Query Segmentation Using Conditional Random Fields

Xiaohui Yu and Huxia Shi
York University
Toronto, ON, Canada, M3J 1P3
xhyu@yorku.ca,huxiashi@cse.yorku.ca

ABSTRACT

A growing mount of available text data are being stored in relational databases, giving rise to an increasing need for the RDBMSs to support effective text retrieval. In this paper, we address the problem of keyword query segmentation, i.e., how to group nearby keywords in a query into *segments*. This operation can greatly benefit both the quality and the efficiency of the subsequent search operations. Compared with previous work, the proposed approach is based on Conditional Random Fields (CRF), and provides a principled statistical model that can be learned from query logs and easily adapt to user preferences. Extensive experiments on two real datasets confirm the effectiveness the efficiency of the proposed approach.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Information Storage and Retrieval—*Information Search and Retrieval*; H.2.4 [**Information Systems**]: Database Management—*Systems*

General Terms

Algorithms, Experimentation

1. INTRODUCTION

The ever increasing need to provide coherent support of text retrieval in various settings, such as health care and customer relationship management, has spurred a great deal of research activities in both databases and information retrieval communities. In this paper, we are concerned with providing effective search of text information in relational databases. Even though most commercial RDBMSs provide integrated full-text search capabilities, they require the users to be familiar with the database schema and know how to use a query language like SQL. In contrast, keyword search, rooted in IR and popularized by the Web, provides a simple yet effective way for users to query and explore the underlying documents. Performing keyword search over text data stored in relational databases, involves more than straightforward applications of keyword search technologies developed in IR, and presents new challenges that have to be addressed. One such challenge is that in relational databases, the expected query result is no longer a list of relevant documents, and correspondingly, the search space is no longer a collection of documents (as in IR). Recent studies [12] have revealed that the search spaces in this setting are generally much larger than those in traditional IR. The reason for the increase in search space is due to the need of assembling keyword-matching tuples from different tables into a complete view (such as a join network of tuples). For example, the keywords in the query "*Green Mile Tom Hanks*" may each match a number of tuples in the database, and those matches have to be assembled together (e.g., through foreign key/primary key joins) to produce a result. In general, the search space is exponential in the number of keywords in the query.

To address the problems identified above, Pu and Yu [12] have introduced a pre-processing stage to keyword search called *query segmentation*, which aims to group nearby keywords in the query into *segments*. For example, the query "*Green Mile Tom Hanks*" can be segmented into "*Green Mile*" and "*Tom Hanks*". Segments, instead of individual keywords, will then be used as units for finding matches in the databases. By doing so, one can greatly reduce the search space involved. In many cases, there exist more than one way of segmenting a query. Therefore, in [12], dynamic programming algorithms are proposed to solve the optimal segmentation problem.

Note that, however, the approach proposed in [12] aims at optimizing for a heuristically defined scoring function, and thus lacks a solid theoretical foundation. Another drawback of that approach is that it does not consider the usage pattern, and therefore cannot easily adapt to user preferences.

In this paper, we present a principled approach to query segmentation based on the statistical model of conditional random fields (CRF), a descriptive model that has been successfully applied to a variety of sequence processing tasks[6, 11]. We first propose a two-stage model where we first label the keywords and then group them into segments. We then present an integrated model that combines those two steps, in order to further improve the quality of segmentation. Extensive experiments are conducted to compare the effectiveness of the proposed models with that proposed in [12], as well as a naive greedy approach. Results show that the integrated model beats the competitors especially in "hard" cases where ambiguity in queries is high. We also

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
KEYS'09, June 28, 2009, Providence, Rhode Island, USA.
Copyright 2009 ACM 978-1-60558-570-3/09/06 ...$5.00.

observe from the experiments that the proposed models are much more efficient than that proposed in [12], showing an improvement by at least two orders of magnitude.

The rest of the paper is organized as follows. Section 2 introduces related work. Section 3 defines the problem of query segmentation. Section 4 discusses two non-statistical approaches that are potential competitors to our approach. Section 5 presents the CRF-based models, and Section 6 reports on the experimental results. Section 7 concludes this paper and points out possible directions for future work.

2. RELATED WORK

Research on keyword search over structured data has attracted a great deal of attention in recent years. Early work [1, 4, 5] in this area focuses on finding the best *join networks* (or *joined tuple trees*) to connect matching tuples from different tables according to some predefined scoring functions. It has been shown that the problem of finding the optimal join network is NP-complete w.r.t. the number of relevant tables [5, 2]. Therefore, some heuristic approaches for generating the top-k candidate join networks have been proposed [2, 4] to address the efficiency issue. There has also been some work on improving the effectiveness of keyword search through the use of more meaningful ranking criteria [7, 9]. Beyond traditional relational databases, Markovwetz et. al. [10] considers the problem of performing keyword search over relational data streams, Wu et al. [14] addresses the issue of keyword search over relational data with star-schema commonly found in OLAP applications, where the focus is on searching for relevant data subspaces instead of joined networks. Keyword search over XML data has also been extensively studied [3, 8].

The query segmentation has been shown as an important problem complementary to the above database keyword search algorithms [12]. Pu and Yu propose to optimize the segmentation according to a scoring function that consists several components such as IR matching quality and segment lengths. Note that, however, the scoring function proposed in [12] is heuristically defined and thus lacks theoretical foundations. In contrast, the solutions we propose in this paper provide a principled probabilistic model based on CRF[6, 11] that can be learned from past search history and adapted to user feedback.

Tan and Peng[13] have considered the query segmentation problem in the Web context. However, due to the distinction between structured databases and unstructured text data, techniques proposed therein cannot be easily applied to query segmentation in relational databases.

3. PROBLEM DEFINITION

We first formally define the problem of query segmentation. We define *tokens* as strings that are considered individual units, and *terms* as sequences of tokens. For a given database D, a *database token* is defined as a token that appears in D at least once, and the set of all tokens in D is denoted by $\textsc{Token}^D$. Similarly, a *database term* is defined as a term that appears in D at least once, and the set of all terms in D is denoted by $\textsc{Term}^D$. A *query* of length n is defined as an ordered keyword sequence $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$. A *segment* $S = \langle x_i, \ldots, x_j \rangle$ is a subsequence of keywords in the query. A segment S is *valid* if this subsequence appears at least once in the database, i.e., $S \in \textsc{Term}^D$. A *segmentation* $\mathbb{S} = \langle S_1, \ldots, S_n \rangle$ is then defined as a sequence of non-overlapping segments that completely cover all keywords in query $\mathbf{x}$. A segmentation is valid if and only if all $S \in \mathbb{S}$ are valid.

A query may have many possible valid segmentations. For example, query "Star Wars Clone" can have three valid segmentations:

$\mathbb{S}_1 = \langle \ \langle Star \rangle \ \langle Wars \rangle \ \langle Clone \rangle \ \rangle$
$\mathbb{S}_2 = \langle \ \langle Star \ \ Wars \rangle \ \langle Clone \rangle \ \rangle$
$\mathbb{S}_3 = \langle \ \langle Star \ \ Wars \ \ Clone \rangle \ \rangle$

In $\mathbb{S}_1$, three keywords are interpreted as parts of three different movie titles or company names. $\mathbb{S}_2$ considers "*Star Wars*" as a movie title and "*Clone*" as a part of another movie title, which splits the query into two segments. $\mathbb{S}_3$ puts all three keywords in one segment as they match the movie title "Star Wars Clone Wars". For a given query, our goal is to find the optimal segmentation that maximizes a certain measure, which will become clear in the sequel.

The query segmentation problem can be considered as part of the more general problem of keyword query cleaning [12], which also considers the issue of spelling correction. In this paper, we focus on query segmentation, assuming that there are no spelling errors in the given queries. Spelling errors can be handled in a similar fashion as that proposed in [12] through a pre-processing step that expands the keywords into similar tokens in the database.

4. NON-STATISTICAL ALGORITHMS

Before preceding to describe the CRF-based models, we first briefly mention two non-statistical algorithms that can be potential competitors to the proposed models.

Greedy Search A naive algorithm to solve the segmentation problem is to greedily search for valid segments. That is, starting with the first keyword in a given query, we keep including the next keyword into the current segment until adding the new keyword would make the segment no longer valid. Then we start another segment, and repeat the above process until we reach the end of the query. This algorithm is summarized in Algorithm 1.

Algorithm 1 $\textsc{GreedySegments}(\mathbf{x})$: computes optimal segments using greedy search

```
1: n = number of tokens in query x.
2: optSegs = new list of segments.
3: start = 1
4: for i = 2 ... n do
5:    if segment <start, i> is not valid then
6:       optSegs.add(<start, i − 1>)
7:       start = i
8:    end if
9: end for
10: optSegs.add(<start, n>)
11: return optSegs
```

Keyword Query Cleaning: [12] addresses the segmentation problem as well as spelling correction. Each keyword in a given query is first expanded a set of m similar tokens in the database. A dynamic programming algorithm is then used to search for the segmentation that optimizes a scoring function consists of three components. It uses TF-IDF weighting to measure the quality of each keyword match in the IR sense, favors longer segments, and penalizes spelling

corrections.

5. QUERY SEGMENTATION USING CONDITIONAL RANDOM FIELDS

Our approach to solving the segmentation problem is based on CRF, which has been shown to outperform generative models such as the Hidden Markov Model (HMM) and local conditional models such as the Maximum-Entropy Markov Model (MEMM) in sequence labeling tasks [6]. In this section, we present two CRF-based models that model a conditional probability distribution over a label sequence $\mathbf{y} = \langle y_1, \ldots, y_n \rangle$ given an input sequence $\mathbf{x}$, with each x_i in $\mathbf{x}$ being assigned a label y_i from the possible set of labels $\mathcal{L}$. The choice of labels is different between the two models we present here, and will become clear in the sequel.

In both models, the relationship between y_{i-1} (the label of the previous keyword), y_i (the label of the current keyword), x_{i-1} (the previous keyword), and x_i (the current keyword) can be captured by a set of *feature functions* $\mathcal{F} = \{f_1, f_2, \ldots, f_m\}$. In first order CRF, we let the feature functions take the form of $f_j(y_{i-1}, y_i, x_(i-1), x_i, i) = \delta(y_{i-1} = L_1)\delta(y_i = L_2)\delta(x_{i-1} = w_1)\delta(x_i = w_2)$, where $L_1, L2 \in \mathcal{L}$, $\delta(c)$ is an indicator function that takes the value of 1 if c is true and 0 otherwise, and w_1, w_2 are keywords.

If we associate with each feature function f_j a weight λ_j, then our CRF model can be formulated as

$$p_{\boldsymbol{\lambda}}(\mathbf{y}|\mathbf{x}) = \frac{1}{Z_{\boldsymbol{\lambda}}(\mathbf{x})} \cdot \prod_{i=1}^{n} \exp\left(\sum_{j=1}^{m} \lambda_j f_j(y_{i-1}, y_i, x_{i-1}, x_i, i)\right), \tag{1}$$

where $Z_{\boldsymbol{\lambda}}(\mathbf{x})$ is a normalizing factor,

$$Z_{\boldsymbol{\lambda}}(\mathbf{x}) = \sum_{\mathbf{y}'} \prod_{i=1}^{n} \exp\left(\sum_{j=1}^{m} \lambda_j f_j(y'_{i-1}, y'_i, x_i, i)\right).$$

The model assigns a probability to each possible sequence given the input sequence $\mathbf{x}$ through the feature vectors.

Training the model involves maximizing the likelihood of the training set $D = \{\mathbf{x}_k, \mathbf{y}_k\}_{k=1}^{N}$ w.r.t. $\boldsymbol{\lambda}$. The log-likelihood function is convex and thus can be optimized using standard methods like gradient ascent or quasi-Newton methods. The training data can be obtained from query logs, which contain keyword queries and the correct labels for the keywords. Alternatively, as bootstrapping when no such query log is available, training data can be obtained by randomly generating queries based on the underlying data, which will be further explained in Section 6.

When deployed, the CRF model finds the best label sequence $\mathbf{y}^*$ for query $\mathbf{x}$ as $\mathbf{y}^* = \text{argmax}_{\mathbf{y}} p(\mathbf{y}|\mathbf{x})$. Since all features just depend on the previous label, an efficient algorithm based on recursive calculation (Viterbi algorithm) can be used to find the best sequence.

5.1 The basic CRF model

In the first model we propose, segmentation is done in two stages. In the first stage, the labels for the keywords are identified, and in the second stage, the given query is segmented based on those labels. In this model, the label set $\mathcal{L}$ consists of all the column names in the database. For example, $\mathcal{L}$ = { MOVIE, ACTOR, COMPANY ... } for the IMDB movie dataset. Correspondingly, a feature function can take the following form: $f_7(y_{i-1}, y_i, x_{i-1}, x_i, i) = \delta(y_{i-1} = \text{ACTOR})\delta(y_i = \text{ACTOR})\ \delta(x_{i-1} =' Tom')\delta(x_i =' Hanks')$.

Adding a boundary between two adjacent keywords with different labels, we can "cut" the query into a potential segmentation $\mathbb{S}$. In our example of "*Green Mile Tom Hanks*", the best label sequence would be ⟨ MOVIE, MOVIE, ACTOR, ACTOR ⟩. Then it will be cut to $\langle\langle Green\ Mile\rangle\ \langle Tom\ Hanks\rangle\rangle$. One might be attempted to conclude that the optimal segmentation can be obtained by this method. This, however, could be wrong, because some $S \in \mathbb{S}$ may *not* be valid segments. For example, "*Johnny Depp Orlando Bloom*", which is labeled as ⟨ ACTOR, ACTOR, ACTOR, ACTOR ⟩, will be segmented to ⟨⟨ Johnny Depp Orlando Bloom ⟩⟩ if only labels are used to generate the segmentation. Apparently, this is not a valid segmentation because no term in the database contains all those four keywords.

The next task is therefore to further breakdown each invalid segment $S \in \mathbb{S}$ to a set of valid segments. Since there can be more than one such set of segments, we would like to identify the optimal one. To this end, we propose the following MAXSCORE algorithm, which ranks the candidate segmentations according to a scoring function.

The MAXSCORE algorithm defines a scoring function SCORE(S) for each segment S. The score of one segmentation $\mathbb{S}$ is defined as

$$\text{SCORE}(\mathbb{S}) = \sum_{S \in \mathbb{S}} \text{SCORE}(S) \tag{2}$$

The optimal segmentation we want to find is the one that maximizes the total score for all segments, i.e., $\mathbb{S}^*$ = argmax{SCORE($\mathbb{S}$)}. The scoring function we choose is SCORE(S) = $e^{|S|}$, where $|S|$ denotes the length of the segment S. SCORE(S) is defined this way to give preference to longer segments.

The naive method to compute the optimal segmentation is to enumerate all possible segmentations. The method is very inefficient, requiring $O(n^2)$ accesses to the database (or the index of database terms when it is present) in order to validate all candidate segments. To improve the efficiency, we propose an algorithm that requires less database (or index) access. This algorithm centers around the notion of MAXTERM, which is defined as follows. Segment $S = \langle i, j\rangle$ is a MAXTERM if $\langle i, j\rangle$ is a valid segment, and $\langle i-1, j\rangle$ is not a valid segment, and $\langle i, j+1\rangle$ is not valid segment. The algorithm MaxTermSearch shown in Algorithm 2 finds all MAXTERMS with only $O(n)$ database accesses for a given query of length n.

After all MAXTERMS are found, a further step is taken to compute the optimal segmentation. No database or index access is required in this step. This step involves constructing a tree where each node corresponds to a candidate segmentation, and performing tree search procedure as follows.

1. Create the finest segmentation with only one token in each segment. Use it as the root of the tree.
2. Create child nodes for each current leaf node in the tree, by merging two adjacent segments in that leaf node. Only child nodes containing valid segments are added to the tree. Validation can be done by consulting the set of MAXTERMS. Repeat this step until no more nodes can be added to the tree.
3. Calculate the segmentation scores of all leaf nodes. (The internal nodes do not have to be considered, as

Algorithm 2 MaxTermSearch(**x**): finds all MaxTerms in input query

```
1: n = number of tokens in query x.
2: maxTerms = new list of segments.
3: head = 1; end = 2
4: headMove = false
5: while head <= n do
6:   if headMove is true then
7:     if segment <head, end> is valid then
8:       end++; headMove = false
9:     else
10:      if (end − head) == 1 then
11:        maxTerms.add(<end, end>)
12:        head++; end++
13:        headMove = false
14:      else
15:        head++
16:      end if
17:    end if
18:  else
19:    if segment <head, end> is valid then
20:      end++
21:    else
22:      maxTerms.add(<head, end − 1>)
23:      head++
24:      end − head == 1?end++ : headMove = true
25:    end if
26:  end if
27: end while
28: return maxTerms
```

child nodes always have higher scores than their parents due to the convexity of the scoring function for segments.)

4. The segmentation corresponding to the leaf node with the highest score is chosen as the optimal segmentation.

In summary, the basic CRF model first computes the label for each keyword in a given query, based on which a "preliminary" segmentation is done by grouping adjacent keywords with the same label into the same segment. The invalid segments resulting from the preliminary segmentation is further broken down, and the optimal segmentation for each such invalid segment is computed through the tree search procedure described above.

5.2 The enhanced CRF model

We find from the experiments that above CRF model does not always provide satisfactory segmentation performance. It is also desirable to integrate labeling and segmentation into one step for better segmentation accuracy. To this end, we enhance the CRF model by replacing the set of labels used in training and inferencing (labeling). In addition to the column name, each label in the new label set $\mathcal{L}$ now contains an extra component indicating a keyword's position in a segment. Essentially, each label is a column-position pair. One might be attempted to use an integer as the position component to label the exact position of a keyword in a segment, but that will result in an explosion in the size of the label set and thus in the number of feature functions, which makes learning very difficult. To avoid this explosion, only two position labels, "S" for the first keyword in segment and "N" for others, are used. Under this new setting, the label of our example "*Green Mile Tom Hanks*" now becomes ⟨ S-Movie, N-Movie, S-Actor, N-Actor ⟩.

A given query is segmented according to the position information in the label sequence **y**. All labels started with S are considered as segment boundaries. This model may also produce invalid segments like the first model. Therefore, a post-processing stage is carried out, in which all invalid segments are further split by the aforementioned MaxScore algorithm. By enriching the label set, the enhanced model sees a dramatically increase in segmentation accuracy in comparison with to the basic model, at the expense of a larger feature function set and higher labeling cost. The detailed experimental results are presented in the next section.

5.3 Adapting to user preferences

The proposed CRF models can naturally adapt to user preferences. For example, when a system based on the proposed models present to the user the list of the candidate segmentations for a given query, we record the query as well as the choice of segmentation made by the user in the query log. This query and its correct segmentation serve as a training instance for the CRF model, based on which the model parameters can be updated. For the same query, different users may choose different segmentations. This preference is naturally captured by the model since different user choices result in different training instances. As more and more keyword queries are performed, the accuracy of the segmentation is expected to increase. The rate of the adaptation can be controlled by a learning rate, which dictates how much a new training instance obtained from user feedback counts relative to past instances. This learning rate can be tuned to satisfy different user requirements, such as preference to recent training data, and the desired speed of model adaptation.

6. EXPERIMENTS

Extensive experiments were conducted on two real datasets, including the IMDB dataset, and the FootMart sample database shipped with SQL Server 2005. The first dataset is obtained in exactly the same way as what is done in [9] and [12]. The IMDB data contains information on movies, actors, directors, etc., with 9,839,026 tuples. The raw text files in this data set are converted to relational data as described in [9], and the text attributes are then indexed. The FoodMart data, an OLAP database, stores information on products, customers, etc., and contains 428, 049 tuples.

Our implementation was done in Java. Apache Lucene (http://lucene.apache.org) was used to index the text attributes in the datasets. For each data set, we built a term index which considers each term as a unit for indexing (a document in Lucene's terminology). Experiments were carried out on a Notebook PC running Windows XP, with a Pentium Dual Core CPU running at 1.8GHz, 1GB of memory, and 160GB of hard drive.

In order to systematically study the behavior of the proposed algorithms, and to minimize the subjectivity in the experimental evaluation, the training and test queries are generated by randomly sampling from the data. Specifically, the queries are generated as follows. For each query to be generated, d data values (cells) are sampled from the data set, and for each sampled value, only c contiguous words are

kept. The words taken from the d different data values are then concatenated to form the query. 10-fold cross validation is performed using the set of queries obtained, and the accuracy measured.

Let $\mathcal{S}_{\mathbf{x}}$ and $\mathcal{S}'_{\mathbf{x}}$ be the true and predicted sets of segments for a query $\mathbf{x}$. The prediction accuracy for $\mathbf{x}$ is defined as $A_{\mathbf{x}} = 1 - \frac{|\mathcal{S}_{\mathbf{x}} - \mathcal{S}'_{\mathbf{x}}| + |\mathcal{S}'_{\mathbf{x}} - \mathcal{S}_{\mathbf{x}}|}{|\mathcal{S}_{\mathbf{x}}|}$.

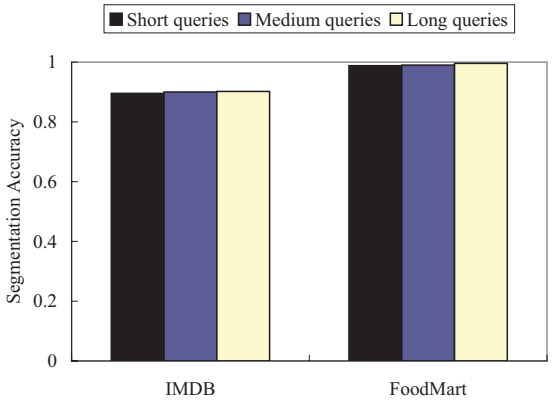

Figure 1: The segmentation accuracy of the enhanced CRF model on FoodMart and IMDB datasets

Figure 1 illustrates the segmentation accuracy of the enhanced CRF model on the two data sets. The short queries are generated using parameters $d = 2$ and $c = 3$. Each query normally has 6 keywords. The medium queries, generated with parameters $d = 5$ and $c = 3$ have 15 or a bit less keywords in each of them. The parameters $d = 10$ and $c = 3$ are used to generate long queries. The segmentation accuracy for the three different types of queries are very similar, which indicates that the CRF model is not sensitive to query length. The result on FoodMart is slightly higher than that on IMDB. In all cases, the accuracy is higher than or close to 90%. In what follows, we will show the experimental results on the IMDB dataset only.

One common cause for segmentation error is that there may exist ambiguity in deciding which segment a keyword should belong when that keyword can form valid segments with both the preceding and the following keywords. We call this situation an ambiguous connection between two segments. Experiments are conducted to evaluate how the different algorithms perform under different levels of ambiguity present in the queries,. The number of ambiguous connections in one query is used to measure the ambiguity level of this query.

The segmentation accuracy of the four different algorithms on medium queries of different levels of ambiguity is compared in Figure 2. As shown in Figure 2, the enhanced CRF model enjoys higher accuracy than the other methods except for the case when the ambiguity level is zero. Compared with other methods, the enhanced CRF also shows a slower decline in accuracy when the ambiguity increases, demonstrating the robustness of the CRF model against ambiguities in segmentation.

How CRF model adapts to user preferences is illustrated in Figure 3. For the experiments, the user preference is added by sampling a subset of the database terms with a higher probability and sampling other terms with a rela-

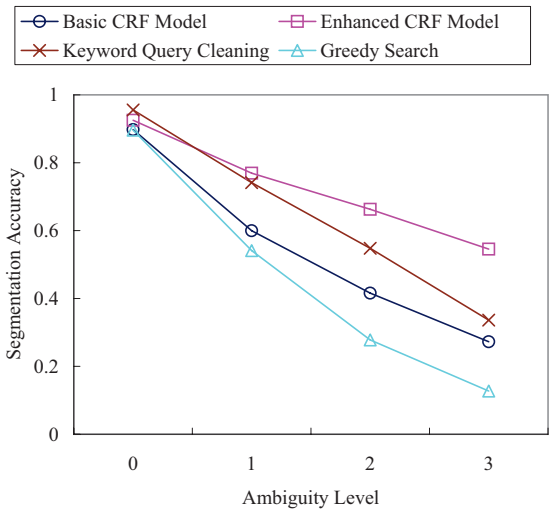

Figure 2: The segmentation accuracy at different ambiguity levels

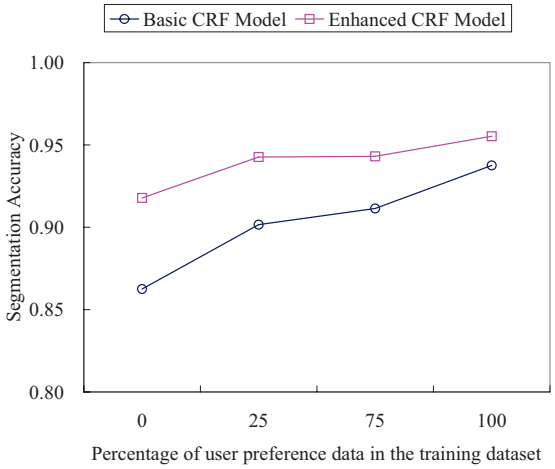

Figure 3: Segmentation accuracy with different portions of user preference data in the training data set

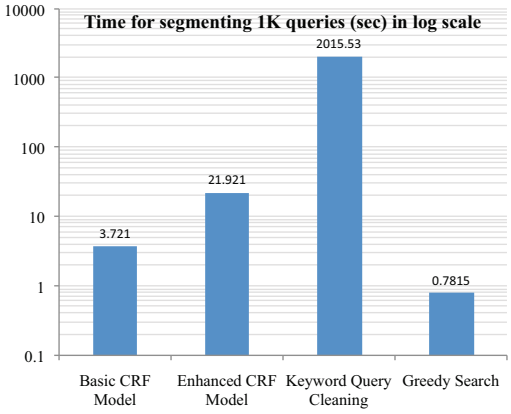

Figure 4: The time for segmenting 1K medium queries on the IMDB dataset

tively lower probability. We call data thus generated user preference data. This is to simulate the fact the user often query only a specific part of the database. The training data sets used to learn the CRF models contains different proportions of user preference data. (Other data in the training dataset are generated normally using the aforementioned random generation method.) As the percentage of user preference data in the training data increases, the segmentation accuracy of both the basic and enhanced CRF models keeps improving, demonstrating the capability of the proposed CRF models to adapt to user preferences.

Figure 4 shows the time used by different algorithms in segmenting 1,000 medium queries. To have a fair comparison in efficiency, the spelling correction feature of keyword query cleaning algorithm is disabled in this experiment. As shown in Figure 4, the proposed CRF models demonstrate two to three orders of magnitude improvement over the keyword query cleaning algorithm, thanks to the highly efficient inference (labeling) process of CRF. Due to the smaller number of feature functions used, the basic CRF is more efficient than the enhanced CRF, at the expense of segmentation accuracy. The Greedy Search algorithm takes the least amount of time because of its simplicity, but it fares poorly in accuracy in non-trivial tasks when ambiguity exists in queries. Other than keyword query cleaning, all algorithms use less than 0.02 second to segment one medium query.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented two CRF-based models for query segmentation, which can greatly help reduce the search space for keyword search in relational databases. They are principled probabilistic models that can be automatically trained and adapted to user preferences. Experiments have demonstrated the effectiveness of the proposed approach. For future work, we would like to study how to improve the CRF-model by accommodating spelling errors in the keywords as well as keywords that do not appear in the database. We would also like to investigate how to perform online segmentation when the keywords in a query appear in a streaming fashion.

8. REFERENCES

[1] Sanjay Agrawal, Surajit Chaudhuri, and Gautam Das. DBXplorer: enabling keyword search over relational databases. In *SIGMOD*, pages 627–627, 2002.

[2] Bolin Ding, Jeffrey Xu Yu, Shan Wang, Lu Qin, Xiao Zhang, and Xuemin Lin. Finding top-k min-cost connected trees in databases. In *ICDE*, pages 836–845, 2007.

[3] Lin Guo, Feng Shao, Chavdar Botev, and Jayavel Shanmugasundaram. Xrank: ranked keyword search over xml documents. In *SIGMOD*, pages 16–27, 2003.

[4] Vagelis Hristidis, L. Gravano, and Yannis Papakonstantinou. Efficient IR-style keyword search over relational databases. In *VLDB*, 2003.

[5] Vagelis Hristidis and Yannis Papakonstantinou. Discover: keyword search in relational databases. In *VLDB*, pages 670–681, 2002.

[6] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, pages 282–289, 2001.

[7] Fang Liu, Clement Yu, Weiyi Meng, and Abdur Chowdhury. Effective keyword search in relational databases. In *SIGMOD*, pages 563–574, 2006.

[8] Ziyang Liu and Yi Cher. Reasoning and identifying relevant matches for xml keyword search. *Proc. VLDB Endow.*, 1(1):921–932, 2008.

[9] Yi Luo, Xuemin Lin, Wei Wang, and Xiaofang Zhou. SPARK: top-k keyword query in relational databases. In *SIGMOD*, pages 115–126, 2007.

[10] Alexander Markowetz, Yin Yang, and Dimitris Papadias. Keyword search on relational data streams. In *SIGMOD*, pages 605–616, 2007.

[11] David Pinto, Andrew McCallum, Xing Wei, and W. Bruce Croft. Table extraction using conditional random fields. In *SIGIR*, pages 235–242, 2003.

[12] Ken Q. Pu and Xiaohui Yu. Keyword query cleaning. *Proc. VLDB Endow.*, 1(1):909–920, 2008.

[13] Bin Tan and Fuchun Peng. Unsupervised query segmentation using generative language models and wikipedia. In *WWW*, pages 347–356, 2008.

[14] Ping Wu, Yannis Sismanis, and Berthold Reinwald. Towards keyword-driven analytical processing. In *SIGMOD*, pages 617–628, 2007.