**RSEARCH: Enhancing Keyword Search in Relational Databases Using Nearly Duplicate Records**

Xiaochun Yang, Bin Wang, Guoren Wang, Ge Yu

Key Laboratory of Medical Image Computing (Northeastern University), Ministry of Education
School of Information Science and Engineering, Northeastern University, China
{yangxc, binwang, wanggr, yuge}@mail.neu.edu.cn

**Abstract**

The importance of supporting keyword searches on relations has been widely recognized. Different from the existing keyword search techniques on relations, this paper focuses on nearly duplicate records in relational databases due to abbreviation and typos. As a result, processing keyword searches with duplicate records involves many unique challenges. In this paper we discuss the motivation and present a system, RSEARCH, to show challenges in supporting keyword search using nearly duplicate records and key techniques including identifying nearly duplicate records and generating results efficiently.

**1 Introduction**

RDBMSs are very popular to store a huge amount of data due to their rigid schema specifications and mature query processing techniques. Conventional RDBMSs provide SQL interfaces and require users to understand how data is stored in it. However, some users might not know how to write an SQL to get what they are interested in, instead, they hope RDBMSs provide IR-style facilities that allow users to access the database using a set of keywords. Many recent work have studied the problem of keyword search in relational databases [1, 2, 5–7, 11–14]. They mainly focus on the mapping between keywords and data in relations and explore different semantic meanings to explain the query results without considering correlations among data.

In relational databases, duplication among tuples is one of typical data correlation. Informally, we say two records are nearly duplicate if they identify the same real-world entity. Nearly duplicate records exists in many databases due to data was collected from heterogenous sources. Figure 1 shows part of records in a real database: Science Citation Index Expanded (SCI-E)\(^1\). The data was collected from different publishers who use different format of author names and journal/conference names in their reference list. It is not surprising to see many nearly duplicate records appear as individual tuples in relations, since a real-world entity might be expressed using different values due to abbreviation, type errors, and etc.

For instance, in Figure 1, relation Source contains two duplicate records \(s_4\) and \(s_5\). Although journal names are different, we can easily find the journal name of \(s_4\) is an abbreviation of \(s_5\). The situation becomes a little complex in relation Author: three records \(a_2\), \(a_3\), and \(a_4\) contain the same author name, however, if we examine

\(^1\)http://isiknowledge.com
their published paper manually, we found $a_2$ and $a_4$ represent the same author Alon Y. Halevy in $a_1$, whereas $a_3$ does not. Such phenomena result in the following two problems:

(i) users might retrieve wrong search results, or

(ii) users might miss some information that they are really interested in.

We use Example 1 to illustrate the problems.

**Example 1:** We used a 2-keyword query, *Alon Y. Halevy* and *2009* to search the SCI-indexed paper of Alon Y. Halevy published in 2009 in the Science Citation Index Expanded database, unfortunately we got nothing. Instead, when using *Halevy A* and *2009* we could get 5 results, 3 of them were written by Alon Y. Halevy (marked by “✓” in Figure 2(a)), and the other two results were not correct. When we used *Halevy AY* and *2009*, we got a different result shown in Figure 2(b).

Ideally, we want to exclude irrelevant results and collect all correct results from relational databases to increase both precision and recall. In this paper, we present a system, RSEARCH, to analyze data between tuples.
in a relation and identify nearly duplicate records to enhance the keyword search ability.

In the remaining part of this paper, we introduce the architecture of RSearch in Section 2. In Section 3 we study several challenges that arise naturally when considering nearly duplicate records in the keyword search processing, and introduce the basic idea of our techniques. Finally, we discuss future directions in Section 4.

2 The RSEARCH System: Architecture

Figure 3 shows the system architecture of RSEARCH. It mainly consists of five modules: Nearly Duplicate Records Identifier, Indexer, Node Locator, Result Generator, and Result Ranker. We describe the process followed by a brief overview of the various modules.

- **Nearly Duplicate Records Identifier.** The Nearly Duplicate Records Identifier analyzes data correlation in relations, identifies nearly duplicate records, and generates a database graph.

  A relational database can be modeled as a database graph $G = (V, E_f, E_d)$. Each tuple in the database corresponds to a vertex in $V$, and the vertex is associated with all attribute values in the tuple. An edge $e \in E_f$ from one vertex to another one represents a foreign-key relationship. Different from the existing graph-based approaches [2, 5, 9, 13], we use $E_d$ to express another type of edges, which we call duplicate edges to express the relationship between two nearly duplicate records. For simplicity, the graph can also be modeled as an undirected graph. The graph can have weights based on different semantics [2, 13]. For example, Figure 4 shows the database graph of the database tuples in Figure 1. The dotted edges represent duplicate edges.

- **Indexer.** The Indexer constructs indices based on database graph $G = (V, E_f, E_d)$. In addition, the Indexer maintains nearly duplicate records in indices. The index structure is a trie. Any token of string type in the relational database can be expressed as a path from root to a leaf in the trie. We use a node to express each of values of other data types, like integer, float, date, and etc.

- **Node Locator.** Once a user issues a query, the Node Locator accesses Indexer and retrieves matches to each token in the given keywords in the database graph $G = (V, E_f, E_d)$. We call such nodes keyword nodes. Besides locating keywords nodes, the Indexer uses duplicate edges $E_d$ in the database graph $G$ to locate nodes that donot contain keywords but are regarded as the same entities with the keyword nodes. We call them shadow nodes. There is an duplicate edge in between a keyword node and a shadow node.
Figure 4: The database graph of the database tuples in Figure 1.

- **Result Generator.** Based on the two kinds of located nodes: keyword nodes and shadow nodes, the Result Generator decides how to connect them to generate results. Notice that, a search result might not contain any keyword nodes, but is also interested by users. For example, the path $a_2 - w_2 - p_7 - s_4$ in Figure 4 should be an interested result to the 2-keyword query Alon Halevy and VLDB Journal, although the result does not contain any keyword nodes. Here, both $a_2$ and $s_4$ are shadow nodes. Without considering duplicate edges in the database graph, the system could not generate this result.

- **Result Ranker.** The Result Ranker ranks the results generated by the Result Generator based on different ranking functions, such as variants of term frequency, the number of keyword matches, query result size, weight in the database graph, and etc. For simplicity, in this paper, let the weight of duplicate edges be 0. Notice that, when using the number of keyword matches, a shadow node is also a match to its corresponding keyword.

3 Key Modules in RsEARCH

In this section, we discuss key modules in RSEARCH and address several challenges of keyword search in relational databases with nearly duplicate records.

3.1 Identifying Nearly Duplicate Records

Two nearly duplicate records might look similar, for example, in Figure 1 records $s_2$ and $s_3$ in relation Source are similar in name and ISSN columns. In fact, only partial attributes in a record could identify duplicates. For example, the attribute email in Author, ISSN in Source, and the combination of attributes sid, vol(number), and page in Paper could be used to identify duplicates. We call such attributes duplicate identifiers.

Duplicate identifier can help us to discriminate most of the nearly duplicate records. For instance, email is a typical duplicate identifier to distinguish a person from others. Using email in Author, we know $a_1$ and $a_2$ are duplicate, and the two records $a_5$ and $a_6$ are also duplicate.

Formally, given a relation $R$ with a set of attributes $U = \{id, A_1, \ldots, A_n\}$, a similarity function $\delta$, and a similarity threshold $\lambda$. A subset of attributes $X \subseteq U - \{id\}$ is a duplicate identifier on a relation $R$, if the following statement holds: “If for any two tuples $t_i$ and $t_j$ of $R$ agree on $X$ (i.e. $t_i[X] = t_j[X]$), their corresponding values $t_i[A_1, \ldots, A_n]$ and $t_j[A_1, \ldots, A_n]$ are similar (i.e. $\delta(t_i[A_1, \ldots, A_n], t_j[A_1, \ldots, A_n]) \leq \lambda$).” We say $X$ approximately determines $R$, denoted $X \sim R$.

Many similarity functions have been used to evaluate the closeness of two records, such as edit distance, jaccard similarity, cosine similarity, and etc [3, 4]. However, it is hard to determine which similarity function(s) and threshold value(s) should be used to evaluate similarity for different attributes.
In addition, we cannot use duplicate identifier to discriminate all nearly duplicate records. For example, we donot know whether a4 and a5 are duplicate since a4 has no corresponding email address. In this case, we need more complex way to do the discrimination. For example, Halevy A is the abbreviation of Halevy Alon, and both of them write the same paper p2. Therefore, we infer that a4 and a5 are also duplicate.

The above observations show that automatically identifying duplicates is technically very nontrivial, however a manually solution is labor intensive and obviously undesirable.

RSEARCH provides a smart way to identify nearly duplicate records using following steps.

(1) Finding candidate duplicate identifiers. For each relation R, the Nearly Duplicate Record Identifier firstly finds possible duplicate identifier X (if any) in R by extracting X \(\sim R\). Obviously, an id attribute of R is not a duplicate identifier. Because the number of possible duplicate identifiers is exponential in the number of remaining n attributes \(A_1, \ldots, A_n\) in R, efficient search techniques and pruning heuristics are essential ingredients of these approaches. Different from the soft key discovery algorithm developed in [8], we examine each attribute \(A_i\) in R and prune attributes based on the following two heuristic rules:

(i) If at least one distinct value in an attribute \(A_i\) is frequent, then \(A_i\) should not be a duplicate identifier. For example, neither attribute year nor citation-times in Figure 1 could be a duplicate identifier.

(ii) If the cardinality of an attribute \(A_i\) is approximately equals to the cardinality of R, then \(A_i\) could not be a duplicate identifier. For example, title of papers is not a duplicate identifier.

All these operations can be done easily using SQLs inside the RDBMS. For example, we can easily use the expression \(\text{COUNT} (\text{DISTINCT} \ A_i)\) to count the number of distinct values in \(A_i\) and determine whether \(A_i\) should be pruned according to the maximum count number.

(2) Verification of duplicate identifiers using single attribute. For the remaining attributes in R, we verify \(A_i \sim A_j \ (i \neq j)\) by computing similarity between values in \(A_j\). If \(A_i \sim A_j\) does not hold, we prune \(A_i\) from the candidate set of duplicate identifiers. The basic idea is to partition values in \(A_i\) firstly. For each partition in \(A_i\), we find the corresponding values in \(A_j\) and adopt our approximate string matching approaches [10, 15] on them to verify whether these values are similar. If they are not similar, we then conclude \(A_i \sim A_j\) does not hold.

Notice that, as we mentioned in the above, it is hard to choose a suitable similarity function and a threshold to determine whether two records on \(A_j\) are close enough. RSEARCH begins with a “Show me more similar samples” and learns a good similarity function and threshold based on the samples.

If \(A_i\) approximately determines each attribute in \(U - \{id\}\), we say \(A_i\) is the duplicate identifier. We conclude records in each partition of \(A_i\) are duplicates.

### 3.2 Result Generator: Propagation along Duplicate Edges

There are many approaches to generate results in a database graph without duplicate edges. L. Qin et al. in [12] summarizes those approaches into three types of semantics: connected tree semantics, distinct root semantics, and distinct core semantics.

Duplicate edges make the database graph more complicated and we need traverse the graph along these duplicate edges. We use connected tree semantics [2, 7, 11] to explain the problem\(^2\). Consider a 3-keyword query, Luna, Alon Halevy, and uncertain. Figure 5 shows the query results. Figure 5(a) is a result without considering any duplicate edges, which means Luna and Alon Halevy coauthored a paper p2 on uncertain. When considering duplicate edges, RSEARCH generates the other four results. The result in Figure 5(b) shows

\(^2\) Notice that our approach is orthogonal to the existing graph-based keyword search approaches, which can be adopted easily by considering duplicate edges in the database graph.
Figure 5: Query results.

the same meaning with the one in Figure 5(a) but using a keyword node $a_7$ and a shadow node $a_4$, which is propagated from the keyword node $a_5$. The result in Figure 5(c) means Luna and Halevy A write paper $p_2$ and $p_1$ separately. The title of these two papers contain uncertain. The result in Figure 5(d) means Luna published a paper about uncertain in a journal, meanwhile the journal publishes another paper wrote by Alon Y. Halevy. The result in Figure 5(e) has the similar meaning with the fourth result.

Different from the existing approaches, when a database graph contains duplicate edges, a critical problem is to decide when and how to propagate along duplicate edges. In RSEARCH, the Node Locator firstly locates all keyword nodes and traverses the database graph $G$ using its foreign-key edges $E_f$. The Result Generator finds all results using keyword nodes, if any. For each result $T$, it expands it by propagating along duplicate edges. It uses a shadow node $n_s$ to replace the corresponding keyword node $n_k$ in $T$ through a duplicate edge $e$. If $n_s$ is reachable to the result through foreign-key edges $E_f$, we say it is a valid propagation. The Result Generator then generate another results using $E_f$. If the Result Generator could not generate a result using keyword nodes and $E_f$, it needs use duplicate edges $E_d$ to generate more results by using the same policy of the existing approaches.

4 Conclusions and Future Work

We motivated our work by considering nearly duplicate records in relational databases and show challenges in developing a keyword search system for RDBMSs. We introduce RSEARCH system that can provide keyword search results with higher precision and recall when considering nearly duplicate records in relations. We briefly present two key techniques in RSEARCH including automatically identifying nearly duplicate records and generating query results efficiently.

Future research directions include further analyzing effects on search results of rich data correlations inside relational database. Besides nearly duplicate records, there are many types of data correlations, such as mutual exclusive of tuples, association rules in relational database. Such data correlations will greatly affect the precision and recall of search results, as well as the search efficiency. However, none of the existing approaches on relational keyword search considers data correlations. Another critical issue is to evaluate relational database search systems. So far, there are many approaches and semantics of generating search results for given keywords. The quality of a relation keyword search system greatly depends on users preference. We expect to allow maximum flexibility for using different semantics according to different preferences.
5 Acknowledgment

The work is partially supported by the National Nature Science of China (Nos. 60828004, 60973018) and the Center Education Fundamental Scientific Research (No. N090504004).

References