Efficient Record Linkage Using a Compact Hamming Space

Dimitrios Karapiperis*, Dinusha Vatsalan†, Vassilios S. Verykios*, and Peter Christen†

*Hellenic Open University
School of Science and Technology
Patras, Greece
{dkarapiperis, verykios}@eap.gr

†The Australian National University
Research School of Computer Science
Canberra ACT 0200, Australia
{dinusha.vatsalan, peter.christen}@anu.edu.au

ABSTRACT

Record linkage, the process of identifying similar records that correspond to the same real-world entities across databases, is a well-established research problem in the database, data mining, and information retrieval communities. Computing distances between string values of records is the key component in order to determine the similarity of the represented entities. Due to the typically large volumes of records, a two-step process is followed. A blocking mechanism is first applied for grouping similar records together, and then a matching mechanism is performed for comparing the records which have been inserted into the same block. However, there does not exist any efficient blocking/matching mechanism which provides theoretical guarantees for identifying similar records which consist of strings. Towards this end, we put forth the novel notion of embedding string-based records into a Hamming space, where such a mechanism exists. The size of these embeddings is kept as small as needed in order to guarantee the correspondence of distances in that space to the types of errors that exist between strings, e.g., a missing or a modified character. We build embeddings whose size is 120 bits for representing accurately four fields of a publicly available data set. We also present a distance threshold-aware blocking technique for higher accuracy rates compared to blocking approaches which ignore the specified threshold. Our empirical study conducted on real-world data sets shows the efficacy achieved by our embedding method as compared to several existing solutions.

1. INTRODUCTION

The integration of data from disparate sources is increasingly being required as an important step towards the identification of similar entities across different sources. Known as entity resolution, record linkage, and data matching, the process of integrating data is an important problem in many data mining and knowledge engineering applications [10]. A wide range of real-world applications, including health-care, government services, crime and fraud detection, national security, and businesses, require entity resolution techniques in order to enrich data quality and empower accurate decision making [2].

Since unique entity identifiers, which would allow a simple join between records, are often not available in databases, a common practice is to use personal identifying attributes, such as names and addresses. Due to the quadratic complexity of the number of comparisons required and the commonly large volumes of records, a two-step process is followed [29]. In the first step, a blocking mechanism is applied, which reduces the comparison space efficiently by creating blocks with potentially similar records. Then, in the second step, only the records within the same block are compared with each other. Moreover, the values of these attributes, which are well correlated with the entities being linked, often contain variations, errors, and misspellings which require the use of approximate matching solutions [2].

A widely adopted criterion that is used to determine the similarity between the string values of these attributes is their edit distance [20], which is the minimum number of character edit operations required to transform one string value into the other. Unfortunately, thus far there does not exist an efficient blocking/matching mechanism that works directly on records, which contain string values, and simultaneously provides theoretical guarantees for identifying all similar record pairs using the edit distance as the metric for determining their similarity. Furthermore, computing the edit distances for a large number of record pairs imposes a considerable non-negligible overhead. This can be tolerated for the traditional context of an off-line process but is not suitable for many emerging recent applications that require nearly real-time analysis, especially if they involve streaming data [5, 33].

For these reasons, a common practice is to embed string values into a metric space where such an efficient blocking/matching solution exists. For example, Hamming [17] and Jaccard [18] Locality-Sensitive Hashing (LSH) based blocking/matching mechanisms work in a Hamming and in a Jaccard space, respectively. Representing a string as a small-sized binary sequence in a Hamming space results in a particularly lightweight structure. Moreover, Hamming distance, which is the number of bits in which two binary sequences differ, can be computed very fast. These two features render those embeddings a perfect fit for distributed and real-time settings. One such example of a real-world application is a health surveillance system that continuously integrates data from hospitals and pharmacy stores by performing a large number of distance computations in real-time.

Also, during the matching step, it is common practice to follow a decision model by applying to each record pair a rule, which classifies a pair of records as a matching or as a non-matching pair according to some distance thresholds specified for each attribute. Setting such a threshold arbitrarily or on an empirical basis may either impose additional unnecessary running time or generate incomplete results. LSH-based blocking mechanisms [17, 18] consider each record as an entity ignoring completely such classification rules during the blocking step. This record-level approach falls short in the presence of such rules, especially when different thresholds are specified for each attribute.

In this paper, we propose an embedding method of strings into a compact binary Hamming space where both the embeddings are of small size and the distances in that space correspond to certain types of errors, e.g., an accidentally deleted character, between strings. Therefore, one can specify accurately the threshold(s) required by the used blocking/matching mechanism in the embedding space. Furthermore, we adapt the blocking mechanism to the used classification rule and report the formal guarantees provided for identifying any similar pair by using the newly adapted mechanism. To the best of our knowledge, such an attribute-level LSH-based blocking/matching technique has not been proposed in the literature before.

The contributions of this paper are:

- An efficient embedding method of strings into a compact Hamming space resulting in lightweight, in terms of size, embeddings.
- A guaranteed correspondence of distances in the embedding space to certain types of errors between strings.
- An attribute-level LSH-based blocking scheme, which adapts to the used classification rule.
- An experimental evaluation of the proposed method compared with existing embedding solutions and using real-world data sets.

In the next section, we review the relevant literature, and in Section 3 we formulate the problem and motivate its importance. We outline the building components of our proposed method in Section 4, and in Section 5 we describe this method in detail. We empirically evaluate and compare our approach with existing embedding solutions in Section 6, and we conclude this paper with future research directions in Section 7.

2. RELATED WORK

A long line of research has been conducted in record linkage and various methods for computing similarities between records in an approximate manner have been proposed. We refer the interested reader to some recent surveys [2, 10]. Many techniques have been developed for the blocking and matching step aimed at reducing the comparison space and identifying as many similar record pairs as possible [3, 10].

For the blocking step, several approaches [6, 8, 12, 34] have been developed with the aim of being scalable to large data sets without sacrificing quality. Nevertheless, there are two methods which had great impact on the research community. The first is the sorted neighborhood method [12], including all its variants, which first sorts all records from the participating data sets and then uses a fixed-sized sliding window over the sorted records in order to compare the pairs which are formulated within that window. The second is the canopy clustering technique [6] that relies on the idea of using a computationally cheap clustering approach to create high-dimensional overlapping clusters, from which blocks of candidate record pairs can then be generated. These methods though do not provide any guarantees for identifying record pairs that are similar nor scale well to large volumes of records.

Recently, randomized blocking/matching techniques, which mainly rely on Locality-Sensitive Hashing (LSH) [1], have received much attention [9, 15, 17, 18]. These techniques work in some metric space into which string values are embedded preserving their initial distances as accurately as possible. The most appealing property of these distance-based randomized techniques is that they provide theoretical guarantees for identifying each similar record pair in the embedding space with high probability.

For the matching step, a large body of work has also been conducted on similarity joins [35, 36, 11, 21, 24, 25, 30, 31, 32] where efficient and scalable approximate joins are facilitated by using several metrics during the matching step such as the edit, Jaccard, and cosine metrics [2]. Especially, the authors in [35, 21, 25, 30, 31] have devised efficient techniques for finding similar string values using the edit distance metric, however they focus on individual such values, whereas our work proposes a solution for finding similar records, which usually consist of multiple strings. All the above-mentioned metrics though use strings or high-dimensional vectors of integers, which are not suitable structures for highly demanding environments either in terms of communication or of computational cost. Three other state-of-the-art embedding methods, introduced in [14, 18, 27], are used as our competitors and are presented in detail in Section 6.

3. PROBLEM DEFINITION

Let us assume that two data custodians, who own databases A and B, respectively, engage themselves into a process for identifying the common entities among their records. They are allowed to make use of the services offered by an independent party, whom we call Charlie. The two data custodians agree to use a common set of n_f attributes, each denoted by f_i where $i = 1, \ldots, n_f$, based on which they can exchange and compare their records. We denote by $u_{\mathcal{E}}^{(f_i)}$ the distances between the values of attribute f_i, and by $\vartheta_{\mathcal{E}}^{(f_i)}$ the specified threshold for this attribute. They also need to provide an additional attribute, let us call it *Id*, for the role of an identifier of each record. The data custodians submit their records to Charlie whose duty is to identify any pairs of *similar* records that belong to different data sets.

Definition 1 (A Similar record pair). *A record pair $r_A \in A$ and $r_B \in B$ is considered as similar if for each attribute f_i, it holds that $u_{\mathcal{E}}^{(f_i)} \leq \vartheta_{\mathcal{E}}^{(f_i)}$ in the metric space $(\mathcal{E}, d_{\mathcal{E}})$, where $\mathcal{E}$ is the original space in which the string values of all records of A and B exist, and $d_{\mathcal{E}}$ is the edit distance used as the metric on $\mathcal{E}$.*

Due to the generally large size of the data sets at hand, Charlie will use a randomized blocking/matching mecha-

Algorithm 1 Mapping a q-gram to a position of a q-gram vector.

```
Input: a $q$-gram $gr$.
Output: The index $ind$ in the $q$-gram vector.
1: $ind = 0$
2: for $i = 1, \dots, q$ do
3:     $ch = gr[i]$        // Extract each character $ch$ from $gr$.
4:     $ind = ind + ord(ch) \times |S|^{q-i}$        // Function $ord(\cdot)$
       returns the order (zero-based) of character $ch$ in $S$.
5: end for
```

nism, which both handles efficiently large volumes of data, and provides theoretical guarantees of performance in approximate matching by identifying each similar record pair with a specified (high) probability. However, such known mechanisms work in different metric spaces than $\mathcal{E}$. For this reason, Charlie embeds the string values into a Hamming metric space $(\mathcal{H}, d_{\mathcal{H}})$, where such a mechanism already exists. Additionally, there should exist a guaranteed correspondence between distances in $\mathcal{H}$ and the type of errors in $\mathcal{E}$ by keeping the size of the embeddings in $\mathcal{H}$ as small as possible. Given this correspondence, the thresholds required can be easily specified in $\mathcal{H}$, which will result in identifying each pair of records regarded as similar in $\mathcal{E}$.

Definition 2 (An Efficient Embedding Method). *Given an efficient blocking/matching mechanism in $\mathcal{H}$, construct an embedding method of strings from $\mathcal{E}$ into $\mathcal{H}$, where errors in $\mathcal{E}$ are identifiable in $\mathcal{H}$ separately for each attribute f_i using embeddings whose sizes are as small as possible depending on the lengths of the strings in f_i.*

4. BACKGROUND

In this section, we outline the building blocks utilized by our proposed scheme.

4.1 q-gram Vectors

A q-gram vector is a deterministic structure for representing a string value in the Hamming space. Such structures have been used in [18] and [19] for representing distinct attribute values and whole records, respectively. Each position of a q-gram vector represents a distinct q-gram which is a group of q consecutive characters in a string value. By assuming that the alphabet S of q-grams is the set of the upper-case letters, the size of a q-gram vector is $m = |S|^q = 26^q$ positions. Let us denote a bijection $F : \{gr_1, gr_2, \dots, gr_m\} \to \{0, \dots, m-1\}$, which maps each q-gram for a certain alphabet S to an integer, termed also as the index ind of that q-gram. The logic behind F is illustrated in Algorithm 1. Therefore, by this mapping, we obtain a set of indexes, denoted by U_s, that indicates which positions of the respective q-gram vector will be set to 1. Figure 1 illustrates how a string is represented by a bigram[1] vector. A record-level q-gram vector is built by concatenating the corresponding attribute-level q-gram vectors and its size is $\overline{m} = n_f \times m$. The space in which these record-level q-gram vectors exist is $\mathcal{H} = \{0,1\}^{\overline{m}}$.

4.2 Hamming LSH-based Blocking/Matching

Due to the large number of records that occur in many of today's databases, the randomized Hamming Locality-Sensitive Hashing (LSH) technique [1], denoted by HB, is

[1]Bigrams are the q-grams with $q = 2$.

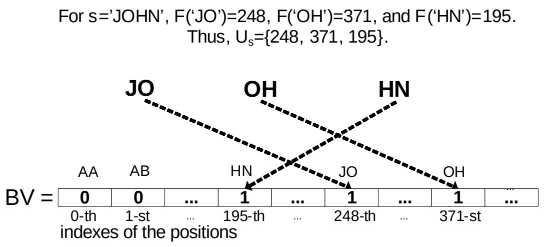


Figure 1: Representing the string *'JOHN'* as a bigram vector denoted by BV.

Table 1: Interpretation of the most used variables throughout this paper.

T_l	An independent blocking group where $l = 1, \dots, L$ used by the HB.
h_l	A composite hash function used to specify the bucket of some T_l into which a c-vector, defined in Section 5.2, is stored.
K	The number of base hash functions used by a h_l.
U_s	The set of indexes of the respective q-grams of a string s.
$d_{\mathcal{S}}$	The metric applied on space $\mathcal{S}$ where $\mathcal{S} \in \{\mathcal{H}, \widehat{\mathcal{H}}, \mathcal{E}, \mathcal{J}\}$.
$u_{\mathcal{S}}$	Distance measured by using metric $d_{\mathcal{S}}$.
$\vartheta_{\mathcal{S}}$	The specified distance threshold in space $\mathcal{S}$.
$\overline{m}_{opt}$	The optimal size of a record-level c-vector.
n_f	The number of common attributes which participate in the linkage process.
f_i	An attribute where $i = 1, \dots, n_f$. When used as a superscript in parentheses, it denotes the attribute-level value, e.g., $u_{\mathcal{H}}^{(f_i)}$, $K^{(f_i)}$, $h_l^{(f_i)}$ etc.
$b^{(f_i)}$	The average number of q-grams of the values of the corresponding attribute f_i.

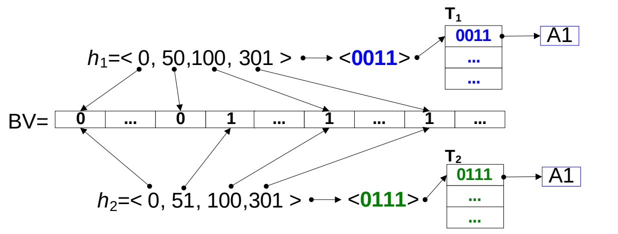
used as the blocking/matching technique in order to identify each similar pair of record-level q-gram vectors, which exist in $\mathcal{H}$, with high probability. Mechanism HB utilizes L independent hash tables, termed also as blocking groups. Each hash table, denoted by T_l where $l = 1, \dots, L$, consists of key-bucket pairs where a bucket hosts a linked list which is aimed at grouping similar q-gram vectors. Moreover, each hash table has been assigned a composite hash function h_l which consists of a fixed number K of base hash functions. A base hash function applied to a q-gram vector returns the value of its j-th position where $j \in \{0, \dots, \overline{m}-1\}$ chosen uniformly at random.

Definition 3 (A Hamming LSH family). *A family ϕ of composite hash functions has the following key property for any pair of record-level q-gram vectors denoted by $QV_1, QV_2 \in \mathcal{H}$ whose Hamming distance is $u_{\mathcal{H}}$ [1]:*

$$\textit{If } u_{\mathcal{H}} \le \vartheta_{\mathcal{H}} \textit{ then } \Pr[h_l(QV_1) = h_l(QV_2)] \ge p^K, \quad (1)$$

where p denotes the success probability of a base hash function and is equal to $p = 1 - \frac{\vartheta_{\mathcal{H}}}{\overline{m}}$.

Intuitively, the smaller the Hamming distance is, the higher the probability for a h_l to produce the same result. The result of a h_l, which constitutes the blocking key, applied to



T1
0011
...
...
A1
h1=< 0, 50,100, 301 > <0011>
BV= 0 ... 0 1 ... 1 ... 1 ...
T2
0111
...
...
A1
h2=< 0, 51, 100,301 > <0111>

Figure 2: Hashing a record-level bigram vector BV, with Id='A1', by h_1 and h_2. For illustration purposes, we set $K = 4$ and $L = 2$.

a q-gram vector specifies into which bucket, termed also as block, this q-gram vector will be stored[2]. Figure 2 illustrates how a record-level bigram vector is hashed.

During the matching step, we scan the buckets of each T_l and formulate pairs of q-gram vectors, which belong to different data sets. By using this redundant blocking scheme, we amplify the probability of identifying similar q-gram vectors, but we also increase both the utilized space and the running time in order to store the generated T_l's. We therefore determine the optimal number of the T_l's that should be utilized by setting [1]:

$$L = \lceil \frac{\ln(\delta)}{\ln(1 - p^K)} \rceil. \quad (2)$$

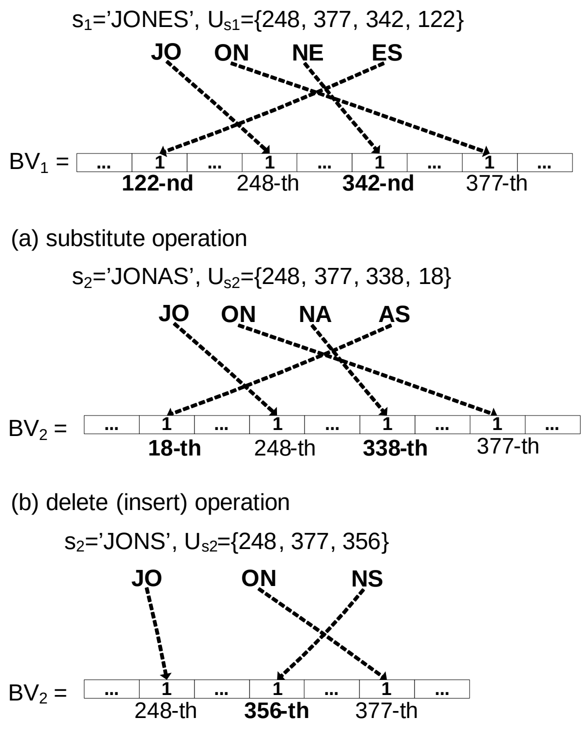
Each similar q-gram vector pair will be returned with high probability $1 - \delta$, as δ is usually set to a small value, say $\delta = 0.1$. The value for K can be set empirically since the correctness of the scheme is guaranteed by setting L appropriately. In [16], a method for choosing the optimal value for K is presented, where the authors by sampling record pairs and by experimenting with several values for K, choose the value that minimizes the estimated running time. The value of K can be set empirically since the completeness, with respect to the identification of the matching pairs, of the mechanism is guaranteed by Equation (2), by deriving the optimal value for L. The value of K should be sufficiently large because otherwise the blocking keys will not reflect the variations of the bit sequences of the q-gram vectors. The direct side-effect of this deficiency will be the generation of a small number of buckets in each T_l, which will be overpopulated by mostly dissimilar pairs.

Not surprisingly though, q-gram vectors render inefficient and cumbersome the HB mainly due to their sparsity as will be explained in Section 5.2. In that section, towards mitigating this sparsity and reducing their size, we propose an alternative embedding scheme of the string values into $\mathcal{H}$ in order to leverage the efficiency of HB.

5. AN EFFICIENT EMBEDDING METHOD

In this section, we instantiate our method, termed as cBV-HB, which relies on embedding string values by using their respective q-grams into a compact Hamming space. The reason for choosing this space is twofold. Firstly, we argue that by corresponding types of errors in $\mathcal{E}$ to distances in $\mathcal{H}$, one can easily specify the distance threshold(s) required by HB. Secondly, the HB mechanism is a fast and accurate method as experimentally demonstrated in [17], and it works in Hamming spaces. We next (a) illustrate the correspondence between distances in $\mathcal{H}$ and types of errors in $\mathcal{E}$ by using the q-gram vectors, (b) propose an embedding method which can be used by HB for efficient identification of similar record pairs, and (c) present an attribute-level LSH-based blocking technique which brings the importance of the classification rule into the blocking step.

[2]More precisely, we store only the corresponding Id's.



s1='JONES', Us1={248, 377, 342, 122}
JO ON NE ES
BV1 = ... 1 ... 1 ... 1 ... 1 ...
122-nd 248-th 342-nd 377-th
(a) substitute operation
s2='JONAS', Us2={248, 377, 338, 18}
JO ON NA AS
BV2 = ... 1 ... 1 ... 1 ... 1 ...
18-th 248-th 338-th 377-th
(b) delete (insert) operation
s2='JONS', Us2={248, 377, 356}
JO ON NS
BV2 = ... 1 ... 1 ... 1 ...
248-th 356-th 377-th

Figure 3: The indexes in bold indicate the differing bigrams between s_1 and s_2, which result in distances being equal to 4 and 3 in $\mathcal{H}$ for the substitute and delete (insert) operations, respectively ($m = 676$).

5.1 Corresponding Types of Errors in $\mathcal{E}$ to Distances in $\mathcal{H}$

An error between a pair of string values is formally quantified by edit distance. In the literature, several variants of edit distance exist using different perturbation operations, as the errors are termed in the edit distance context[3]. We consider the basic perturbation operations (substitute, insert, and delete) defined for the Levenhstein distance [20] and correspond these basic operations to distances in $\mathcal{H}$ by using the q-gram vectors. We present an illustrative running example where the initial error-free string value is s_1='*JONES*', unless otherwise stated and we set $q = 2$ (bigrams). The perturbed values are stored in variable s_2. In this example, we use two bigram vectors, denoted by BV_1 and BV_2 for representing s_1 and s_2, respectively.

Substitute perturbation operation: This type of perturbation operation changes a single character in s_1 and materializes the main reason for errors and misspellings commonly found in string values. Assume the value s_2='*JON**A**S*',

[3]We use the terms *perturbation operation* and *error* interchangeably. Usually, a perturbation operation occurs intentionally and an error unintentionally.

the distance $u_{\mathcal{H}}$ between BV_1 and BV_2 is 4, as shown in Figure 3, due to the 4 differing bigrams, which are *'NE'* and *'ES'* in s_1 and *'NA'* and *'AS'* in s_2[4]. Distance may be smaller in case a differing bigram overlaps with a common bigram. For instance, by perturbing s_1 = *'SH**A**NNEN'* as s_2 = *'SH**E**NNEN'*, we produce two differing bigrams in s_1, which are *'HA'* and *'AN'*, and two more in s_2 namely *'HE'* and *'EN'*. The latter though overlaps with a common bigram found in both s_1 and s_2. Therefore, only bigrams *'HA'*, *'AN'*, and *'HE'* affect $u_{\mathcal{H}}$ which in this case is 3. We conclude that for this type of perturbation operation $u_{\mathcal{H}} \leq 4 \times u_{\mathcal{E}}$.

Delete and insert perturbation operations: Another common error that emerges when typing string values is the omission of a character. This causes the generation of a smaller number of bigrams for s_2. As an illustration, by setting s_2 = *'JONS'*, we get two differing bigrams in s_1, which are *'NE'* and *'ES'* and only 1 in s_2, which is *'NS'* (see Figure 3). Hence, for the delete operation, it holds that $u_{\mathcal{H}} \leq 3 \times u_{\mathcal{E}}$. Likewise, the insert is quite similar to the delete operation (s_2 = *'JONE**A**S'*) since it is essentially as a delete operation in s_1.

The above-mentioned observations, which hold for any q-gram vector pair with $q \geq 2$, lead to the definition of an upper bound for a distance $u_{\mathcal{E}}$ which corresponds to a $u_{\mathcal{H}}$ scaled by a constant factor α:

$$u_{\mathcal{H}} \leq \alpha \times u_{\mathcal{E}}. \quad (3)$$

The factor α depends on the type of the applied perturbation operation, as explained before, and determines the deviation between distances from $\mathcal{E}$ to $\mathcal{H}$, commonly termed as *distortion* [13].

In the Jaccard space $\mathcal{J}$, which consists of the sets U_s where s stands for each possible string value, by using the Jaccard metric [2] the distance between s_1 = *'JON**E**S'* and s_2 = *'JON**A**S'* is $u_{\mathcal{J}} = 1 - |U_{s_1} \cap U_{s_2}|/|U_{s_1} \cup U_{s_2}| \simeq 0.667$. Comparing though s_1 = *'WASH**I**NGTON'* with s_2 = *'WASH**A**NGTON'*, the distance is affected by the length of the strings resulting in $u_{\mathcal{J}} \simeq 0.364$. In contrast, the Hamming distance is constantly $u_{\mathcal{H}} = 4$ in both cases. Hence using the Jaccard metric, one should take into account the length of strings in order to set the threshold appropriately. This task is not easy or sometimes is not feasible at all.

5.2 Embedding Strings into a Compact Space

In this subsection, we propose an embedding method, which preserves the distances from $\mathcal{E}$, into a compact space that consists of small-sized embeddings. These embeddings can be particularly useful in highly demanding distributed environments, which deal with large volumes of records, for efficient communication. Our method adapts the size of the q-gram vectors to the expected number of characters that a record holds as needed. For example, the average number of bigrams for the *LastName* attribute of the NCVR database [4], which is a large publicly available data set that we will use in our experiments in Section 6, is only 5.0 bigrams. Therefore, the generated attribute-level bigram vectors would be quite sparse due to the few bigrams produced by each string value. This sparsity though has negative effects during the application of the HB. The h_l's by sampling randomly bit positions from such q-gram vectors mostly choose 0's, which has as a side-effect the formulation of a small number of overpopulated buckets. Thus, the HB boils down to an inefficient *all-pairs* comparison process. Moreover, by assuming n_f attributes, the size of a record-level q-gram vector would be quite large, namely $\mathcal{O}(n_f \times m)$ bits. On account of these drawbacks, we embed the string values of each attribute into a new space $\widehat{\mathcal{H}}$ which also uses the Hamming metric. This space consists of compact q-gram vectors, termed as *c-vectors*, which are of size $m^{(f_i)} \ll |S|^q$, that will be exactly specified later, for each attribute f_i. We introduce the dependency of the size of c-vectors on the average number $b^{(f_i)}$ of q-grams of the values of the corresponding attribute f_i. This dependency will allow us to be as efficient as possible by adjusting the sizes accordingly and simultaneously preserving the distances from $\mathcal{H}$. Towards this end, we hash the indexes in U_s of a string value s by randomly chosen, pairwise independent hash functions of the form $g(x) = [(ax+b) \mod P] \mod m$, where $x \in U_s$, P is a large prime number (e.g., $2^{31}-1$) and a, b are randomly chosen integers from $(0, P)$.

Figure 4 shows the creation of a c-vector by hashing the bigrams of a string. However, during the hash operations of the elements of U_s, a number of collisions may occur if two elements of U_s hash to the same index in the c-vector. This happens because the number of all possible q-grams is much larger than $m^{(f_i)}$. A collision is formally defined as $g(x) = g(y)$ for any $x, y \in U_s$ with $x \neq y$ and the probability $\Pr[g(x) = g(y)]$ is $\frac{1}{m^{(f_i)}}$. By considering the guarantees quoted in the previous section, the collisions, in which differing q-grams of a pair of c-vectors participate, affect the distances in $\widehat{\mathcal{H}}$. These collisions result in misleadingly classifying non-matching as matching pairs.

As an illustration, let us assume the c-vectors with $q = 2$, generated by the values s_1 = *'JON**E**S'* and s_2 = *'JON**A**S'* of an attribute f_i. We expect the distance to be $u_{\mathcal{H}}^{(f_i)} = u_{\widehat{\mathcal{H}}}^{(f_i)} = 4$ due to the differing bigrams generated by the suffices *'NES'* and *'NAS'*. However, if during the hash operations of s_2, the results corresponding to the bigrams *'NA'* and *'AS'* collide, then the $u_{\widehat{\mathcal{H}}}^{(f_i)}$ will be 1 bit less than the $u_{\mathcal{H}}^{(f_i)}$. Therefore, the value for $m^{(f_i)}$ should be adequately specified so that both the HB can be efficiently applied and the distances should be preserved.

The phenomenon of collisions is described in the Birthday Paradox Problem [23] on which the following lemma relies. In the calculations below, we drop superscript (f_i) for better readability.

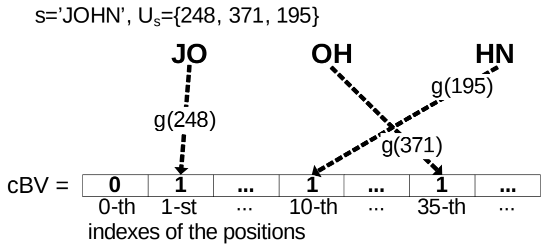


Figure 4: Representing the string *'JOHN'* as a c-vector denoted by cBV.

[4]We pad the first and the last character, e.g., *'_JONES_'* in order to include all the characters in 2 bigrams.

LEMMA 1. *The expected number of collisions $E[c]$ by hashing b q-grams for attribute f_i to a c-vector with size m is:*

$$E[c] = b - E[v], \tag{4}$$

where $E[v]$ denotes the expected number of positions which both hold 1 and no collisions have occurred.

PROOF. *The indexes of positions of a c-vector, representing a string s of an attribute f_i, are uniformly chosen by $g(\cdot)$, therefore the probability of choosing any position for each $x \in U_s$ is $\frac{1}{m}$. Let the indicator variable I_j denote for the position with index j, where $j \in \{0, \ldots, m-1\}$, the content in that position. The probability that a certain position with index j is not chosen ($I_j = 0$) after hashing b q-grams is:*

$$\Pr[I_j = 0] = (1 - \frac{1}{m})^b. \tag{5}$$

Thus, the probability that the position with index j is finally chosen is $\Pr[I_j = 1] = 1 - \Pr[I_j = 0]$. The expected number v of positions holding 1 is:

$$E[v] = \sum_{j=0}^{m-1} E[I_j] = mE[I_j] = m(1 - (1 - \frac{1}{m})^b). \tag{6}$$

Then, by subtracting $E[v]$ from b, we arrive at the desired result. □

Let us denote by ρ the maximum number of collisions we can tolerate during the generation of the c-vectors. Then, using ρ and the above lemma, we state the following theorem:

THEOREM 1. *By expecting b q-grams in the corresponding strings, the optimal size of the c-vectors for some attribute f_i is:*

$$m_{opt} = \lceil \frac{b - \rho}{1 - e^{-r}} \rceil, \tag{7}$$

where $E[c] \leq \rho$ with confidence $1 - r$.

PROOF. *We first specify the value of ρ and expand Equation (4) by using Equation (6) as follows:*

$$\begin{aligned} E[c] = b - m(1 - (1 - \frac{1}{m})^b) \leq \rho \Rightarrow \\ m(1 - (1 - \frac{1}{m})^b) \geq b - \rho. \end{aligned} \tag{8}$$

By using the fact that $-(1 - \frac{1}{m})^b \geq -e^{\frac{-b}{m}}$, we derive an upper bound for the left hand side of the second inequality in (8) which is written as:

$$m(1 - e^{-\frac{b}{m}}) \geq b - \rho. \tag{9}$$

We then substitute $\frac{b}{m}$ with a constant r, where $r < 1$ since it always holds that $b < m$. This constant denotes the ratio between the number of q-grams to the size m. Intuitively, smaller values for r increase our confidence, quantified as $1-r$, that collisions will not occur at the cost of a larger size m. Finally, we solve for m in (9), and derive the optimal size, as illustrated in Equation (7), where we keep only the equal sign, since we want to be as optimal as possible. □

In Section 6, we show experimentally that by setting $r < 1/3$, we just increase m_{opt} without earning a lot in terms of accuracy.

Algorithm 2 Matching the c-vector pairs formulated in the buckets of the T_l's.

```
Input: $cBV_B \in B$
1: $\mathcal{C} \leftarrow$ new $UniqueCollection()$
2:         // $\mathcal{C}$ is a collection of unique $Id$'s.
3: for $l = 1, \ldots, L$ do
4:     $Id\_list \leftarrow T_l.get(h_l(cBV_B))$
5:         // Object $Id\_list$ is a linked list of $Id$'s.
6:     for $i = 1, \ldots, Id\_list.size()$ do
7:         $Id \leftarrow Id\_list[i]$
8:         if ($not$ $\mathcal{C}.contains(Id)$) then
9:             $cBV_A \leftarrow retrieve(Id)$
10:            $rule(cBV_A,\ cBV_B)$
11:        // A classification rule applied for each $c$-vector pair.
12:            $\mathcal{C}.add(Id)$
13:        end if
14:    end for
15: end for
```

Table 2: Primitive operations used by Algorithm 2.

$get(x)$	Return the linked list, to which the specified key x is mapped.
$size()$	Return the size of a linked list.
$contains(x)$	Return *true* if the unique collection contains element x.
$retrieve(x)$	Retrieve a c-vector with $Id = x$ from the data store.
$add(x)$	Add value x to a unique collection.

For example, by assuming $b^{(f_1)} = 5.1$ and $b^{(f_3)} = 20.0$ from Table 3, by setting in (7) $\rho = 1$ and $r = 1/3$, we derive values $m_{opt}^{(f_1)} = 15$ and $m_{opt}^{(f_3)} = 68$, respectively. We use the ceiling function $\lceil \cdot \rceil$ to $m_{opt}^{(f_i)}$ because the size of a c-vector should be an integer.

For each attribute, Charlie transforms the strings he receives from Alice and Bob into c-vectors using the optimal size $m_{opt}^{(f_i)}$ by sampling randomly and uniformly strings from the data sets and computing $b^{(f_i)}$. By concatenating the attribute-level c-vectors, Charlie then builds the record-level structures, whose size $\overline{m}_{opt}$ is compact and adapted to the needs of each attribute.

5.3 Outline of the Blocking/Matching Step

Let us denote by cBV_A and cBV_B the record-level c-vectors which belong to data sets A and B, respectively. We first hash each cBV_A and store its Id in the buckets of the T_l's. Then, we hash each cBV_B to the T_l's in order to formulate c-vector pairs for performing the distance computations. Due to the *redundant* blocking model that we follow, certain pairs of c-vectors might be formulated in several T_l's. On account of this redundancy, we incorporate a de-duplicating mechanism in HB in order to prevent the repetitive distance computations of duplicate pairs as can be seen in Algorithm 2. For each bucket that cBV_B maps to, we retrieve the Id's already stored therein (line 4), and query them against a collection of unique elements[5] (line 8). If an Id is not found in that collection, then the corresponding distance computation is performed otherwise it is dropped. Table 2 quotes a description of each primitive operation used by Algorithm 2. We have to note that our method is capable of handling an arbitrary number of data

[5]This collection is instantiated by a *HashSet* object in Java programming language.

sets (two or more) belonging to different data custodians.

5.4 Attribute-level LSH-based Blocking

Mechanism HB assumes record-level c-vectors where by sampling randomly and uniformly their bits builds the T_l's. During the matching step, a decision model is applied in order to classify the formulated record pairs as matching or as non-matching. In its simplest form, a decision model might be a classification rule, which applies a logical condition to the values of each attribute by comparing them to an attribute-level threshold. Therefore, there is no guarantee that c-vector pairs are formulated according to the classification rule during the blocking step. For instance, by using the attributes in Table 3, a classification rule might be $(u^{f_1} \leq 4) \wedge (u^{f_2} \leq 8)$[6]. This rule is more strict to errors in the values of the first attribute while being more tolerant to errors in the values of the second attribute. HB though is unaware of that rule, and uses the underlying values of attributes on an equal basis. In this subsection, we propose a method for adjusting the HB mechanism to the classification rule. This adjustment has as a result the formulation of c-vector pairs which are much closer in terms of distance to the logic of the classification rule.

To begin with, the hash functions during the blocking step should use each attribute separately rather than sampling bits uniformly from the record-level c-vectors. To this end, we choose a value for each attribute-level $K^{(f_i)}$, in the same sense as we have described in Section 4.2. A $K^{(f_i)}$ specifies the number of base hash functions for each $h_l^{(f_i)}$ which work on the attribute-level c-vectors corresponding to attribute f_i. By assuming (a) n_c attributes, where $n_c \leq n_f$, that participate in each rule, (b) independence among the string values of each attribute, and (c) the attribute-level success probability of a base hash function is $p^{(f_i)} = 1 - \frac{\vartheta^{(f_i)}}{m_{opt}^{(f_i)}}$, we state the following definitions for any pair of record-level c-vectors that exhibit distances $u^{(f_i)} \leq \vartheta^{(f_i)}$ as follow:

Definition 4 (AND operator). *By using the AND operator ($\wedge$) on certain attributes in the classification rule, the probability of a record-level c-vector pair to collide into the same bucket of a T_l is:*

$$p_\wedge \geq \prod_{i=1}^{n_c} (p^{(f_i)})^{K^{(f_i)}}. \tag{10}$$

Using the AND operator, the structure of the blocking groups used, described in Section 4.2, is maintained. The blocking keys for each f_i are concatenated resulting in a compound blocking key that is finally inserted into some T_l.

Definition 5 (OR operator). *By using the OR operator ($\vee$) on certain attributes in the classification rule, the probability of a record-level c-vector pair to collide into the same bucket of any $T_l^{(f_i)}$ is:*

$$p_\vee \geq (p^{(f_1)})^{K^{(f_1)}} + (p^{(f_2)})^{K^{(f_2)}} - (p^{(f_1)})^{K^{(f_1)}} \times (p^{(f_2)})^{K^{(f_2)}}. \tag{11}$$

Without loss of generality, in Equation (11), we show only the case where $n_c = 2$ attributes. For a larger number n_c of attributes, the inclusion-exclusion principle [22] should be used.

[6]We drop the space subscript from distances and thresholds, since from now on we focus on $\widehat{\mathcal{H}}$.

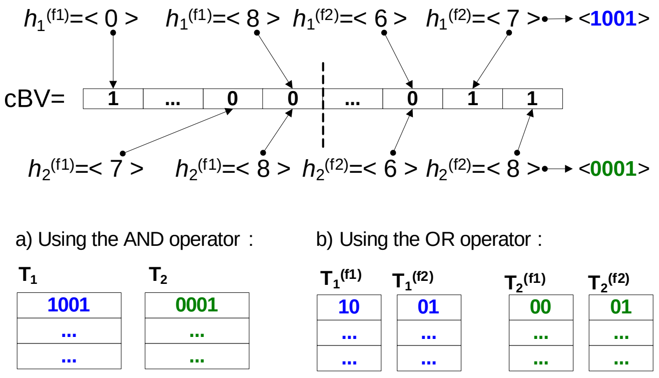


Figure 5: Applying attribute-level hashing to a c-vector cBV which consists of 2 attribute-level c-vectors. For illustration purposes, we set $K = 4$ ($K^{(f_i)} = 2$) and $L = 2$.

When using the OR operator, the structure of the blocking groups changes considerably. For each attribute f_i, which is part of the OR rule, for each blocking group, we build an independent hash table, denoted by $T_l^{(f_i)}$, that stores the blocking keys of this specific attribute. Therefore, given n_c attributes in the OR rule, we end up with $L \times n_c$ hash tables. Intuitively, one would classify a c-vector pair as a matching one, if this pair is formulated in at least one $T_l^{(f_i)}$, regardless of the remaining outcomes.

Definition 6 (NOT operator). *By using the NOT operator ($\neg$) on a certain attribute, the probability of any pair of record-level c-vectors not to collide into the same bucket of a $T_l^{(f_i)}$ is:*

$$p_\neg \geq 1 - (p^{(f_i)})^{K^{(f_i)}}. \tag{12}$$

The NOT operator assumes one attribute f_i and thus one hash table for each blocking group. One would assume the *true* value as outcome, if a certain pair has not been formulated in the corresponding $T_l^{(f_i)}$'s.

These revised probability bounds adjust the number L of the blocking groups[7] used by substituting p^K in Equation (2). The new value of L is larger using an AND rule, and smaller using an OR rule than the standard record-level LSH-based blocking approach. Using these operators, we build the basic classification rules which are depicted in Figure 5. The blocking group of the NOT operator does not include any modifications because we just change what we consider as a true outcome.

In addition, by using these basic rules and their corresponding blocking groups, we may compose compound classification rules which consist of several *subrules*. Such compound rules might be:

- $C_1 = [(u^{(f_1)} \leq \vartheta^{(f_1)}) \wedge (u^{(f_2)} \leq \vartheta^{(f_2)})] \vee [(u^{(f_3)} \leq \vartheta^{(f_3)}) \wedge (u^{(f_4)} \leq \vartheta^{(f_4)})]$ where two separate blocking structures for the AND subrules should be built. The first blocking structure comprises the blocking groups of attributes f_1, and f_2, while the second one contains the attributes f_3, and f_4. During the blocking mechanism, the blocking keys will be built from the

[7]The value of L may vary among the blocking structures used.

corresponding attribute-level c-vectors and will be inserted into the corresponding T_l's. Since an OR operator joins the two subrules, a pair will be returned if it will be formulated in the blocking structure of either subrule. Thus, during the matching phase, for each c-vector from data set B, we formulate all possible pairs in all the corresponding buckets. Then, a pair is considered as a matching one, if it is formulated in either blocking structure.

- $C_2 = [(u^{(f_1)} \leq \vartheta^{(f_1)}) \vee [(u^{(f_2)} \leq \vartheta^{(f_2)})] \wedge [(u^{(f_3)} \leq \vartheta^{(f_3)}) \vee (u^{(f_4)} \leq \vartheta^{(f_4)})]$ where four separate blocking structures for the OR operators should be built. The main difference between C_1 and C_2 is that using C_2, a pair should be formulated in both blocking structures of the subrules in order to be considered as a matching pair.

- $C_3 = (u^{(f_1)} \leq \vartheta^{(f_1)}) \wedge [\neg(u^{(f_2)} \leq \vartheta^{(f_2)})]$. Using C_3, a pair is returned if it is formulated in the blocking structure for f_1, but not found in the blocking structure for f_2.

The space needed for building the blocking groups of a rule using an AND operator is $\mathcal{O}(L)$, while using an OR operator is $\mathcal{O}(n_c \times L)$.

6. EVALUATION

In this section, we describe the experimental settings, the baseline methods, the data sets used, as well as the achieved results. We conducted experiments using two publicly available real-world databases which are (a) the NCVR database [4] and (b) the DBLP bibliography database[8]. By using both these data sets, which exhibit different properties such as the average lengths of string values, we obtain several insights through the experimental results. The attributes used are listed in Table 3.

We developed a software prototype which by using as input the above-mentioned databases, extracts records and creates two data sets, denoted by A and B, respectively, where one can specify the perturbation frequency, number of perturbation operations, and number of perturbed records in B for each chosen record in A. We apply (a) a light perturbation scheme, termed as *PL*, where we perturb the values of one randomly chosen attribute, and (b) a heavy scheme, termed as *PH*, where we apply one perturbation to the values of the first two attributes and two perturbations to the values of the third attribute. We notate the thresholds for each perturbation scheme as $\vartheta_{PL}^{(f_i)}$ and $\vartheta_{PH}^{(f_i)}$ regardless of the used space. The number of records in A (and B) is $1,000,000$, while the probability of choosing a record from A in order to apply a perturbation scheme and then place it in B, is set to 0.5. The experiments were executed on a dual-core Pentium PC with 32 GB of main memory. The software components are developed using the Java programming language (JDK 1.7) and are available from the authors.

Quality measures. The Pairs Completeness (*PC*), Pairs Quality (*PQ*), and Reduction Ratio (*RR*) measures [3] are employed to evaluate the quality of both our method and the baseline methods which are discussed in detail below. The set of truly matching record pairs is denoted by M and the set of identified matching pairs by $\mathcal{M}$. The accuracy in finding the matching record pairs is indicated by the *PC* measure, which is equal to $PC = |\mathcal{M} \cap M|/|M|$. The *PQ* measure shows the efficiency in generating mostly matching pairs with respect to candidate pairs, namely $PQ = |\mathcal{M} \cap M|/|CR|$, where CR is the set of candidate pairs. The *RR* metric indicates the percentage in the reduction of the comparison space $A \times B$, which is equal to $RR = 1.0 - |CR|/|A \times B|$. We ran each experiment 50 times and plotted the average values of these measures in the figures shown below.

[8]`http://dblp.uni-trier.de/xml/`

Table 3: Attribute-level parameters used for each type of data set by using bigrams.

	attribute	$b^{(f_i)}$	$m_{opt}^{(f_i)}$	$K^{(f_i)}$
NCVR	f_1=*FirstName*	5.1	15	5
	f_2=*LastName*	5.0	15	5
	f_3=*Address*	20.0	68	10
	f_4=*Town*	7.2	22	
			$\overline{m}_{opt} = 120$	
DBLP	f_1=*FirstName*	4.8	14	5
	f_2=*LastName*	6.2	19	5
	f_3=*Title*	64.8	226	12
	f_4=*Year*	3.0	8	
			$\overline{m}_{opt} = 267$	

6.1 Baseline Methods

We compare our approach *cBV-HB* with three state-of-the-art embedding approaches for record linkage. The first is the h-CC algorithm of ***HARRA*** [18], where two de-duplicated data sets are linked. In this approach, all attribute values of a record are represented by a single bigram vector. However, setting the same position of a bigram vector by identical bigrams, which belong to different attributes, may lead to ambiguous evaluation of distances and consequently to reduced accuracy. *HARRA* employs the Min-Hash LSH-based blocking/matching mechanism which uses the Jaccard metric, as described in Section 5.1, for performing the distance computations. Since *HARRA* selects arbitrary values for K and L, by experimenting for better results, we set $L = 30$ and $L = 90$ ($K = 5$) for each perturbation scheme, respectively. We performed several distance computations using the vector space that *HARRA* works, where one cannot focus on separate attributes, using perturbed string values and ended up choosing $\vartheta_{PL} = 0.35$ and $\vartheta_{PH} = 0.45$[9] as a nice balance between accuracy and efficiency. During the blocking phase, we hash those vectors by applying random permutations of their indexes and we choose the index of the minimum non-zero element of these permutations as the result of each base hash function. However, we mostly end up with an index holding 0, which implies that more elements of each permutation should be used until we find an index that is set to 1. As a result, similar records are inserted into different buckets. The blocking and matching mechanisms are conducted iteratively and separately for each T_l. When two records are classified as a matching pair, they are subsequently excluded from the remaining iterations.

[9]All thresholds are set after experimenting exhaustively using the initial and corresponding perturbed values.

Another method we compare our approach with is ***BfH*** presented in [17], which uses the HB, as described in Section 4.2, on Bloom filters. A Bloom filter is a data structure used to represent the elements of a set in order to support membership queries efficiently in terms of time and space required. It has been shown in [27] that by embedding string values into Bloom filters, distances from the original space are preserved. More specifically, a Bloom filter is a bitmap array initialized with zeros and created by hashing the bigrams of a string value by using independent composite cryptographic hash functions such as MD5 and SHA1 [26]. Field-level Bloom filters are created using a size of 500 bits by using 15 cryptographic hash functions for each bigram, as proposed in [27]. We set $K = 30$ and $\delta = 0.1$ while thresholds are set as $\vartheta^{(f_i)}_{PL} = 45$ ($L = 4$), $\vartheta^{(f_1)}_{PH} = \vartheta^{(f_2)}_{PH} = 45$, and $\vartheta^{(f_3)}_{PH} = 90$ ($L = 43$). We have to note that these attribute-level thresholds are used only during the matching step. A key observation for the Bloom filter space, which is a high-dimensional binary Hamming space, is that distances are affected by the number of bigrams. For example, using the field-level Bloom filters, described before, the Hamming distance between *'J**O**HN'* and *'J**A**HN'* is 54. In contrast, the Hamming distance between *'SC**A**LABILITY'* and *'SC**E**LABILITY'* is equal to 37. This variation in distance, although in both cases there exists a single error in the initial string values, causes difficulties in specifying effectively the distance threshold.

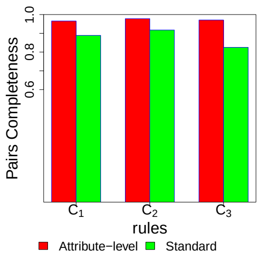
Finally, we compare *cBV-HB* with the ***StringMap*** algorithm [14] which is used to embed string values into a Euclidean space. Initially for each attribute, *StringMap* iterates the strings of both data sets in order to form d orthogonal directions (axes). Each such direction is specified by two strings, termed as pivots, whose distances are as far from each other as possible. Yet, the process of specifying the pivot values is quite expensive since it includes several iterations of the data sets. Then, for each string, we compute its coordinates on these d axes, which results in a vector of values of dimensionality d. As the authors suggest [14], dimensionality d is set to 20 for each attribute and thresholds $\vartheta^{(f_i)}_{PL} = 4.5$, $\vartheta^{(f_1)}_{PH} = \vartheta^{(f_2)}_{PH} = 4.5$, and $\vartheta^{(f_3)}_{PH} = 7.7$, for each scheme respectively. We utilize the Euclidean LSH-based blocking/matching mechanism [17], specifically developed for finding similar points in Euclidean spaces. As in *BfH*, the above-mentioned thresholds are used only during the matching step. The value of K is set to 5 which generates $L = 29$ and $L = 194$ blocking groups [7] for each perturbation scheme, respectively. We call this method ***SM-EB*** due to the combination of *StringMap* and the blocking/matching mechanism used.

6.2 Experimental Results

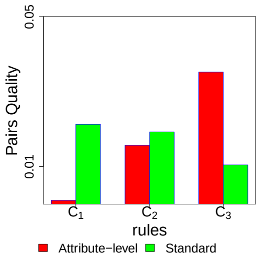
Accuracy. In the first series of experiments, for the rules:

- $C_1 = (u^{(f_1)} \leq \vartheta^{(f_1)}) \wedge (u^{(f_2)} \leq \vartheta^{f_2}) \wedge (u^{(f_3)} \leq \vartheta^{(f_3)})$,
- $C_2 = [(u^{(f_1)} \leq \vartheta^{(f_1)}) \wedge (u^{(f_2)} \leq \vartheta^{(f_2)})] \vee (u^{(f_3)} \leq \vartheta^{(f_3)})$, and
- $C_3 = (u^{(f_1)} \leq \vartheta^{(f_1)}) \wedge [\neg(u^{(f_2)} \leq \vartheta^{(f_2)})]$,

we measured the PC and PQ rates of our attribute-level blocking and compare them with the rates of the standard LSH-based approach, which, as described in Section 4.2, during the blocking phase samples bits uniformly from the



(a) Pairs Completeness rates.



(b) Pairs Quality rates.

Figure 6: Attribute-level Pairs Completeness and Pairs Quality evaluation using the NCVR-based data sets.

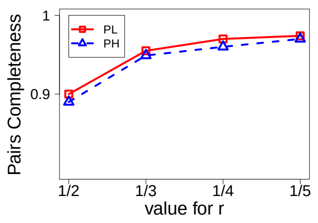
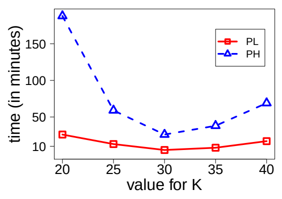


Figure 7: Evaluation of accuracy by setting several values to confidence r, from Equation (7), and measuring the PC rates using the NCVR-based data sets ($K = 35$).

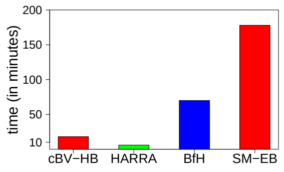
whole record-level c-vector. Clearly, Figure 6(a) shows that by using this rule-aware blocking phase, the PC rates are constantly higher than those using the standard LSH-based blocking approach. Nevertheless, the largest difference lies in C_3 where the standard approach is unable to articulate the NOT operator, comparing and discarding rather late any such non-matching pairs. Conversely, those pairs, in the rule-aware blocking phase, are not formulated at all and are never brought for comparison. The PQ rates, illustrated in Figure 6(b), for C_1 are lower than the standard approach due to the larger number of blocking groups required. For C_2, the utilization of two hash tables in a blocking group, because of the OR operator, drops initially the PQ rates which balance later, during the matching phase, due to the better quality of the formulated pairs. We then experimented by setting several values to confidence r, for less collisions from Equation (7), and measured the corresponding PC rates. Since we want to be as optimal as possible, the choice of $r = 1/3$ exhibits both high PC rates and the sizes of the c-vectors are kept to a desired level. As can be seen in Figure 7, we do not gain a lot in terms of accuracy by setting $r < 1/3$.

Running time. By choosing several values for K we measure the elapsed running time. Specifically, for both perturbation schemes we vary K between 20 and 40, which results in generating different values for L[10] (blocking groups). Figure 8(a) clearly illustrates that there is a near-optimal value of K, which is 30 for both perturbation schemes, that minimizes the running time. This is quite reasonable because by increasing K we adjust the *selectivity* of our block-

[10]As shown in Section 4.2, L depends on K, δ, $\vartheta_{\widehat{\mathcal{H}}}$, and $\overline{m}_{opt}$.

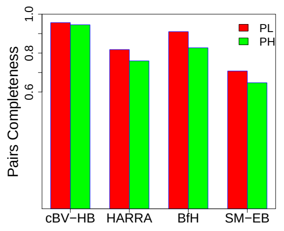


(a) Applying *cBV-HB* by using several values for K.

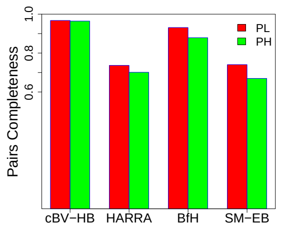


(b) Time needed for converting the data sets into the respective embeddings of each method.

Figure 8: Running time evaluation using the NCVR-based data sets.

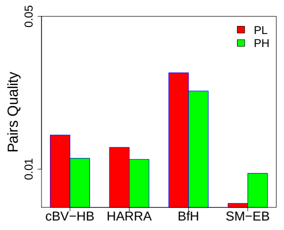


(a) Using the NCVR-based data sets.

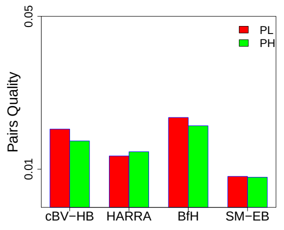


(b) Using the DBLP-based data sets.

Figure 9: Pairs Completeness (accuracy) evaluation.



(a) Using the NCVR-based data sets.



(b) Using the DBLP-based data sets.

Figure 10: Pairs Quality evaluation.

ing mechanism, i.e., buckets are populated with *Id*'s corresponding mostly to similar *c*-vectors. The consequence of higher selectivity is a decrease in running time due to the smaller number of the formulated *c*-vector pairs. Nevertheless, there is a value for K ($K = 35$) where the time needed for building the blocking groups dominates the total running time which starts to increase.

Comparison with the baseline methods. For the next set of experiments, we first measured the average number $b^{(f_i)}$ of bigrams of string values for each attribute listed in Table 3. We underline the difference in the values of $b^{(f_i)}$ between the two sources of data sets and evaluate its impact on the experimental results which follow below. For scheme *PL*, since there is a single perturbation operation, we set $K = 30$, $\delta = 0.1$, and $\vartheta_{PL}^{(f_i)} = 4$, which generate $L = 6$, and $L = 3$ blocking groups, without applying attribute-level blocking, for each source of data sets used. For scheme *PH*, we apply attribute-level blocking by using the rule C_1, as defined previously, and the parameters in Table 3. We set the thresholds as $\vartheta_{PH}^{(f_1)} = \vartheta_{PH}^{(f_2)} = 4$, and $\vartheta_{PH}^{(f_3)} = 8$, which yield $L = 178$ and $L = 62$ blocking groups, respectively.

In Figure 8(b), we evaluate the time needed in order to embed the data sets into the space required by each method. The bigram vectors used by *HARRA* require the least amount of time because a single vector is used for the bigrams generated of the whole record with the side-effects in accuracy, which will be discussed below. The vectors utilized by *SM-EB* exhibit a large amount of time due to the distance computations performed for specifying the pivots.

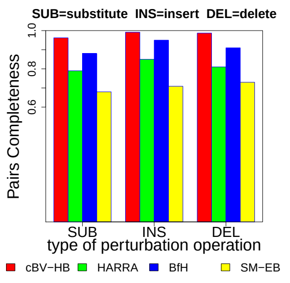
Figures 9(a) and 9(b) show that the *PC* rates of our method are constantly above 95% by using both sources of data sets. These figures also indicate that cBV-HB is the only method which exhibits stable *PC* rates regardless of the source of data sets used. Furthermore, by applying *PH* in the presence of a certain classification rule our method adjusts to it during the blocking step by generating the required number of blocking groups separately for each attribute as the rule defines. However, this attribute-level adjustment requires a larger number of blocking groups, which results in reduced *PQ* rates due to the larger number of the formulated pairs (Figures 10(a) and 10(b)).

The *PC* rates of *SM-EB* are rather low, as Figure 9(a) suggests, especially when using *PH*. This happens due to the insufficient distance-preserving property of the used embedding method. This insufficiency has also another drawback, which is the population of blocks with truly non-matching pairs. These pairs, although being similar in the Euclidean space, exhibit large distances in the original space, which results in low *PQ* rates, as can be clearly seen in Figures 10(a) and 10(b).

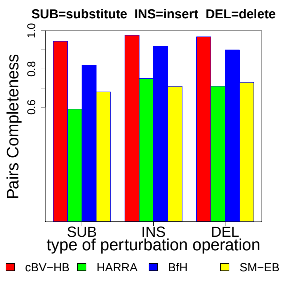
In *HARRA*, the early removal of records in each iteration may lead to missed matching pairs as well, since those records do not participate in the subsequent iterations. Initially, the *PC* rates of *HARRA* were below 0.77. We had to increase considerably the number of blocking groups[11] in order to achieve better rates, which were approximately equal to 0.82 as shown in Figure 9(a). However, the side-effect of increasing the number of blocking groups was the low *PQ* rates as illustrated in Figures 10(a) and 10(b). Especially by using the DBLP-based data sets, the larger number of bigrams combined with the utilization of a single bigram vector for all attributes in a record increased considerably the probability of comparing bigram vectors with identical bigrams belonging to different attributes. This disambiguation, as expected, deteriorated the *PC* rates which fell below 0.75 (Figure 9(b)).

The Bloom filters, which are used by *BfH*, seem to preserve the initial distances from the original space, as confirmed by the high *PC* rates. However, the accuracy guarantees, provided by the HB, refer to the Bloom filter space and there is no study in the literature that corresponds distances from that space to distances in $\mathcal{E}$ with a specified distortion. The authors in [17] provide only some empirical observations with respect to this correspondence without any rigorous justifications. The dependency of distances, in the Bloom filter space, on the length of the initial string

[11]We actually doubled the number of blocking groups in order to give more chances to similar records to be grouped together.

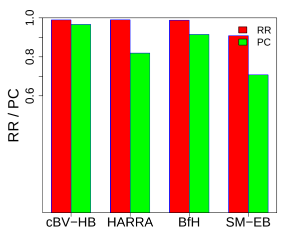


(a) Applying *PL*.

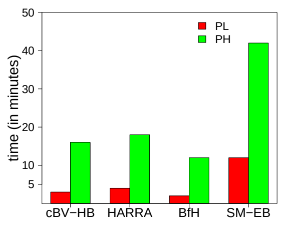


(b) Applying *PH*.

Figure 11: Evaluating Pairs Completeness by focusing on each type of perturbation operation by using the NCVR-based data sets.



(a) Evaluating the reduction of the comparison space in conjunction with accuracy using perturbation scheme *PL*.



(b) Time elapsed for producing the results.

Figure 12: Evaluating the efficiency of each method by using the NCVR-based data sets.

values, as demonstrated previously, explains the increased accuracy by using the DBLP-based data sets (Figure 9(b)). On the contrary, distances in $\widehat{\mathcal{H}}$ depend only on the type of error and by no means on the length of the initial string values. The *PQ* rates of *BfH* are slightly higher than *cBV-HB* mainly due to the larger number of the cryptographic hash functions used for creating the Bloom filters, which results in a larger number of positions set to 1. These bit patterns hashed by the h_l's produce a larger number of buckets in each T_l, which host a smaller number of pairs than *cBV-HB*.

Another factor that does not affect the *PC* rates of our method is the type of the applied perturbation operation as demonstrated in Figures 11(a) and 11(b) by applying *PL* and *PH*, respectively. Our method attains excellent performance and the *PC* rate barely falls below 0.95 only when applying the substitute operations. In general, we observe that all methods exhibit a lower *PC* rate for pairs which have been perturbed by the substitute operation, which indicates a higher distortion in all spaces. For *PH*, only *BfH* exhibits comparable performance, as can be seen in Figure 11(b). However, the two operations performed in the values of the *Address* attribute resulted in missing some pairs especially when both were substitute operations.

Figure 12(a) illustrates together the *RR* and the *PC* rates so that one can easily evaluate the efficiency of each method. *RR* is high for all the compared methods except for *SM-EB* where the formulated blocks are overwhelmed by non-matching pairs. The reduction of the comparison space though keeps up with high accuracy only for cBV-HB and *BfH*, where our method performs better than *BfH* in terms of accuracy by at least 5%. Overall, this provides additional validation of the robustness and practicality of our method. These high *RR* rates affect the running time positively which is below 5 minutes for *PL* for both methods, as depicted in Figure 12(b). By applying *PH* though, the running time increases due to the larger number of blocking groups generated for this perturbation scheme. In *HARRA*, the early pruning of records in each iteration reduces the running time but the results are far from accurate. As expected, *SM-EB* exhibits the highest running time by a large margin among all the compared methods due to the large number of the formulated vector pairs.

7. CONCLUSIONS AND FUTURE EXTENSIONS

In this paper we have proposed a method to embed strings into a compact binary Hamming space in order to apply HB which is an efficient blocking/matching mechanism. The embeddings are of small size, e.g., a record of four strings is represented by 120 bits, and simultaneously the initial distances are preserved as the supporting set of experiments confirmed. Furthermore, we have adapted the LSH-based blocking mechanism to the used classification rule for highly accurate results. We have considered and provided formal guarantees for rules using the AND, OR, and NOT operators. In addition, we have also demonstrated the use of compound classification rules, which include several subrules. For the future, we aim to investigate a distance-preserving and lightweight embedding method for the Jaro-Winkler metric, which was specifically developed for measuring distances between attributes that denote personal information such as names, surnames, or addresses. We also aim to extend the experimental part by comparing the effectiveness of our method with the baselines in identifying records with missing or non-standardized values. The initial results indicate that by applying *PH*, the gain in accuracy compared to the baselines is larger. Another interesting research avenue could be the adaptation of our method to the privacy-preserving context by applying two-party techniques [28]. The compact data structures used for representing the records could be an ideal fit in the protocols introduced in [17, 19] which are used for comparing those records in a secure manner.

8. REFERENCES

[1] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *CACM*, 51(1):117 – 122, 2008.

[2] P. Christen. *Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Springer, Data-Centric Sys. and Appl., 2012.

[3] P. Christen. A survey of indexing techniques for scalable record linkage and deduplication. *TKDE*, 24(9):1537 – 1555, 2012.

[4] P. Christen. Preparation of a real voter data set for record linkage and duplicate detection research. Tech. Rep., The Australian National Univ., 2013.

[5] P. Christen, R. Gayler, and D. Hawking. Similarity–aware indexing for real-time entity resolution. In *CIKM*, pages 1565 âĂŞ– 1568, 2009.

[6] W. W. Cohen and J. Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *SIGKDD*, pages 475–480, 2002.

[7] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *SCG*, pages 253 – 262, 2004.

[8] T. de Vries, H. Ke, S. Chawla, and P. Christen. Robust Record Linkage Blocking Using Suffix Arrays and Bloom Filters. *TKDD*, 5(2), 2011.

[9] E. Durham. *A Framework For Accurate Efficient Private Record Linkage*. PhD thesis, Vanderbilt Univ., US, 2012.

[10] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *TKDE*, 19(1):1–16, 2007.

[11] L. Gravano, P. Ipeirotis, P. Koudas, and D. Srivastava. Text joins for data cleansing and integration in an RDBMS. In *ICDE*, pages 729–731, 2003.

[12] M. Hernandez and S. Stolfo. The merge/purge problem for large databases. In *SIGMOD*, pages 127–138, 1995.

[13] G. Hjaltson and H. Samet. Properties of embedding methods for similarity searching in metric spaces. *TPAMI*, 25(5):530 – 549, 2003.

[14] L. Jin, C. Li, and S. Mehrotra. Efficient Record Linakge In Large Data Sets. In *DASFAA*, pages 137–146, 2003.

[15] D. Karapiperis and V. Verykios. A distributed framework for scaling up LSH-based computations in privacy preserving record linkage. In *BCI*, pages 102 – 109, 2013.

[16] D. Karapiperis and V. Verykios. A Distributed Near-Optimal LSH-based Framework for Privacy-Preserving Record Linkage. *COMSIS*, 11(2):745–763, 2014.

[17] D. Karapiperis and V. Verykios. An LSH-based Blocking Approach with a Homomorphic Matching Technique for Privacy-Preserving Record Linkage. *TKDE*, 27(4):909 – 921, 2015.

[18] H. Kim and D. Lee. Fast Iterative Hashed Record Linkage for Large-Scale Data Collections. In *EDBT*, pages 525 – 536, 2010.

[19] M. Kuzu, M. Kantarcioglu, A. Inan, E. Bertino, E. Durham, and B. Malin. Efficient privacy-aware record integration. In *EDBT*, pages 167 – 178, 2013.

[20] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707 – 710, 1966.

[21] G. Li, D. Deng, J. Wang, and J. Feng. Pass-join: a partition-based method for similarity joins. In *PVLDB*, pages 253–264, 2011.

[22] I. Miller and J. Freund. *Probability and Statistics for Engineers*. Prentice–Hall, 1977.

[23] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge Univ. Press, 1995.

[24] A. M. N. Koudas and D. Srivastava. Flexible string matching against large databases in practice. In *VLDB*, pages 1086–1094, 2004.

[25] J. Qin, W. Wang, Y. Lu, C. Xiao, and X. Lin. Efficient exact edit similarity query processing with the asymmetric signature scheme. In *SIGMOD*, pages 1033–1044, 2011.

[26] B. Schneier. *Applied cryptography: protocols, algorithms, and source code in C*. Wiley, 1996.

[27] R. Schnell, T. Bachteler, and J. Reiher. Privacy-preserving record linkage using Bloom filters. *Central Medical Inf. and Decision Making*, 9, 2009.

[28] D. Vatsalan, P. Christen, and V. Verykios. Efficient two-party private blocking based on sorted nearest neighborhood clustering. In *CIKM*, pages 1949 – 1958, 2013.

[29] D. Vatsalan, P. Christen, and V. Verykios. A taxonomy of privacy-preserving record linkage techniques. *JIS*, 38(6):946–969, 2013.

[30] J. Wang, J. Feng, J. Wang, and G. Li. Trie-join: Efficient trie-based string similarity joins with edit-distance constraints. In *PVLDB*, pages 1219–1230, 2010.

[31] W. Wang, C. Xiao, X. Lin, and C. Zhang. Efficient approximate entity extraction with edit distance constraints. In *SIGMOD*, pages 759–770, 2009.

[32] M. Weis, F. Naumann, U. Jehle, J. Lufter, , and H. Schuster. Industry-scale duplicate detection. In *PVLDB*, pages 1253–1264, 2008.

[33] S. Whang, D. Marmaros, and H. Garcia-Molina. Pay–as–you–go entity resolution. *TKDE*, 25(5):1111–1124, 2013.

[34] S. E. Whang, D. Menestrina, G. Koutrika, M. Theobald, and H. Garcia-Molina. Entity resolution with iterative blocking. In *SIGMOD*, pages 219–232, 2009.

[35] C. Xiao, W. Wang, and X. Lin. Ed–join: an efficient algorithm for similarity joins with edit distance constraints. In *PVLDB*, pages 933–944, 2008.

[36] C. Xiao, W. Wang, X. Lin, and J. Yu. Efficient similarity joins for near duplicate detection. In *WWW*, pages 131–140, 2008.