Scaling up Copy Detection

Xian Li
SUNY at Binghamton
xianli@cs.binghamton.edu

Xin Luna Dong
AT&T Labs-Research
lunadong@research.att.com

Kenneth B. Lyons
AT&T Labs-Research
kbl@research.att.com

Weiyi Meng
SUNY at Binghamton
meng@cs.binghamton.edu

Divesh Srivastava
AT&T Labs-Research
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 size and number of data sources, so can slow down truth finding by one to two orders of magnitude compared with the corresponding techniques without considering copying. In this paper, we study *how to improve efficiency and scalability of copy detection on structured data*.

Our algorithm builds an inverted index for each shared value and orders the index entries by how much sharing the 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. In addition, we design 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.

1. INTRODUCTION

As we enjoy the abundance of useful information on the Web, we are often confused and misguided by low-quality data, which can be out-of-date, inaccurate, or erroneous. Typically, we expect a true value, which reflects the real world, to be provided by more sources than any particular false one, so we can take the value provided by the largest number of sources as the truth. Unfortunately, as shown in a recent study [13], data copying is common on the Web; a false value can spread through copying and become even more popular than the true value. This makes truth discovery extremely tricky. In [13] the authors show the promise of considering copying in truth discovery: even the basic copy-detection techniques proposed for structured data [6] can significantly improve truth-discovery results in the presence of copied values and fix half of the errors made by naive voting or considering only source accuracy. In addition, understanding the copying relationships between sources can help with conducting in-depth data analysis, studying dissemination of information, protecting the rights of data providers, and so on.

Despite the importance of copy detection, it is not clear if existing techniques can be applied on Web-scale data, where millions of sources can provide data in the same domain [4]. Most of current copy-detection techniques [2, 6, 7] consider each pair of sources and examine every shared data item (*i.e.*, an attribute of a record): if many incorrect values are shared, they conclude with copying since two independent sources rarely make the *same* mistakes on a large number of data items. In addition, since we often do not know the correctness of data, most of the proposed techniques conduct copy detection and truth finding iteratively until convergence, re-applying copy detection at each round. Such techniques cannot scale when the sizes of the sources, the number of the sources, or the number of iterations is large. Indeed, the aforementioned recent study [13] points out that even on medium-sized data (55 sources and 16,000 data items), considering copying slows down truth finding by one to two orders of magnitude compared with the corresponding techniques without considering copying. In the Big Data environment, not only are the data sources growing rapidly in size, but also there are more and more sources emerging; scalability is important for successfully conducting copy detection in such an environment. However, research on copy detection for structured data is still in its infancy and the current focus is mainly on effectiveness of the techniques. In this paper, we study the complementary problem of *how to improve scalability of copy detection.*

Before we describe our novel techniques for scalable copy detection on structured data, 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 wrong values are treated as strong evidence for copying whereas sharing correct ones are 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 correctness 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 between them.

We improve scalability in several ways and make the following contributions. First, we design an inverted index, where each entry corresponds to a provided value for a particular data item and

contains the data sources that provide this value. 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 rank 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 3).

Second, we perform pruning to further improve scalability for copy detection. Since we order entries in the inverted index in decreasing order of their scores, we consider strong evidence early as we scan the index. Once we have accumulated enough evidence for deciding copying or no-copying, we can stop without considering every shared value (Section 4).

Third, we perform incremental copy detection in later iterations. 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 5).

Fourth, we apply copy detection using a sample of data items from the sources. We design a sampling strategy that reduces the number of data items we consider while still obtaining very similar results as on the whole data set (Section 6).

Finally, we experimented on a variety of real data sets, showing high scalability of the proposed techniques (Section 7). In particular, we show that (1) using the inverted index itself can tremendously reduce the number of source pairs we consider and reduce copy-detection time by one to two orders of magnitude; (2) pruning and incremental detection can significantly reduce the number of data items we consider for each pair and further reduce copy-detection time by nearly one order of magnitude; and (3) a careful sampling can in addition reduce execution time by orders of magnitude without sacrificing the quality of copy detection.

In this paper we show that even a single server can detect copying for thousands of sources in seconds. We leave extensions for parallel detection for future work, described in Section 9.

2. PRELIMINARIES

We start with reviewing copy-detection techniques for structured data and describe the opportunities for scalability improvement.

2.1 Copy detection

Copy detection for structured sources was recently studied in [2, 5, 6, 7]. 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 we know which data items are shared between the sources. A source S_1 is considered as a *copier* of S_2 if S_1 copies a subset of data from S_2. *Copy detection* aims at *finding copying between sources in* $\mathcal{S}$.

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, and they are uniformly distributed. 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.

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

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 $S_1 \rightarrow S_2$ (or $S_2 \leftarrow S_1$) copying by S_1 from S_2, and by $S_1 \perp S_2$ no-copying between them[2]. Assuming there is no mutual copying (S_1 copies from S_2 *and* S_2 copies from S_1), we then have

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

Here, $0 < \alpha < .5$ is the a-priori probability of a source copying from another one and $\beta = 1 - 2\alpha$. We consider there is no copying if $Pr(S_1 \perp S_2|\Phi) > .5$. 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 conditions). Thus, we can rewrite Eq.(1) as follows.

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

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

1. Providing the same value v: We measure for each source $S \in \mathcal{S}$ its *accuracy*, denoted by $A(S)$, as the percentage of its correct values over all provided values. This can be considered as the probability of S providing a correct 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 that we assume there are n uniformly distributed false values). In practice, we are often not sure which value is correct. 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; 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 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 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,

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

Combining Eq.(3-5), we have

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

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

Table 1: Ten sources providing data on capitals of five US states. 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	*New York*	*Miami*	*Houston*
S_3	0.2	*Atlantic*	Phoenix	*New York*	*Miami*	*Arlington*
S_4	0.4	*Atlantic*	Phoenix	*New York*	Orlando	*Houston*
S_5	0.6	*Union*	*Tempe*	Albany	Orlando	Austin
S_6	0.01		*Tempe*	*Buffalo*	*Palm Bay*	*Dallas*
S_7	0.25	Trenton		*Buffalo*	*Palm Bay*	*Dallas*
S_8	0.2	Trenton	*Tucson*	*Buffalo*	*Palm Bay*	*Dallas*
S_9	0.99	Trenton			Orlando	Austin

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,

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

$$C_{\rightarrow}(D) = \ln(1-s). \quad (8)$$

It has been proved that $C_{\rightarrow}(D)$ 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. Note that we often do not know the value probabilities and source accuracies beforehand and need to compute them, and this computation often requires knowledge of the copying relationship. Iterative computation of copying probability, value correctness, and source accuracy until convergence is proposed to solve this chicken-and-egg problem [2, 6]. We skip the details on computation of value correctness and source accuracy [6], as they are not the focus of this paper.

EXAMPLE 2.1. *Consider 10 data sources that describe capitals for 5 states in the US (Table 1). 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 we know that NJ.Atlantic *has probability .01 to be true. Then, $C_{\rightarrow}(D_1) = C_{\leftarrow}(D_1) = \ln(.2 + .8 \cdot \frac{.01*.2+.99*.8}{.01*.2*.2+.99*\frac{.8*.8}{50}}) = 3.89$, showing that sharing this false value is strong evidence for copying. We compute for other items similarly and eventually $C_{\rightarrow} = C_{\leftarrow} = 3.89 + 1.6 + 3.86 + 3.83 - 1.6 = 11.58$. 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 correct and each of them has a contribution .01 (details skipped). Eventually, $C_{\rightarrow} = C_{\leftarrow} = .01 * 4 = .04$ and $Pr(S_0 \perp S_1|\Phi) = .79$, so copying is unlikely. □

2.2 Opportunities for scalability improvement

According to Bayesian analysis, we can detect copying in a pairwise fashion. For each pair of sources, we (1) compute for each shared $D \in \mathcal{D}$ the contribution scores $C_{\rightarrow}(D)$ and $C_{\leftarrow}(D)$; (2) accumulate the scores and compute $C_{\rightarrow}$ and $C_{\leftarrow}$; (3) apply Eq.(2) for probability computation. We repeat this process in every iteration. If the results converge in l rounds, the time complexity is $O(l|\mathcal{D}||\mathcal{S}|^2)$. It is not scalable if there are a large number of sources, a large number of data items, or a lot of iterations.

There are several opportunities for improving scalability of copy detection. First, for some source pairs 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 detection 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 source pairs. Section 3 describes how we can explore this opportunity by building and using an inverted index.

Second, for some source pairs that share a lot of false values, we can determine that they are dependent after we observe only a subset of these false values; this can reduce the number of data items we examine for those source 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 that they are dependent without knowing the rest of the provided values. Section 4 explores this opportunity for single-round copy detection.

Third, in the iterative process, the changes on value probability and source accuracy between two consecutive rounds after the second round are typically very small. Thus, we can do copy detection incrementally in later rounds; this can reduce the number of data items we consider for each source pair in those rounds. Section 5 explores this opportunity and describes incremental copy detection.

Fourth, we can sample a subset of data items from $\mathcal{D}$ for copy detection; this can also reduce the number of data items we consider. Section 6 describes our sampling approach.

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 correctness. In practice, however, early pruning can improve efficiency with slight loss of accuracy. We show in experiments (Section 7) the effectiveness and scalability of our techniques.

Finally, we note that it is possible to adapt Fagin's NRA (Non-Random Access) algorithm [9] for top-k search to improve efficiency of copy detection. We maintain for each particular value of a particular data item a list of contribution scores for the pairs of sources that share the value, and order the scores in decreasing order. In addition, we maintain a list containing the accumulated contribution scores from different values for source pairs 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. We show in experiments (Section 7) that even generating the input to NRA (*i.e.*, the ordered lists) for our context can incur longer time than our proposed approaches.

3. INVERTED INDEX

We first describe an important building block in our solution–the *inverted index*, which facilitates in exploring many aforementioned opportunities for scalability improvement. Inverted index was originally used in Information Retrieval [15] and we next describe how we adapt it for copy detection.

Building the index: An important component in copy detection is to find for each pair of sources the values 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. We wish to first consider sharing of values that serve as strong evidence for copying; thus, 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; we choose the maximum one, denoted by $M(D.v)$. The next proposition shows that we can compute $M(D.v)$ only from providers (*i.e.*, sources) with the maximum or minimum accuracy.

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 $A_{min} \leq \frac{1}{1+\frac{nP(D.v)}{1-P(D.v)}}$, $M(D.v)$ is obtained 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 $P(D.v) < .5$, $M(D.v)$ is obtained when S_2 has the minimum accuracy and S_1 has the second minimum accuracy;*
- *Otherwise, $M(D.v)$ is obtained when S_1 has the minimum accuracy and S_2 has the second minimum accuracy.* □

PROOF. According to Eq.(3-6), we examine $X = \frac{1+PA}{A(S_1)+PA\frac{1-A(S_1)}{n}}$, where $P = \frac{1}{P(D.v)} - 1$ and $A = \frac{1}{A(S_2)} - 1$. We can prove that (1) when $A(S_2)$ decreases, both A and X increase; and (2) when $A(S_1)$ increases, X increases when $PA > n$ and decreases when $PA < n$. These lead to the results. □

We can now formally define the 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 an ordered list $\mathbf{E}$ 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. *the entry contains a set $\bar{S}(E)$ of sources that provide $D_E.v_E$;*
3. *E is associated with probability $P(E)$ for $D_E.v_E$ being true and with* contribution score *$C(E) = M(D_E.v_E)$.*

The entries in $\mathbf{E}$ are in decreasing order of contribution scores. □

The way we define the inverted index and order the entries has many good properties that make it amenable to additional optimizations. We also show in experiments (Section 7.3) the benefit of our ordering scheme.

PROPOSITION 3.3. *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 correct values and we can skip them too. To simplify the computation, we consider the entries from the end of the inverted index and denote by $\bar{E} \subseteq \mathbf{E}$ the subset of entries at the end of the index 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 $Pr(S_1 \perp S_2 | \Phi) > \frac{1}{1+\frac{\alpha}{\beta}(\frac{\beta}{2\alpha}+\frac{\beta}{2\alpha})} = .5$ 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.

Table 2: 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	$S_0, S_1, \mathbf{S_5}, \mathbf{S_9}$

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 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 3.4. *Let r be the number of source pairs for which we maintain scores.* INDEX *takes time $O(r \cdot |\mathcal{D}|)$ and space $O(r)$ and obtains the same binary results as pairwise detection.* □

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

EXAMPLE 3.5. *Continue with Ex.2.1. Table 2 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.*

If we apply INDEX*, the last two entries in the index 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 shares 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.* □

4. SINGLE-ROUND DETECTION

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 3.3) 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.

4.1 Reducing examined shared values

We consider binary decisions for copying. According to Eq.(2), to guarantee $Pr(S_1 \perp S_2|\Phi) > .5$ (no-copying), we should have $1 > \frac{\alpha}{\beta}(e^{C_\rightarrow}+e^{C_\leftarrow})$; this must be true if $C_m = \max\{C_\rightarrow, C_\leftarrow\} < \ln\frac{\beta}{2\alpha}$. On the other hand, to guarantee $Pr(S_1 \perp S_2|\Phi) \leq .5$ (copying), we should have $1 \leq \frac{\alpha}{\beta}(e^{C_\rightarrow} + e^{C_\leftarrow})$; this must be true if $C_m \geq \ln\frac{\beta}{\alpha}$. If $\ln\frac{\beta}{2\alpha} \leq C_m < \ln\frac{\beta}{\alpha}$, we need to apply Eq.(2) to decide if there is copying. Thus, we use two thresholds: $\theta_{ind} = \ln\frac{\beta}{2\alpha}$ for no-copying and $\theta_{cp} = \ln\frac{\beta}{\alpha}$ for copying. Note that 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: $Pr(S_1 \perp S_2|\Phi) > .9$, $Pr(S_1 \perp S_2|\Phi) < .1$ and otherwise; we can compute the thresholds accordingly.

Given a pair of sources S_1 and S_2, as we scan the index, we can maintain for $C_\rightarrow$ a maximum score and a minimum one, denoted by $C_\rightarrow^{max}$ and $C_\rightarrow^{min}$ respectively; similarly we maintain $C_\leftarrow^{max}$ and $C_\leftarrow^{min}$. We can terminate early for this pair in two cases: (1) if at some point $C_\rightarrow^{min} \geq \theta_{cp}$ *or* $C_\leftarrow^{min} \geq \theta_{cp}$, we must have $C_m \geq \theta_{cp}$ and can conclude copying; and (2) if $C_\rightarrow^{max} < \theta_{ind}$ *and* $C_\leftarrow^{max} < \theta_{ind}$, we must have $C_m < \theta_{ind}$ and can conclude no-copying. If none of the conditions is satisfied after we scan the whole index, we apply Eq.(2) to compute the probability of copying.

To compute the minimum and maximum scores, as we scan each entry E, we maintain three auxiliary data structures: (1) for each $S \in \bar{S}(E)$, we maintain $n(S)$ as the number of observed values for S; (2) for each pair $S_1, S_2 \in \bar{S}(E)$, we maintain $n(S_1, S_2)$ as the number of observed shared values; (3) for each pair $S_1, S_2 \in \bar{S}(E)$, we maintain $C_\rightarrow^0(S_1, S_2)$ and $C_\leftarrow^0(S_1, S_2)$ as the sum of contribution scores from already observed shared values.

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. The contribution score from the observed common values is $C_\rightarrow^0(S_1, S_2)$. The score for each of the remaining $l(S_1, S_2) - n(S_1, S_2)$ items is negative, $\ln(1-s)$. Thus,

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

For $C_\rightarrow^{max}$, 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 $C_\rightarrow^{max}$ is the sum of their scores.

- *Data items with observed shared values:* The accumulated score from such items is $C_\rightarrow^0(S_1, S_2)$.
- *Data items with observed unshared values:* According to Proposition 3.3, 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 $n(S_1)$ and $n(S_2)$. Let $\bar{N}_1$ be 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 number of overlapping items is $|\bar{N}_1 \cup \bar{N}_2|$, satisfying $|\bar{N}_1 \cup \bar{N}_2| \geq \max\{|\bar{N}_1|, |\bar{N}_2|\} = \max\{n(S_1) \cdot \frac{l(S_1,S_2)}{|\bar{D}(S_1)|}, n(S_2) \cdot \frac{l(S_1,S_2)}{|\bar{D}(S_2)|}\}$, denoted by h. Thus, the two sources provide different values for at least $h - n(S_1, S_2)$ data items, and the maximum score is $(h - n(S_1, S_2))\ln(1-s)$.
- *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 3.3, 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

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

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 the first time to observe S, 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 $n(S_1, S_2) = C_{\rightarrow/\leftarrow}^0(S_1, S_2) = 0$ 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 $n(S_1, S_2)$ by 1 and update $C_{\rightarrow/\leftarrow}^0(S_1, S_2)$.

   (2) Compute $C_{\rightarrow/\leftarrow}^{min}(S_1, S_2)$. If either is above θ_{cp}, conclude copying and remove the pair from $\mathbf{Q}$.

   (3) Compute $C_{\rightarrow/\leftarrow}^{max}(S_1, S_2)$. If both are below θ_{ind}, conclude no-copying and remove the pair from $\mathbf{Q}$.

Step III. Do 1 and 3 in Step II (so only for pairs encountered before) for each $E \in \bar{E}$.

Step IV. After we scan the index, for each pair $(S_1, S_2) \in \mathbf{Q}$, we have $C_\rightarrow = C_\rightarrow^{min}$ (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 $C_\rightarrow^{max}$, 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 difference happens rarely, as we observed in our experiments (Section 7). 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.3.5. 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 $C_{\rightarrow/\leftarrow}^0(S_2, S_3) = 3.89$. By Eq.(9) we compute $C_{\rightarrow/\leftarrow}^{min} = 3.89 - 1.6 * (5-1) = -2.51$. For maximum scores, $h = 1$ so by Eq.(10) $C_{\rightarrow/\leftarrow}^{max} = 3.89 + 0 + 4.05 * (5-1) = 20.09$. We see this pair again at entry* NY.NewYork. *We update $C_\rightarrow^0(S_2, S_3) = 3.89 + 3.86 = 7.75$, so $C_\rightarrow^{min} = 7.75 - 1.6 * (5-2) = 2.95 > 2.08 = \theta_{cp}$ 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 $C_{\rightarrow/\leftarrow}^0(S_2, S_3) = .01 * 3 = .03$, 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, less than* BOUND.

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

4.2 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), so the minimum scores can be above θ_{cp} only after we observe at least $T^{min} = \lceil \frac{\theta_{cp} - \max\{C^{min}_{\rightarrow}, C^{min}_{\leftarrow}\}}{M - \ln(1-s)} \rceil$ shared values. Accordingly, once we compute $C^{min}_{\rightarrow}$, we set timer T^{min}; when we examine a new shared value, we reduce T^{min} by 1, and do not re-compute $C^{min}_{\rightarrow/\leftarrow}$ until $T^{min} = 0$. We stop computing T^{min} once it is close to 1 (in experiments we stop when $T^{min} \leq 2$).

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 $T^{max}_0 = \frac{\max\{C^{max}_{\rightarrow}, C^{max}_{\leftarrow}\} - \theta_{ind}}{M - \ln(1-s)}$ more different values. At entry E we have already seen $h - n(S_1, S_2)$ different values, so we need to see in total $T^{max}_0 + h - n(S_1, S_2)$ different values; this would transform to seeing $T^{max}_1 = \lceil (T^{max}_0 + h - n(S_1, S_2)) \cdot \frac{|\bar{D}(S_1)|}{l(S_1, S_2)} \rceil$ entries for S_1 or $T^{max}_2 = \lceil (T^{max}_0 + h - n(S_1, S_2)) \cdot \frac{|\bar{D}(S_2)|}{l(S_1, S_2)} \rceil$ entries for S_2. Thus, when we examine a shared value later, we do not re-compute $C^{max}_{\rightarrow/\leftarrow}$ until $n(S_1) \geq T^{max}_1$ or $n(S_2) \geq T^{max}_2$. Note that $C^{max}_{\rightarrow}$ and $C^{max}_{\leftarrow}$ are also reducing as we scan the index since contributions from the data items that we have not seen for S_1 or S_2 are also diminishing; thus, unlike for the lower bound, $C^{max}_{\rightarrow}$ and $C^{max}_{\leftarrow}$ may hit θ_{ind} before we check them next time and so we may check more shared values than necessary. However, this would not affect the conclusion, as we observe the same termination condition.

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* $C^0_{\rightarrow/\leftarrow} = 5$. *Then* $C^{min}_{\rightarrow/\leftarrow} = 5 - (101 - 1) * 1.6 = -155 < 2.08$. *Suppose* $M = 4$, *then* $T^{min} = \lceil \frac{2.08 - (-155)}{4 - (-1.6)} \rceil = 29$, *so we do not compute* $C^{min}_{\rightarrow/\leftarrow}$ *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* $C^{max}_{\rightarrow/\leftarrow} = 5 + 0 + (101 - 1) * 4 = 405$. *Then,* $T^{max}_0 = \frac{405 - 1.39}{4 - (-1.6)} = 72$. *Suppose* $\frac{l(S_1,S_2)}{|\bar{D}(S_1)|} = \frac{l(S_1,S_2)}{|\bar{D}(S_2)|} = .8$, *then* $T^{max}_1 = T^{max}_2 = \lceil \frac{(72+1-1)}{.8} \rceil = 90$. *So we do not compute* $C^{max}_{\rightarrow/\leftarrow}$ *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 7) show that HYBRID can further reduce computation and copy-detection time.

5. INCREMENTAL DETECTION

Section 4 considered copy detection in one single round; this section considers the iterative process. Our observation is that although there can be changes from round to round, after the second round the changes on value probability and source accuracy 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. In this section we base our discussions on the HYBRID algorithm.

5.1 Classifying changes

Both changes on value probability and changes on source accuracy can affect copy detection. For comparison we record for each source S the accuracy used previously in score computation, denoted by $A_{old}(S)$, and for each value $D.v$ the probability used previously, denoted by $P_{old}(D.v)$. Note that the recorded accuracy or probability may not be the one from the previous round, but can be from some early round in case the change since then is small.

We distinguish between big and small changes. For source accuracy we have a threshold ρ_A (we can set a default threshold or choose the one corresponding to the maximum gap between the differences). If source S satisfies $|A(S) - A_{old}(S)| > \rho_A$, we just recompute the scores for each pair containing S and update $A_{old}(S)$ to $A(S)$.[3] Otherwise, we incrementally update the contribution score as we describe next.

For a value $D.v$, we consider the change on $M(D.v)$ 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 $M(D.v)$ on the same two accuracies used in the round with $P_{old}(D.v)$. We set a threshold ρ_V for big score difference in the same way as we set ρ_A. We call an entry E a *big-change* entry if the change of $C(E)$ is larger than ρ_V, and a *small-change* entry otherwise. We *update scores on big-change entries first, and then on small-change ones only when necessary*.

5.2 Updating for one pair of sources

Source pairs with copying: We first explain our strategy for source pairs where we concluded with copying in the previous round. To prepare for incremental copy detection in later rounds, we maintain for each source pair (1) the *decision point*; that is, the entry at which we make our decision (for source pairs where we apply INDEX, the decision point is the last entry); (2) number of shared values before the decision point and that after the decision point; and (3) the final score of this round, denoted by $\hat{C}_{\rightarrow}$ and $\hat{C}_{\leftarrow}$, which will be taken as the starting score for the next round. In the round when we conduct copy-detection from scratch, we compute $\hat{C}_{\rightarrow}$ (resp. $\hat{C}_{\leftarrow}$) as follows. Recall that the computation of $C^{min}_{\rightarrow}$ assumes that there is no value shared after the decision point, but now suppose from the bookkeeping we know that n values are shared after the decision

[3] When there are many big-accuracy-change sources (percentage above a threshold), we can directly apply HYBRID; similar if we have a lot of values whose probabilities change a lot.

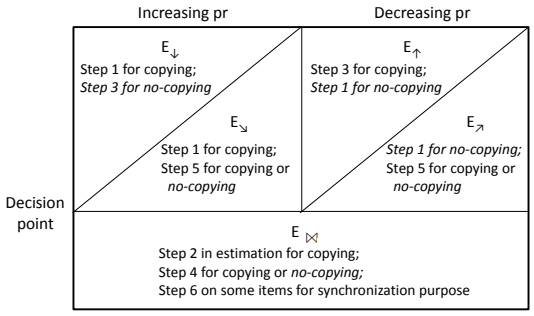


Figure 1: Entry categories. The steps for no-copying pairs are in italic font.

Table 3: 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.

point, so we can remove the penalty $(\ln(1-s))$ for them and compute $\hat{C}_{\rightarrow} = C^{min}_{\rightarrow} - n \cdot \ln(1-s)$. Note that $C^{min}_{\rightarrow} < \hat{C}_{\rightarrow} < C_{\rightarrow}$, because for the shared entries after the decision point, $\hat{C}_{\rightarrow}$ does not apply penalty but does not accumulate the contribution either.

For a pair of sources, we can categorize their shared entries into five categories (see Fig.1): (1) $\bar{E}_{\downarrow}$: big-change entries whose contribution scores decrease (value probabilities 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 first illustrate the idea using an example.

EXAMPLE 5.1. *Continue with the motivating example. There are five rounds before convergence; Table 3 shows the source accuracy and value probability computed in each round (for simplicity, we show only for the first 5 sources and their values). Consider incremental detection at Round 3; it considers the accuracies and value probabilities from Round 1 as old ones and those from Round 2 as new ones. Table 4 shows the inverted index. We set $\rho_A = .2$ and $\rho_V = 1$. There is no source with big accuracy change, but there are 2 entries with big score changes (in italic), 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 decision and 1 value after decision. 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. Thus, the score is not decreased and we can terminate for this pair without further examination.*

Table 4: 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 → .03	.17	↗	3.97
FL.Miami	S_2, S_3	.05 → .03	.21	↗	3.83
NJ.Atlantic	S_2, S_3, S_4	.07 → .03	.39	↗	3.96
NY.Albany	S_0, S_1	.07 → .77	-2.49	↓	.52
NJ.Trenton	S_0, S_1	.9 → .95	-.12	↘	1.31
NY.NewYork	S_2, S_3, S_4	.84 → .16	1.69	↑	3.17
AZ.Phoenix	$S_0 - S_4$	.94 → .95	-.01	↘	1.45
FL.Orlando	S_1, S_4	.9 → .92	-.01	↘	.78
TX.Austin	S_0, S_1	.9 → .93	-.01	↘	.51

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 decision; recall that $\theta_{cp} = 2.08$ and $\theta_{ind} = 1.39$, so we have 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 *was $.44/1.53$ in Round 2 and is $.24/.09$ now. The largest score difference for the other three items can be computed from* NJ.Trenton*, which has the largest score decrease among small-change entries, and it is .015. 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.* □

We now describe how we update $\hat{C}_{\rightarrow}$ (resp. $\hat{C}_{\leftarrow}$) in five steps.

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 of such entries, we use the maximum change, denoted by Δ_{ρ}, which we estimate from the entry with the largest score decrease below ρ_V. 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; to maintain precise score for $\hat{C}_{\rightarrow}$, we remove estimation Δ_1 and record $\hat{C}_{\rightarrow} + \Delta_1$ as the starting value of $\hat{C}_{\rightarrow}$ in the next round.

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 $\Delta_2 = m \cdot |\bar{E}_{\bowtie}|$ (recall that $|\bar{E}_{\bowtie}|$ is recorded from the previous round). If at this point $\max\{\hat{C}_{\rightarrow}, \hat{C}_{\leftarrow}\} \geq \theta_{cp}$ already holds, we can stop and record $\hat{C}_{\rightarrow} + \Delta_1 - \Delta_2$ as the new score.

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. Once $\max\{\hat{C}_{\rightarrow}, \hat{C}_{\leftarrow}\} \geq \theta_{cp}$ is satisfied, we stop and record the new score as $\hat{C}_{\rightarrow} + \Delta_1 - \Delta_2$.

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. Once $\max\{\hat{C}_{\rightarrow}, \hat{C}_{\leftarrow}\} \geq \theta_{cp}$, we stop, updating the decision point, the number of shared entries before and after the decision point, and record the new score as $\hat{C}_{\rightarrow} + \Delta_1 - \Delta_2$.

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. Once $\max\{\hat{C}_{\rightarrow}, \hat{C}_{\leftarrow}\} \geq \theta_{cp}$, we stop and record the new score as $\hat{C}_{\rightarrow} + \Delta_1$ ($\Delta_2 = 0$ at this point). If the condition is not satisfied until the end, we apply Bayesian analysis to decide if we need to change our decision to no-copying.

These five 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.

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. 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 $C_{\rightarrow}^{max} = -4.75, C_{\leftarrow}^{max} = -4.3$, sharing 1 value before decision and 0 value after decision. Since they share 4 data items, they must provide different values on 3 data items, same as our estimation. Also, the entry where we compute M for this pair (*FL.Orlando*) does not have a big score change. Thus, $\hat{C}_{\rightarrow/\leftarrow} = C_{\rightarrow/\leftarrow}^{max}$. 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.* □

5.3 Synchronization for pairs of sources

There is in addition a *synchronization* problem when we update scores for different source pairs. Recall that for each value $D.v$ we maintain $P_{old}(D.v)$, which is the probability used in previous score computation; however, we may use different probabilities for different source pairs, as they may not terminate at the same point.

EXAMPLE 5.3. *Continue with Ex.5.1. Entry* NY.Albany *has a big score decrease. Pair (S_0, S_1) was determined to have copying in Round 2 and so uses its new probability .77 in score updates in Round 3. Pair (S_0, S_5) (note that S_5 is not shown in Table 4 for simplicity), however, was determined to have no copying in Round 2 and so continues using its old probability .07.*

Entry AZ.Phoenix *has a small score decrease. Pair (S_0, S_1) goes through all five steps and uses its new probability .95. Pair (S_0, S_2) was determined to have no copying in Round 2 and so continues using its old probability .94. Pair (S_2, S_3) terminates before this entry in Round 2, so does not use either probability.* □

As the example shows, there are two types of desynchronization. First, in the first pass, an entry in $\bar{E}_{\downarrow}$ (resp. $\bar{E}_{\uparrow}$) affects only copying pairs before the decision points, so can cause desynchronization between copying pairs and no-copying ones that terminate in the first pass and have later decision points. Second, an entry considered in later passes affects only a few pairs of sources that have not terminated yet, causing desynchronization between these pairs and others. Storing an old probability for each source pair can be extremely expensive; instead, we synchronize for different pairs as follows.

1. To remove Type-I desynchronization, we maintain two old scores for each data item, one for copying pairs and one for no-copying pairs, and decide whether the change is big separately.

Table 5: Number of computations for Example 5.5.

	Rnd 3	Rnd 4	Rnd 5
BOUND+	102	102	102
INCREMENTAL	54	29	0
For synchronization	24	4	0

Thus, an entry can be a big-change entry for copying pairs but small-change entry for no-copying pairs. In our example, for NY.Albany, we maintain .77 for copying pairs and .07 for no-copying pairs.

2. To remove Type-II desynchronization in Step 3 and Step 5, once we update the score for one copying (resp., no-copying) pair, we update the score for every other copying pair with a later decision point. In our example, for AZ.Phoenix we use the new probability .95 for (S_0, S_1), so we maintain .95 for copying pairs but keep .94 for no-copying ones. Meanwhile, we synchronize for other copying pairs; however, since AZ.Phoenix is after the decision points for copying pairs between $S_2 - S_4$, no update is needed.

3. To avoid Type-II desynchronization in Step 4 for copying (resp. no-copying) pairs, we (1) apply the new value probability for entries with big score decrease, and (2) apply the old value probability for the rest of the entries and reduce Δ_ρ for each entry with small score decrease. If we do not terminate after Step 4, Step 5 will adjust scores for the small-change entries, and we add Step 6 in which we adjust scores for entries with big score increase.

4. We may change our decision on a source pair from copying to no-copying (vice versa). This change would cause desynchronization between this pairs with other no-copying pairs. We set the decision point for such pairs as before the first entry (so the only non-empty set is $\bar{E}_{\bowtie}$), reset $\hat{C}_{\rightarrow/\leftarrow} = 0$, and essentially recompute their scores from scratch in the next round. In our example, we do so for pair (S_0, S_1) at the end of Round 3 as we change decision on it. We handle source pairs that contain big accuracy change for one source in a similar way, since score recomputation is needed.

5.4 Putting it together

Algorithm INCREMENTAL updates scores for all source pairs in three passes of index scanning. It requires more space for bookkeeping across rounds, but in practice it recomputes scores for much fewer entries.

PROPOSITION 5.4. *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.* INCREMENTAL *takes time $O(re')$ and space $O(|\mathcal{E}| + r + |\mathcal{S}|)$ for a single round.* □

EXAMPLE 5.5. *Table 5 compares the number of computations of* BOUND+ *and* INCREMENTAL *in Rounds 3-5 for our example. We observe that although synchronization causes overhead and counts for 34% of computations, the total number of computations for* INCREMENTAL *is still 73% lower than that for* BOUND+. □

6. SAMPLING

When there are a lot of data items, even with early pruning we may still need to consider a lot of data items for each pair of sources before termination. This is because initially both $C_{\rightarrow/\leftarrow}^{min}$ and $C_{\rightarrow/\leftarrow}^{max}$ are functions of the number of shared data items (*i.e.*, $|\bar{D}(S_1) \cap \bar{D}(S_2)|$); the more shared data items, the smaller $C_{\rightarrow/\leftarrow}^{min}$ and the larger $C_{\rightarrow/\leftarrow}^{max}$, and the longer it can take to terminate.

One obvious solution to reduce the number of data items is to sample a certain percentage of the data items. However, a naive uniform random sampling does not work when some sources provide only a small number of items: we may sample too few data items for such sources to draw the right conclusion for copying. To

Table 6: 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

solve the problem, we wish to guarantee that from each source we sample at least $M > 0$ data items[4]. We do so in two steps.

1. Randomly sample a set $\bar{D}$ of data items with the given sampling rate.
2. Check each source $S \in \mathcal{S}$. If S provides at most M data items, add all of its data items into the sample $\bar{D}$. If S has only $k < M$ data items sampled (*i.e.*, $|\bar{D}(S) \cap \bar{D}| < M$), randomly sample $M - k$ data items from the rest of S's data items and add them to $\bar{D}$.

7. EXPERIMENTAL RESULTS

This section presents experimental results validating the efficiency and effectiveness of the 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.

7.1 Experiment settings

Data: We experimented on four data sets (see Table 6 for an overview). Two data sets came from an online bookstore aggregator *AbeBooks.com*: *Book-CS* was extracted by searching computer science books[5]; it contains 894 sources (*i.e.*, book stores), 1265 books, and 2528 data items including the title and author list of each book (there are missing values for some books). *Book-full* includes books of all categories crawled from *AbeBooks* in 9/2009; it contains 3182 sources, 81,352 books, and 147,431 data items.

The other two data sets were crawled from 55 Deep Web sources among top-200 results by Google for keyword search "stock price quotes" and "AAPL quotes". We focused on 1000 stocks, including the 30 symbols from Dow Jones Index, the 100 symbols from NASDAQ Index (3 symbols appear in both Dow Jones and NASDAQ), and randomly chose 873 symbols from the other symbols in Russell 3000. We collected data one hour after the stock market closed on each work day and considered 16 attributes that were provided by at least one third of the sources and stay stable after market close (see [13] for details of data collection). *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 the latter contains $16,000 * 10 = 160,000$ data items (stock markets are closed during weekends but there was update on 7/4).

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.*, all sources in *Stock-1day* provide over 90% of the stocks and 80% sources cover 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 2.2.
- 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 why such sampling rates shortly).
- INDEX implements algorithm INDEX (Section 3).
- BOUND and BOUND+ each applies the corresponding algorithm (Section 4) 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 4)
- INCREMENTAL applies HYBRID in the first two rounds and Algorithm INCREMENTAL (Section 5) in later rounds. It sets $\rho_A = .2$ and $\rho_V = 1.0$ according to observations of the largest gaps.
- SCALESAMPLE samples 1% of data items on *Stock-2wk* and 10% on the other data sets as described in Section 6 ($M = 4$), then applies INCREMENTAL on the sampled data.
- FAGININPUT generates the input to Fagin's NRA algorithm as described at the end of Section 2.2.

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

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

Measures: We measure three aspects of our results.
Efficiency: We measure efficiency by (1) the number of computations in copy detection (as described in the examples in Sections 3-5), and (2) the execution time.

Copy-detection correctness: We examined how the various methods for improving scalability may sacrifice the results of copy detection; thus, we compared their results with those of PAIRWISE. *Precision* measures among the output copying pairs, which percentage is also output by PAIRWISE; *Recall* measures among the output copying pairs by PAIRWISE, which percentage 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. For *Book-CS*, we used a gold standard containing author lists verified from book title pages for 100 randomly selected books. For *Stock-1day*, we used a gold standard containing 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*. We report three measures: (1) *Fusion accuracy* measures the percentage of correct truth-finding results among all data items in the gold standard; (2) *Fusion difference* measures the percentage of truth-finding results different from those when applying PAIRWISE; and (3) *Accuracy variance* measures the average difference of the source accuracy computed when applying PAIRWISE and when applying the specific copy-detection method.

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

[4] We discuss in Section 7 how to set M and the sampling rate.

[5] We thank the authors of [18] for providing us the data.

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

Table 7: 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	Fusion accu	Fusion diff	Accu var	Prec	Rec	F-msr	Fusion 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	-	-	-	-	-	-
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 8: Execution time and the time reduction 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%	-	-	684	94%	-	-
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%

7.2 Performance overview

We first compare the various methods on each data set. Table 7 reports copy-detection and truth-finding correctness, and Table 8 reports execution time. We make the following observations.

First, naive sampling (SAMPLE1 and SAMPLE2) did improve the efficiency a lot, but not as much as INCREMENTAL and SCALESAMPLE. Indeed, 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 and very low accuracy on truth discovery on *Book-CS*; in this data set a lot of data sources provide only a few books, so a random sampling can lead to inaccurate decisions.

Second, our proposals for improving scalability work very well. Without sampling, INCREMENTAL finished in 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. 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 are not considered at all. Also, we observe from Table 6 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 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 end 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 our sampling strategy. 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 M sampled data items takes longer time for large data sets.

Table 9: Execution-time ratio w.r.t. FAGININPUT.

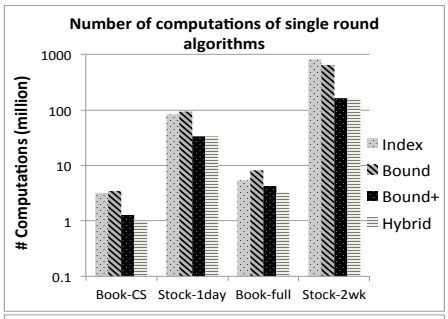
	Book-CS	*Stock-1day*	*Book-full*	*Stock-2wk*
HYBRID	.87	.76	.99	.67
INCREMENTAL	.30	.27	.22	.19

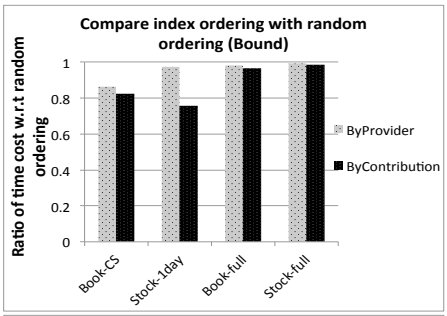
Finally, Table 9 shows the execution time ratio of our methods versus FAGININPUT. FAGININPUT has two drawbacks. First, it has to compute the contribution scores from each shared value for each source pair; thus, HYBRID is 18% faster than FAGININPUT on average for a single round. Second, it is not clear how to generate the input lists incrementally in later rounds; thus, INCREMENTAL is 75% faster than FAGININPUT on average for all rounds.

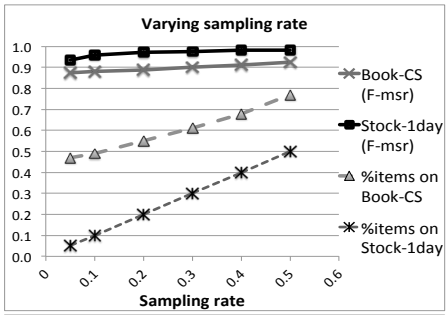
7.3 Single-round algorithms

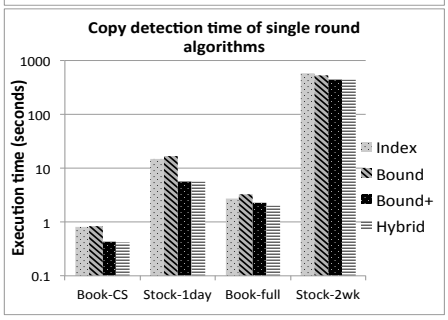
We next examine single-round algorithms in more detail. We first compared INDEX, BOUND, BOUND+, and HYBRID on their numbers of computations (for all rounds together) and copy-detection (see Figure 2). We have three observations. First, for three out of four data sets BOUND conducts more computations and finished in longer time. Although it reduces the number of data items for consideration, it introduces a big overhead for computing 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%. 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.

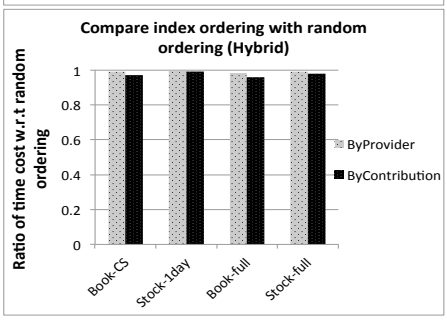
We then examined various ways of ordering entries in the inverted index: RANDOM orders the entries randomly; BYPROVIDER orders the entries in increasing order of the number of providers











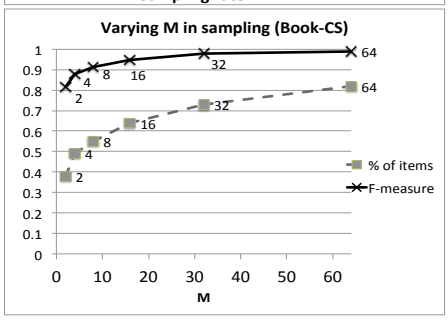


Figure 2: Single-round algorithms.

Figure 3: Different index ordering.

Figure 4: Sampling rate and M.

Table 11: Execution time ratio of INCREMENTAL vs HYBRID, percentage of pairs terminated at each pass of incremental detection, and synchronization overhead for #computations.

	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%
Synchronization	61%	62%	47%	77%

(*i.e.*, sources); and BYCONTRIBUTION orders 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 put some correct but not widely provided values towards the beginning of the index and so can have more computation than BYCONTRIBUTION.

7.4 Incremental algorithms

We conducted three experiments to understand how incremental detection improves efficiency. First, Table 10 shows the changes of copy-detection results round by round. We observe a big change from the first round to the second round; this is because the first round assumes all values have the same probability to be true whereas the second round has a better clue on value correctness. Starting from the third round, the changes are quite small and the numbers of added and removed copying pairs are below 10%. For this reason, we apply incremental detection starting from the third round.

Table 11 shows 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 then show in Table 11 how many pairs terminate at each of the three passes and what is the overhead of synchronization in terms of the number of computations. 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. We also observe that synchronization indeed has a big overhead: on average 62% of computations are spent on synchronization; however, it is worthwhile given the big efficiency gain.

Table 12: Comparing different sampling methods.

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

7.5 Sampling

Finally, we compare our sampling strategy, called SOURCEBASED, with sampling rate 10%, with two naive sampling strategies as described in SAMPLE1 and SAMPLE2, which we call BYITEM and BYCELL respectively. To ensure a fair comparison, the sampling rate for BYITEM is decided by the percentage of sampled data items in SOURCEBASED, and the sampling rate for BYCELL (and SAMPLE2) is decided by the percentage of sampled cells in SOURCEBASED. For example, SOURCEBASED sampled 49% data items and 65% cells on *Book-CS*, so we applied a sampling rate of 49% for BYITEM and 65% for BYCELL; SOURCEBASED sampled 10% data items and 10% cells on *Stock-1day*, so BYITEM and BYCELL applied the same sampling rate (10%). Table 12 shows the quality of copy-detection results when we apply INDEX. The three sampling methods obtain the same results on *Stock-1day* since the sources all have a high coverage in that data set; SOURCEBASED obtains the best results on *Book-CS* even though it selects the same

Table 10: Fraction of copying pairs added or removed from each previous round and F-measure w.r.t results of the final round.

	Book-CS								*Stock-1day*					
	Rnd 1	Rnd 2	Rnd 3	Rnd 4	Rnd 5	Rnd 6	Rnd 7	Rnd 8	Rnd 1	Rnd 2	Rnd 3	Rnd 4	Rnd 5	Rnd 6
Added	-	.334	.053	.016	.007	.005	.003	.004	-	1.150	.081	.003	.003	.003
Removed	-	.504	.034	.013	.006	.008	.004	.001	-	.052	.006	.003	0	0
F-measure	.518	.936	.974	.983	.988	.993	.995	1	.594	.954	.998	.998	.999	1

number of data items as BYITEM and the same number of cells as BYCELL, since it guarantees that we select at least $M = 4$ data items from each source when possible.

We also examined the effect of sampling rate and the value of M; Figure 4 shows the results of SOURCEBASED. First, the percentage of sampled data items is exactly the same as the sampling rate for *Stock-1day*, but much higher than the sampling rate for *Book-CS*; in other words, M plays a big role when there are a lot of low-coverage sources. Second, as expected, the higher the sampling rate, the higher the F-measure of copy detection. On *Stock-1day*, we observe a big jump when the sampling rate grows from .05 to .1 and then the increase slows down. In contrast, on *Book-CS* we obtain similar F-measures when the sampling rate is .05 and is .1, since SOURCEBASED is controlled more by M when sampling rate is low on this data set. Third, on *Book-CS* when M increases, the F-measure of copy detection increases as well. We observe a big jump when M grows from 2 to 4 and then the increase slows down. How to set the sampling rate and the value of M is a trade-off between efficiency and effectiveness. We observe that a sampling rate of .1 and $M = 4$ work well on our data sets.

8. RELATED WORK

Improving scalability of copy detection has been intensively studied for text documents and software programs (surveyed in [8]). For documents, where each document can be considered as a text sequence, 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 [14] 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. [17] 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 [10].

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 aspects. 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 ranked 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 design in addition algorithms for pruning and incremental copy detection, which are not discussed for document copy detection.

Inverted indexes are also used in many other applications: information retrieval uses inverted index to search keywords [15]; set similarity join uses inverted index to find overlapping strings or q-grams [1, 11, 12, 16]. In our context, copy detection requires scanning of the index and so calls for a wise ordering of the entries. Finally, we show by experiments that if we apply Fagin's NRA algorithm [9] for copy detection, even generating the input to NRA cost more time than our algorithms.

9. CONCLUSIONS

This paper proposed various methods for improving efficiency and 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 can finish fast on large data sets. Future research directions include extending it for more sophisticated copy-detection techniques [2, 5] and adjusting it for the cloud-computing environment.

10. 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] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.

[10] H. Garcia-Molina, L. Gravano, and N. Shivakumar. dSCAM: Finding document copies across multiple databases. In *PDIS*, 1996.

[11] M. Hadjieleftheriou, A. Chandel, N. Koudas, and D. Srivastava. Fast indexes and algorithms for set similarity selection queries. In *ICDE*, 2008.

[12] C. Li, J. Lu, and Y. Lu. Efficient merging and filtering algorithms for approximate string searches. In *ICDE*, 2008.

[13] X. Li, X. L. Dong, K. B. Lyons, W. Meng, and D. Srivastava. Truth finding on the deep web: Is the problem solved? *PVLDB (to appear)*, 2013.

[14] U. Manber. Finding similar files in a large file system. In *USENIX*, pages 1–10, 1994.

[15] C. D. Manning, P. Raghavan, and H. Schutze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[16] S. Sarawagi and A. Kirpal. Efficient set joins on similarity predicates. In *Sigmod*, 2004.

[17] S. Schleimer, D. S. Wilkerson, and A. Aiken. Winnowing: Local algorithms for document fingerprinting. In *Proc. of SIGMOD*, 2003.

[18] X. Yin, J. Han, and P. S. Yu. Truth discovery with multiple conflicting information providers on the web. In *Proc. of SIGKDD*, 2007.