An Efficient Duplicate Record Detection Using q-Grams Array Inverted Index

Alfredo Ferro, Rosalba Giugno, Piera Laura Puglisi, and Alfredo Pulvirenti

Dept. of Mathematics and Computer Sciences
University of Catania
{ferro,giugno,lpuglisi,apulvirenti}@dmi.unict.it

Abstract. Duplicate record detection is a crucial task for data cleaning process in data warehouse systems. Many approaches have been presented to address this problem: some of these rely on the accuracy of the resulted records, others focus on the efficiency of the comparison process. Following the first direction, we introduce two similarity functions based on the concept of q-grams that contribute to improve accuracy of duplicate detection process with respect to other well known measures. We also reduce the number and the running time of record comparisons by building an inverted index on a sorted list of q-grams, named *q-grams array*. Then, we extend this approach to perform a clustering process based on the proposed q-grams array. Finally, an experimental analysis on synthetic and real data shows the efficiency of the novel indexing method for both record comparison process and clustering.

Keywords: Duplicate record detection, q-grams, inverted index, bitmaps, clustering.

1 Introduction

The quality of data can significantly affect the reliability of decision support systems. Real-world data are neither carefully controlled for quality nor defined in a consistent way due to the fact that they have huge size and come from multiple sources [10]. Thus, data mining techniques have to take into account incomplete or missing values, constraints violations, noisy and inconsistent data. Misspellings and different conventions result in a multiple and not unique representation of objects. The process of resolving such identification problems refers to the *data heterogeneity* term [3].

There are two different types of data heterogeneity: *structural* and *lexical*. Structural heterogeneity occurs when the fields of the tuples in the database have different structures in different databases [8]. For example, in one database the customer address might be recorded in one field named `addr`, while in another database the same information might be stored in multiple fields (`street`, `city`, `state` and `zipcode`). Lexical heterogeneity occurs when the tuples have identically structured fields across databases, but the data use different representations to refer to the same real-world object. For example, in one database

T.B. Pedersen, M.K. Mohania, and A M. Tjoa (Eds.): DaWaK 2010, LNCS 6263, pp. 309–323, 2010.

the first and last name of a person can be stored using this format: *Ilary Patricia Doe*, while another database can use different convention: *I. P. Doe*. Based on this consideration, two records can be considered *equivalent* if they are semantically equal. The similarity between records is computed by metrics which measure the semantic equivalence through a score. Record pairs with high similarity scores (above a specified threshold) are treated as duplicates. In this paper, we focus on the problem of lexical heterogeneity.

In addition to the accuracy of classifying records pairs into matches and mismatches, the central issue consists of improving the speed of comparisons. For this reason, many techniques have been proposed to reduce the quadratic complexity of comparing two tables and the running time of a single comparison.

After a brief review of existing techniques, we propose a new method, called DDEBIT, that efficiently solves the Duplicate record DEtection problem using BITmaps. By introducing two similarity functions based on q-grams [9] and Monge-Elkan [17] distances, DDEBIT improves the accuracy of comparison process with respect to other well known approaches. Next, we show that DDEBIT can also be used to cluster records in a table. Finally, in the experimental section we demonstrate the efficacy of our method on synthetic and real data.

2 Related Work

Duplicate record detection typically relies on string comparison techniques such as, edit distance [13], Smith-Waterman [19], Jaro [11], Monge-Elkan [17] and others [18,23]. Although effective, all these measures are computationally expensive (quadratic on the length of the strings) and therefore, they are not suitable for record detection cause of the size of databases. Therefore, approximate similarity metrics have been designed [21,20,9]. They are based on the notions of q-grams and positional q-grams [22].

Related to efficiency, different methods have been proposed to support duplicate detection process. Common techniques, such as *Standard Blocking* [12], divide the database into blocks and compare only the records that fall into the same block. The blocking is performed by grouping records having the same *blocking key*, that could be an attribute value or a concatenation of more attribute values. The risk is to miss some matches due to errors in the blocking step (records assigned to the wrong block) or having false mismatches caused by the failure of comparing records that do not agree on the blocking field.

The *Sorted Neighborhood Approach* [14] is based on the idea that the probability that similar records will be closer after sorting data, according to the value of the blocking key, is high.

Suffix-array indexing [1] uses a suffix array [15] as an inverted index to dynamically generate blocks of associated records using blocking keys.

In [2], blocking key values are converted into lists of bigrams (q-grams with q=2) alphabetically sorted. Then, sub-lists of all possible combinations are built using a threshold and inserted in an inverted index. The number of sub-lists depends on both the length of the key value and the threshold. In [5], Christen

and Gayler present *similarity-aware* and *materialized similarity-aware* inverted indexes for large real-world data sets. The basic idea is to store the similarity in the inverted index and use it to reduce the computation at query time.

Monge and Elkan [17] proposed a technique called *Canopy clustering* to improve the performance of record comparisons by assuming that duplicate detection is transitive. They use a union-find structure in which duplicate records are merged into a cluster and only a representative of the cluster is kept for subsequent comparisons. Several methods define metric similarity as a canopy distance [7,4]. In particular McCallum et al. [16] propose the use of *canopies* to cluster large high-dimensional datasets. The key idea is to use a cheap and approximate distance measure to efficiently group records into overlapping clusters. Then, a more expensive function can be used to achieve a better accuracy when similar record pairs are compared.

3 DDEBIT: A Duplicate Record Detection Algorithm Based on Bitmaps and q-Grams

Here, we introduce a new approach to efficiently solve the duplicate record detection problem by extending the strategy adopted in [16]. The method is based on the use of two similarity functions together with an efficient indexing technique.

First, a lightweight function filters out record pairs with a lower 'global' similarity. This produces a set of candidate pairs whose global similarity is above a minimum *loose threshold*. Next, a more accurate function performs comparisons between candidate pairs that passed the above filter. This function detects potentially duplicated records whose 'local' similarity is above a *tight threshold*.

The intuition behind this method is that record pairs with a higher global similarity are likely to be duplicated and can be successively compared to determine a better accuracy using a finer and local similarity function. Moreover, the increase of accuracy due to the combination of global and local similarities reduces the number of potential false positives and negatives.

To support an efficient duplicate detection process in large datasets, we propose a two-phase approach:

1. **Create index.** We implement an indexing technique based on a sorted list of q-grams named *q-grams array*. For each q-gram, we generate an inverted index by using bitmaps.
2. **Record detection.** After indexing q-grams in the previous phase, we use bitmaps to perform a fast comparison process or an efficient clustering.

Next Sections describe the method in more details.

3.1 DDEBIT Similarity Functions

Before introducing the new similarity functions, we report some basic properties of q-grams.

Basic Definitions. Let Σ be a finite alphabet of size $|\Sigma|$. Let $s_1 \in \Sigma^*$ be a string of length n. The **q-grams** are short characters substrings of length q of the database strings. Given a string s_1, its **positional q-grams** are obtained by sliding a window of length q over the characters of s_1. Q-grams at the beginning and the end of the string can have fewer than q characters. Therefore, new characters "#" and "$" not in Σ are used to extend the string (1) by prefixing it with q-1 occurrences of # and (2) suffixing it with q-1 occurrences of $.

For example, the positional q-grams of length q=2 for string *tom_smith* are: { (1,#t), (2,to), (3,om), (4,m_), (5,_s), (6,sm), (7,mi), (8,it), (9,th), (10,$h$$) }. The set of all positional q-grams of a string s_1 is the set of all the $|s_1|$ + q - 1 pairs constructed from all q-grams of s_1. The intuition behind the use of q-grams is that when two strings s_1 and s_2 have a small edit distance, they have many q-grams in common. The use of positional q-grams will involve comparing positions of matching q-grams within a certain distance. With the appropriate use of hash-based indexes, the average time required for computing the q-gram overlap between two strings s_1 and s_2 is $O(\max(|s_1|, |s_2|))$.

Follows the formal definitions of the proposed similarity functions. These are inspired by Monge-Elkan [17] and q-grams [9] distances. The first one better detects typographical errors and block movements whereas, the q-gram distance allows a quick computation of the edit distance to filter non-similar strings.

Definition 3.11. (**Lightweight similarity**). *Let R be a relation and X, Y be strings representing two tuples in R. Let q_X and q_Y be the sets of distinct q-grams of X and Y, respectively. The function $d_{light,q}$ is defined as follows:*

$$d_{light,q}(X, Y) = \frac{|q_X \bigcap q_Y|}{\max(|q_X|, |q_Y|)} \tag{1}$$

This function computes the ratio of the number of distinct q-grams that two strings have in common over the number of q-grams of the longest string. This score represents a 'global' similarity on the compared strings. It does not take into account the position of common q-grams within the strings. In Figure 1 (a), for each string, the distinct bigrams (q=2) are listed in a lexicographical order. The computation of $d_{light,2}$, for all possible string pairs, is reported in Figure 1 (b). In some cases, d_{light} returns a low value also for pairs that could represent the same real world entity. Here, the strings S_2 and S_3 differ on an high number of q-grams, caused by the word 'professor' and a typographical error (Stewen in place of Steven). To refine the score of this type of string pairs, a more accurate function combines d_{light} with Monge-Elkan distance [17].

Definition 3.12. (**Accurate similarity**). *Let R be a relation and X, Y be strings representing two tuples in R. Let X_i and Y_j be the tokens or atomic strings (i.e. a sequence of alphanumeric characters delimited by punctuation characters) of strings X and Y, respectively. Let $|X|$ and $|Y|$ be the number of tokens of X and Y respectively, and $|X| \geq |Y|$. The function $d_{acc,q}$ is defined as follows:*

ID	String	List of distinct and sorted q-grams	Distinct q-grams
S_1	Prof. Brown Steven	#b #p #s br en ev of ow pr ro st te ve wn f$ n$	16
S_2	Stewen Brown	#b #s br en ew ow ro st te we wn n$	12
S_3	Professor Brown Steven	#b #p #s br en ev es fe of or ow pr ro so ss st te ve wn n$ r$	21
S_4	Steven Brown, executive	#b #e #s br cu ec en ev ex iv ow ro st te ti ut ve xe wn e$ n$	21
S_5	Dr. Brown, executive	#b #d #e br cu dr ec ex iv ow ro ti ut ve xe wn e$ n$ r$	19

(a)

$d_{light,2}(d_{\_acc,2})$	S_1	S_2	S_3	S_4	S_5
S_1	1 (1)	0,62 (0,62)	0,71(0,96)	0,5 (0,7)	0,36 (0,53)
S_2		1 (1)	0,47 (0,60)	0,47 (0,57)	0,31 (0,50)
S_3			1 (1)	0,57 (0,70)	0,38 (0,40)
S_4				1 (1)	0,76 (0,83)
S_5					1 (1)

(b)

Fig. 1. (a) List of sorted q-grams. (b) $d_{light,2}$ and $d_{acc,2}$ scores for all string pairs.

$$d_{acc,q}(X,Y) = \frac{1}{|X|}\sum_{i=1}^{|X|}\max_{j=1}^{|Y|} d_{light,q}(X_i,Y_j) \tag{2}$$

Before computing $d_{acc,q}$, X_i and Y_j are divided into q-grams and $d_{light,q}$ is computed for all token pairs in order to find the best matches. The sum of these maximal scores is then normalized by the maximum number of tokens given by $|X|$. Intuitively, this function computes the number of common q-grams locally, giving more importance to tokens similarity than global strings similarity. Notice that, to increase the accuracy of similarity of tokens X_i which are prefixes of Y_j, $d_{light,q}$ within $d_{acc,q}$ is replaced with a function $d_{prefix}(X_i,Y_j) = 1 - \frac{|q_{Y_j} \setminus q_{X_i}|}{|q_{Y_j}||q_{X_i}|}$ which yields a score that better captures the prefix similarity. Furthermore, since $d_{acc,q}$ is a local similarity measure, missing short tokens (i.e. X_i with no more than 4 bigrams) could cause lower and biased values for $d_{acc,q}$. Thus, pseudo-counters β_i proportional to $1/|q_{X_i}|$ are added to the similarity measure $d_{acc,q}$. In Figure 1 (b), the values of $d_{acc,2}$ are also reported for strings in Figure 1 (a). Typically, $d_{acc,q}$ increases the score of a strings pair with respect to $d_{light,q}$ in presence of prefixes, missing of short words, exact matches or high similarity between token pairs. The function $d_{acc,q}$ can be generalized to work with records of k fields.

Definition 3.13. *Let R be a relation, let r_x and r_y be two records in R projected on k fields. Let p_i, $1 \leq i \leq k$, be a weight associated to i-th field.*

$$d_{acc_k,q}(r_x, r_y) = \sum_{i=1}^{k} p_i \times d_{acc,q}(r_x[i], r_y[i]) \quad (3)$$

where $p_1 + p_2 + ... + p_k = 1$ *and* $r_x[i]$ *is the projection of* r_x *to the* $i-th$ *field.*

Here, the comparison key is composed of k fields, and $d_{acc,q}$ is computed separately for each field. The contribution of the $i-th$ field depends also on the value of p_i, that can give a higher or lower weight to the $i-th$ attribute in the comparison process. Unfortunately, this function is quadratic with respect to the length of the strings. To overcome this limitation, we introduce a secondary memory indexing technique on the list of distinct q-grams. This index makes similarity computation linear in the number of q-grams of the longest string.

3.2 DDEBIT Indexing

The indexing phase contributes to quickly detect candidate records having an high global similarity (i.e. high values of $d_{light,q}$) and to reduce the running time of $d_{acc,q}$. The key idea is based on the concept of *q-grams array*. Given a table T, a `q-grams array` is a sorted list of all distinct q-grams in T. Prefix q-grams of the form $\#c_1..c_q$ are listed at the beginning, followed by q-grams without special symbols, which are followed by suffix q-grams ending with the symbol \$. In our implementation, we use bigrams (i.e. q=2) sorted in a lexicographical order.

Figure 3 reports the pseudocode of the algorithm Create_Index. Given the dataset and the list of fields used for comparisons, the algorithm generates three index files. For each record of the dataset, all distinct q-grams are inserted in the q-grams array. More precisely, in line {1..3}, the q-grams array index is created and stored in secondary memory (dataset.grm). In line {5, 6}, for each element of the q-grams array, an inverted index is generated allocating a bitmap of n bits (n is the size of the dataset). For each record and for each field, the algorithm stores the position of its q-grams (obtained from the q-grams array) in the index file (dataset.idx). Moreover, if a q-gram occurs in the $i-th$ record, the corresponding bitmap is updated (line 13) by setting the $i-th$ bit. Finally, in line {15, 16} bitmaps are stored in a binary file (dataset.bin). Figure 2 (a) shows an example of dataset and corresponding index files.

Concerning the complexity, let m be the number of records, n be the size of the q-grams array and r be the number of q-grams of the longest string. The running time of *Create_Index* algorithm is *O(m*n)*, whereas the secondary memory required by the index files is *O(n)* for the q-grams array, *O(m*r)* for saving q-grams ids and *O(m*n)* for the binary file. Finally, the method requires *O(n*m)* bits of memory.

3.3 DDEBIT Record Comparisons

The duplicate record detection problem treated in this paper is defined as follows.

Definition 3.31. *Let* B *be a base table and* Q *be a query table. Let* l *be the loose threshold and* t *be the tight threshold. Let* $r_x \in Q$ *and* $r_y \in B$ *be two records*

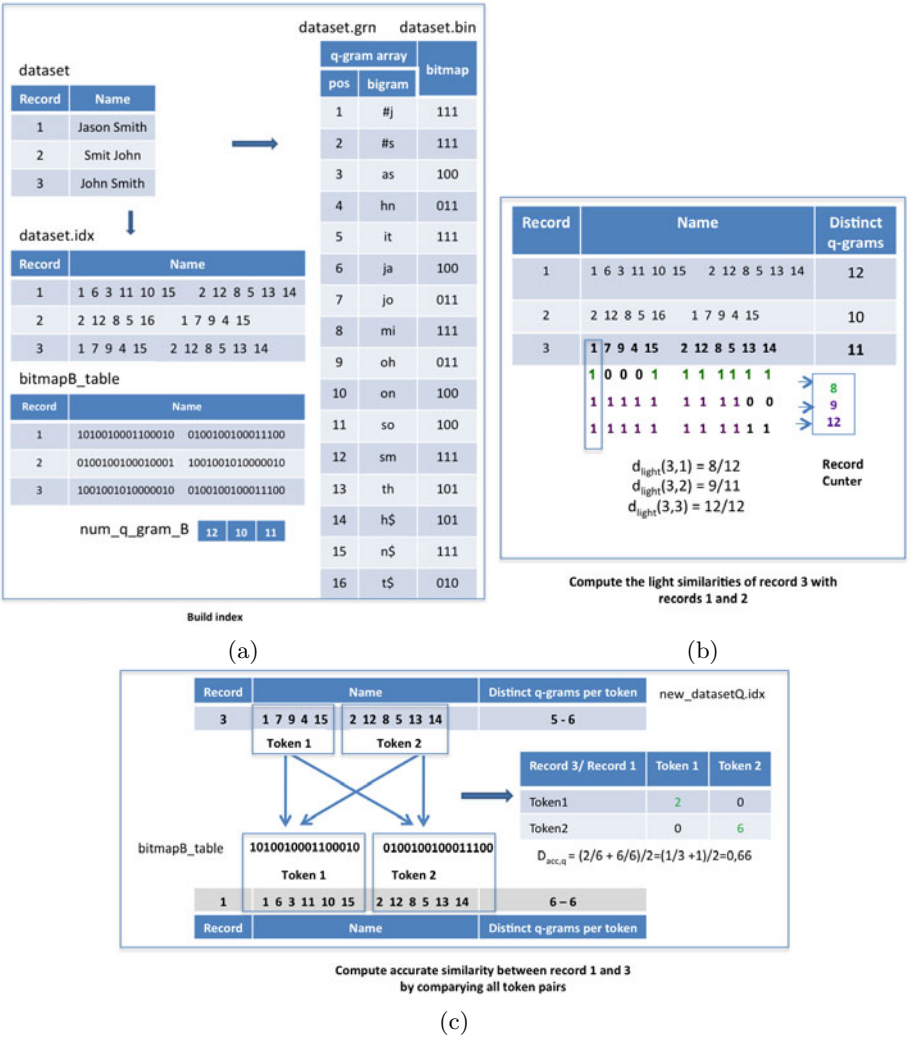


Fig. 2. (a) Indexing construction. (b) d_{light} computation. (c) d_{acc} computation.

projected on k fields. r_x *is considered a duplicate of* r_y*, if* $d_{light,q}(x, y) \geq l$ *and* $d_{acc_k,q}(r_x, r_y) \geq t$.

The novelty introduced by DDEBIT consists in reducing both the number and the cost of the expensive record comparisons using information stored in the index files previously generated for each dataset. More precisely, the inverted indexes stored in the binary files (dataset.bin) are used to quickly compute $d_{light,q}$, while record index files (dataset.idx) are used to reduce the complexity of $d_{acc,q}$. Notice that, when base table B and query table Q do not coincide index files (.grm, .idx, and .bin) are constructed for both B and Q.

DDEBIT_Create_Index(D dataset of n records, a list of k fields $(f_1, f_2, \ldots, f_k)$)
OUTPUT: Three index files: dataset.grm, dataset.idx, dataset.bin

```
1  for each record rec_i in D do // Create file dataset.grm
2      insert_all_q_grams (rec_i, q-grams_array);
3  write (q-gram_array, dataset.grm);
4
5  for each q-gram q_j in q-gram_array do // Allocate bitmaps
6      bitmap[q_j] = allocate_bitmap (n); // n is the size of D
7
8  for each record rec_i in D do // Create file dataset.idx
9      for each field f_y do
10         for each q-gram q_j do
11             pos_q_gram = find_q_gram_position (q_j, q-gram_array);
12             write (pos_q_gram, dataset.idx);
13             bitmap[q_j] [rec_i] = 1; // Update bitmap for q_j
14
15 for each q-gram q_i in q-gram_array do // Create file dataset.bin
16     write (bitmap[q_j], dataset.bin);
```

Fig. 3. The Create Index procedure of DDEBIT

The pseudo code of duplicate detection algorithm is showed in Figure 4. In line {1..4}, for each base record, the algorithm (i) retrieves for each field and for each token, all q-grams from record index and loads them in memory by setting a bitmap whose size is equal to the length of the q-grams array, and (ii) maintains the total number of distinct q-grams. The construction of *bitmapB_table* (line 3) is not expensive; for each base record r_b, it is linear in the total number of q-grams of r_b. In line 5, the algorithm remaps the ids of q-grams in datasetQ.idx with the corresponding ids from the datasetB.grm. If a q-gram of query table does not occur in the q-grams array of base table, its id is set to -1. In lines {7..19}, the comparison process between query table and base table is performed. DDEBIT, by using information stored in the binary file, maintains a counter associated to each base record storing the number of common q-grams with the query records (line 10). Such a counter allows to quickly detect base records sharing at least one q-gram with the query record (see Figure 2). We compute (line 14) the $d_{light,q}$ similarity between the query record and each base record whose id corresponds to a value of record_counter greater than 0. If the value of $d_{light,q}$ is above the loose threshold, the algorithm computes $d_{acc_k,q}$ to verify the local similarity. The candidate records are considered duplicated if $d_{acc_k,q}$ is above the tight threshold (see Figures 2 (b),(c) for an example of computation).

Using information stored in the index files, if g is the number of q-grams of the query records, $d_{acc,q}$ requires $O(g)$ comparisons. The reason behind this complexity relies on fact that, for each q-gram i of a query record, the method tests only the i-th bit of the bitmap corresponding to each token of the base record. Thus, the complexity of the method is *O(m*n*g)*, where m is the number of records of the query table, n is the number of records of the base table, and

DDEBIT_Duplicate_Detection(*datasetB.*∗: Base table, *datasetQ.*∗: Query table; a list of k fields (f_1, f_2, ..., f_k); loose threshold l; tight threshold t)
OUTPUT: for each query record, a list of potential duplicated records

```
1  for each record rec_b in B do //Load base table in memory
2      for each field f_i in rec_b do //Read record ids from file datasetB.idx
3          bitmapB_table[rec_b][f_i]= Allocate_and_set_List_bitmaps();
4      num_q-grams_B[rec_b]=total_num_distinct_q-grams();
5  new_datasetQ.idx = Remap(datasetQ.idx, datasetB.grm);
6
7  for each record rec_q in Q do // Comparison process
8      for each q-gram q_j in rec_q do
9          for each i − th set bit of datasetB.bin[q_j] do
10             record_counter[i]++;
11     num_q-grams_Q[rec_q]=total_num_distinct_q-grams();
12
13     for each i such that record_counter[i]> 0 do //Find candidate pairs
14         d_light= record_counter[i] / max(num_q−grams_B[i],num_q−grams_Q[rec_q]);
15         if (d_light == 1.0 ) then output(Exact match);
16         else if ( d_light ≥ l ) then
17                 d_acc_k = compute_score(bitmapB_table[i], new_datasetQ.idx[rec_q]);
18                 if (d_acc_k ≥ t) then
19                    output (Record i and record rec_q are duplicated!);
```

Fig. 4. The Duplicate_Detection procedure of DDEBIT

$g << n$ is the maximum number of q-grams in the records. Moreover, the method requires *O(n*m)* bits of main memory.

3.4 Application to Clustering

The key idea for detecting duplicates presented in this paper can be used for clustering. Extending a strategy proposed in [16], we introduce a new clustering technique which consists of partitioning the dataset into non-overlapping clusters. Using the similarity measures d_{light} and d_{acc}, the method groups into the same cluster records considered duplicated according to the strategy described in the previous sections. Differently from comparisons process, number of comparisons are slightly reduced. This is due to the fact that a record associated to a cluster will not be used for further comparisons. The complexity is *O(n*g*c)*, where c is the number of clusters, n is the number of records, and $g << n$ is the maximum number of q-grams in the records.

4 Performance Analysis

Experimental analysis was performed on a server HP Proliant DL380 with 4GB RAM, equipped with Linux Debian Operating System. DDEBIT was implemented in C++ language. In all our experiments, q has been set to 2.

Accuracy of DDEBIT measures. We compared DDEBIT with q-grams and positional q-grams [9], Jaro [11], Jaro Winkler [23] and edit distance [13]. To perform comparisons, we used Python implementation of these metrics available in Febrl package[1]. We measured the accuracy of similarity metrics on real datasets concerning information about restaurants, by comparing 331 tuples from Zagat's website with 533 tuples from the Dept. of Health website. We selected the fields `name`, `street`, `city`, and `phone` and computed Recall, Precision and the F1 measure defined as $\frac{2 \times Recall \times Precision}{Recall + Precision}$.

Table 1 shows the accuracy of the metrics related to the best performances for each method. Results clearly show that the proposed method has a better behavior with respect to the other metrics. DDEBIT outperforms q-grams and edit distance since it uses a combination of local and global similarities. Moreover, due to the fact that DDEBIT better captures the similarity in presence of prefixes and missing of short tokens, it also yields an higher value of F1 with respect to Jaro and Jaro Winkler. However, a limitation of DDEBIT is that it does not detect string swaps between attributes neither potential similar strings which differ on an high number of tokens or contain missing values.

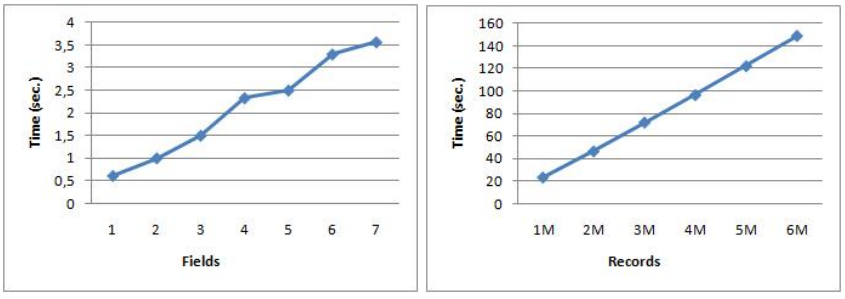
Table 1. Comparison among DDEBIT and q-grams, positional q-grams, edit distance, Jaro and Jaro-Winkler

Metric	Tight(Loose) threshold	Real Duplicates	Correct Pred. Duplicates	Predicted Duplicates	Precision	Recall	F1
q-gram(q=2)	0,7	112	107	210	0,51	0,96	0,66
Positional	0,7	112	49	61	0,8	0,44	0,57
Jaro	0,8	112	78	178	0,44	0,70	0,54
Edit	0,75	112	72	140	0,51	0,64	0,57
Jaro Winkler	0,85	112	103	173	0,60	0,92	0,72
DDEBIT	0,8 (0,65)	112	97	106	0,92	0,87	**0,89**
DDEBIT	0,75 (0,6)	112	107	134	0,8	0,96	0,87
DDEBIT	0,7 (0,6)	112	110	181	0,61	0,98	0,75

Efficiency and accuracy of DDEBIT method. By evaluating the indexing construction time on synthetic data[2], we observed a linear trend on both the number of fields (see Figure 5 (a)) and the size of the dataset (see Figure 5 (b)). Concerning the record comparisons time, by performing a self comparison of a base table of 10k and varying number of fields, DDEBIT took a maximum running time of 15,41 sec. (Figure 6 (a)). In Figure 6 (b) we report the average time for query record varying the size of the dataset. The query time increases for larger datasets due to the higher number of candidate records. Figure 7 (a) reports a record comparison between a query table of size 10k and a base table of size 100k. A naive approach (which does not use filtering) performs 10k*100k

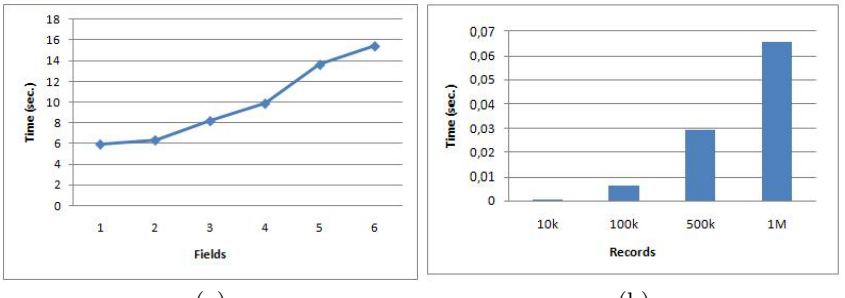
[1] http://sourceforge.net/projects/febrl/

[2] We used the generator provided at http://dbgen.sourceforge.net/.



(a) (b)

Fig. 5. DDEBIT indexing time. (a) Size of dataset is 100k. (b) Number of fields is 4.

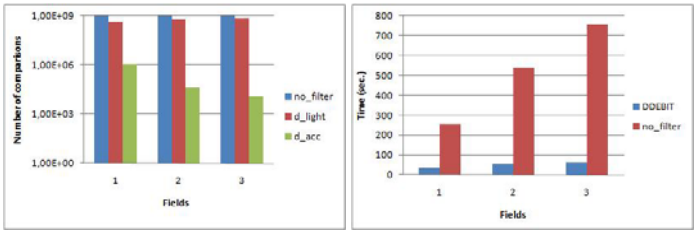


(a) (b)

Fig. 6. DDEBIT record comparisons time with light threshold and tight threshold equal to 0.7 and query table of size 10k. (a) Time w.r.t. fields. (b) Number of fields is 4.

comparisons, computing $d_{acc,q}$ on all possible record pairs. $d_{light,q}$ computes an high number of comparisons however the cost of $d_{light,q}$ is lower than $d_{acc,q}$ (common q-grams are retrieved directly from index files). Moreover, it reduces the number of expensive comparisons filtering out non similar record pairs. For example, by setting $k = 3$, $d_{acc_{3,2}}$ performs only 12.294 record comparisons in place of 10k*100k. In Figure 7 (b), DDEBIT drops down the running time up to the 92% with respect to the naive approach. Although the different programming languages, we compared DDEBIT with SB (Standard Blocking with Soundex encoding) [6], QI (Q-gram Indexing) [2], SA and MSA [5] (Similarity Aware and Materialized Similarity Aware using Jaro winkler and 1-gram comparisons metrics)[3]. We generated uniformly distributed synthetic datasets having size ranging from 500k to 1.5M. We computed the index construction time (Figure 8 (a) top) and the average query time (Figure 8 (a) bottom) using a query table of 2k records.

[3] We thank prof. Peter Christen for providing the software.



(a) (b)

Fig. 7. (a) Number of comparisons using $d_{light,q}$, $d_{acc,q}$ and the naive approach. (b) Running time of DDEBIT vs the naive approach.

We also compared the accuracy of the above methods on synthetic datasets of size 100 with different distributions (uniform, Poisson and zipf) and 4 fields. Each query record is a duplicate of one base record with 1 modification per field and 3 modifications per record. We also used two real data selecting randomly 100 records from *ucdPeopleMatch*[4] dataset which contains prefixes and swap of words within attributes, and *cora*[5] dataset containing citations clustered into groups referring to the same scientific paper.

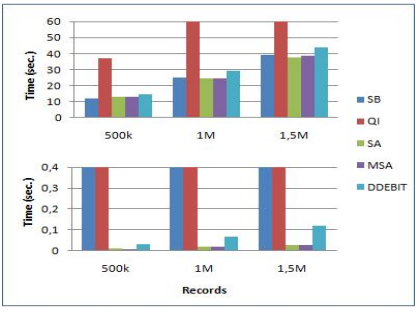
For real datasets, the number of true matches is slightly increased with respect to the other algorithms (see Figure 8 (b)). Figure 8 (c) shows the Recall of DDEBIT for different values of light and tight thresholds for the cora dataset. Notice that, using proper thresholds, DDEBIT is able to reach a very high Recall. Finally, Figure 8 (d) contains different F1 values varying the weights p_i associated to the attributes, used in $d_{acc_k,q}$. Assigning 1/2 to *title* and 1/6 to the other attributes, DDEBIT yields a very good accuracy with respect to the uniform weights assignment (first row in the table).

Clustering. In order to test the accuracy of DDEBIT clustering, we generated synthetic datasets (using the Febrl generator) having at most 4 modifications per attribute and 15 modifications per record. Each dataset contains one duplicate per record. We used a light threshold equal to 0.6 and a tight threshold equal to 0.65. Results clearly show that F1 (see Figure 9 (a)) is higher using more fields in the comparisons process, due to the fact that $d_{acc,q}$ is highly discriminating and gives a more accurate score using a larger number of fields.

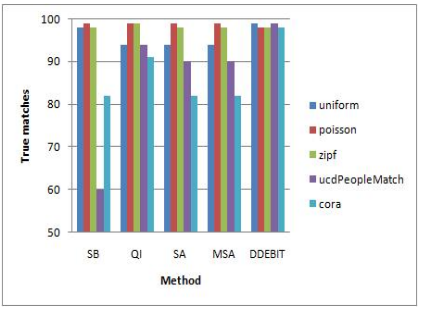
We also tested running time of our clustering approach. Figure 9 (b) shows that, by fixing the number clusters to 10 and varying the size of dataset, DDEBIT has a better running time with respect to fix the percentage of duplicates per record (10%) and varying the number of clusters.

The proposed clustering technique does not take into account, for further comparisons, records already assigned to a cluster. This improves clustering efficiency at cost of inserting some records in wrong blocks. Finally, in Figure 10 we report

[4] http://fingolfin.user.cis.ksu.edu/ERICRAWLER/data/
[5] http://www.hpi.uni-potsdam.de/naumann/projekte/repeatability/datasets/



(a)



(b)

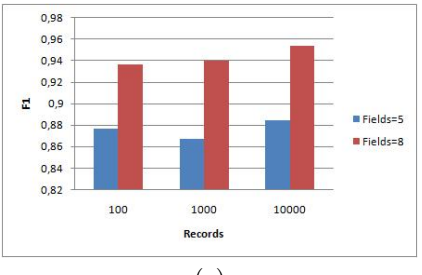
	Recall wrt loose threshold and tight threshold			
	Loose Threshold			
	0,9	0,8	0,7	0,6
0,95	0,34	0,39	0,39	0,36
0,9	0,57	0,68	0,7	0,7
0,85	0,65	0,84	0,86	0,86
0,8	0,66	0,93	0,95	0,95
Tight threshold 0,75	0,66	0,95	0,98	0,98
0,7	0,66	0,95	0,98	0,98
0,65	0,66	0,95	0,99	0,99
0,6	0,66	0,95	0,99	0,99
0,55	0,66	0,95	0,99	1

(c)

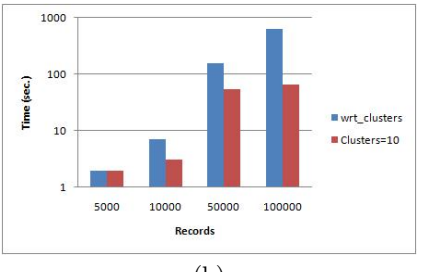
Weight (p_i) per field				Accuracy of cora dataset		
Author	Title	Conference	Year	Precision	Recall	F1
1/4	1/4	1/4	1/4	0,890909	0,924528	0,90741
1/2	1/6	1/6	1/6	0,816667	0,924528	0,86726
1/6	**1/2**	**1/6**	**1/6**	**0,961538**	**0,943396**	**0,95238**
1/6	1/6	1/2	1/6	0,723077	0,886792	0,79661
1/6	1/6	1/6	1/2	0,877193	0,943396	0,90909
1/3	1/3	1/6	1/6	0,907407	0,924528	0,91589
1/6	1/3	1/3	1/6	0,907407	0,924528	0,91589
1/6	1/3	1/6	1/3	0,909091	0,943396	0,92593

(d)

Fig. 8. (a) DDEBIT vs other techniques on synthetic datasets. Top shows index construction time, bottom query time. (b) True matches (0-100) on different datasets. (c) DDEBIT Recall on cora dataset. (d) DDEBIT accuracy on cora dataset wrt p_i.



(a)



(b)

Fig. 9. Clustering. (a) F1 w.r.t. the size of dataset. (b) Running time with fields=5.

a direct comparison between DDEBIT clustering technique and a naive approach that does not filter candidate record pairs. DDEBIT slightly reduces execution time and the number of comparisons. Moreover, accuracy is lower in some cases (records = 1000), because $d_{acc,q}$ introduces some false positives or negatives; but we observed an increased value of Recall in the other cases (records equal to 10k and 20k), due to the fact that $d_{light,q}$ sometimes discards false positives that $d_{acc,q}$ would consider as duplicates.

Records	Time (sec.)		Number of comparisons		Recall	
	DDBEBIT	no_filter	DDBEBIT	no_filter	DDEBIT	no_filter
1000	0,12	324	263	250035	0,998	1
2000	0,33	13,63	1619	1000925	1	1
10000	6,3	457	10007	24816509	0,997	0,9954
20000	19,83	1640	5001	98649409	0,9977	0,9913

Fig. 10. Clustering vs naive approach. Fields=5, light and tight threshold equal to 0.7.

5 Conclusions and Future Work

In this paper we proposed DDEBIT, an indexing method for fast and accurate duplicate record detection. The novelty of this approach consists of using a two-step similarity comparisons process which results more efficient and effective than using a single measure. Moreover, comparisons are optimized by using a q-grams array indexed with bitmaps. Finally, the efficiency of DDEBIT has been confirmed by a new clustering technique based on it. Future work will investigate the memory usage reduction through an efficient bitmap compression and the refinement of similarity metrics to increase accuracy of resulted records.

References

1. Aizawa, A., Oyama, K.: A fast linkage detection scheme for multi-source information integration. In: International Workshop on Challenges in Web Information Retrieval and Integration, pp. 30–39 (2005)
2. Baxter, R., Christen, P., Churches, T.: A comparison of fast blocking methods for record linkage. In: ACM SIGKDD 2003 Workshop on Data Cleaning, Record Linkage, and Object Consolidation, pp. 25–27 (2003)
3. Chatterjee, A., Segev, A.: Data manipulation in heterogeneous databases. ACM SIGMOD Record 20, 64–68 (1991)
4. Chaudhuri, S., Ganjam, K., Ganti, V., Motwani, R.: Robust and efficient fuzzy match for online data cleaning. In: SIGMOD 2003, pp. 313–324 (2003)
5. Christen, P., Gayler, R.: Towards scalable real-time entity resolution using a similarity-aware inverted index approach. Proceedings of AusDM 2008, Glenelg, Adelaide 87, 30–39 (2008)
6. Christen, P., Churches, T.: Febrl: Freely extensible biomedical record linkage Manual (2002)
7. Cohen, W., Richman, J.: Learning to match and cluster large high-dimensional data sets for data integration. In: SIGKDD 2002 (2002)
8. Elmagarmid, A., Ipeirotis, P., Verykios, V.: Duplicate record detection: A survey. TKDE 19 (2007)
9. Gravano, L., Ipeirotis, P.G., Jagadish, H.V., Koudas, N., Muthukrishnan, S., Srivastava, D.: Approximate string joins in a database (almost) for free. In: VLDB 2001, pp. 491–500 (2001)

10. Han, J., Kamber, M.: The data warehouse ETL toolkit: Practical techniques for extracting, cleaning, conforming, and delivering data. John Wiley and Sons, Chichester (2004)
11. Jaro, M.A.: Unimatch: A record linkage system: User's manual. Technical report, U.S. Bureau of the Census, Washington, D.C (1976)
12. Jaro, M.A.: Advances in record linkage methodology as applied to matching the 1985 census of tampa, florida. Journal of the American Statistical Society 84, 414–420 (1989)
13. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions and reversals. Doklady Akademii Nauk SSSR 163, 845–848 (1965)
14. Hernandez, M., Stolfo, S.: The merge/purge problem for large databases. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (1995)
15. Manber, U., Myers, G.: Suffix arrays: a new method for on-line string searches. SIAM Journal on Computing 22, 935–948 (1993)
16. McCallum, A., Nigam, K., Ungar, L.H.: Efficient clustering of high-dimensional data sets with application to reference matching. In: ACM SIGKDD, pp. 169–178 (2000)
17. Monge, A.E., Elkan, C.P.: An efficient domain-independent algorithm for detecting approximately duplicate database records. In: Proceedings of DMKD 1997, pp. 23–29 (1997)
18. Ramos, J.: Using tf-idf to determine word relevance in document queries. In: Proceedings of the First Instructional Conference on Machine Learning (2003)
19. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. Journal of Molecular Biology 147, 195–197 (1981)
20. Sutinen, E., Tarhio, J.: On using q-gram locations in approximate string matching. In: Spirakis, P.G. (ed.) ESA 1995. LNCS, vol. 979, pp. 327–340. Springer, Heidelberg (1995)
21. Ukkonen, E.: Approximate string matching with q-grams and maximal matches. Theoretical Computer Science 92, 191–211 (1992)
22. Ullman, J.: A binary n-gram technique for automatic correction of substitution, deletion, insertion, and reversal errors in words. The Computer Journal 20, 141–147 (1977)
23. Winkler, W.E.: The state of record linkage and current research problems. In: Statistics of Income Division (1999)