

[9] A. Ghosh, S. Kale, and P. McAfee. Who moderates the moderators?: crowdsourcing abuse detection in user-generated content. In *EC*, pages 167–176, 2011.

[10] C. Gokhale, S. Das, A. Doan, J. F. Naughton, N. Rampalli, J. Shavlik, and X. Zhu. Corleone: Hands-off crowdsourcing for entity matching. In *SIGMOD Conference*, pages 601–612, 2014.

[11] D. R. Karger, S. Oh, and D. Shah. Iterative learning for reliable crowdsourcing systems. In *NIPS*, pages 1953–1961, 2011.

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

[13] T. Papenbrock, A. Heise, and F. Naumann. Progressive duplicate detection. *Knowledge and Data Engineering, IEEE Transactions on*, 27(5):1316–1329, 2015.

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

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

[16] J. Wang, G. Li, T. Kraska, M. J. Franklin, and J. Feng. Leveraging transitive relations for crowdsourced joins. In *SIGMOD Conference*, pages 229–240, 2013.

[17] S. E. Whang, P. Lofgren, and H. Garcia-Molina. Question selection for crowd entity resolution. *PVLDB*, 6(6):349–360, 2013.

[18] S. E. Whang, D. Marmaros, and H. Garcia-Molina. Pay-as-you-go entity resolution. *Knowledge and Data Engineering, IEEE Transactions on*, 25(5):1111–1124, 2013.

[19] W. E. Winkler. Overview of record linkage and current research directions. In *Bureau of the Census*. Citeseer, 2006.