A Syntactic Approach for Searching Similarities within Sentences

Federica Mandreoli
DII - Univ. di Modena e Reggio Emilia
via Vignolese, 905
Modena - Italy
fmandreoli@unimo.it

Riccardo Martoglia
DII - Univ. di Modena e Reggio Emilia
via Vignolese, 905
Modena - Italy
rmartoglia@unimo.it

Paolo Tiberio
DII - Univ. di Modena e Reggio Emilia
via Vignolese, 905
Modena - Italy
ptiberio@unimo.it

ABSTRACT

Textual data is the main electronic form of knowledge representation. Sentences, meant as logic units of meaningful word sequences, can be considered its backbone. In this paper, we propose a solution based on a purely syntactic approach for searching similarities within sentences, named *approximate sub^{2}sequence matching*. This process being very time consuming, efficiency in retrieving the most similar parts available in large repositories of textual data is ensured by making use of new filtering techniques. As far as the design of the system is concerned, we chose a solution that allows us to deploy approximate sub^{2}sequence matching without changing the underlying database.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Information filtering, search process*; I.5.4 [**Patter Recognition**]: Applications—*Text processing*

General Terms

Algorithm, design, performance

Keywords

Sentence similarity measure, sequence matching

1. INTRODUCTION

Textual data is the main electronic form of knowledge representation. With the advent of databases, storing large amounts of textual data has become an effortless and widespread task. On the other hand, exploiting the full potentiality of unstructured repositories and thus understanding the utility of the information they contain is a much more complex task, strictly connected to the application they serve. Sentences, meant as logic units of meaningful word sequences, can be considered the backbone of textual data. Searching in sentence repositories often requires to go beyond exact matching to determine the sentences which are similar or close to a given query sentence. The similarity involved in this process can be based just on the syntax of the sentence, disregarding its semantical content while identifying similar word sequences. Many applications may benefit from such a facility, from EBMT (Example-Based Machine Translation) systems to syntactical document similarity search and the correlation of independent sentence repositories. We argue that the kind of similarity matching useful for most of the applications we consider should go beyond the search for whole sentences. The similarity matching we refer to attempts to match *any* parts of data sentences against *any* query parts. Although complex, this kind of search enables the detection of similarities that could otherwise be unidentified.

In this paper, we propose a solution based on a purely syntactic approach for searching similarities within sentences, named *approximate sub^{2}sequence matching*, and fitting into the DBMS context, which represents the most common choice adopted by the above cited applications for managing their large amount of textual data. The underlying similarity measure is exploitable for any language since it is based on the similarity between sequences of terms such that the parts most close to a given one are those which maintain most of the original form and contents. Applying an approximate sub^{2}sequence matching algorithm to a given query sentence and a collection of data sentences is extremely time consuming. Efficiency in retrieving the most similar parts available in the sentence repository is ensured by exploiting filtering techniques. We introduce two new filters for the approximate sub^{2}sequence matching which quickly discard sequences that cannot match, efficiently ensuring no false dismissals and few false positives.

2. APPROXIMATE SENTENCE MATCHING

The problem of searching similarities between sentences is addressed by introducing a syntactic approach which analyzes the sentence contents in order to find similar parts. We consider a sentence as a sequence of terms and we characterize the problem of approximate matching between sentences as a problem of searching for similar sequences corresponding to the whole sentences or parts of them.

In this context, we adopt the *edit distance* [6] as similarity measure between (parts of) sentences. More pre-

CIKM'02, November 4–9, 2002, McLean, Virginia, USA.

cisely, given two sentences as sequences of terms S_1 and S_2, $ed(S_1[i_1 \dots j_1], S_2[i_2 \dots j_2])$ denotes the edit distance between the two parts $S_1[i_1 \dots j_1]$ and $S_2[i_2 \dots j_2]$.

The operation of approximate matching that we introduce in the following definition extends the notion of subsequence/whole matching [3] in order to locate (parts of) sentences that match (parts of) query sentences.

Definition 1. Given a collection of query sentences Q and a collection of data sentences D not necessarily distinct, a distance threshold d and a minimum length $minL$, find all pairs of sequences $(S_1[i_1 \dots j_1], S_2[i_2 \dots j_2])$ such that $S_1 \in Q$, $S_2 \in D$, $(j_1 - i_1 + 1) \geq minL$, $(j_2 - i_2 + 1) \geq minL$ and $ed(S_1[i_1 \dots j_1], S_2[i_2 \dots j_2]) \leq d$.

Applying an approximate sub^{2}sequence matching algorithm to a given query sentence and a collection of data sentences is extremely time consuming. The main challenge is thus to find filtering techniques suitable for the problem introduced in Def. 1. Such filtering techniques should operate on whole sentence pairs and efficiently hypothesize a small set of them as matching candidates. As to sentence content, only that of the candidate answers will be further analyzed. To this purpose, we introduce two new filters for the approximate sub^{2}sequence matching, namely sub^{2}count filter and sub^{2}position filter, which quickly discard sequences that cannot match, efficiently ensuring no false dismissals and few false positives. *sub^{2}Count filtering* relies on a minimum number of common short subsequences of length q (known as q-grams) which two matching sentences are required to share. *sub^{2}Position filtering* takes into account the relative positions of individual q-gram matches requiring them to be relatively close.

The approximate sub^{2}sequence matching problem can be easily expressed in any database system supporting user-defined functions (UDFs), such as Oracle and DB2. The immediate practical benefit of our techniques is that approximate search can be widely and efficiently deployed without changes to the underlying database. Let `D` be a table containing the data sentences and `Q` an auxiliary table storing the query sentences, which is created on-the-fly. Both tables share the same schema `(COD, SENT)`, where `COD` is the key attribute and `SENT` the sentence. In order to enable approximate sub^{2}sequence matching processing through filtering techniques based on q-grams, the database must be augmented with the data about q-grams corresponding to the data and query sentences, maintained in `D` and `Q` respectively, and stored in two auxiliary tables `Qq` and `Dq` with the same schema `(COD,POS,Qgram)`. For each sentence S, its positional q-grams are represented as separate tuples in the above tables, where `POS` identifies the position of the q-gram `Qgram`. The positional q-grams of S share the same value for the attribute `COD`, which serves as the foreign key attribute to the table storing S.

The SQL expression exploiting filtering techniques for approximate sub^{2}sequence matches has the form pictured in Figure 1. It shows that filters can be expressed as an SQL expression and efficiently implemented by a commercial relational query engine. The involved SQL expression joins the auxiliary tables for q-gram sentences, `Dq` and `Qq`, with the query table `Q` and the data table `D` to retrieve the sentence pairs to be further analysed for approximate sub^{2}sequence matches. The sub^{2}Position filtering algorithm is implemented by means of an UDF function `sub2Position(S1,S2,minL,d)`.

```
SELECT    S1.COD AS COD1, S2.COD AS COD2
FROM      Q S1, Qq S1q, D S2, Dq S2q
WHERE     S1.COD = S1q.COD
AND       S2.COD = S2q.COD
AND       S1q.Qgram = S2q.Qgram
          -- position filtering
AND       sub2Position(S1.SENT, S2.SENT, minL, d)
          -- count filtering
GROUP BY  S1.COD, S2.COD
HAVING    COUNT(*) >= minL + 1 - (d + 1)*q
```

Figure 1: Query for approximate sub^{2}sequence match filtering

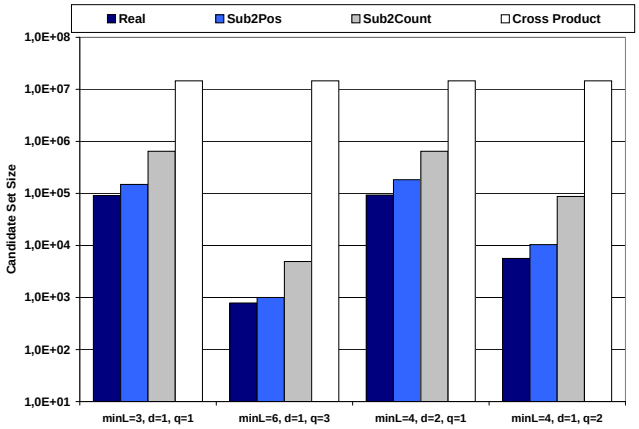

Figure 2: Filtering tests

The sub^{2}Count filtering is implemented by comparing the number of q-gram matches with the length of the involved sentences.

3. EXPERIMENTAL EVALUATION

Performances were tested in relation to the efficiency of the proposed search techniques. The main objective of filtering techniques is to reduce the number of candidate answer pairs. Obviously, the more filters are effective the more the size of candidate answers gets near to the size of the answer set.

In order to examine how effective each filter and each combination of filters is we ran different queries, enabling different filters each time, and measured the size of the candidate set with respect to the cross product of the query sentences and data sentences. We conducted experiments on a collection of 34550 reference sentences against 421 query sentences. Performance trends were observed under the parameters that are associated with our problem, that is the minimum length $minL$, and the number of allowed errors d. The most meaningful experiments are shown in Fig. 2. Obviously, the sub^{2}Position filter always filters better than the sub^{2}Count filter since, besides counting the number of equal words, it also considers their positions. For this reason, we did not take into account possible combinations of filters. In particular the sub^{2}Count filter gave a candidate answer that was between 0.003% to 11% of the cross-product size and sub^{2}Position filter filters from five to tens of times better than the sub^{2}Count filter. In any case, the sub^{2}Count filter works better with q values greater than 1 and preferably

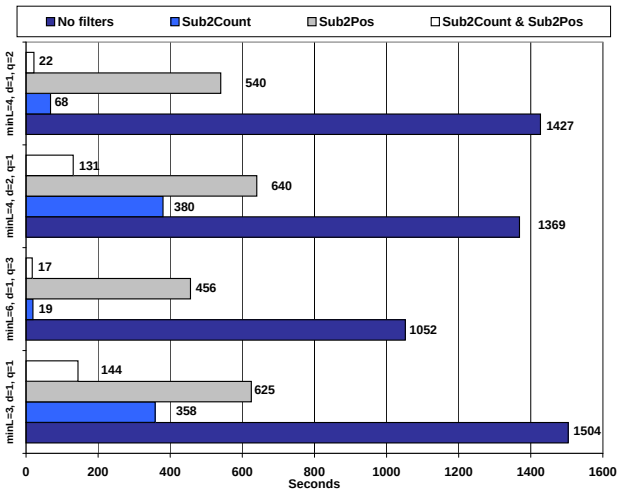

Figure 3: Running time tests

smaller than 4. The comparison of the two alternatives having $minL = 4$ shows it: setting $q = 2$ doubles the filtering performance. Even more evident is the case of $q = 3$ where the reduction of all the data sets is more than 99.8% .

Another key aspect of filtering techniques is their efficiency. Indeed, for a filter to be useful its response time should not be greater than the processing time of just the match algorithm on the whole cross product. In order to examine how efficient each combination of filters and matching algorithm is, we ran different queries, enabling different filters each time, and measured the response time. Figure 3 presents the response times of the experiments detailed in the effectiveness of filters paragraph. In particular, it shows the times required to get the answer sentence pairs for each possible combination of filters (denoted as Sub2Count and Sub2Pos). The assessment and evaluation of the obtained values focus on determining the best choice for filters with respect to the parameter values. Enabling filtering techniques reduces the response time of at least 7 times in the worst case. In particular, any combination that includes the sub^{2}Count filter always improves the prototype performances. Indeed, even if it filters less than the sub^{2}Position filter (see Fig. 2), it plays an important role by pruning out a large portion of sentence pairs and thus leaving a small set of them on which the sub^{2}Position filter or directly the matching algorithm is applied. Moreover, its execution requires a small amount of time since it relies on the facilities offered by the DBMS.

4. RELATED WORK

A large work body has been devoted to the problem of matching sequences with respect to a similarity measure. Starting from the works of Faloutsos et al. [3] addressing the problem of whole and subsequence matching for sequences of the same length, the problem has been considered in different fields such as text and information processing, genetics (e.g. [1]), and time series (e.g. [5]). In particular, the paper [1] presents a fast algorithm for all-against-all genetic sequence matching. They adopt a suffix tree as indexing method in order to efficiently compare all possible sequences.

As far as text and information processing is concerned, the work [6] is an excellent survey on the current techniques to cope with the problem of string matching allowing errors. The problem has been addressed by proposing solutions based on specific algorithms (e.g. [7]), indexes (e.g. [2]), and filters (e.g. [4]). Such solutions are limited to the problem of string matching and substring matching, where in the latter case the main objective is to verify if a pattern string is contained in a given text without necessarily locating the exact position of the occurrences. As to indexes, customized secondary storage indexes or indexing techniques for arbitrary metric spaces have to be supported by the DBMS in order to be useful for techniques accessing large amounts of data stored in databases, such as the approach we propose. Amongst the others, we found the work [4] of particular interest. It presents some filtering techniques relying on a DBMS for approximate string joins and it offered the starting ideas for our work.

5. CONCLUSIONS

Searching similarities between sentences is essential for many applications. The main contributions of this paper are the proposal of a similarity measure between sequences of terms exploitable for any language and the definition of the sub^{2}sequence approximate matching which searches for matches within sentences. Efficiency and portability are ensured by the introduction of ad-hoc filtering techniques and a mapping into plain SQL expressions, respectively. We showed that the performance of the matching processing for the reference applications can widely benefit from the adoption of our techniques.

6. REFERENCES

[1] R. Baeza-Yates and G. Gonnet. A Fast Algorithm on Average for All-Against-All Sequence Matching. In *Proc. of the Int'l Workshop and Symposium on String Processing and Information Retrieval (SPIRE)*, pages 16–23, 1999.

[2] A. Cobbs. Fast Approximate Matching Using Suffix Trees. In *Proc. of the 6th Int'l Symposium on Combinatorial Pattern Matching (CPM)*, pages 41–54, 1995.

[3] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast Subsequence Matching in Time-Series Databases. In *Proc. of the 1994 ACM SIGMOD Int'l Conf. on Management of Data*, pages 419–429, 1994.

[4] L. Gravano, P. Ipeirotis, H. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate String Joins in a Database (Almost) for Free. In *Proc. of 27th Int'l Conf. on Very Large DataBases (VLDB)*, 2001.

[5] T. Kahveci and A. Singh. Variable Length Queries for Time Series Data. In *Proc. of the 17th Int'l Conf. on Data Engineering (ICDE)*, pages 273–282, 2001.

[6] G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.

[7] G. Navarro and R. Baeza-Yatesa. New and faster filters for multiple approximate string matching. *Random Structures and Algorithms*, 20(1):23–49, 2002.

[8] C. Rick. A New Flexible Algorithm for the Longest Common Subsequence Problem. Technical report, University of Bonn, Computer Science Department IV, 1994.