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

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

[1]http://isiknowledge.com

Author

id	name	email
a_1	Alon Y. Halevy	alon@cs.washington.edu
a_2	Halevy A	alon@cs.washington.edu
a_3	Halevy A	avinoams@clalit.org.il
a_4	Halevy A	
a_5	Halevy Alon	halevy@google.com
a_6	Halevy AY	halevy@google.com
a_7	Dong XL	lunadong@research.att.com

Write

id	aid	pid
w_1	a_1	p_6
w_2	a_2	p_7
w_3	a_3	p_5
w_4	a_4	p_1
w_5	a_4	p_2
w_6	a_4	p_4
w_7	a_5	p_2
w_8	a_6	p_3
w_9	a_7	p_2

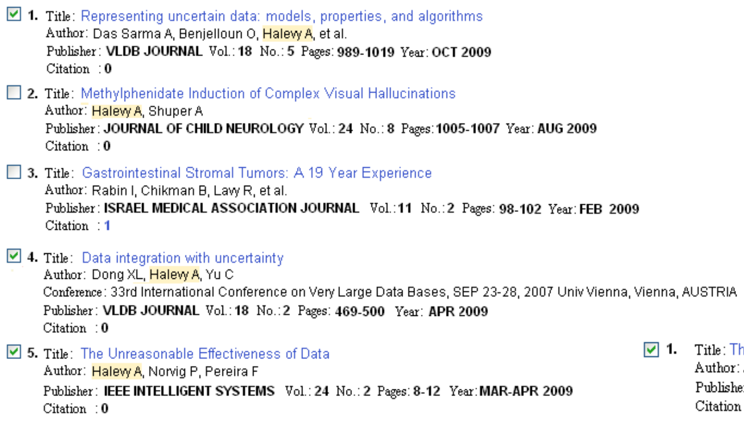
Source

id	name	ISSN
s_1	Communication of the ACM	0001-0782
s_2	IEEE Intelligent System	1541-1672
s_3	Journal of Child Neurology	0883-0738
s_4	VLDB J.	1066-8888
s_5	VLDB Journal	1066-8888

Paper

id	title	sid	year	vol(number)	page	citation-times
p_1	Data integration with uncertainty	s_5	2009	18(2)	469-500	0
p_2	Representing uncertain data...	s_5	2009	18(5)	989-1019	0
p_3	The Claremont Report on Database Research	s_1	2009	52(6)	56-65	0
p_4	The Unreasonable Effectiveness of Data	s_2	2009	24(2)	8-12	0
p_5	... Complex Visual Hallucinations	s_3	2009	24(8)	1005-1007	0
p_6	Schema mediation ... semantic data sharing	s_4	2005	14(1)	68-83	11
p_7	MiniCon: ... answering queries using views	s_4	2001	10(2-3)	182-198	33

Figure 1: Sample database in Science Citation Index Expanded (SCI-E).



✓ 1. Title: Representing uncertain data: models, properties, and algorithms
Author: Das Sarma A, Benjelloun O, Halevy A, et al.
Publisher: VLDB JOURNAL Vol.: 18 No.: 5 Pages: 989-1019 Year: OCT 2009
Citation : 0

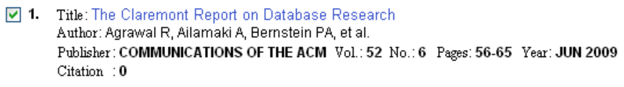
☐ 2. Title: Methylphenidate Induction of Complex Visual Hallucinations
Author: Halevy A, Shuper A
Publisher: JOURNAL OF CHILD NEUROLOGY Vol.: 24 No.: 8 Pages: 1005-1007 Year: AUG 2009
Citation : 0

☐ 3. Title: Gastrointestinal Stromal Tumors: A 19 Year Experience
Author: Rabin I, Chikman B, Lavy R, et al.
Publisher: ISRAEL MEDICAL ASSOCIATION JOURNAL Vol.: 11 No.: 2 Pages: 98-102 Year: FEB 2009
Citation : 1

✓ 4. Title: Data integration with uncertainty
Author: Dong XL, Halevy A, Yu C
Conference: 33rd International Conference on Very Large Data Bases, SEP 23-28, 2007 Univ Vienna, Vienna, AUSTRIA
Publisher: VLDB JOURNAL Vol.: 18 No.: 2 Pages: 469-500 Year: APR 2009
Citation : 0

✓ 5. Title: The Unreasonable Effectiveness of Data
Author: Halevy A, Norvig P, Pereira F
Publisher: IEEE INTELLIGENT SYSTEMS Vol.: 24 No.: 2 Pages: 8-12 Year: MAR-APR 2009
Citation : 0

(a) Input *Halevy A* and *2009*.



✓ 1. Title: The Claremont Report on Database Research
Author: Agrawal R, Ailamaki A, Bernstein PA, et al.
Publisher: COMMUNICATIONS OF THE ACM Vol.: 52 No.: 6 Pages: 56-65 Year: JUN 2009
Citation : 0

(b) Input *Halevy AY* and *2009*.

Figure 2: Keyword search results.

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 phenomenons 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 "$\surd$" 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

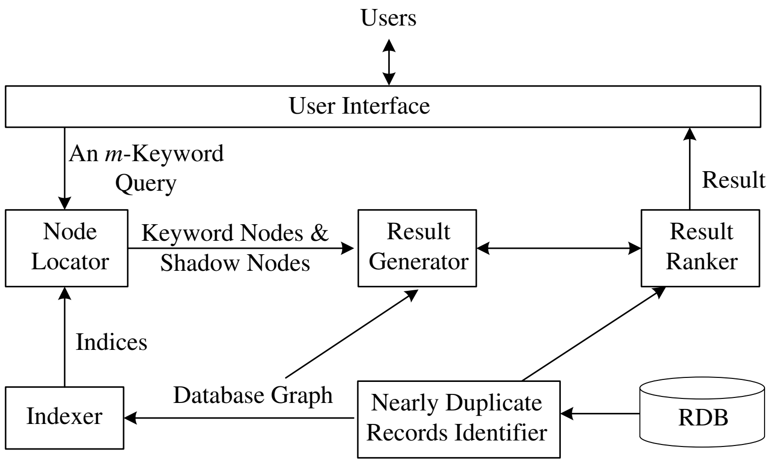


Figure 3: Architecture of RSEARCH.

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.

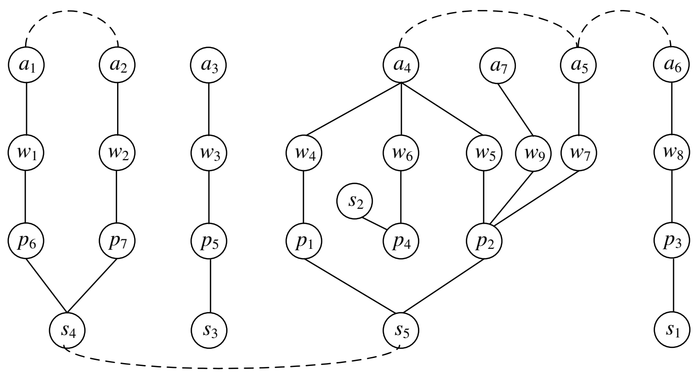


Figure 4: The database graph of the database tuples in Figure 1.

- *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 δ, and a similarity threshold λ. 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 $X \rightsquigarrow R$.

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 a_4 and a_5 are duplicate since a_4 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 p_2. Therefore, we infer that a_4 and a_5 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 identifers. For each relation R, the *Nearly Duplicate Record Identifier* firstly finds possible duplicate identifer X (if any) in R by extracting $X \rightsquigarrow R$. Obviously, an *id* attribute of R is not a duplicate identifer. Because the number of possible duplicate identifers 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 identifer. For example, neither attribute *year* nor *citation-times* in Figure 1 could be a duplicate identifer.

   (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 identifer. For example, *title* of papers is not a duplicate identifer.

   All these operations can be done easily using SQLs inside the RDBMS. For example, we can easily use the expression COUNT(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 identifers using single attribute. For the remaining attributes in R, we verify $A_i \rightsquigarrow A_j$ $(i \neq j)$ by computing similarity between values in A_j. If $A_i \rightsquigarrow A_j$ does not hold, we prune A_i from the candidate set of duplicate identifers. 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 $A_i \rightsquigarrow A_j$ 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 p_2 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.

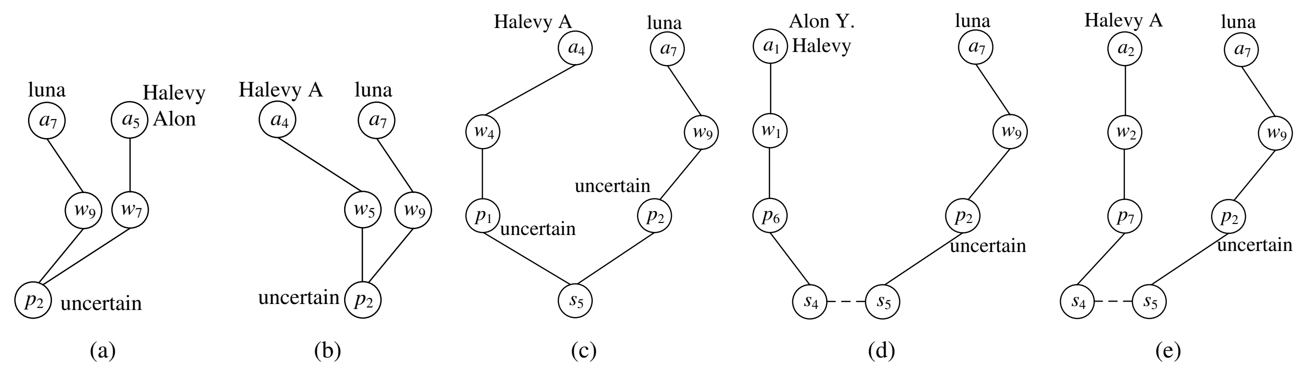


Figure 5: Query results.

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

[1] S. Agrawal, S. chaudhuri, and G. Das. DBXplorer: A system for keyword-based search over relational databases. In *ICDE*, pages 5-16, 2002.

[2] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *ICDE*, pages 431-440, 2002.

[3] R. Bayardo, Y. Ma, and R. Srikant. Scaling up all pairs similarity search. In *WWW Conference*, pages 131-140, 2007.

[4] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *VLDB*, pages 491-500, 2001.

[5] H. He, H. Wang, J. Yang, and P. S. Yu. BLINKS: ranked keyword searches on graphs. In *SIGMOD*, pages 305-316, 2007.

[6] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient IR-style keyword search over relational databases. In *VLDB*, pages 850-861, 2003.

[7] V. Hristidis and Y. Papakonstantinou. DISCOVER: keyword search in relational databases. In *VLDB*, pages 670-681, 2002.

[8] I. F. Ilyas, V. Markl, P. Haas, P. Brown, and A. Aboulnaga. CORDS: Automatic discovery of correlations and soft functional dependencies. In *SIGMOD*, pages 647-658, 2004.

[9] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB*, pages 505-516, 2005.

[10] C. Li, B. Wang, and X. Yang. Vgram: Improving performance of approximate queries on string collections using variablelength grams. In *VLDB*, pages 303-314, 2007.

[11] Y. Luo, X. Lin, W. Wang, and X. Zhou. Spark: top-k keyword query in relational databases. In *SIGMOD*, pages 115-126, 2007.

[12] L. Qin, J. X. Yu, and L. Chang. Keyword search in databases: the power of RDBMs. In *SIGMOD*, pages 681-694, 2009.

[13] L. Qin, J. X. Yu, L. Chang, and Y. Tao. Querying communities in relational databases. In *ICDE*, pages 724-735, 2009.

[14] A. Simitsis, G. Koutrika, and Y. E. Ioannidis. Précis: from unstructured keywords as queries to structured databases as answers. *VLDB J.*, 17(1): 117-149, 2008.

[15] X. Yang, B. Wang, and C. Li: Cost-Based Variable-Length-Gram Selection for String Collections to Support Approximate Queries Efficiently. In *SIGMOD*, pages 353-364, 2008.