Supporting Similarity Operations based on Approximate String Matching on the Web

Eike Schallehn[1], Ingolf Geist[1], and Kai-Uwe Sattler[2]

[1] Dpt. of Computer Science, University of Magdeburg
P.O. Box 4120, 39106 Magdeburg, Germany
schallehn|geist@iti.cs.uni-magdeburg.de
[2] Dpt. of Computer Science and Automation, Technical University of Ilmenau
P.O. Box 100565, 98684 Ilmenau, Germany
k.sattler@computer.org

Abstract. Querying and integrating sources of structured data from the Web in most cases requires similarity-based concepts to deal with data level conflicts. This is due to the often erroneous and imprecise nature of the data and diverging conventions for their representation. On the other hand, Web databases offer only limited interfaces and almost no support for similarity queries. The approach presented in this paper maps string similarity predicates to standard predicates like substring and keyword search as offered by many of the mentioned systems. To minimize the local processing costs and the required network traffic, the mapping uses materialized information on the selectivity of string samples such as q-samples, substrings, and keywords. Based on the predicate mapping similarity selections and joins are described and the quality and required effort of the operations is evaluated experimentally.

1 Introduction

The growing amount of information publicly or locally available from a growing number of databases and other informations systems in networks raises the need for an integrated or mediated access to this information. Among the many problems to be solved is the resolution of data level conflicts in weakly related or overlapping data sets from different sources. Similarity-based operations became one way to address this problem in data integration scenarios. Unfortunately, the support for such operations in current data management solutions is rather limited. And worse, interfaces provided over the Web are even more limited and almost always do not allow any similarity-based lookup of information.

The principal idea of the presented approach is to provide a pre-selection for string similarity operations by using string containment operations as provided by most information systems. Regarding the pre-selection this approach is similar to those by Gravano et al. introduced in [5]. Contrary to their pre-selection strategy, the one presented here is not only applicable in a scenario were integrated data sets or data sets in general are materialized in one database, but allows re-writing string similarity queries for the virtual integration of autonomous sources. This way, it is applicable in Web integration scenarios.

The proposed pre-selection is based on the edit or Levenshtein distance, which expresses the dissimilarity of two strings by the minimal number k of operations necessary to transform a string to a comparison string. A basic observation described by Navarro et al. in [15] is, that if we pick any $k + 1$ non-overlapping substrings of one string, at least one of them must be fully contained in the comparison string. This corresponds to *Count Filtering* as introduced by Gravano, where the number of common q-grams (substrings of fixed length q) in two strings is used as a criterion.

The problem is, we can not use additional filtering techniques like described in [5] to further refine the pre-selection, because we can not access the necessary information in a non-materialized scenario. And, if we choose inappropriate substrings, the candidate sets can be huge. In this case, the question is: which substrings are appropriate? Obviously, we can minimize the size of the intermediate result by finding the $k + 1$ non-overlapping substrings having the best selectivity when combined in one disjunctive query. Then, processing a string similarity predicate requires the following steps:

1. Transform the similarity predicate to an optimal disjunctive substring pre-selection query considering selectivity information
2. Process the pre-selection using standard functionality of the information system yielding a candidate set
3. Process the actual similarity predicate within a mediator or implemented as a user defined function in standard DBMS

While this sketches only a simple selection, we will describe later on, how for instance similarity joins over diverse sources can be executed.

The remainder of this paper is structured as follows. After giving an overview of related work in Section 2 we will describe the mapping of string similarity predicates in Section 3. Based on the described mapping we outline how similarity selections and joins can be performed in an integration scenario in Section 4. Section 5 describes necessary data structures and algorithms for maintaining information on substring selectivity required for the predicate mapping. Finally, we present experimental results in Section 6 and conclude with a short summary and outlook in Section 7.

2 Related Work

The roots of this research stem from similarity operations in Information Retrieval and probabilistic data processing. Spatial and similarity joins were first addressed for materialized scenarios and data values that either represented points in a mutidimensional metric space or could be mapped to such a space. A recent overview is given by Koudas and Sevcik in [12]. Though searching and performing more complex operations in multidimensional spaces is well researched, string data would have to be mapped to such a space of fixed dimensionality to apply the previously mentioned approaches. This can be done using for instance FastMap introduced by Faloutsos and Lin in [3], which is based on metric multidimensional scaling, and was actually used for this purpose by Jin et al. as described in [10]. Nevertheless, this approach requires a fully materialized data set, the full domain of string values to define the mapping, and according interfaces to perform a similarity search based on a vector representation of a string. Based on Fuhr's

probabibilistic datalog ([4]) in [2] Cohen described a related approach for performing joins based on textual similarity, contrary to shorter strings used here. For this purpose he applied document vector representations as known from Information Retrieval.

Our approach is based on the edit or Levenshtein distance for string values. A short overview of similarity measures in general is included in [18] by Santini and Jain, while Navarro gives an overview of approximate string matching in [14]. Based on distance and similarity measures for string values according similarity operations were introduced in recent research, such as the previously mentioned approach by Jin et al. in [10]. In [5] and [6] Gravano et al. present and refine an approach to perform joins based on similarity of string attributes through efficient pre-selections of materialized q-grams. Schallehn and Sattler in [19] use temporarily created Tries for similarity operations based on string similarity research by Shang and Merret ([20]). Nevertheless, all these approaches are only applicable if the data sets to be joined are locally materialized or the source systems do not have to process the similarity predicates.

In addition to the previously mentioned research, the approach presented here builds on [15] by Navarro and Baeza-Yates. They use q-gram indexes for approximate searches within texts and q-samples chosen by their selectivity for querying in an information retrieval context. An overview of indexing techniques for approximate string matching is given for instance in [16]. Possible index structures are suffix trees, suffix arrays as well as q-gram and q-sample indexes.

The count-suffix tree was proposed by Krishnan et al. [13] and refined by Jagadish et al. in [9]. Especially, the pruned version of a count-suffix tree is useful for substring selectivity estimation with tight space requirements. Pruned q-gram tables are very similar to end-biased histograms [7]. The highest frequency information are maintained and all lower frequencies are put into one bucket and estimated with one frequency. Because a q-gram index contains only strings with a fixed length q the problem that shorter and longer strings are estimated with the same selectivity does not occur. In our approach count-tries are used for storing information about q-grams of varying lengths q. These tries can be seen as count-suffix trees as described in [13, 9], which contain only prefixes of the suffix of a supported lengths, i.e. the number of levels is pruned. Query-based sampling [1, 8] is a method to obtain source descriptions from text-databases, i.e. tokens and their corresponding frequency information. Based on this idea we adapted the general approach to q-grams and substring queries.

Other work related to our approach deals with the implementation of data integration operators (mainly joins) in the presence of sources with limited query capabilities. The specification of query capabilities is addressed e.g. in [21], where the set of queries accepted by a source (or wrapper) is described using Datalog variants and is used for a capbilities-based rewriting. For the implementation of join operations on sources with limited capabilities the bind join was introduced in [17], where tuples from the results of one relation are used to fetch the corresponding tuples from the second relation even if this source does not support a full table scan.

3 Mapping Similarity predicates

We consider a predicate like $edist(x, y) \leq k$ as part of a join condition, where x and y represent attribute names, or of a selection criterion, where one may represent a literal search string. We use the classic definition of the edit distance, which includes only insertion, deletion, and replacement. In this case, for a threshold k the number of required non-overlapping substrings is $n = k + 1$, because all of the above mentioned operations can only modify one substring each. A common derivative in addition allows transpositions of characters and increases the number of sub-strings to be considered to $n = 2k + 1$, because every transposition can modify two substrings.

Considering what kind of substring is most suitable, let us assume a predicate $edist('Vincent\ van\ Gogh', stringAttribute) \leq 1$. Assuming we have selectivity information $sel(a)$ about any substring $a = s[i, j]$, $0 \leq i < j < length(s)$ of $s \in \Sigma^*$ over an alphabet Σ available as discussed later in Section 5, we may choose the following substrings for pre-selection predicates:

- **Arbitrary Substrings:** 'Vincent van' ∨ ' Gogh'
- **Fixed length substrings (q-samples):** 'Vinc' ∨ 'Gogh' (here $q = 4$)
- **Tokens:** 'Vincent' ∨ 'Gogh'

All three obviously must yield a candidate set including the correct result, but they differ largely regarding their selectivity. Intuitively, longer strings have a better selectivity, because every additional character refines the query. This consideration would render the transformation to q-samples as the least effective one. On the other hand, there is an overhead for managing and using selectivity information. Storing such information for arbitrary strings requires complex data structures to be efficient and considerable memory resources. In general, choosing a suitable substring paradigm implies a trade-off between several aspects.

Selectivity: as mentioned above, the selectivity of longer substrings is always better than or, in the unlikely worst case, equal to a shorter substring, $sel(s[i, j]) \geq sel(s[k, l]), 0 \leq k \leq i \leq j \leq l < length(s)$. Choosing a small q as for instance 3 or 4 will likely return more intermediate results and this way introduce a high overhead for transfer and local processing.

Maintenance: independently of what data structure we use for maintaining selectivity information, the required data volume grows dramatically with the (possible) length of the substrings due to a combinatoric effect for each additional position. Hence, a greater q increases the necessary overhead for global processing and the global resource consumption.

Applicability: we run into problems if a comparison string is not long enough to derive the necessary number of substrings such as tokens or q-samples. For instance, if the allowed edit distance is $k = 3$ and $q = 5$ a disjunctive pre-selection must contain $n = k + 1 = 4$ q-samples of length 5, i.e. the minimal required length of the mapped search string is $l_{min} = n * q = 20$. Obviously, it is not possible to derive the necessary 5-samples from the string 'Vincent van Gogh'.

Source capabilities: we consider two kinds of sources regarding the query capabilities, those allowing substring and those allowing keyword searches. For the latter, only tokens are suitable for composing pre-selection queries.

3.1 Substring decomposition

The optimal solution to the addressed problem regarding selectivity performs the mapping *map_substring* in terms of a complete decomposition of the search string s into $n = k + 1$ non-overlapping substrings. The decomposition consists of positions $pos[0] \ldots pos[n]$ with $pos[0] = 0$ and $pos[n] = length(s)$ such that the concatenation $s = s[pos[0], pos[1] - 1]s[pos[1], pos[2] - 1] \ldots s[pos[n-1], pos[n] - 1]$ of the substrings is equal to the search string. An optimal decomposition would yield the minimal selectivity $1 - \Pi_{i=0}^{n-1}(1 - sel(s[pos[i], pos[i+1] - 1]))$. Here we assume independence between the selected query strings.

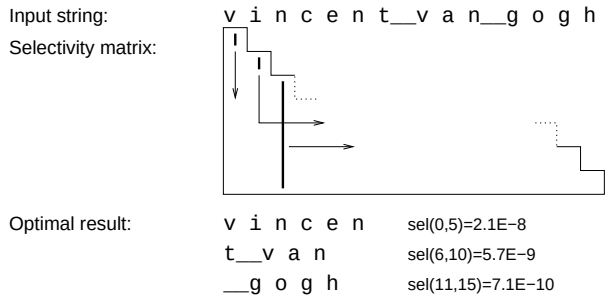


Fig. 1. Finding selective substrings for $k = 2$, hence $n = k + 1 = 3$

The algorithm sketched in Figure 1 uses a lower triangular matrix A where a_{ij} represents the selectivity of substring $s[i, j]$, hence, $0 \leq i \leq j < length(s)$. If a count suffix trie is used for storing selectivity information, as shown in Section 5, this matrix can be generated from $length(s)$ path traversals in the trie. An exhaustive search is quite expensive for long strings, but it can be tuned by skipping high selectivities in the upper region of the triangular matrix. Furthermore, starting with a decomposition of equal length substrings and stepwise adjusting this decomposition by moving adjacent cut positions represents a greedy approach yielding sufficient results regarding the selectivity quickly. The disadvantage here is that we need selectivity information on the variable length substrings $s[pos[i], pos[i+1] - 1]$. Possible solutions and problems for the storage and retrieval of this information is outlined in Section 5, but obviously it requires much more resources than managing the same information for q-samples as introduced in the following.

3.2 q-samples

The main advantage of using q-samples, i.e. non-overlapping q-grams of fixed length q, for a mapping *map_qgram* of an edit distance predicate to a disjunctive source query results from the straightforward maintenance of according selectivity information, as shown later on in Section 5.

To find the best possible combination of n q-samples from a single string s with $length(s) \geq n * q$ an algorithm basically works as shown in Figure 2. In a first

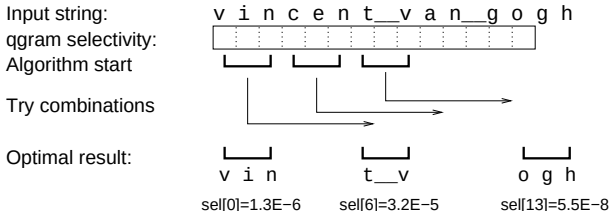


Fig. 2. Finding selective 3-samples for $k = 2$, hence $n = k + 1 = 3$

step selectivity information for all contained q-grams is retrieved from data structures described in Section 5 and represented in an array $sel[i] = sel(s[i, i + q]), 0 \leq i < length(s) - q$. As shown later on, this can be accomplished in $O(length(s))$ time. Among the number of all possible combinations we have to find the positions $pos[i], 0 \leq i < n$ with $\forall j, k : 0 \leq j < k < n \wedge pos[k] - pos[j] > q$ that optimizes the selectivity of the disjunctive source query, i.e. yields the minimal overall selectivity $1 - \Pi_{i=1}^{n}(1 - sel[pos[i]])$. This selectivity estimation can further be used to decide, if the pre-selection actually should be performed on the data source. If the selectivity exceeds some selectivity threshold and can not be performed efficiently, i.e. it yields too many intermediate results, the query can be rejected. As the number of possible combinations is $\Pi_{i=1}^{n}(length(s) - (n * q))$ an exhaustive search can become very expensive, especially if the mapping has to be applied during a bind-join on a great number of long strings as shown in Section 4. Alternatively, a greedy algorithm with $O(length(s))$ was implemented yielding sufficiently selective combinations, in most cases equal to the result of the exhaustive search.

The selectivity of the resulting pre-selection

$$\sigma_{\bigvee_{i=1}^{n} substring(s[pos[i],pos[i]+q],stringAttribute)}$$

can further be improved by not only considering the retrieved q-samples at $pos[i]$, but also the bounding substring, resulting in a complete decomposition of s. In the given example this may be '**vin**cen' and '**t_v**an_g' and '**ogh**', which can easily be derived. Though we can not estimate the selectivity of this query based on the given information, unless we move to the approach presented in the previous subsection, it must be better or at least equal to our estimation made based on q-gram selectivity. Another refinement of the presented approach would be to dynamically determine q based on the string length and the number of required q-samples, e.g. $q := \lfloor length(s)/n \rfloor$. This would solve the problem of applicability for shorter strings mentioned above, and improve the selectivity of the pre-selection for longer strings. The disadvantage is that we would need selectivity information for various length q-grams. Finally, if q is fixed and the applicability condition $length(s) \geq n * q$ does not hold, we may decide to nevertheless send a disjunctive query to the source, containing $m = \lfloor length(s)/q \rfloor < n$ substrings. Though this may not yield all results to the query, it still yields the subset containing $k - (n - m)$ differences in the string representations. Of course, the source query should only be executed, if the estimated selectivity $1 - \Pi_{i=1}^{m}(1 - sel[pos[i]])$ is below a threshold granting efficient processing and transfer of the pre-selection.

3.3 Tokens

Considering only substrings of a fixed or variable length would neglect the query capabilities of a great number of sources providing keyword search instead of substring search. To support such interfaces we can apply a mapping *map_token* to choose a set of tokens $T = \{t\}$ derived from our search string s using common delimiters like spaces, commas, etc. Managing and retrieving selectivity information for keywords can be based on standard approaches from information retrieval like the $TF * IDF$ norm. Therefore, it is quite straightforward as outlined in Section 5. Finding an optimal combination is also easier than with q-samples or substrings. The disadvantages of the approach are the in general worse selectivity of keywords compared to the other approaches, a relatively big space overhead for managing selectivity information compared to q-grams, and problems with the applicability. The latter results from the fact that $k + 1$ tokens have to be derived, which often may not be possible, e.g. it is impossible to derive a pre-selection for a query like

$$\sigma_{edist('ErnestHemingway',authorName)\leq 2}$$

because the threshold $k = 2$ implies the need of $n = 3$ tokens, which are not available. The selectivity problems occur because we can not take advantage of longer substrings, we can not take advantage of token-spanning substrings, and a probability growing with n of having one or more relatively un-selective keywords in our pre-selection.

4 Similarity-based Operations

The selectivity-based mapping of similarity predicates can be used for rewriting and executing similarity queries on Web sources. In this way we can support approximate string matching in global queries even if the source systems support only primitive predicates such as *substring(a, b)*, e.g. in form of SQL's "`a like '%b%'`" predicate or *keyword(a, b)* representing an IR-like keyword containment of phrase b in string a. In the following we use a generalized form *contains(a, b)* that has to be replaced by the specific predicate supported by the source system.

With regard to approximate string matching in Web queries we focus on two operations: the similarity selection returning tuples satisfying a string similarity condition, the similarity join combining tuples from two relations based on an approximate matching criterion. In the following we describe strategies for implementing these operators using selectivity-based mapping.

4.1 Similarity-based selections

Intuitively, a similarity selection $\tilde{\sigma}_{SIM(s,attr)}r(R)$ is an operation returning all tuples satisfying a similarity condition *SIM*(s, *attr*) where *attr* $\in R$ with a similarity value greater or equal than a given threshold:

$$\tilde{\sigma}_{SIM(s,attr)}r(R) = \{t \mid t \in r(R) \wedge SIM(s, t.attr) \geq threshold\}$$

A particular variant of such a similarity predicate considered here is the edit distance:

$$\tilde{\sigma}_{edist(s,attr)\leq k} r(R) = \{t \mid t \in r(R) \wedge edist(s, t.attr) \leq k\}$$

Without loss of generality we focus on simple predicates only. Complex predicates, e.g. connected by $\vee$ or $\wedge$ can be handled by applying the following steps to each atomic predicate and taking into account query capabilities of the sources. However, query capability issues are addressed in other works (see Sect. 2). Furthermore, we assume that source systems do not support such predicates but only the primitive predicate *contains(a, b)* introduced above. Now, the problem is to rewrite a query containing $\tilde{\sigma}_{SIM}$ in the following form:

$$\tilde{\sigma}_{SIM} \rightarrow \tilde{\sigma}_{SIM}(\sigma_{PRESIM}(r(R)))$$

where σ_{PRESIM} is pushed to the source system and $\tilde{\sigma}_{SIM}$ is performed in the mediator.

Assuming *SIM* is an atomic predicate of the form $edist(s, attr) \leq k$ the selection condition *PRESIM* can be derived using the mapping functions *map_qgram*, *map_substring*, *map_token* from Section 3 which we consider in the generalized form *map*. This mapping function returns a set $\{q\}$ of q-samples, substrings, or keywords according to the mappings described in Section 3. The disjunctive query represented by this set in general contains $k+1$ strings, unless the length of s does not allow to retrieve this number of substrings. In this case, the next possible smaller set is returned, representing a query returning a partial result as described before. In any case, the estimated selectivity of the represented query must be better than a given selectivity threshold.

Based on this we can derive the expression *PRESIM* from the similarity predicate as follows:

$$PRESIM := \bigvee_{\forall q \in map(s)} contains(q, attr)$$

In case of using the edit distance as similarity predicate we can further optimize the query expression by applying length filtering. This means, we can omit the expensive computation of the edit distance between two strings s_1 and s_2 if $|\,\mathrm{length}(s_1) - \mathrm{length}(s_2)| \geq k$ for a given maximum distance values k. This holds, because in this case the edit distance value is already $\geq k$. Thus, the final query expression is

$$\tilde{\sigma}_{edist(s,attr)<k}(\sigma_{|\,\mathrm{length}(s)-\mathrm{length}(attr)|\leq k}(\sigma_{PRESIM}(r(R))))$$

where the placement of the length filtering selection depends on the query capabilities of the source. A second optimization rule deals with complex disjunctively connected similarity conditions of the form $SIM(s_1, attr) \vee SIM(s_2, attr)$. In this case the preselection condition can be simplified to

$$\bigvee_{\forall q_1 \in map(s_1)} contains(q_1, attr) \vee \bigvee_{\forall q_2 \in map(s_2)} contains(q_2, attr)$$

A general problem that can occur in this context are query strings exceeding the length limit for query strings given by the source system. This has to be handled by

splitting the query condition into two or more parts $PRESIM_1 \ldots PRESIM_n$ and building the union of the partial results afterwards:

$$\tilde{\sigma}_{SIM}(\sigma_{PRESIM_1}(r(R)) \cup \cdots \cup \sigma_{PRESIM_n}(r(R)))$$

Obviously, the above mentioned optimization of applying length filtering can be used here, too.

4.2 Similarity join

Based on the idea of implementing similarity operations by introducing a pre-selection we can realize similarity join operations, too. A similarity join $r_1(R_1)\tilde{\bowtie}_{SIM} r_2(R_2)$ where the join condition is a approximate string criterion of the form $SIM(R_1.attr_1, R_2.attr_2) > threshold$ or $edist(R_1.attr_1, R_2.attr_2) \leq k$. As in the previous sections we consider in the following only simple edit distance predicates.

A first approach for computing the join is to use a bind join implementation. Here, we assume that one relation is either restricted by a selection criterion or can be scanned completely. Then, the bind join works as follows (Fig. 3). For each tuple of the outer relation r_1 we take the (string) value of the join attribute $attr_1$ and perform a similarity selection on the inner relation. This is performed in the same way as described in Section 4.1 by mapping the string to a set of q-grams, sending the selection to the source and post-process the result by applying the similarity predicate. Next, each tuple of this selection result is combined with the current tuple of the outer relation.

foreach $t_1 \in r_1(R_1)$ **do**
  $s := t(R_1.attr_1)$
  foreach $t_2 \in \tilde{\sigma}_{edist(s,attr_2)}(\sigma_{PRESIM}(r_2(R_2)))$ **do**
    output $t_1 \circ t_2$

Fig. 3. Bind join

The roles of the participating relations (inner or outer relation) are determined by taking into account relation cardinalities as well as the query capabilities. If a relation is not restricted using a selection condition and does not support a full table scan it has to be used as inner relation. Otherwise, the smaller relation is chosen as the outer relation in order to reduce the number of source queries. Obviously, a bind join implementation requires $|r_1| + 1$ source queries if no constraint on the result from r_1 exists.

A significant reduction of the number of the source queries can be achieved by using a semi-join variant. Here, one of the relations is first processed completely. The string values of the join attribute are collected and the *map* function is applied to each of them. The resulting set $\mathcal{S}$ of q-grams, tokens or substrings is used to build a single pre-selection condition. Next, this pre-selection is sent to the source. Finally, the result is joined with the tuples from the first relation using the similarity condition (Fig. 4).

If the pre-selection condition exceeds the query string limit of the source, the pre-selection has to be performed in multiple steps. In the best case, this approach requires

$\mathcal{S} := \emptyset$
foreach $t_1 \in r_1(R_1)$ **do**
  $\mathcal{S} := \mathcal{S} \cup map(t(R_1.attr_1))$
$r_{tmp} := \sigma_{\bigvee_{s \in \mathcal{S}} contains(s, attr_2)}(r_2(R_2))$
foreach $t_1 \in r_1(R_1)$ **do**
  foreach $t_2 \in r_{tmp}$ **do**
    if $edist(t_1(R_1.attr_1), t_2(R_2.attr_2)) < k$
      output $t_1 \circ t_2$

Fig. 4. Semi join

only 2 source queries assuming that the first relation is cached in the mediator or 3 source queries otherwise. The worst case depends on the query length limit as well as the number of derived q-grams. However, if the number of queries is greater than $|r_1| + 1$ one can switch always to the bind join implementation.

5 Managing Selectivity Information

In the previous sections we described the mapping of similarity-based predicates to substring and keyword queries. In this section we shortly review and adapt data structures to store selectivity information and algorithms to extract these information.

5.1 Data structures

There are various kinds of data structures to store information for approximate string matching, for instance described by Navarro in [16]. For the purpose of matching, these structures hold pointers to the occurrences of substrings in the text. As for selectivity estimation the number of occurrences are interesting, and not the positions themselves, the data structures were adapted to hold counts instead of pointers. Based on the kind of string decomposition possible data structures are

- full count-suffix trees (FST) or pruned count-suffix trees (PST) for arbitrary length substrings,
- hash tables or pruned hash tables which store fixed length q-grams or tokens and their corresponding counts, and
- count tries (CT) or pruned count tries (PCT), that store count information of tokens or q-grams of variable length q.

Count-Suffix tree: a suffix tree is a trie that stores not only all strings s of a database but also all suffixes $s[i, length(s) - 1], 0 \leq i < length(s)$ of s. The count-suffix tree is a variant of a suffix tree which does not store pointers to the occurrences of the substrings $s[i, j]$ but maintains the count of substrings $C_{s[i,j]}$. As each node corresponds to a substring $s[i, j]$ the count value $C_{s[i,j]}$ can have two meanings: (*i*) $C_{s[i,j]}$ describes the number of strings in which $s[i, j]$ is contained or *(ii)* $C_{s[i,j]}$ denotes the number of

occurrences of the substring $s[i, j]$. In our further investigations we assume the second case. The count assigned to the root node N is the number of all suffixes in the database.

The space requirements of a full count-suffix tree can be prohibitive for estimating substring selectivity because of limited available space and high costs for substring selectivity estimation, especially if we assume the operations presented in Sec. 4. Therefore, the pruned count-suffix tree was presented by Krishnan et al. in [13]. This data structure maintains only the counts of substrings that satisfy a certain pruning rule. Examples for a rule are: maintain only the top-n levels of the suffix-tree, i.e. retain only substrings with a length $length(a) \leq l_{max}$, or retain all nodes that have a count $C_a > p_{min}$, where p_{min} is the pruning threshold. A pruned count-suffix tree can be used to summarize the selectivity information of arbitrary substrings, which is the first kind of the pre-selection predicates.

q-gram hash table: The second kind of substring decomposition uses q-samples that allow cheaper storage and computation costs. The selectivity information are stored in hash tables, which contain q-grams extracted from the string in the database. Each entry in a hash table $\mathcal{H}_q$ consists of a q-gram with length q as key and the assigned count information C_{qgram}. To access the information efficiently the address is computed by a hash function, similar to Karp-Rabin [11], i.e. the hash value of a q-gram can be computed from the hash value of the previous q-gram in a string in a time of $O(1)$. These kind of hash functions are useful in constructing as well as in searching for selectivity information. In order to reduce the storage costs the hash table can be pruned using a pruning rule. A typical pruning rule is based on the count: maintain only those q-gram entries with a count greater than a given threshold, i.e. $C_{qgram} > p_{min}$. To support q-samples of varying length selectivity information of q-grams with differnt length has to be maintained. A straightforward solution can use several hash tables for different length q. However, this approach causes a considerable redundancy addressed in the next paragraph.

Count trie (CT): as mentioned in [9] a count-suffix tree can be pruned by different rules apart from minimum counts. In order to find a compressed representation of q-grams of different lengths by the maximum height of the count-suffix tree a pruning rule $p \leq q$ means, for each suffix s_i of a string s only the part $s_i[0, q]$ is stored in the count-suffix tree. For each suffix, which now represents a q-gram, the count of occurrences is maintained. Furthermore, if only q-grams of a certain minimum length p_{min} should be maintained, the pruning rule can be extended to $p_{min} \leq q \leq p_{max}$. As almost all q-grams are a prefix of (q+i)-grams, the compression rate is very high. Additional pruning based on the counts can be performed corresponding to the q-gram hash tables. Thus, the data structure can hold information for the selectivity estimation for q-grams. The structure is not supposed to help estimating the selectivity information for arbitrary substrings like the count-suffix tree described above. Corresponding to pruned count-suffix trees and pruned hash tables, it is possible to construct pruned count tries. In a pruned count trie only nodes that have a count greater than a threshold $p_{mincount}$ are maintained.

5.2 Estimation of selectivity values

Count suffix-trees: decomposition into arbitrary substrings requires selectivity information for each substring in the query string. These information are stored in a full count-suffix tree or in a pruned count-suffix tree. To efficiently retrieve all selectivity information from an FST, a suffix-tree is constructed from the query string s, called ST_s. Subsequently, the tree ST_s is traversed in pre-order and each traversal step is repeated immediately in the FST. Thus, in each step the selectivity is assigned to the corresponding position in the lower triangular matrix (see Sec. 3.1). The selectivity of a substring $s[i,j]$ $0 \leq i < j \leq length(s)$ itself is computed by $sel(s[i,j]) = \frac{C_{s[i,j]}}{N}$ with $C_{s[i,j]}$ the count value associated to the node of the FST which represents the substring, and N the count value associated to the root node. Traversing the suffix-tree of query string s and the count-suffix tree in parallel reduces the search costs compared to querying each substring separately. If a substring is not found, its selectivity is assumed to be 0.

q-gram hash table: the decomposition of the query string s into q-samples requires the selectivity information of all q-grams of s. The cost for computing the selectivity information is $O(length(s))$. If a q-gram is contained in a hash table $\mathcal{H}_q$, the estimated selectivity is $sel(qgram) = \frac{C_{qgram}}{N}$ with C_{gram} the associated count value and N the number of all q-grams in the database. Using a pruned hash table some q-grams might not appear in the table. In this case, the estimated selectivity is simply the pruning selectivity $sel_p = \frac{p-1}{N}$ with p the minimum count. This kind of strategy works well for low pruning limits. Token tables are used in a straightforward way. To each token t the token count C_t in the complete database is stored. The selectivity of the token is defined as $sel(t) = \frac{C_t}{N}$ with N sum of all token counts in the database. In a pruned token table not found tokens are estimated with $sel(t) = \frac{p-1}{N}$ with p the pruning limit.

Count trie: a count tries can be used to estimate selectivity information for q-grams of a length q with $p_{min} \leq q \leq p_{max}$. For each q-gram $s[i, i+q]$ with $0 \leq i < length(s) - q$ of a query string s the count trie CT has to traversed from the root node to a node on level q to find the associated count information $C_{s[i,i+q]}$. Thus, the costs of obtaining selectivity information are $O(length(s) * q)$. The selectivity is computed by $sel(s[i, i+q]) = \frac{C_{s[i,i+q]}}{N}$ with N the number of q-grams assigned to root of the count trie.

5.3 Building and maintaining selectivity information

In this paper we assume uncooperative Web sources, i.e. sources provide only limited query capabilities, e.g. via a Web interface or a restricted Web service. However, either a substring or a keyword search has to be supported.

First, an initial description of the substring distribution is needed. If this can not be retrieved from a cooperative source or some kind of vocabulary, a possible approach uses *query-based sampling* as described in [1]. There, query-based sampling is used for the construction of source descriptions of text databases. The obtained descriptions allow the selection of relevant sources in distributed text retrieval. The general approach can be described by the following steps:

1. Select an initial query string
2. Send the query string to the source and execute it
3. Retrieve top (first or sample) k results (tuples)
   (a) extract substrings according the selected type
   (b) update the statistic information of the summary structure
4. while no exit condition is reached, randomly select a new query string from the learned substrings and continue with step 2.

A query string is either an arbitrary substring, a q-gram, or a token. The corresponding query is a substring query for the former two cases and a keyword query for the latter case. As substring queries do not rank the results, there are two approaches in step 3: select the first k or sample k results. Selecting only the first k returned tuples allows a fast computation for the sample tuples. If the results are ordered, the extracted frequency information and substrings are too biased. This problem can be solved by using the sample k approach. The sample can be drawn from the arriving results by applying a reservoir sampling algorithm. The randomly selected tuples are used to build the summary structure. The disadvantage is the retrieval of the complete query results necessary to sample the data.

The ideas behind query-based sampling are the following. At first, the selectivity information can be seen as an ordered "stop word list", i.e. we want to avoid substrings with a bad selectivity. But, substrings occuring with a high frequency are extremely well approximated with query based sampling, as shown in the evaluation in Section 6. This way we can avoid big result sets even with a relatively small ratio of sampled tuples.

6 Evaluation

For evaluation purposes we used a real-life data set containing detailed information about cultural assets lost or stolen from private persons and museums in Europe during and after the Nazi regime. Because the gathered data is often imprecise or erroneous, similarity based operations are important in this application scenario and are already part of the application. This current research targets the integration with similar databases available over the Web. The experiments dealt with a collection of approximately 60000 titles of cultural assets. The data set contains a great number of duplicates with identical and similar values, e.g. 14% of the tuples have identical duplicates, 2.2% of the tuples have duplicates with an edit distance of 1, and 1.8% of the tuples have duplicates with an edit distance of 2. To evaluate the key criteria described in the following, this data set as well as necessary index structures were materialized in one local Oracle 9i database and queries were mapped to SQL substring queries for pre-selection. The required mapping and further evaluation was implemented in a mediator on top of the database system using Java. The considered queries were similarity self-joins on this one table. The key criteria considered during evaluation are the selectivity of generated pre-selections, the quality of our selectivity estimation, and the applicability to actual data values. Figure 5 shows the average selectivity we achieved with the proposed q-samples approach for a varying maximum edit distance k and varying q. The size of the candidate sets retrieved from the database were reasonable, especially for $q = 4$ and $q = 5$, 100 to 300 of the approximate 60000 original titles.

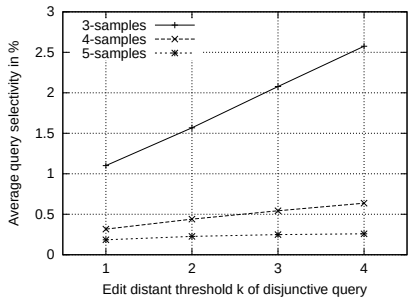
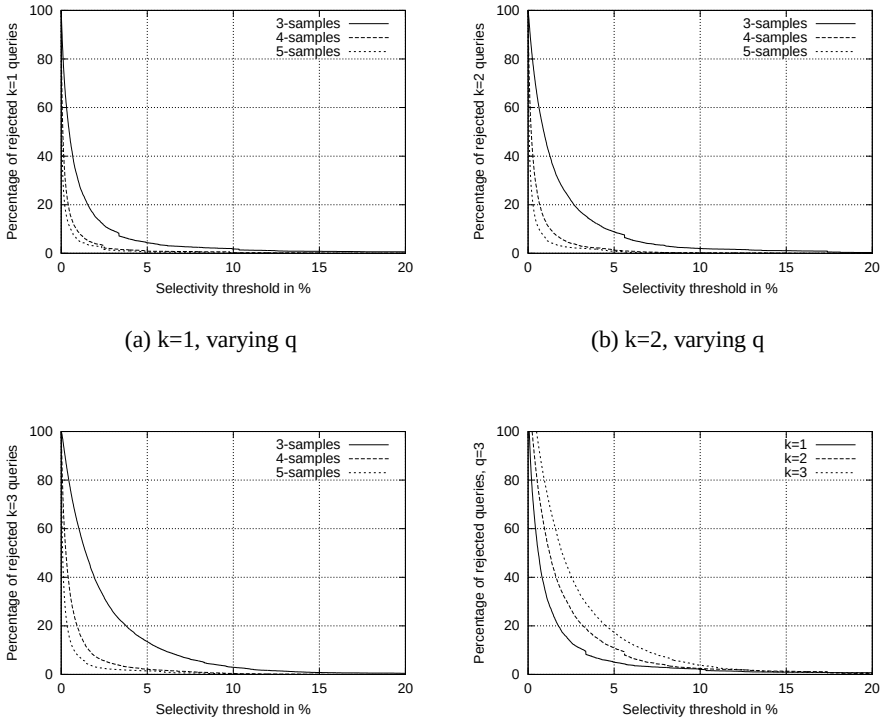


Fig. 5. Average selectivity for varying q and k



(a) k=1, varying q

(b) k=2, varying q

(c) k=3, varying q

(d) varying k, q=3

Fig. 6. Cumulative selectivity distribution for varying q and k

To answer the question, how many queries provide a good selectivity, beneath a given threshold, which also can be used to reject a query if the intermediate result would exceed a reasonable limit, we investigated the selectivity distribution of queries created

from every tuple in the data set. The results are shown in Figure 6 for varying q and k. For example, in Figure 6(c) where $k = 3$, if we set the selectivity threshold to 5%, we have to reject approximately 3% of the queries using 4-samples and 5-samples and approximately 14% of queries using 3-samples. Though the former observation may seem quite bad, actually the edit distance threshold of $k = 3$ is not realistic for most applications, where real duplicates often have a distance of 1 or 2. The effect improves for smaller k as shown in Figure 6(d), where for the the same selectivity threshold we see that for $k = 2$ we only have to reject 10% and for $k = 1$ only 5% of our queries. Again, for longer substrings with $q = 4$ and $q = 5$ the queries perform far better, as seen in Figures 6(a) and 6(b).

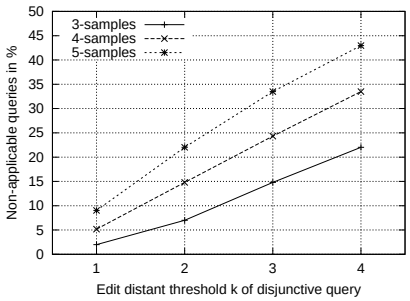
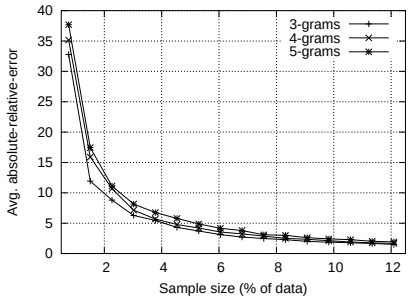


Fig. 7. Applicability for varying q and k

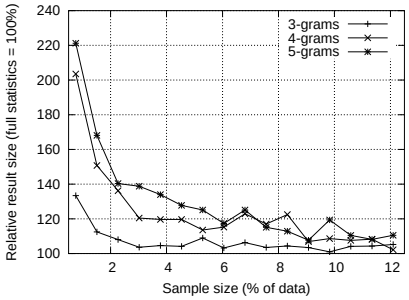
A problem with the presented approach occurs, if the number of required q-samples can not be retrieved from a given search string, because the latter is not long enough for the decomposition. Figure 7 shows how often this was the case with our data set and for varying q and k, i.e. the query strings s did not fulfill the condition $length(s) \leq q * (k + 1)$. Though, in this case we can still step back instead of reject and send a query containing less than $k + 1$ q-samples providing at least a subset of the result as mentioned before. Nevertheless, while greater q benefit the selectivity they hinder the applicability when many short query strings exist. Therefore, the parameters q, k, and a possible selectivity threshold have to be chosen carefully based on characteristics of a given application.

In Sec. 5.3 we described the usage of query-based sampling in order to build selectivity summaries for uncooperative sources. Now we have to evaluate the quality of these information and the impact on the pre-selection predicates. First, the differences between full scan estimation and estimation based on a certain sample size are investigated. From all possible q-grams 2000 were selected into a query set $\mathcal{Q}$. Subsequently, we computed the average of the absolute-relative-errors defined as

$$e = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} \frac{\text{abs}(\text{sel}(q) - \text{est}(q))}{\text{sel}(q)},$$



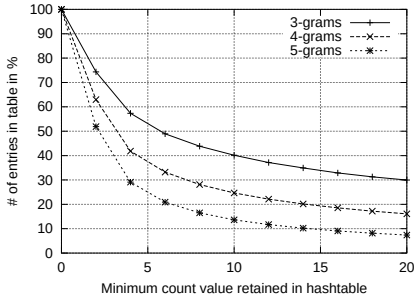
(a) Sample size estimation errors



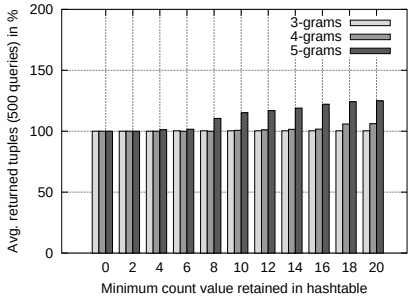
(b) Quality of the pre-selection decisions

Fig. 8. Sample size vs. Quality

with $\text{sel}(q)$ the selectivity of q based on full statistics, i.e. the real selectivity, and $\text{est}(q)$ the estimated selectivity based on a sample of specific size. The results are illustrated in Fig. 8(a). As expected, the error is decreasing with bigger sample sizes. However, the error is quite significant with a factor of 5 for 5% sample size. But that only means, that for small samples we give rather conservative estimations. And, the most important thing is, the relative order of q-grams is retained. Furthermore, high ranked q-grams, i.e. those, which have to be avoided, are approximated well in the sample.



(a) q-gram table sizes vs. pruning limits



(b) Quality of pre-selection vs. pruning limits

Fig. 9. Results for pruned q-gram tables

Following the evaluation of the estimation errors the influence of the errors to the pre-selection results have to be shown. Therefore, we generated a sample set of queries $\mathcal{Q}_2$ which contains 500 strings randomly selected from the database. We measured the

average number of tuples returned by the pre-selection condition for an edit distance $k = 2$, i.e. a disjunction of three q-sample substring queries. Here, we assumed the average result size of substring queries created with full statistics as 100%. Fig. 8(b) shows the result sizes of pre-selection queries created using selectivity information from different sample sizes. Even the precision of query based sampling selectivity estimation is not very high, the query results are close to full statistics. That has several reasons. The selectivity estimation of high ranked q-grams is rather high and ranking similarity is high. Thus, even if the selectivity estimation is not perfect, the relative order of the q-grams is good.

Finally, we evaluated the influence of the pruning limit on sizes of the storage structures as well as on the quality of pre-selection. The results are illustrated in Figures 9(a) and 9(b) respectively. Especially for 4- and 5-grams pruning reduces the storage costs decisively, e.g. with a pruning limit $p_{min} = 15$ the size of the 5-gram table reduces to 10% of the original size. Nevertheless, the quality of the estimations and result set sizes based on the estimations are very good as seen in Figure 9(b).

7 Conclusions and Outlook

In this paper we presented an approach for querying based on string similarity in a virtually integrated and heterogeneous environment. String similarity is expressed in terms of the edit distance, and global queries are re-written to a source query using standard interfaces to efficiently select a candidate set and a global part of the query that actually computes the correct result within a mediator. To grant efficiency, queries are re-written to disjunctive source queries based on selectivity information for q-samples, arbitrary substrings, or tokens.

As the approach in general is quite new, there is of course a great number of open questions, which require further research or could not be discussed here in detail due to space limitations. The currently achieved results of fetching only a small fraction of a percent of the original data in most scenarios may be suitable for many applications, but for large data volumes this already may be prohibitive. On the other hand, while a complete decomposition of a search string to substrings is optimal regarding the selectivity, the necessary overhead seems impractical in most applications. We pointed toward mixed approaches and are currently researching further ways for selectivity estimation.

Using the string edit distance for similarity operations does not fully reflect real-world requirements, were similarity is most often specific to attribute semantics of the given application, e.g. the similarity of presentations of a persons name can be judged much better using a specific similarity measure. Nevertheless, the general principle of pre-selection by query re-writing remains applicable, as well as many aspects of mapping a given value based on selectivity. A framework for query processing should provide generalized operations clearly separated from application-specific aspects.

References

1. J. Callan and M. Connell. Query-based sampling of text databases. *ACM Trans. Inf. Syst.*, 19(2):97–130, 2001.

2. W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In L. M. Haas and A. Tiwary, editors, *Proceedings ACM SIGMOD, 1998, Seattle, Washington, USA*, pages 201–212. ACM Press, 1998.
3. C. Faloutsos and K.-I. Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *ACM SIGMOD 1995*, pages 163–174, 1995.
4. N. Fuhr. Probabilistic datalog – A logic for powerful retrieval methods. In *Proceedings of the Eighteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Retrieval Logic, pages 282–290, 1995.
5. 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 2001*, pages 491–500, 2001.
6. L. Gravano, P. G. Ipeirotis, N. Koudas, and D. Srivastava. Text joins in an RDBMS for web data integration. In *WWW 2003*, pages 90–101, 2003.
7. Y.E. Ioannidis and V. Poosala. Balancing histogram optimality and practicality for query result size estimation. In M.J. Carey and D.A. Schneider, editors, *ACM SIGMOD 1995*, pages 233–244, 1995.
8. P.G. Ipeirotis and L. Gravano. Distributed search over the hidden web: Hierarchical database sampling and selection. In *VLDB 2002*, pages 394–405, 2002.
9. H.V. Jagadish, O. Kapitskaia, R.T. Ng, and D. Srivastava. One-dimensional and multi-dimensional substring selectivity estimation. *The VLDB Journal*, 9(3):214 – 230, dec 2000.
10. L. Jin, C. Li, and S. Mehrotra. Efficient record linkage in large data sets. In *Eighth International Conference on Database Systems for Advanced Applications (DASFAA '03), March 26-28, 2003, Kyoto, Japan*, 2003.
11. R.M Karp and M.O. Rabin. Efficient randomized pattern-matching algorithms. *IBM Journal of Research Developments*, 31(2), mar 1987.
12. N. Koudas and K. C. Sevcik. High dimensional similarity joins: Algorithms and performance evaluation. *IEEETKDE: IEEE Transactions on Knowledge and Data Engineering*, 12, 2000.
13. P. Krishnan, J.S. Vitter, and B.R. Iyer. Estimating Alphanumeric Selectivity in the Presence of Wildcards. In H.V. Jagadish and I.S. Mumick, editors, *ACM SIGMOD 1996*, pages 282–293, 1996.
14. G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.
15. G. Navarro and R. Baeza-Yates. A practical q-gram index for text retrieval allowing errors. *CLEI Electonic Journal*, 1(2), 1998.
16. G. Navarro, R.A. Baeza-Yates, E. Sutinen, and J. Tarhio. Indexing Methods for Approximate String Matching. *IEEE Data Engineering Bulletin*, 24(4):19 – 27, dec 2001.
17. M.T. Roth and P.M. Schwarz. Don't scrap it, wrap it! a wrapper architecture for legacy data sources. In M. Jarke, M.J. Carey, K.R. Dittrich, F.H. Lochovsky, P. Loucopoulos, and M.A. Jeusfeld, editors, *VLDB 1997*, pages 266–275, 1997.
18. S. Santini and R. Jain. Similarity measures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(9):871–883, 1999.
19. E. Schallehn, K. Sattler, and G. Saake. Efficient Similarity-based Operations for Data Integration. *Data and Knowledge Engineering Journal*, 48(3):361–387, 2004.
20. H. Shang and T. H. Merrett. Tries for approximate string matching. *IEEE Transactions on Knowledge and Data Engineering*, 8(4):540–547, 1996.
21. V. Vassalos and Y. Papakonstantinou. Describing and using query capabilities of heterogeneous sources. In M. Jarke, M.J. Carey, K.R. Dittrich, F.H. Lochovsky, P. Loucopoulos, and M.A. Jeusfeld, editors, *VLDB 1997*, pages 256–265, 1997.