Scaling up Copy Detection

Xian Li [1], Xin Luna Dong [2], Kenneth B. Lyons [3], Weiyi Meng [1], Divesh Srivastava [3]

[1] *Computer Science Department of Binghamton University, {xianli, meng}@cs.binghamton.edu*

[2] *Google Inc, lunadong@google.com*

[3] *AT&T Labs-Research, {kbl, divesh}@research.att.com*

***Abstract*—Recent research shows that copying is prevalent for Deep-Web data and considering copying can significantly improve truth finding from conflicting values. However, existing copy detection techniques do not scale for large sizes and numbers of data sources, so truth finding can be slowed down by one to two orders of magnitude compared with the corresponding techniques that do not consider copying. In this paper, we study *how to improve scalability of copy detection on structured data*.**

Our algorithm builds an inverted index for each *shared* value and processes the index entries in decreasing order of how much the shared value can contribute to the conclusion of copying. We show how we use the index to prune the data items we consider for each pair of sources, and to incrementally refine our results in iterative copy detection. We also apply a sampling strategy with which we are able to further reduce copy-detection time while still obtaining very similar results as on the whole data set. Experiments on various real data sets show that our algorithm can reduce the time for copy detection by two to three orders of magnitude; in other words, truth finding can benefit from copy detection with very little overhead.

I. Introduction

As we enjoy the abundance of information on the Web, we are often confused and misguided by low-quality data, which can be out-of-date, incomplete, or erroneous. Recently, Li et al. [12] showed that even for domains such as stock and flight, conflicting values are provided by different Deep Web sources on 70% of the *data items* (*e.g.*, closing price of a stock). In addition, although well-known authoritative sources, such as *NASDAQ* for stock and *Orbitz* for flight, often have fairly high accuracy, they may not have the desired coverage. Many applications, such as integrating Web-scale data and building knowledge bases from the Web, call for advanced *data fusion* techniques to resolve conflicts from different sources and identify values that reflect the real world.

Typically, we expect that the values provided by many sources are likely to be true. Unfortunately, data copying is common on the Web: a false value can spread through copying and become quite popular. Thus, we need to detect copying and discount data from copiers for truth finding. Most of current copy-detection techniques on structured data [2], [6], [7], [15] share two features. First, they exploit the intuition that copying is likely if *many false values* are shared, since this is unlikely to happen between independent sources. Second, since the truthfulness of data is often unknown *a priori*, they iteratively conduct copy detection, truth finding, and source-accuracy computation until convergence. In [12] the authors show that the copy-detection techniques of [6] can significantly improve truth-finding results in the presence of copied values and fix half of the errors made by naive voting or considering only source accuracy. Detecting copying between Web sources is also important to finding the truths in building knowledge bases [9], which are widely used in Web search. Copy detection is also valuable in studying dissemination of information, protecting the rights of data providers, and so on.

Despite its importance, research on copy detection for structured data is still in its infancy, focusing mainly on effectiveness of the techniques. Current techniques examine every shared data item between every pair of sources in each iteration to detect copying, so do not scale when the sizes of the sources, the number of the sources, or the number of iterations is large. As pointed out in [12], even on a medium-sized data set (55 sources and 16,000 data items), conducting copy detection would slow down data fusion by one to two orders of magnitude. In the Big Data environment where the data sources are growing rapidly in size and millions of sources in the same domain are emerging [4], scalability is important for successful copy detection. Consequently, we propose and study the problem of *how to improve scalability of copy detection for structured data.*

Before we describe our techniques, it is instructive to consider scalable techniques that have been proposed for discovering copying on other types of data, such as text documents and software programs (surveyed in [8]). Each document or program can be considered as a text sequence. Reuse of sufficiently large text fragments is taken as evidence for copying and copy detection essentially looks for such fragments. To improve scalability, proposed techniques create *signatures* (or *fingerprints*) of each text fragment and build an *inverted index* for all or selected signatures. A pair of documents or programs are compared only if a sufficiently large number of signatures are shared. Directly applying such techniques on structured data would be inadequate for two reasons. First, different values need to be treated differently in copy detection: sharing false values is treated as strong evidence for copying whereas sharing true ones is considered as a possible coincidence and treated only as weak evidence; thus, whether copying is likely depends not only on the number of shared values but also on the truthfulness of the shared values. Second, there is no natural way to order structured data (records and attributes); thus, large text fragments may not be shared by sources even if there is copying.

We propose a comprehensive set of index structures and algorithms for this problem and make the following contributions. First, we design an inverted index, where each entry corresponds to a shared value for a particular data item and lists the data sources that provide this value. The presence of a source in an index entry guarantees its absence in all entries that correspond to other values for the same data item. We associate each entry with a score, derived from the probability of the value being false; the higher the score, the stronger the evidence that sharing the entry can serve for detecting copying. We process the entries in decreasing order of the scores and consider a pair of sources only if they share at least some value with a high score (Section III).

Second, we propose pruning algorithms to further improve scalability for copy detection. Since we process index entries in decreasing order of their scores, we consider strong evidence early as we scan the index. Once we have accumulated enough evidence to decide copying or no-copying, we can stop without considering every shared value (Section IV).

Third, we develop incremental algorithms for iterative copy detection. We observe that between consecutive iterations, our truth-finding decisions typically change very slightly, and so do our decisions on copying. Instead of detecting copying from scratch in each iteration, we refine our decisions incrementally from previous iterations (Section V).

Finally, we experimented on a variety of real data sets, showing high scalability of the proposed techniques and big performance gains over simple sampling strategies (Section VI). We show that our algorithms together can speed up copy detection by two to three orders of magnitude, and can detect copying for thousands of sources in seconds even on a single server. With our algorithms, copy detection can significantly improve truth finding with very little overhead.

Our index and pruning techniques also shed light on other applications that require computing similarity by accumulating weighted evidence; for example, in record linkage different attributes may have different weights in the computation of record similarity.

II. Preliminaries

We review copy-detection techniques for structured data and describe the opportunities to improve scalability.

A. Copy detection

Copy detection for structured sources was recently studied in [2], [5], [6], [7], [15]. They consider a domain $\mathcal{D}$ of *data items*, each describing a particular aspect of a real world object, such as the capital of a state. They consider a set $\mathcal{S}$ of *data sources*, each providing data for a subset of data items in $\mathcal{D}$; we denote by $\bar{D}(S)$ the items provided by $S \in \mathcal{S}$. Schema mapping and entity resolution are assumed to have been performed so it is known which data items are shared between the sources (errors in these stages can be treated as wrongly provided data). A source S_1 is considered as a *copier* of S_2 if S_1 copies a subset of data values from S_2. *Copy detection* aims at *finding copying between sources in* $\mathcal{S}$.

Bayesian analysis: The key idea in copy detection is to examine the values shared between a pair of sources.[1] It is assumed that each data item is associated with a single true value that reflects the real world, but there are in addition $n > 1$ false values in the domain for each data item, and they are uniformly distributed.[2] Thus, the likelihood that two independent sources share the same false value is typically low. As a result, sharing false values on a large number of data items serves as strong evidence for copying.

Based on this intuition, Bayesian analysis is conducted for copy detection [2], [5], [6], [7]. Consider two sources S_1 and S_2 and let Φ be the observation on their data. Denote by $S_1 \to S_2$ (or $S_2 \leftarrow S_1$) copying by S_1 from S_2, and by $S_1 \perp S_2$ no-copying between them.[3] It is assumed that there is no mutual copying (S_1 copies from S_2 *and* S_2 copies from S_1), so

$$Pr(S_1 \perp S_2|\Phi) = \frac{\beta Pr(\Phi|S_1 \perp S_2)}{\beta Pr(\Phi|S_1 \perp S_2) + \alpha Pr(\Phi|S_1 \to S_2) + \alpha Pr(\Phi|S_1 \leftarrow S_2)} \quad (1)$$

Here, $0 < \alpha < .5$ is the a-priori probability of a source copying from another one and $\beta = 1 - 2\alpha$. Assuming independence between data items, and denoting by Φ_D the observation on data item D, we have $Pr(\Phi|S_1 \perp S_2) = \Pi_{D \in \mathcal{D}} Pr(\Phi_D|S_1 \perp S_2)$ (similar for other cases). Thus, we can rewrite Eq.(1) as follows.

$$Pr(S_1 \perp S_2|\Phi) = \frac{1}{1 + \frac{\alpha}{\beta}(\Pi_{D \in \mathcal{D}} \frac{Pr(\Phi_D|S_1 \to S_2)}{Pr(\Phi_D|S_1 \perp S_2)} + \Pi_{D \in \mathcal{D}} \frac{Pr(\Phi_D|S_1 \leftarrow S_2)}{Pr(\Phi_D|S_1 \perp S_2)})}. \quad (2)$$

Computation of Eq.(2) would require computing $\Pi_{D \in \mathcal{D}} \frac{Pr(\Phi_D|S_1 \to S_2)}{Pr(\Phi_D|S_1 \perp S_2)}$; we denote its logarithms by $C_{\to} = \sum_{D \in \mathcal{D}} \ln \frac{Pr(\Phi_D|S_1 \to S_2)}{Pr(\Phi_D|S_1 \perp S_2)}$ (similarly, $C_{\leftarrow} = \sum_{D \in \mathcal{D}} \ln \frac{Pr(\Phi_D|S_1 \leftarrow S_2)}{Pr(\Phi_D|S_1 \perp S_2)}$). Essentially, $C_{\to}$ and $C_{\leftarrow}$ accumulate the contribution from each data item $D \in \mathcal{D}$. We denote the *contribution score* from D to $C_{\to}$ by $C_{\to}(D) = \ln \frac{Pr(\Phi_D|S_1 \to S_2)}{Pr(\Phi_D|S_1 \perp S_2)}$ and compute it as follows (similar for $C_{\leftarrow}$).

1. Providing the same value v: For each source $S \in \mathcal{S}$, its *accuracy*, denoted by $A(S)$, is measured as the fraction of its true values over all provided values. This can be considered as the probability of S providing a true value for a data item. Then, the probability of S_1 and S_2 independently providing the (same) true value is $A(S_1)A(S_2)$, and that for the same false value is $\frac{1-A(S_1)}{n} \cdot \frac{1-A(S_2)}{n} \cdot n = \frac{(1-A(S_1))(1-A(S_2))}{n}$ (recall it is assumed that there are n uniformly distributed false values). In practice, we are often not sure which value is true. Let $P(D.v)$ be the probability of value v being true for D, then

$$Pr(\Phi_D|S_1 \perp S_2) = P(D.v)A(S_1)A(S_2) + (1 - P(D.v))\frac{(1 - A(S_1))(1 - A(S_2))}{n}. \quad (3)$$

Now consider copying. Let s be the *selectivity* of copying[4]; that is, the probability that the copier copies on a particular item. When S_1 copies from S_2 on D (with probability s), they must provide the same value. The probability of our observed

[1]Advanced techniques also consider coverage and formatting of data items [5], for which we can extend our techniques.

[2]This assumption can be relaxed to take value distributions into account [6], but is used here for simplicity.

[3]We can also extend our techniques to distinguish direct copying from co-copying and transitive copying [5] and we skip the details.

[4]α, n, s are inputs and can be set/refined according to [5], [6].

TABLE I
MOTIVATING EXAMPLE. FALSE VALUES ARE IN ITALIC FONT. AN EMPTY CELL CORRESPONDS TO A MISSING VALUE FROM A SOURCE.

	Accu	NJ (D_1)	AZ (D_2)	NY (D_3)	FL (D_4)	TX (D_5)
S_0	0.99	Trenton	Phoenix	Albany		Austin
S_1	0.99	Trenton	Phoenix	Albany	Orlando	Austin
S_2	0.2	*Atlantic*	Phoenix	*NewYork*	*Miami*	*Houston*
S_3	0.2	*Atlantic*	Phoenix	*NewYork*	*Miami*	*Arlington*
S_4	0.4	*Atlantic*	Phoenix	*NewYork*	Orlando	*Houston*
S_5	0.6	*Union*	*Tempe*	Albany	Orlando	Austin
S_6	0.01		*Tempe*	*Buffalo*	*PalmBay*	*Dallas*
S_7	0.25	Trenton		*Buffalo*	*PalmBay*	*Dallas*
S_8	0.2	Trenton	*Tucson*	*Buffalo*	*PalmBay*	*Dallas*
S_9	0.99	Trenton			Orlando	Austin

value then depends on the likelihood that S_2 provides the value. The probability of S_2 providing the true value is $A(S_2)$ and that for a false value is $1 - A(S_2)$. Thus, the probability for our observation of S_2's data on D, denoted by $\Phi_D(S_2)$, is

$$Pr(\Phi_D(S_2)) = P(D.v)A(S_2) + (1 - P(D.v))(1 - A(S_2)). \quad (4)$$

When S_1 does not copy from S_2 on D (with probability $1 - s$), the probability that they both (independently) provide v is the same as $Pr(\Phi_D|S_1 \perp S_2)$. Thus,

$$Pr(\Phi_D|S_1 \to S_2) = (1-s)Pr(\Phi_D|S_1 \perp S_2) + sPr(\Phi_D(S_2)). \quad (5)$$

Combining Eq.(3-5), we have

$$C_\to(D) = \ln(1 - s + s \cdot \frac{Pr(\Phi_D(S_2))}{Pr(\Phi_D|S_1 \perp S_2)}). \quad (6)$$

2. Providing different values: When two sources provide different values, the copier cannot copy (the probability is $1 - s$) and they independently provide different values. Thus,

$$Pr(\Phi_D|S_1 \to S_2) = (1 - s)Pr(\Phi_D|S_1 \perp S_2); \quad (7)$$

$$C_\to(D) = \ln(1 - s). \quad (8)$$

It has been proved that $C_\to(D)$ is positive when S_1 and S_2 share the same value on D and negative otherwise, and it is larger when the shared value has a lower $P(D.v)$ (*i.e.*, v is more likely to be false) [6]. In other words, sharing a value serves as evidence for copying and vice versa, and sharing a false value serves as strong evidence for copying.

Example 2.1: Consider 10 data sources that describe capitals for 5 states in the US (Table I); their accuracy measures are shown on the second column. There is copying between $S_2 - S_4$ and between $S_6 - S_8$. We set $\alpha = 0.1$, $s = 0.8$, and $n = 50$.

Consider S_2 and S_3 as an example. Starting with D_1, they provide the same value so we apply Eq.(6). Suppose that NJ.Atlantic has probability .01 to be true. Then, $C_\to(D_1) = C_\leftarrow(D_1) = \ln(.2 + .8 \cdot \frac{.01*.2+.99*.8}{.01*.2*.2+.99*\frac{.8*.8}{50}}) = 3.89$, showing that sharing this false value is strong evidence for copying. We compute for other items similarly and eventually $C_\to = C_\leftarrow = 3.89+1.6+3.86+3.83-1.6 = 11.58$. Applying Eq.(2) computes $Pr(S_2 \perp S_3|\Phi) = .00004$, so copying is very likely.

Now consider S_0 and S_1, which also share 4 values. However, suppose we know that these values are all true and each of them has a contribution .01 (details skipped). Eventually, $C_\to = C_\leftarrow = .01 * 4 = .04$ and $Pr(S_0 \perp S_1|\Phi) = .79$, so copying is unlikely. □

TABLE II
ITERATIONS FOR THE MOTIVATING EXAMPLE.

	Rnd 1	Rnd 2	Rnd 3	Rnd 4	Rnd 5
S_0	0.75	0.94	0.96	0.98	0.99
S_1	0.98	0.99	0.99	0.99	0.99
S_2	0.38	0.23	0.21	0.2	0.2
S_3	0.38	0.23	0.21	0.2	0.2
S_4	0.58	0.43	0.41	0.4	0.4

(a) Source accuracy.

	Rnd 1	Rnd 2	Rnd 3	Rnd 4	Rnd 5
NJ.Trenton	0.9	0.95	0.96	0.97	0.97
NJ.Atlantic	0.07	0.03	0.02	0.01	0.01
AZ.Phoenix	0.94	0.95	0.95	0.95	0.95
NY.Albany	0.07	0.77	0.88	0.92	0.94
NY.NewYork	0.84	0.16	0.08	0.03	0.02
FL.Orlando	0.9	0.92	0.92	0.92	0.92
FL.Miami	0.05	0.03	0.04	0.03	0.03
TX.Austin	0.9	0.93	0.95	0.96	0.96
TX.Houston	0.04	0.03	0.02	0.02	0.02

(b) Probability of values in the index.

Iterative computation: We often do not know value probability $P(D.v)$ and source accuracy $A(S)$, and computing them often requires knowledge of the copying relationship (details in [6]). An iterative approach has been proposed as follows [2], [6]: starting with assuming the same accuracy for each source, each round iteratively computes copying probability, value truthfulness, and source accuracy, until convergence. For our motivating example, there are five rounds before convergence. Table II shows the source accuracy and value probability computed in each round (for simplicity, we show only for the first 5 sources and their values).

B. Opportunities for scalability improvement

Previous works [5], [6] conduct copy detection in an exhaustive fashion. For each pair of sources, the algorithm, called PAIRWISE, does the following: (1) compute for each shared $D \in \mathcal{D}$ the contribution scores $C_\to(D)$ and $C_\leftarrow(D)$; (2) accumulate the scores and compute $C_\to$ and $C_\leftarrow$; (3) apply Eq.(2) for probability computation. This process is repeated in every round. If the results converge in l rounds, the time complexity is $O(l|\mathcal{D}||\mathcal{S}|^2)$. PAIRWISE is not scalable if the number of sources or data items is large, or there are many iterations. The algorithm proposed in [2] also examines every pair of sources, so has similar complexity to PAIRWISE.

There are several opportunities for improving scalability of copy detection. First, for some pairs of sources that share no value at all or just a few true values, we can determine that they are independent without going through all of the shared data items; this can reduce the number of source pairs we examine. For our motivating example, PAIRWISE requires examining 45 pairs of sources; however, among them 18 pairs (such as S_0 and S_6) do not share any value, and the pair of S_0 and S_5 share only two true values. We can skip these pairs. Section III describes how we can explore this opportunity by building and using a specialized inverted index.

Second, for some pairs of sources that share a lot of false values, we can determine copying after we observe only a subset of these false values; this can reduce the number of data items we examine for those pairs. In our motivating example, S_2 and S_3 share 4 values, including 3 false ones; actually, after observing 2 false values, we can already determine copying without knowing the rest of the provided values. Section IV explores this opportunity for single-round copy detection.

Third, in the iterative process, the changes in value probability and source accuracy between two consecutive rounds after the second round are typically very small. Thus, we can do copy detection incrementally to consider fewer data items for each pair of sources in later rounds. Section V explores this opportunity and describes incremental copy detection.

Ideally, these aforementioned optimizations should tremendously reduce computation and thus execution time, while leading to the same (binary) decision on copying relationships, and also on value truthfulness. In practice, however, early pruning may improve efficiency with a slight loss of accuracy. We show in experiments (Section VI) the effectiveness and scalability of our techniques.

We have also explored Fagin's NRA (No Random Access) algorithm [10] for top-k search to speed up copy detection. We maintain for each value of a data item a list of contribution scores for the pairs of sources that share the value, and order the pairs in decreasing score order. We also maintain a list containing the accumulated contribution scores from different values for pairs of sources that have such differences. Then, $C_{\rightarrow}$ (similar for $C_{\leftarrow}$) for a particular pair of sources is the sum of the scores from all lists. To find copying, we can apply NRA to find the pairs with top values of $C_{\rightarrow}$ and $C_{\leftarrow}$ and stop when $C_{\rightarrow}$ and $C_{\leftarrow}$ lead to the conclusion of no-copying. However, we show in experiments (Section VI) that even generating the input to NRA (*i.e.*, the ordered lists) for our problem is slower than our proposed approaches.

III. Inverted Index

We first describe an important building block in our solution–the *inverted index*, which facilitates the exploration of many aforementioned opportunities for scalability improvement. Inverted indexes were originally used in Information Retrieval [14] and we describe the adaptation for copy detection.

Building the index: An important component in copy detection is to find for each pair of sources the values, not just the items, they share. We can facilitate this process with an inverted index, where each entry corresponds to a value v for a data item D, denoted by $D.v$, and contains the sources that provide v on D. Note that the presence of source S in the entry for $D.v$ guarantees that S is not present in any of the entries for $D.v', v' \neq v$.

Intuitively, we wish to first consider sharing of values that serve as strong evidence for copying, as it provides the opportunity to prune weak evidence for copying. We order the entries according to their contribution scores to $C_{\rightarrow}$ and $C_{\leftarrow}$. Note, however, that according to Eq.(3-6) the contribution from sharing $D.v$ can be different for different pairs of sources with various accuracy; we choose the maximum one, denoted by $\hat{M}(D.v)$. The next proposition shows that we can compute $\hat{M}(D.v)$ only from providers (*i.e.*, sources) with the maximum or minimum accuracy (proofs omitted to save space).

TABLE III
Inverted index for the motivating example. The two sources used to compute the contribution scores are in bold.

Value	Pr	Score	Providers
AZ.Tempe	0.02	4.59	$\mathbf{S_5}, \mathbf{S_6}$
NJ.Atlantic	0.01	4.12	$S_2, \mathbf{S_3}, \mathbf{S_4}$
TX.Houston	0.02	4.05	$\mathbf{S_2}, \mathbf{S_4}$
NY.NewYork	0.02	4.05	$S_2, \mathbf{S_3}, \mathbf{S_4}$
TX.Dallas	0.02	3.98	$\mathbf{S_6}, \mathbf{S_7}, S_8$
NY.Buffalo	0.04	3.97	$\mathbf{S_6}, \mathbf{S_7}, S_8$
FL.PalmBay	0.05	3.97	$\mathbf{S_6}, \mathbf{S_7}, S_8$
FL.Miami	0.03	3.83	$\mathbf{S_2}, \mathbf{S_3}$
AZ.Phoenix	0.95	1.62	$S_0, S_1, \mathbf{S_2}, \mathbf{S_3}, S_4$
NJ.Trenton	0.97	1.51	$S_0, S_1, \mathbf{S_7}, \mathbf{S_8}, S_9$
FL.Orlando	0.92	0.84	$S_1, \mathbf{S_4}, \mathbf{S_5}, S_9$
NY.Albany	0.94	0.43	$S_0, \mathbf{S_1}, \mathbf{S_5}$
TX.Austin	0.96	0.43	$S_0, S_1, \mathbf{S_5}, \mathbf{S_9}$

Proposition 3.1: Let $D.v$ be a value with probability $P(D.v)$. Let A_{min} be the minimum accuracy among $D.v$'s providers.

- If $A_{min} \leq \frac{1}{1+\frac{nP(D.v)}{1-P(D.v)}}$, $\hat{M}(D.v)$ is obtained by Eq.(6) when S_1 has the maximum accuracy and S_2 has the minimum accuracy;
- If $A_{min} > \frac{1}{1+\frac{nP(D.v)}{1-P(D.v)}}$ and $P(D.v) < .5$, $\hat{M}(D.v)$ is obtained by Eq.(6) when S_2 has the minimum accuracy and S_1 has the second minimum accuracy;
- Else, $\hat{M}(D.v)$ is obtained when S_1 has the minimum accuracy and S_2 has the second minimum accuracy. □

We can now formally define our specialized inverted index.

Definition 3.2 (Inverted Index): Let $\mathcal{D}$ be a set of data items and $\mathcal{S}$ be a set of sources. The *inverted index* for $\mathcal{D}$ and $\mathcal{S}$ contains a set $\mathbf{E}$ of entries, such that for each $E \in \mathbf{E}$,

1) E corresponds to a value $D_E.v_E$, where $D_E \in \mathcal{D}$ and v_E is a value provided by at least two sources on D_E;
2) E is associated with probability $P(E)$ for $D_E.v_E$ being true and with *contribution score* $C(E) = \hat{M}(D_E.v_E)$;
3) the entry contains a set $\bar{S}(E)$ of sources that provide $D_E.v_E$. □

Example 3.3: Continue with Ex.2.1. Table III shows the inverted index for the data, assuming knowledge of value probability. As an example, entry NJ.Atlantic has probability 0.01 and contribution score 4.12, computed from pair (S_4, S_3), with the highest and lowest accuracy among providers of NJ.Atlantic. Note that there is no entry for value NJ.Union, AZ.Tucson, or TX.Arlington, as each of them is provided by a single source. Also note that for any entries for the same data item, such as NJ.Atlantic and NJ.Trenton, there is no overlap between their sources.

The following properties show that processing the entries in decreasing order of their contribution scores not only helps quickly accumulate strong evidence for copying, but also helps compute the upper bound of the contribution scores, making it amenable to additional optimizations. We also show in experiments (Section VI-C) that this processing order significantly improves over random ordering.

Proposition 3.4: For each pair of sources $S_1, S_2 \in \mathcal{S}$ and index entry $E \in \mathbf{E}$, the following properties hold for $C_{\rightarrow/\leftarrow}(D_E)$.

- If $S_1, S_2 \in \bar{S}(E)$, $C_{\rightarrow}(D_E)$ is computed based on $P(D_E.v_E)$.
- If $S_1 \in \bar{S}(E), S_2 \notin \bar{S}(E)$, but they share item D_E, they provide different values on D_E and $C_{\rightarrow}(D_E) = \ln(1-s)$.
- If neither S_1 nor S_2 has appeared in any entry for D_E before entry E, $C_{\rightarrow}(D_E) \leq C(E)$. □

Optimizing with the index: With the inverted index, we can improve copy detection in three ways. First, copying is unlikely if two sources do not share any value; thus, we can skip source pairs that do not appear in the same entry.

Second, copying is also unlikely if two sources share only a few true values and we can skip them too. To simplify the computation, we consider the entries with the lowest contribution scores and denote by $\bar{E} \subseteq \mathbf{E}$ the subset of entries where $\sum_{E \in \bar{E}} C(E) < \ln \frac{\beta}{2\alpha}$. Then, for source pairs that do not share any value outside $\bar{E}$, $C_{\rightarrow} < \ln \frac{\beta}{2\alpha}$ and $C_{\leftarrow} < \ln \frac{\beta}{2\alpha}$, so $Pr(S_1 \bot S_2 | \Phi) > \frac{1}{1+\frac{\alpha}{\beta}(\frac{\beta}{2\alpha}+\frac{\beta}{2\alpha})} = .5$ and copying is unlikely. Thus, we consider a pair of sources only if they appear together in some entry outside $\bar{E}$.

Third, since each data item for which the two sources provide different values contributes the same negative score $\ln(1-s)$ (Eq.(8)), the accumulated score from these items depends only on the number of these items. This number can be derived from (1) the number of shared items, denoted by $l(S_1, S_2)$, counted at index building time (we can apply techniques for set similarity joins [1] to improve efficiency of counting), and (2) the number of shared values, denoted by $n(S_1, S_2)$, counted at index scanning time.

We next describe an algorithm, INDEX, that uses the inverted index for copy detection. Instead of considering each pair of sources, INDEX scans the inverted index in decreasing order of contribution scores and proceeds in three steps.

1) For each entry $E \in \mathbf{E} \setminus \bar{E}$ and each pair of sources $S_1, S_2 \in \bar{S}(E)$, (1) compute the contribution from E and update $C_{\rightarrow}$ and $C_{\leftarrow}$ for (S_1, S_2), and (2) maintain $n(S_1, S_2)$.
2) For each entry $E \in \bar{E}$, do the same as in Step 1 but only for pairs encountered before.
3) After scanning the whole index, for each already considered pair (S_1, S_2), (1) update scores for data items where different values are provided by adding $\ln(1-s)(l(S_1,S_2) - n(S_1,S_2))$, and (2) compute copy probability accordingly.

Proposition 3.5: Let r be the number of source pairs for which we maintain scores. INDEX takes time $O(r \cdot |\mathcal{D}|)$ and space $O(r)$, obtaining the same binary results as PAIRWISE. Note that index building has a much lower complexity: $O(|\mathcal{S}||\mathcal{D}|)$. □

The next example shows that the INDEX algorithm can considerably improve the efficiency of copy detection.

Example 3.6: Continue with the motivating example. For INDEX, the last two entries in the index (Table III) form the set $\bar{E}$ ($.43 + .43 < \ln \frac{.8}{.2} = 1.39$). There are only 26 pairs of sources that occur in entries outside $\bar{E}$; for example, S_0 and S_5 share only values in $\bar{E}$, so we do not need to consider this pair. In total INDEX needs to examine 51 shared values and have $51 * 2 + 26 * 2 = 154$ computations (2 additional computations for each pair of sources on different values) for copy detection. Note that pairwise detection requires examining 45 pairs of sources and 183 shared data items, so in total conducting $183 * 2 = 366$ computations. For this example, INDEX cuts computation by more than half. □

IV. DETECTION IN ONE ROUND

INDEX does not need to consider every pair of sources and thus can save computation; however, for each pair it considers, it still examines all shared values. The properties of the inverted index (Proposition 3.4) make it possible to terminate after we examine only a subset of shared values for a pair. First, when we observe a lot of high-score (low-probability) entries to which both sources belong, we may conclude with copying early. Second, when we observe a lot of entries to which one of the two sources belongs and a lot of high-score entries to which neither source belongs, we may conclude with no-copying early. This section describes how we can speed up copy detection by making early decisions.

A. Reducing examined shared values

Given a pair of sources S_1 and S_2, as we scan the index, we can maintain for $C_{\rightarrow}$ a maximum and a minimum score, denoted by $C_{\rightarrow}^{max}$ and $C_{\rightarrow}^{min}$ respectively; similarly we maintain $C_{\leftarrow}^{max}$ and $C_{\leftarrow}^{min}$. If the minimum scores are large enough to conclude copying, or the maximum scores are small enough to conclude no-copying, we can terminate early. For such pruning, we need to (1) decide the termination conditions and (2) compute maximum and minimum scores.

Termination conditions: We first consider binary decisions for copying. According to Eq.(2), to guarantee $Pr(S_1 \bot S_2 | \Phi) > .5$ (no-copying), we should have $1 > \frac{\alpha}{\beta}(e^{C_{\rightarrow}} + e^{C_{\leftarrow}})$; this must be true if $C_{\rightarrow}^{max} < \ln \frac{\beta}{2\alpha}$ and $C_{\leftarrow}^{max} < \ln \frac{\beta}{2\alpha}$. Thus, we define threshold $\theta_{ind} = \ln \frac{\beta}{2\alpha}$ for no-copying. On the other hand, to guarantee $Pr(S_1 \bot S_2 | \Phi) \leq .5$ (copying), we should have $1 \leq \frac{\alpha}{\beta}(e^{C_{\rightarrow}} + e^{C_{\leftarrow}})$; this must be true if $C_{\rightarrow}^{min} \geq \ln \frac{\beta}{\alpha}$ or $C_{\leftarrow}^{min} \geq \ln \frac{\beta}{\alpha}$. Thus, we define threshold $\theta_{cp} = \ln \frac{\beta}{\alpha}$ for copying. If none of the conditions is satisfied after we scan the whole index, we apply Eq.(2) to compute the probability of copying.

If we instead wish to compute real copying probabilities when it is between $[.1, .9]$ (or some other values close to 0 or 1), we can consider three different cases: $Pr(S_1 \bot S_2 | \Phi) > .9$, $Pr(S_1 \bot S_2 | \Phi) < .1$ and otherwise; we can compute the thresholds accordingly.

Maximum/Minimum score computation: When we scan each entry E, we update $C_{\rightarrow}^{min}$ and $C_{\rightarrow}^{max}$ for every pair $S_1, S_2 \in \bar{S}(E)$ as follows (similar for $C_{\leftarrow}$). First, $C_{\rightarrow}^{min}$ is obtained when the two sources share only the observed common values and no other value. Let $C_{\rightarrow}^{0}(S_1, S_2)$ be the sum of scores from observed common values. The score for each of the remaining items is negative, $\ln(1-s)$. Let $n_0(S_1, S_2)$ be the number of observed shared values and recall that $l(S_1, S_2)$ denote the number of shared items. Then,

$$C_{\rightarrow}^{min}(S_1, S_2) = C_{\rightarrow}^{0}(S_1, S_2) + (l(S_1, S_2) - n_0(S_1, S_2)) \ln(1-s). \quad (9)$$

For $C^{max}_{\rightarrow}$, we need to consider the already scanned entries containing only S_1 or S_2, or neither of them. We thus compute scores for three subsets of data items and $C^{max}_{\rightarrow}$ is the sum of their scores.

- *Data items with observed shared values:* The accumulated score from such items is $C^0_{\rightarrow}(S_1, S_2)$.
- *Data items with observed non-shared values:* According to Proposition 3.4, for each data item D shared between S_1 and S_2, if we have seen only S_1 or S_2 appearing in one of its entries, the contribution score for D is negative, $\ln(1-s)$. However, finding the precise number of such items requires recording the set of observed entries for each source and can cost a lot of space; thus, we estimate the minimum number from the numbers of observed values for S_1 and for S_2, denoted by $n(S_1)$ and $n(S_2)$ respectively. Let $\bar{N}_1$ be the overlapping items among the $n(S_1)$ items scanned for S_1–the size of $\bar{N}_1$ is roughly $n(S_1) \cdot \frac{l(S_1,S_2)}{|\bar{D}(S_1)|}$; similar for S_2, denoted by $\bar{N}_2$. The set of overlapping items among all scanned items is $\bar{N}_1 \cup \bar{N}_2$, their size satisfies $|\bar{N}_1 \cup \bar{N}_2| \geq \max\{|\bar{N}_1|, |\bar{N}_2|\} = \max\{n(S_1) \cdot \frac{l(S_1,S_2)}{|\bar{D}(S_1)|}, n(S_2) \cdot \frac{l(S_1,S_2)}{|\bar{D}(S_2)|}\}$, denoted by h. Thus, the two sources provide different values for at least $h - n_0(S_1, S_2)$ data items, and the maximum score is $(h - n_0(S_1, S_2)) \ln(1-s)$.
- *Data items we have not seen for S_1 or S_2:* There are at most $l(S_1, S_2) - h$ such items, and according to Proposition 3.4, the maximum score for each of them is the score of the next unscanned entry, denoted by M. The maximum score for this subset is thus $(l(S_1, S_2) - h) \cdot M$.

In summary, we have

$$C^{max}_{\rightarrow}(S_1, S_2) = C^0_{\rightarrow}(S_1, S_2) + (h - n_0(S_1, S_2)) \ln(1-s) + (l(S_1, S_2) - h) \cdot M. \quad (10)$$

Algorithm and analysis: From the previous analysis, we design algorithm BOUND, which proceeds in four steps.

Step I. Build the inverted index and initialize $l(S_1, S_2)$ for each pair of sources that occur together in at least one entry of the index. Initialize the active set of source pairs as $\mathbf{Q} = \emptyset$.

Step II. As we scan each entry $E \in \mathbf{E} \setminus \bar{E}$, do the following.

1) For each $S \in \bar{S}(E)$, if it is observed for the first time, set $n(S) = 1$; otherwise, increase $n(S)$ by 1.
2) For each pair $S_1, S_2 \in \bar{S}(E)$ that we observe for the first time, set $n_0(S_1, S_2) = C^0_{\rightarrow/\leftarrow}(S_1, S_2) = 0$ and add it to $\mathbf{Q}$.
3) For each pair $S_1, S_2 \in \bar{S}(E) \cap \mathbf{Q}$, do the following.
   (1) Increase $n_0(S_1, S_2)$ by 1 and update $C^0_{\rightarrow/\leftarrow}(S_1, S_2)$.
   (2) Compute $C^{min}_{\rightarrow/\leftarrow}(S_1, S_2)$. If either is above θ_{cp}, conclude copying and remove the pair from $\mathbf{Q}$.
   (3) Compute $C^{max}_{\rightarrow/\leftarrow}(S_1, S_2)$. If both are below θ_{ind}, conclude no-copying and remove the pair from $\mathbf{Q}$.

Step III. As we scan each entry $E \in \bar{E}$, do 1 and 3 in Step II (but only for pairs encountered before).

Step IV. After we scan the index, for each pair $(S_1, S_2) \in \mathbf{Q}$, we have $n_0(S_1, S_2) = n(S_1, S_2)$, so $C_{\rightarrow} = C^{min}_{\rightarrow}$ (similar for $C_{\leftarrow}$). If both $C_{\rightarrow}$ and $C_{\leftarrow}$ are below θ_{ind}, conclude no-copying; otherwise, apply Eq.(2).

Proposition 4.1: Let r be the number of source pairs that share values, and e be the maximum number of shared entries we process for each pair before concluding. BOUND takes time $O(r \cdot e)$ and space $O(r + |\mathcal{S}|)$. □

As BOUND estimates the number of observed overlapping data items (h) in computing $C^{max}_{\rightarrow}$, the result may be different from pairwise detection. However, the computation of h and the use of M in Eq.(10) make the upper bounds already loose, so the decisions are rarely different, as we observed in our experiments (Section VI). Note that e can be much smaller than $|\mathcal{D}|$, so BOUND often significantly reduces the total number of data items we consider in copy detection. However, computing upper and lower bounds of contribution scores introduces an overhead, so BOUND may not always save computation for each pair of sources, as illustrated next.

Example 4.2: Continue with Ex.3.6. We have $\theta_{cp} = \ln \frac{.8}{.1} = 2.08$ and $\theta_{ind} = \ln \frac{.8}{.2} = 1.39$.

First consider pair (S_2, S_3); recall that they share 4 values (including 3 false ones) and copying is likely. We see them first at entry NJ.Atlantic where $C^0_{\rightarrow/\leftarrow}(S_2, S_3) = 3.89$. By Eq.(9) we compute $C^{min}_{\rightarrow/\leftarrow} = 3.89 - 1.6 * (5-1) = -2.51$. For maximum scores, $h = 1$ so by Eq.(10) $C^{max}_{\rightarrow/\leftarrow} = 3.89 + 0 + 4.05 * (5-1) = 20.09$. We see this pair again at entry NY.NewYork. We update $C^0_{\rightarrow}(S_2, S_3) = 3.89 + 3.86 = 7.75$, so $C^{min}_{\rightarrow} = 7.75 - 1.6*(5-2) = 2.95 > 2.08 = \theta_{cp}$ and we can conclude copying for the pair. While INDEX considers 4 shared values for them and conducts $4*2+2 = 10$ computations, BOUND considers only 2 shared values and conducts $4+1 = 5$ computations.

Now consider pair (S_0, S_1); recall that they share 4 true values and no-copying is likely. When we see them at the third shared entry NY.Albany, we have $C^0_{\rightarrow/\leftarrow}(S_2, S_3) = .01*3 = .03$, so $C^{max}_{\rightarrow/\leftarrow} = .03 + 0 + 0.43*(4-3) = .46 < 1.39 = \theta_{ind}$; we can then conclude no-copying. Thus, BOUND considers 3 shared values and conducts $4*3 = 12$ computations. However, INDEX considers 4 shared values for them but conducts only $4*2+2 = 10$ computations, fewer than BOUND.

In total BOUND considers 26 pairs, 33 shared values, and requires 116 computations. It considers 18 fewer shared values and conducts 38 fewer computations than INDEX. □

B. Reducing computation

Although BOUND reduces the number of shared values we consider, it introduces the overhead of computing $C^{min}_{\rightarrow/\leftarrow}$ and $C^{max}_{\rightarrow/\leftarrow}$. Actually, we do not need to maintain $C^{min}_{\rightarrow/\leftarrow}$ and $C^{max}_{\rightarrow/\leftarrow}$ each time we scan a shared entry; we only need to do so when termination is likely. This can further reduce computation for BOUND.

First, suppose after scanning entry E, we compute $C^{min}_{\rightarrow} < \theta_{cp}$ and $C^{min}_{\leftarrow} < \theta_{cp}$ for source pair (S_1, S_2). The next shared value can increase $C^{min}_{\rightarrow}$ and $C^{min}_{\leftarrow}$ by at most $M - \ln(1-s)$ (recall that M denotes the contribution score of the next entry). Thus, we do not need to re-compute $C^{min}_{\rightarrow/\leftarrow}$ until we have observed at least $T^{min} = \lceil \frac{\theta_{cp} - \max\{C^{min}_{\rightarrow}, C^{min}_{\leftarrow}\}}{M - \ln(1-s)} \rceil$ shared values.

Similarly, suppose after we scan E, we compute $C^{max}_{\rightarrow} \geq \theta_{ind}$ or $C^{max}_{\leftarrow} \geq \theta_{ind}$ for sources (S_1, S_2). A new data item on which S_1 and S_2 provide different values would reduce $C^{max}_{\rightarrow}$ and $C^{max}_{\leftarrow}$ by $M - \ln(1-s)$, so we would resume computing maximum scores when we see $T_0^{max} = \lceil \frac{\max\{C_{\rightarrow}^{max}, C_{\leftarrow}^{max}\}-\theta_{ind}}{M-\ln(1-s)} \rceil$ more different values. At entry E we have already seen $h - n_0(S_1, S_2)$ different values, so we need to see in total $T_0^{max} + h - n_0(S_1, S_2)$ different values. We do not re-compute $C_{\rightarrow/\leftarrow}^{max}$ until $n(S_1) \geq T_1^{max} = \lceil (T_0^{max} + h - n_0(S_1, S_2)) \cdot \frac{|\bar{D}(S_1)|}{l(S_1,S_2)} \rceil$ or $n(S_2) \geq T_2^{max} = \lceil (T_0^{max} + h - n_0(S_1, S_2)) \cdot \frac{|\bar{D}(S_2)|}{l(S_1,S_2)} \rceil$.

Example 4.3: Consider a pair of sources S_1 and S_2 that share 101 data items. Suppose again that $\ln(1 - s) = -1.6, \theta_{cp} = 2.08, \theta_{ind} = 1.39$. Suppose the first shared item we have observed between S_1 and S_2 has contribution $C_{\rightarrow/\leftarrow}^0 = 5$. Then $C_{\rightarrow/\leftarrow}^{min} = 5-(101-1)*1.6 = -155 < 2.08$. Suppose $M = 4$, then $T^{min} = \lceil \frac{2.08-(-155)}{4-(-1.6)} \rceil = 29$, so we do not compute $C_{\rightarrow/\leftarrow}^{min}$ until we have observed 29 other shared values.

For maximum scores, suppose we have not seen other entries containing S_1 or S_2 yet, so $h = 1$ and $C_{\rightarrow/\leftarrow}^{max} = 5+0+(101-1)*4 = 405$. Then, $T_0^{max} = \lceil \frac{405-1.39}{4-(-1.6)} \rceil = 72$. Suppose $\frac{l(S_1,S_2)}{|\bar{D}(S_1)|} = \frac{l(S_1,S_2)}{|\bar{D}(S_2)|} = .8$, then $T_1^{max} = T_2^{max} = \lceil \frac{(72+1-1)}{.8} \rceil = 90$. So we do not compute $C_{\rightarrow/\leftarrow}^{max}$ until $n(S_1) \geq 90$ or $n(S_2) \geq 90$. □

We can improve BOUND accordingly and the result is called BOUND+. Note that BOUND+ has the same asymptotic complexity as BOUND but in practice can save a lot of computation.

Finally, intuitively only when two sources share a lot of data items, we are likely to significantly reduce the number of considered shared values and compensate for the extra cost for bound computation. We can thus apply INDEX for pairs of sources that share only a few data items and apply BOUND+ for the rest of the pairs. We call the resulting algorithm HYBRID. Our experiments (Section VI) show that HYBRID can further reduce computation and copy-detection time.

V. INCREMENTAL DETECTION

Section IV considered copy detection in one single round; this section considers the iterative process. Our observation is that although there can be changes on value probability and source accuracy from round to round, after the second round the changes are typically small and seldom change our copy-detection decisions. A natural thought for improving scalability is to detect copying incrementally after the second round. We base our discussions on the HYBRID algorithm.

A. Overview

Both changes in value probability $P(D.v)$ and changes in source accuracy $A(S)$ can affect copy detection. We distinguish big changes and small changes. If a pair of sources contains a source with big accuracy change, we need to recompute the probability of copying. For the rest of the pairs, we can incrementally update the contribution scores. We *update scores on big-change entries first; only for pairs whose score changes can lead to an opposite decision on copying, we would further consider small-change entries.* The challenge is to reduce the number of entries we consider whenever possible but still reach the same copying decision.

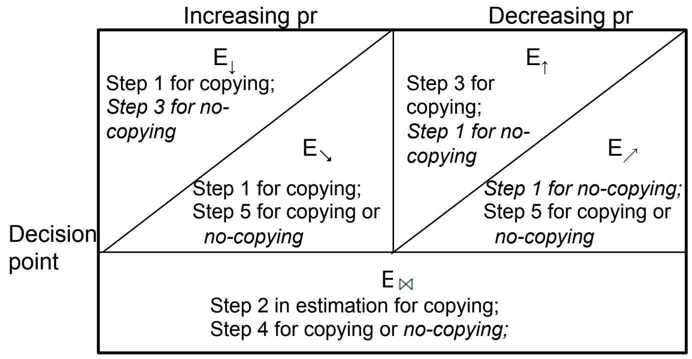

Fig. 1. Entry categories. The steps for no-copying pairs are in italic font.

We denote by $P_{old}(D.v)$ the probability used previously in score contribution. Note that the recorded probability may not be the one from the previous round, but can be from some earlier round when we do the last re-computation. For an entry $D.v$, we consider the change on $\hat{M}(D.v)$ rather than on $P(D.v)$, since a small change of the latter may cause a big change of the former, which eventually matters in score computation. To separate the change of value probability from that of source accuracy, we compute $\hat{M}(D.v)$ on the same two accuracies used in the round with $P_{old}(D.v)$. Finally, to classify big and small changes, we have a threshold ρ. We can either set a default one, or order the changes in decreasing order, choose the maximum gap between two consecutive changes, and set ρ to the change above the gap.

B. Source pairs with copying

We first explain our strategy for source pairs where we concluded with copying in the previous round. Recall that for a pair of sources, we may make our decision before reaching the end of the index; we call the last entry we considered the *decision point*. Accordingly, we can categorize the shared entries between this pair of sources into five categories (see Fig.1): (1) $\bar{E}_{\downarrow}$: big-change entries whose contribution scores decrease (the probabilities of the entries increase) before the decision point; (2) $\bar{E}_{\searrow}$: small-change entries whose scores decrease before the decision point; (3) $\bar{E}_{\uparrow}$: big-change entries whose scores increase before the decision point; (4) $\bar{E}_{\nearrow}$: small-change entries whose scores increase before the decision point; and (5) $\bar{E}_{\bowtie}$: shared entries after the decision point.

Among them, entries in $\bar{E}_{\downarrow}$ and $\bar{E}_{\searrow}$ would decrease the scores and may even change our decision. The high-level idea for our algorithm is to first consider the decreases, and then compensate for score loss with the increases from the other categories until the scores are once again above the threshold θ_{cp}. In the latter process, we consider big increases (from $\bar{E}_{\uparrow}$) first, and small ones (from $\bar{E}_{\nearrow}$) last. We next describe how we update $\hat{C}_{\rightarrow}$ (resp. $\hat{C}_{\leftarrow}$).

Preparation step: As a preparation, in the round when we conduct copy-detection from scratch, we maintain for each pair of sources the number of shared values before the decision point and that after the decision point (the latter can be denoted by $|\bar{E}_{\bowtie}|$). We then compute the final score of this round, denoted by $\hat{C}_{\rightarrow}$ (resp. $\hat{C}_{\leftarrow}$), as $\hat{C}_{\rightarrow} = C_{\rightarrow}^{min} - |\bar{E}_{\bowtie}| \cdot \ln(1-s)$.

Note that the computation of $C^{min}_{\rightarrow}$ assumes that there is no value shared after the decision point and applies penalty $\ln(1-s)$ to each such shared value; the computation of $\hat{C}_{\rightarrow}$ removes this penalty but does not apply their real (positive) contribution; thus, $C^{min}_{\rightarrow} < \hat{C}_{\rightarrow} < C_{\rightarrow}$. We use $\hat{C}_{\rightarrow}$ and $\hat{C}_{\leftarrow}$ as the starting scores for the next round.

Step 1 ($\bar{E}_{\downarrow} \cup \bar{E}_{\searrow}$): Each entry $E \in \bar{E}_{\downarrow}$ may significantly reduce $\hat{C}_{\rightarrow}$. We update $\hat{C}_{\rightarrow}$ by replacing the old score on E, computed by the old value probability and source accuracy, with the new one computed by the new value probability and source accuracy. Each entry $E \in \bar{E}_{\searrow}$ will reduce $\hat{C}_{\rightarrow}$ slightly. Instead of updating the change for each such entry, we use the maximum change, denoted by Δ_{ρ}, which we estimate from the entry with the largest score decrease below ρ. We decrease $\hat{C}_{\rightarrow}$ by $\Delta_1 = \Delta_{\rho} \cdot |\bar{E}_{\searrow}|$. If after these changes $\max\{\hat{C}_{\rightarrow}, \hat{C}_{\leftarrow}\} \geq \theta_{cp}$ still holds, we can stop; otherwise, we conduct Step 2-5 and stop once $\max\{\hat{C}_{\rightarrow}, \hat{C}_{\leftarrow}\} \geq \theta_{cp}$.

Step 2 ($\bar{E}_{\bowtie}$): In case the new score is below θ_{cp}, we look for data entries that can increase them back to above the threshold. Consider the shared entries after the decision point. Each of them should have a minimum contribution score, which can be estimated on the last entry in the index. We denote this score by m and increase $\hat{C}_{\rightarrow}$ by $\Delta_2 = m \cdot |\bar{E}_{\bowtie}|$.

Step 3 ($\bar{E}_{\uparrow}$): Each entry $E \in \bar{E}_{\uparrow}$ can significantly increase $\hat{C}_{\rightarrow}$ and compensate for the loss. We update $\hat{C}_{\rightarrow}$ by replacing the old score on E with the new one.

Step 4 ($\bar{E}_{\bowtie}$): Each $E \in \bar{E}_{\bowtie}$ may also increase $\hat{C}_{\rightarrow}$ a lot. We (1) increase $\hat{C}_{\rightarrow}$ by $C_{\rightarrow}(D_E)$, and (2) subtract m from $\hat{C}_{\rightarrow}$ and Δ_2 to remove our previous estimation on E.

Step 5 ($\bar{E}_{\searrow} \cup \bar{E}_{\nearrow}$): Now the only entries not updated are those with small changes before the decision point. For each $E \in \bar{E}_{\searrow} \cup \bar{E}_{\nearrow}$, we (1) update $\hat{C}_{\rightarrow}$ by replacing the old score on E with the new one, and (2) if $E \in \bar{E}_{\searrow}$, increase $\hat{C}_{\rightarrow}$ by Δ_{ρ} and subtract Δ_{ρ} from Δ_1 to remove our previous estimation on E.

Final step: We remove the estimation, recording $\hat{C}_{\rightarrow} + \Delta_1 - \Delta_2$ as the precise $\hat{C}_{\rightarrow}$ (resp. $\hat{C}_{\leftarrow}$) for the starting point of the next round. We also update the new decision point if needed. If the condition $\max\{\hat{C}_{\rightarrow}, \hat{C}_{\leftarrow}\} \geq \theta_{cp}$ is not satisfied until the end, we apply Bayesian analysis to decide if we need to change our decision to no-copying.

These steps can be combined into three passes of index scanning. The first pass conducts Steps 1 and 2, the second pass conducts Steps 3 and 4, and the third pass conducts Step 5. Figure 1 summarizes the algorithm. We next illustrate the idea using an example.

Example 5.1: Recall that for the motivating example there are five rounds before convergence (Table II). Consider incremental detection at Round 3; it considers value probabilities from Round 1 as old ones and those from Round 2 as new ones. Table IV shows the inverted index. We set $\rho_V = 1$, so there are 2 entries with big score changes (in italics), corresponding to values for NY.

First consider (S_2, S_3). In Round 2 it terminates at entry NY.NewYork, having scores of $C^{min}_{\rightarrow/\leftarrow} = 4.73$, sharing 3 values before the decision point and 1 value after the point. Thus, $\hat{C}_{\rightarrow/\leftarrow} = 4.73 + 1.6 = 6.33$. Among the shared entries before decision, 2 have small increases and 1 has big increase.

TABLE IV
PARTIAL INVERTED INDEX USED IN ROUND 3. ONLY SOURCE $S_0 - S_4$ AND THEIR PROVIDED VALUES ARE SHOWN.

Value	Providers	Pr	ΔScore	Cat.	Score@R_3
TX.Houston	S_2, S_4	$.04 \rightarrow .03$	.17	$\nearrow$	3.97
FL.Miami	S_2, S_3	$.05 \rightarrow .03$	.21	$\nearrow$	3.83
NJ.Atlantic	S_2, S_3, S_4	$.07 \rightarrow .03$	.39	$\nearrow$	3.96
NY.Albany	S_0, S_1	$.07 \rightarrow .77$	-2.49	$\downarrow$	.52
NJ.Trenton	S_0, S_1	$.9 \rightarrow .95$	-.12	$\searrow$	1.31
NY.NewYork	S_2, S_3, S_4	$.84 \rightarrow .16$	1.69	$\uparrow$	3.17
AZ.Phoenix	$S_0 - S_4$	$.94 \rightarrow .95$	-.01	$\searrow$	1.45
FL.Orlando	S_1, S_4	$.9 \rightarrow .92$	-.01	$\searrow$	.78
TX.Austin	S_0, S_1	$.9 \rightarrow .93$	-.01	$\searrow$	.51

Thus, the score is not decreased and we can terminate for this pair without further examination.

Now consider (S_0, S_1). In Round 2 it terminates at the last entry, having scores $C_{\rightarrow} = 1.15, C_{\leftarrow} = 1.66$, and sharing 4 values before the decision point; recall that $\theta_{cp} = 2.08$ and $\theta_{ind} = 1.39$, so we need to apply Eq.(2), computing $P(S_0 \perp S_1 | \Phi) = .32$ and deciding copying. Among the 4 shared values, NY.Albany has big score decrease and the other three have small decreases. The contribution scores for $C_{\rightarrow/\leftarrow}$ from NY.Albany were $.44/1.53$ in Round 2 and are $.24/.09$ now. The largest score difference for the other three items is .015, computed from NJ.Trenton, which has the largest score decrease among small-change entries. Accordingly, we have $\hat{C}_{\rightarrow} = 1.15 + (.24 - .44) - .015 * 3 = .9 < \theta_{cp}$, and $\hat{C}_{\leftarrow} = 1.66 + (.09 - 1.53) - .015 * 3 = .17 < \theta_{cp}$; thus, we may change our decision. Since $\bar{E}_{\uparrow} = \bar{E}_{\bowtie} = \emptyset$, we cannot compensate for the loss of the score. We next reconsider the items with small changes and compute precise scores $\hat{C}_{\rightarrow} = .95 < \theta_{ind}, \hat{C}_{\leftarrow} = .20 < \theta_{ind}$. Therefore, we change our decision for this pair to no-copying. □

C. Source pairs with no-copying

We handle source pairs with no-copying in a similar way. For such pairs, entries in $\bar{E}_{\uparrow}$ and $\bar{E}_{\nearrow}$ would increase the scores and may change our decision, while entries in $\bar{E}_{\downarrow}$ and $\bar{E}_{\searrow}$ would decrease the scores and compensate for the score increase, so we change the order of considering them. Also, Step 2 does not apply for no-copying pairs since we actually need to *reduce* the scores to compensate for its increase. Again, the steps are summarized in Figure 1. In addition, we compute $\hat{C}_{\rightarrow/\leftarrow}$ by Eq.(10) with two changes. First, we use the real number of different values obtained from bookkeeping rather than the estimated one. Second, in case the maximum score M has a big change, we update $\hat{C}_{\rightarrow/\leftarrow}$ upfront in each round.

Example 5.2: Continue with Ex.5.1 and now consider no-copying pair (S_0, S_2). In Round 2 it terminates at entry AZ.Phoenix, having scores $C^{max}_{\rightarrow} = -4.75, C^{max}_{\leftarrow} = -4.3$, sharing 1 value before decision and 0 value after decision. The shared value is in category $\bar{E}_{\searrow}$, so Step 1 does not change the score and we can terminate with the same decision. □

The final algorithm, INCREMENTAL, updates scores for all source pairs in three passes of index scanning. It requires more space for book-keeping across rounds, but in practice it recomputes scores for much fewer entries.

Proposition 5.3: Let r be the number of source pairs that share values, and e' be the maximum number of shared entries

TABLE V
OVERVIEW OF DATA SETS.

	#Srcs	#Items	#Dist-values	#Index-entries
Book-CS	894	2,528	14,930	7,398
Stock-1day	55	16,000	104,611	40,834
Book-full	3,182	147,431	162,961	48,683
Stock-2wk	55	160,000	915,118	405,537

we process for each pair. INCREMENTAL takes time $O(re')$ and space $O(|\mathcal{E}| + r + |\mathcal{S}|)$ for a single round. □

Example 5.4: In Rounds 3-5 for our example, BOUND+ takes 102 computations for each round, while INCREMENTAL reduces it to 54, 29 and 0 respectively. The total number of computations for INCREMENTAL is 73% lower than that for BOUND+. □

VI. EXPERIMENTAL RESULTS

This section presents experimental results validating the efficiency and effectiveness of the inverted index and algorithms proposed in this paper. We show that among the strategies we have proposed, the inverted index can improve the efficiency by one to two orders of magnitude and obtain exactly the same results; pruning and incremental detection together can improve the efficiency by nearly one order of magnitude and obtain very similar results; and a careful sampling can improve the efficiency by orders of magnitude without sacrificing the quality of the results too much.

A. Experiment settings

Data: We experimented on four data sets[5]; Table V provides an overview. Two data sets were crawled from an online bookstore aggregator *AbeBooks.com*: *Book-CS* contains 894 sources (*i.e.*, book stores), 1265 CS books, and 2528 data items including the title and author list of each book (there are missing values for some books); on average 5.9 conflicting values are provided for each data item. *Book-full* contains 3182 sources, 81,352 books of all categories, and 147,431 data items; on average 1.1 conflicting values are provided for each data item. A gold standard for *Book-CS* contains author lists verified from book title pages for 100 randomly selected books.

The other two data sets were crawled from 55 Deep Web sources on 16 attributes of 1000 stocks. *Stock-1day* includes the data on 7/7/2011 and *Stock-2wk* includes the data from 7/1/2011 to 7/14/2011. The former contains $16 * 1000 = 16,000$ data items and on average 6.5 conflicting values are provided for each data item; the latter contains $16,000 * 10 = 160,000$ data items and on average 5.7 conflicting values are provided for each item. A gold standard for *Stock-1day* contains the voting results on the 100 NASDAQ symbols and 100 other randomly selected symbols from 5 popular financial websites: *NASDAQ, Yahoo! Finance, Google Finance, MSN Money,* and *Bloomberg*.

The four data sets have very different features. *Book-full* and *Stock-2wk* contain a large number of data items. *Book-CS* and *Book-full* contain a large number of data sources; however, some sources contain only a few data items (*e.g.*, 85% sources in *Book-CS* each covers at most 1% books). *Stock-1day* and *Stock-2wk* contain much fewer sources, but each source has a much higher coverage (*e.g.*, 80% sources each covers over half of the data items).

Implementation: We implemented various methods for copy detection and describe them as follows.

- PAIRWISE examines each pair of sources as described at the beginning of Section II-B [6].
- SAMPLE1 randomly samples 1% of data items on *Stock-2wk* and 10% on the other data sets, then applies PAIRWISE on the sampled data.
- SAMPLE2 is different from SAMPLE1 on the two *Book* data sets. It considers each data set as a table where each row represents a source and each column represents a data item. It randomly samples data items (columns) until the number of non-empty cells reaches 65% on *Book-CS* and 24% on *Book-full* (we explain the need for such sampling rates shortly).
- INDEX implements algorithm INDEX (Section III).
- BOUND and BOUND+ each applies the corresponding algorithm (Section IV) for each round.
- HYBRID applies INDEX for a pair of sources that share at most 16 data items[6] and applies BOUND+ for other pairs in each round (end of Section IV)
- INCREMENTAL applies HYBRID in the first two rounds and Algorithm INCREMENTAL (Section V) in later rounds. [7] It sets ρ to .2 for source accuracy and to 1.0 for value probability according to observations of the largest gaps on differences of changes.
- SCALESAMPLE applies INCREMENTAL on a sampled data set, where we sample 1% of data items on *Stock-2wk* and 10% on the other data sets, and guarantee sampling at least $N = 4$ data items from each source.
- FAGININPUT generates the input to Fagin's NRA algorithm as described at the end of Section II-B.

In addition, we used the truth-finding algorithm in [6], which considers both copying and source accuracy. We plugged in the aforementioned copy-detection algorithms.

We implemented the algorithms in Java on a Windows machine with Intel Core i5 processor (3.2GHz, 4MB cache, 4.8 GT/s QPI, 8GB memory).

Measures: We measure three aspects of different methods.
Efficiency: We measure efficiency by (1) the number of computations in copy detection (as described in the examples in Sections III-V), and (2) the execution time.

Copy-detection correctness: We examined how the various methods for improving scalability may hurt the results of copy detection; thus, we compared their results with those of PAIRWISE. *Precision* measures among the output copying pairs, what fraction is also output by PAIRWISE; *Recall* measures

[5]The data are at *http://lunadong.com/fusionDataSets.htm*.

[6]We observe empirically that when two sources share fewer than 16 data items, INDEX conducts fewer computations than BOUND+ on average.

[7]Empirically we found that copy-detection and truth-finding results vary a lot in the first two rounds in general, so applying INCREMENTAL in the second round would not save much.

TABLE VI
COPY-DETECTION AND TRUTH-DISCOVERY QUALITY OF VARIOUS ALGORITHMS. EXCEPT FUSION ACCURACY, ALL MEASURES ARE COMPUTED BY COMPARING WITH RESULTS OF PAIRWISE. SAMPLE2 OBTAINS THE SAME RESULTS AS SAMPLE1 ON *Stock* DATA.

Method	*Book-CS*						*Stock-1day*					
	Copy detection			Truth discovery			Copy detection			Truth discovery		
	Prec	Rec	F-msr	Accu	Fusion diff	Accu var	Prec	Rec	F-msr	Accu	Fusion diff	Accu var
PAIRWISE	-	-	-	.890	-	-	-	-	-	.897	-	-
SAMPLE1	.691	.165	.264	.870	.070	.127	.967	.945	.956	.896	.008	.001
SAMPLE2	.886	.696	.779	.880	.029	.089	.967	.945	.956	.896	.008	.001
INDEX	1	1	1	.890	0	0	1	1	1	.897	0	0
HYBRID	.990	.980	.985	.890	.015	.039	1	.970	.985	.897	.002	.001
INCREMENTAL	.985	.975	.980	.890	.015	.037	.993	.947	.969	.897	.003	.001
SCALESAMPLE	.930	.841	.882	.890	.029	.055	.970	.927	.948	.897	.008	.001

TABLE VII
EXECUTION TIME AND THE TIME IMPROVEMENT COMPARED WITH THE PREVIOUS METHOD (SAMPLE1, SAMPLE2, INDEX COMPARING WITH PAIRWISE; OTHERS COMPARING WITH THE METHOD IN THE ABOVE ROW). SAMPLE2 OBTAINS THE SAME RESULTS AS SAMPLE1 ON *Stock* DATA.

Method	*Book-CS*		*Stock-1day*		*Book-full*		*Stock-2wk*	
	Time (s)	Improvement	Time (s)	Improvement	Time (s)	Improvement	Time (s)	Improvement
PAIRWISE	321	-	306	-	11536	-	3408	-
SAMPLE1	3.2	99%	16.2	95%	278	98%	55	98%
SAMPLE2	32	90%	16.2	95%	684	94%	55	98%
INDEX	1.6	99.5%	25.0	92%	47.7	99.6%	573	83%
HYBRID	1.2	24%	15.8	37%	47.2	2%	443	23%
INCREMENTAL	0.4	65%	6.9	56%	7.9	83%	127	72%
SCALESAMPLE	0.3	25%	0.7	90%	3.8	52%	1.4	99%
Total Improvement		**99.91%**		**99.8%**		**99.97%**		**99.96%**

among the output copying pairs by PAIRWISE, what fraction is output by the specific method; *F-measure* is computed by $\frac{2 \cdot precision \cdot recall}{precision + recall}$.

Truth-finding correctness: We also examined how the copy-detection results may affect truth finding. We report three measures: (1) *Fusion accuracy* measures the fraction of correct truth-finding results among all data items in the gold standard; (2) *Fusion difference* measures the fraction of truth-finding results different from those when applying PAIRWISE; and (3) *Accuracy variance* measures the average difference of the source accuracies we compute when applying PAIRWISE and the specific copy-detection method.

We report efficiency on all data sets and other results only on the two small data sets *Book-CS* and *Stock-1day*.

B. Performance overview

We first compare the various methods on each data set. Table VI reports copy-detection and truth-finding correctness, and Table VII reports execution time.

First, naive sampling (SAMPLE1 and SAMPLE2) did improve the efficiency a lot, but not as much as INCREMENTAL and SCALESAMPLE. Indeed, on the *Stock* data sets they are one order of magnitude slower than SCALESAMPLE and on the *Book* data sets they are even slower than INDEX. In addition, SAMPLE1 obtains very low F-measure on copy detection for *Book-CS*, where a lot of data sources provide only a few books, so a random sampling can lead to inaccurate decisions.

Second, our proposed methods for improving scalability work very well. Without sampling, INCREMENTAL finished in about 2 minutes for *Stock-2wk* and seconds for other data sets. In particular, the use of the inverted index in itself (INDEX) on average reduced execution time by 94% and obtains exactly the same results for copy detection and truth discovery as PAIRWISE. It works especially well for the two *Book* data sets (improving by two orders of magnitude) because a lot of source pairs (95.6% on average) do not share any data item and need not to be considered at all. Also, we observe from Table V that on average only 42% values are provided by multiple sources and so are indexed. Pruning (HYBRID) on average reduced execution time further by 21% and changed copy-detection and truth-discovery results very slightly. Incremental detection (INCREMENTAL) on average reduced execution time further by 69% and also changed the results very slightly. The two enhancements together reduced execution time by 77% on average and sacrificed precision and recall of copy detection by at most 5%; they also changed results of truth-discovery very slightly, by up to 1.5%. We observed from our experiments that indexing costs 57% of execution time in INCREMENTAL, but it spent only .9% execution time of PAIRWISE and significantly improves scalability, so is worthwhile.

Third, sampling helps with a small sacrifice on effectiveness: SCALESAMPLE finished within a few seconds for all data sets with reasonable F-measure for copy detection and very similar results for truth discovery. On the *Stock* data sets, the improvement corresponds to the sampling rate: 90% for *Stock-1day* (sampling rate .1) and 99% for *Stock-2wk* (sampling rate .01); in addition, the F-measure and fusion results are very similar to INCREMENTAL, which does not do sampling. On the *Book* data sets, the efficiency was improved but not as much (by 25% and 52% respectively), and the F-measure of copy detection drops. Recall that in these two data sets there are a lot of low-coverage sources, making sampling much harder. Indeed, we ended up sampling 49% data items for *Book-*

CS and 19% items for *Book-full*. However, we obtain much higher F-measure than SAMPLE1 and SAMPLE2, showing effectiveness of sampling at least $N = 4$ data items. Last, we note that sampling in itself has a very small overhead for small data sets (5% of execution time on average) but a larger overhead for large data sets (37% on average); this is because checking whether each source covers N sampled data items takes longer time for large data sets.

Finally, Table X shows the execution time ratio of our methods versus FAGININPUT. FAGININPUT has two drawbacks. First, it has to compute the contribution scores from each shared value for *each source pair*; thus, HYBRID is 18% faster than FAGININPUT on average for a single round. Second, it is not clear how to generate the input lists *incrementally in later rounds*; thus, INCREMENTAL is 75% faster than FAGININPUT on average for all rounds.

C. Single-round algorithms

We next examine single-round algorithms in more detail. We first compare INDEX, BOUND, BOUND+, and HYBRID on their numbers of computations (for all rounds together) and copy-detection time (see Figure 2). We have three observations. First, for three out of four data sets BOUND conducts more computations and finished in longer time than INDEX. Although it reduces the number of data items for consideration, it introduces a big overhead for computing the minimum and maximum scores. Second, BOUND+ speeds up copy detection significantly: on average it reduces the number of computations by 55% and saves copy-detection time by 37% over BOUND. Third, HYBRID further saves 20.3%, 22.9% computations and 4.6%, 11.6% copy-detection time on *Book-CS* and *Book-full* respectively. It does not make a difference on the two *Stock* data sets, because there each pair of sources share a lot of data items.

We then examined various orders of processing entries in the inverted index: RANDOM processes the entries randomly; BYPROVIDER processes the entries in increasing order of the number of providers (*i.e.*, sources); and BYCONTRIBUTION processes the entries in decreasing order of contribution (proposed in this paper). Figure 3 shows the execution time of each of the latter two compared with random ordering for BOUND and HYBRID. We observed that BYCONTRIBUTION is the fastest among the three ordering schemes. When we apply BOUND, it improves over RANDOM by 12% on average and by 24% for *Stock-1day*; it improves over BYPROVIDER by 7% on average and by 22% for *Stock-1day*. When we apply HYBRID, which skips many computations by setting up a timer, the benefit of BYCONTRIBUTION is less evident but it is still the fastest. We also note that although BYPROVIDER is better than RANDOM, it may process some true but not widely provided values towards the beginning and so can incur more computation than BYCONTRIBUTION.

D. Incremental algorithms

To understand how incremental detection improves efficiency, we show in Table VIII the execution time ratio of INCREMENTAL versus HYBRID round by round. Indeed, incremental detection saves execution time significantly: on average it improves over HYBRID by 97% for indexing, 52% for copy detection, and 93.5% in total. We also show in Table VIII how many pairs terminate at each of the three passes We observe that in the first pass 86% pairs terminate for *Book-full* and over 98% pairs terminate for other data sets. This verifies our intuition and explains why INCREMENTAL can save computation significantly.

TABLE VIII
EXECUTION TIME RATIO OF INCREMENTAL VS HYBRID, PERCENTAGE OF PAIRS TERMINATED AT EACH PASS OF INCREMENTAL DETECTION

	Book-CS	*Stock-1day*	*Book-full*	*Stock-2wk*
Round 3	14.0%	6.9%	3.1%	7.3%
Round 4	12.2%	6.8%	3.3%	4.7%
Round 5	10.2%	6.1%	3.4%	4.4%
Round 6	9.6%	6.4%	3.3%	4.9%
Round 7	10.2%	-	3.7%	-
Round 8	9.6%	-	3.1%	-
Round 9	-	-	3.0%	-
Pass 1	99%	98%	86%	99%
Pass 2	0	1%	4%	0
Pass 3	1%	1%	10%	1%

TABLE IX
COMPARING DIFFERENT SAMPLING METHODS.

	Book-CS			*Stock-1day*		
Method	Prec	Rec	F-msr	Prec	Rec	F-msr
SCALESAMPLE	.92	.84	.88	.98	.94	.96
BYITEM	.85	.56	.67	.98	.94	.96
BYCELL	.89	.70	.78	.98	.94	.96

TABLE X
EXECUTION-TIME RATIO W.R.T. FAGININPUT.

	Book-CS	*Stock-1day*	*Book-full*	*Stock-2wk*
HYBRID	.87	.76	.99	.67
INCREMENTAL	.30	.27	.22	.19

E. Sampling

Finally, we compare our sampling strategy, called SCALESAMPLE, with sampling rate 10%, with two naive sampling strategies as described in SAMPLE1 and SAMPLE2, which we call BYITEM and BYCELL respectively; here we apply INCREMENTAL on all samples. To ensure a fair comparison, the sampling rate for BYITEM is decided by the percentage of sampled data items in SCALESAMPLE, and the sampling rate for BYCELL (and SAMPLE2) is decided by the percentage of sampled cells in SCALESAMPLE. For example, SCALESAMPLE sampled 49% data items and 65% cells on *Book-CS*, so we applied a sampling rate of 49% for BYITEM and 65% for BYCELL; SCALESAMPLE sampled 10% data items and 10% cells on *Stock-1day*, so BYITEM and BYCELL applied the same sampling rate (10%). Table IX shows the quality of copy-detection results compared to applying INDEX. The three sampling methods obtain the same results on *Stock-1day* since the sources all have a high coverage in that data set; SCALESAMPLE obtains the best results on *Book-CS* even though it selects the same number of data items as BYITEM and the same number of cells as BYCELL, since it guarantees

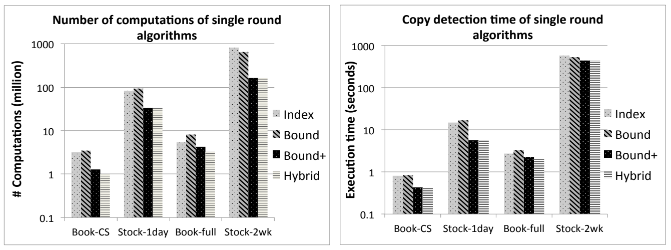

Fig. 2. Single-round algorithms.

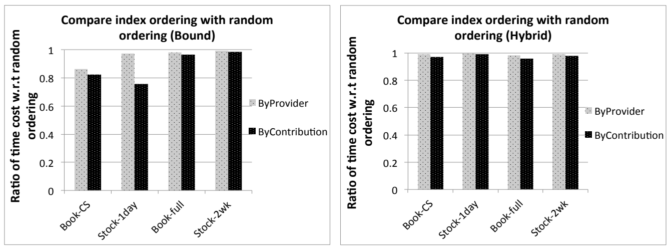

Fig. 3. Different index ordering.

that we select at least $N = 4$ data items from each source when possible.

VII. Related Work

Copy detection has been studied recently in [2], [5], [6], [7], [15]. Prior work has focused on effectiveness rather than efficiency of detection. As our experiments show, our algorithms can improve the efficiency over state-of-the-art algorithms (PAIRWISE) by three or more orders of magnitude, without sacrificing the quality much.

Improving scalability of copy detection has been intensively studied for text documents and software programs (surveyed in [8]). For documents, copy detection considers sharing sufficiently large text fragments as evidence of copying. The naive strategy looks for the longest common subsequences (LCS), but can take time $O(n_1 \cdot n_2)$ for documents of sizes n_1 and n_2 respectively, and needs to compare every pair of documents. The first improvement is to build *fingerprints* for each document and only selectively store and compare the fingerprints. Manber [13] fingerprints each sequence of Q consecutive tokens (Q-*gram*), and builds a *sketch* with Q-grams whose fingerprints are 0 mod K; the space usage is thus only $\frac{1}{K}$ of original documents. Brin et al. [3] divides each document into non-overlapping chunks, where the last unit of each chunk has a fingerprint that is 0 mod K, and sketches each chunk; again, the space usage is expected to be $\frac{1}{K}$ of original documents. Schleimer et al. [16] also fingerprints each Q-gram, but the sketch contains the smallest fingerprint in each K-window; it has the same space usage but is guaranteed to find reuse of text with length of at least $K + Q - 1$. Another improvement is to build an index for the sketches, such that two documents are compared only if they share some fingerprints [11].

We also build an inverted index for the provided values and skip pairs of sources that do not share any value; however, our index is different in many ways. First, each entry in the index is associated with a score, indicating how strong sharing the value can serve as evidence for copying. Second, the entries are processed in decreasing order of the scores, so we consider stronger evidence first and can stop computation for a pair of sources when we have accumulated sufficient evidence for deciding copying or no-copying. Third, source pairs that share only a few entries with small scores will also be skipped for copy detection. Finally, we additionally design algorithms for pruning and incremental copy detection, which have not been discussed for document copy detection.

VIII. Conclusions

Copy detection has been shown to be crucial for truth finding on Web data but meanwhile is a bottle-neck in data fusion. This paper proposed various methods for improving scalability of copy detection on structured data. Experimental results show that the proposed algorithm can reduce copy-detection time by several orders of magnitude and finish fast on large data sets.

Our algorithms provide two opportunities for parallelization in a Hadoop framework. First, when we process each index entry, we can parallelize score computation for each pair of sources in that entry. Second, we can parallelize computation among entries: whereas parallelizing on all entries would be hard given the possibly huge number of entries, BOUND+ provides good insights on which entries can be processed in parallel. Both approaches are likely to be better than the strategy that simply extends PAIRWISE by parallelizing copy detection for each pair of sources, as the total number of pairs can be huge for big data. We leave such extensions and an experimental comparison for future work.

References

[1] A. Arasu, V. Ganti, and R. Kaushik. Efficient exact set-similarity joins. In *VLDB*, pages 918–929, 2006.
[2] L. Blanco, V. Crescenzi, P. Merialdo, and P. Papotti. Probabilistic models to reconcile complex data from inaccurate data sources. In *CAiSE*, 2010.
[3] S. Brin, J. Davis, and H. Garcia-Molina. Copy detection mechanisms for digital documents. In *Sigmod*, 1995.
[4] N. Dalvi, A. Machanavajjhala, and B. Pang. An analysis of structured data on the web. *PVLDB*, 5:680–691, 2012.
[5] X. L. Dong, L. Berti-Equille, Y. Hu, and D. Srivastava. Global detection of complex copying relationships between sources. *PVLDB*, 2010.
[6] X. L. Dong, L. Berti-Equille, and D. Srivastava. Integrating conflicting data: the role of source dependence. *PVLDB*, 2(1), 2009.
[7] X. L. Dong, L. Berti-Equille, and D. Srivastava. Truth discovery and copying detection in a dynamic world. *PVLDB*, 2(1), 2009.
[8] X. L. Dong and D. Srivastava. Large-scale copying detection. In *Sigmod (Tutorial)*, 2011.
[9] X. L. Dong, E. Gabrilovich, G. Heitz, W. Horn, K. Murphy, S. Sun and W. Zhang. From data fusion to knowledge fusion *PVLDB*, 7(10), 2014.
[10] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.
[11] H. Garcia-Molina, L. Gravano, and N. Shivakumar. dSCAM: Finding document copies across multiple databases. In *PDIS*, 1996.
[12] X. Li, X. L. Dong, K. B. Lyons, W. Meng, and D. Srivastava. Truth finding on the Deep Web: Is the problem solved? *PVLDB*, 6(2), 2013.
[13] U. Manber. Finding similar files in a large file system. In *USENIX*, pages 1–10, 1994.
[14] C. D. Manning, P. Raghavan, and H. Schutze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.
[15] G.-J. Qi, C. Aggarwal, J. Han, and T. Huang. Mining collective intelligence in groups. In *WWW*, 2013.
[16] S. Schleimer, D. S. Wilkerson, and A. Aiken. Winnowing: Local algorithms for document fingerprinting. In *Proc. of SIGMOD*, 2003.