Keyword Query Cleaning Using Hidden Markov Models

Ken Q Pu
University of Ontario Inst. of Technology
Oshawa, Ontario
ken.pu@uoit.ca

ABSTRACT

In this paper, we consider the problem of keyword query cleaning for structured databases from a probabilistic approach. Keyword query cleaning consists of rewriting the user query, segmenting the keywords, matching each segment to database items, and finally tagging the segments by their meta-data information. We present an efficient and robust solution using Hidden Markov Models (HMM). By modeling user keyword queries using a generative probabilistic HMM-based model, we construct a HMM from the user specified keyword query (and the database instance). The optimal statistical keyword cleaning is computed as the most likely path of the constructed HMM. Furthermore, we demonstrate how the optimal HMM-based keyword cleaning algorithm can be generalized to compute a stream of clean queries ranked from the most likely clean query to the least likely clean query. Finally, we present the implementation of the proposed system and its preliminary performance.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]:

General Terms

Algorithms

Keywords

keyword search, hidden Markov models, segmentation, query cleaning

1. BACKGROUND AND MOTIVATION

Keyword queries have been adopted as the de-facto feature of unstructured text based information systems, such as the World Wide Web. However, for structured databases, such as the relational database management systems, it is very challenging to build high performance and robust keyword query systems. One of many challenges of processing keyword queries for relational databases is that one must dynamically construct documents in the form of tuple join networks during the search phase. Therefore, one must confront the exponential explosion of the search space.

In our previous work on keyword query cleaning [13], we have demonstrated that the search space is significantly reduced by introducing a query preprocessing phase, which we referred to as *keyword query cleaning*. During keyword query cleaning, we are interested to:

- Rewrite query tokens when necessary to compensate for spelling errors, synonyms and aliases of text words in the database.
- Segment query tokens into groups, which we called *query terms*. Each query term is mapped to potential matches in the database.

We refer to the token rewriting and segmentation as *query rewritings*. In our previous work [13], in order to compute sensible query rewritings, we introduced a cost model which penalizes query token rewriting and rewards merging multiple query tokens into segments provided that the resulting segments yield better match against the data in the database. This allows us to compute the optimal query rewriting. We extended the query rewriting computation to support top-k query rewriting, and online computation of query rewritings from keyword streams.

Despite the positive experimental evidence that the cost model works well for real databases (IMDB, northwind, etc.), the cost function is ad-hoc and fixed. Therefore, theoretically, it lacks the explanation as to why certain query rewritings are better by some more fundamental arguments. Furthermore, practically, the cost model is rigid and cannot be adaptive to user feedback and existing query logs. It is also difficult to extend the cost model to incorporate new ways of cleaning a keyword query.

This paper presents an alternative approach to keyword query cleaning by the means of statistical modeling. Users have certain items of data in the database in mind, and the items of data are expressed as keywords subject to unintentional and intentional mutations. For instance, the user may be thinking of the data of **British Columbia** (a province name), **Coffee** (a product name) and **SUM(Sales)** (an aggregated column name). However, when expressed as a keyword query, one may write "**total bc cofee sales**". We

KEYS'09, June 28, 2009, Providence, Rhode Island, USA.

observe that data items are mutated, permuted and concatenated. For instance, **British Columbia** is transformed to **bc** intentionally, while **Coffee** is mutated to **cofee** unintentionally. Finally **SUM(Sales)** is expressed as **total** ... **sales**. We refer to the nondeterministic process that transforms data items to keywords as *abstraction*.

In this paper, our approach is to model abstraction as a stochastic process. By constructing a precise statistical model that describes how statistically data items are expressed as keywords, we can perform keyword query cleaning by means of statistical inference. Given a user specified keyword query, we can systematically compute the most probable data items that the user had in mind using maximal likelihood estimation. Aside from the more disciplined formulation of the problem, a statistical model offers many practical advantages over a cost model of query rewritings:

- It is more intuitive to understand the statistical model and its limitations [8].
- Statistical model can be learnt from training data using parameter estimation. Thus, a statistical approach to query cleaning is naturally adaptive to the evolutions of the user behaviours and the database content. [5, 6]
- There is a wealth of existing literature on statistical models which have been very successfully applied to model of a wide range of recognition problems. [15, 10]

Let Q_{data} denote the sequence of data items from a structured database that the user intends to express. A data item can be one of many things: a value in a tuple, a XML node or name of an attribute, a table or a XML tag. The abstraction process maps Q_{data} to a keyword query Q_{keyword}.

$$\text{abstract} : Q_{\text{data}} \mapsto Q_{\text{keyword}}$$

A statistical model $\mathbf{M}$ describes the likelihood of Q_{data} mapping to Q_{keyword} by joint probability distribution

$$p_\theta(Q_{\text{data}}, Q_{\text{keyword}})$$

where θ is the model parameters. In particular one may formulate the discriminative distribution $p_\theta(Q_{\text{data}}|Q_{\text{keyword}})$. Keyword query cleaning is the inverse of abstraction:

$$\text{clean} : Q_{\text{keyword}} \mapsto Q_{\text{data}}$$

In a statistical framework, it becomes an inference problem in which we compute the inverse of abstraction as:

$$\text{clean}_{\mathbf{M}} : Q_{\text{keyword}} \mapsto \arg\max_{Q_{\text{data}}} p_\theta(Q_{\text{data}}|Q_{\text{keyword}}) \qquad (1)$$

That is, keyword cleaning is to infer the most likely user data query $\text{clean}_{\mathbf{M}}(Q_{\text{keyword}})$ given the keyword query Q_{keyword}, according to a model $\mathbf{M}$.

Adaptive keyword query cleaning from user feedback and query logs becomes the problem of parameter estimation [7]. Give a query log in the form of $\text{LOG} = \{(Q_{\text{keyword}_i}, Q_{\text{data}_i})\}_{i \in I}$, consisting of the user queries Q_{keyword_i} and their corresponding data items Q_{data_i}, the model parameter can be estimated by:

$$\theta^* = \arg\max_\theta p_\theta(\text{LOG}) = \arg\max_\theta \prod_{i \in I} p_\theta(Q_{\text{data}_i}, Q_{\text{keyword}_i}) \qquad (2)$$

The choice of the model $\mathbf{M}$ is critical to the computational issues. Equation 1 and Equation 2 are optimization problems that often do not have closed form solutions for complex models [14]. In this paper, we present some preliminary but promising success of performing statistical keyword query cleaning using hidden Markov models (HMM). HMM has been widely applied a wide range of modeling applications ranging from image processing to natural language processing to speak recognition [2, 9, 11]. It's well known that HMM is highly robust and has efficient algorithms for inference (the Viterbi algorithm) and parameter estimation (the Baum-Welsh algorithm).

2. MARKOVIAN MODELING OF KEYWORD QUERIES

In a structured database, data is broken into small units which are organized by associations. For instance, for a relational database, data is broken into attribute values, which are associated with tuples. Tuples are further grouped into tables, and are connected by foreign key joins. For XML databases, data is text and attributes, which are grouped into nodes. Nodes are further connected by parent-child relation or IDREF links. From a keyword cleaning perspective, different data models can be uniformly treated as a large graph of item interconnected by edges, similar to a RDF graph.

Let $\mathcal{D} = (\mathbf{U}, \mathbf{E})$ be the database graph. The vertices $\mathbf{U}$ consists of all items in a database: data and schema alike. The edges $\mathbf{E}$ reflect how the items are related by the data model of $\mathcal{D}$.

A keyword query Q_{keyword} is formally defined as a sequence of query tokens $Q_{\text{keyword}} = \langle q_1, q_2, \ldots, q_n \rangle$. However, the user's intention is a sequence of database items from $\mathbf{U}$: $Q_{\text{data}} = \langle u_1, u_2, \ldots, u_m \rangle$ where $u_i \in \mathbf{U}$.

We model the generative model $p(Q_{\text{data}}, Q_{\text{keyword}})$ as a hidden Markov model as shown illustrated in Figure 1.

A hidden Markov model, $\mathbf{H}$, is a tuple (X, Y, A, B, π) where X is a set of states, Y a set of output symbols, $A : X \times X \to \mathbb{R}$ the transition probabilities, $B : X \times Y \to \mathbb{R}$ the output probabilities and finally $\pi : X \times \mathbb{R}$ is the initial state probability distribution.

The model parameters of HMM is the probability distributions $\theta = (A, B, \pi)$. HMM is a stochastic finite state machine. During a run of a HMM $\mathbf{H}$, $\mathbf{H}$ initializes the state nondeterministically according to the distribution π. The state transits from the current state x to the next state x' according to the transition distribution $A(x, x')$. Upon the arrival of a state x, an output symbol y is emitted according to the distribution $B(x, y)$. Only the emitted output symbols are observed while the state transitions are hidden.

HMM offers efficient algorithms for inference and parameter estimation algorithms. Given an observation $\mathbf{y} = \langle y_1, y_2, \ldots, y_n \rangle$, using the Viterbi algorithm [8], one can compute the most likely path of states $\mathbf{x}^* = \arg\max_{\mathbf{x}} p(\mathbf{x}, \mathbf{y})$. Furthermore, given a set of identical and independently dis-

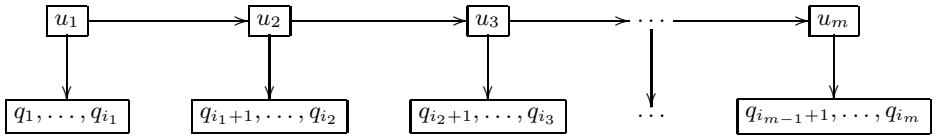

Figure 1: A generative model

tributed (iid) observations $L = \{(\mathbf{x}_i, \mathbf{y}_i)\}$, we can utilize the Baum-Welsh algorithm to estimate the optimal model parameter $\theta^* = (A, B, \pi)$ that best explains the observations L.

2.1 Naive modeling using HMM

Denote $Q_{\text{keyword}}(i, j)$ as the subquery $\langle q_i, q_{i+1}, \ldots, q_j \rangle$. We construct a HMM $\mathbf{H}$ with state space of $X = \mathbf{U}$, and the output symbols $Y = \{Q_{\text{keyword}}(i, j) : 0 \le i \le j \le n\}$ are all possible subqueries of Q_{keyword}. A run of $\mathbf{H}$ describes how a user abstracts Q_{data}. The user starts nondeterministically at some database item d_1, corresponding to the initialization of $\mathbf{H}$. Subsequent database items u_i are entered depending on the previously entered database items $\langle u_1, u_2, \ldots, u_{i-1} \rangle$. We assume a Markovian property: u_i only depends on u_{i-1}. This corresponds to the state transitions of $\mathbf{H}$. Finally, each database item u_i is mutated into a segment of query tokens $\langle q_k, q_k + 1, \ldots \rangle$. The mutation is modeled by the symbol emission of $\mathbf{H}$. One may think of the states (database items u_i) as the true entities that the user intends to express in the keyword query. The sequence of the entities is governmented by the Markovian property. The output symbols (query segments, $\langle q_k, \ldots \rangle$) are the true entities + user injected alterations, either as spelling error or abbreviations. The HMM assumes independence between output symbols – this is reasonably true in our case: how one database entity is altered does not affect the alternation of the next database entity.

There is one subtle difference between our modeling and classical HMMs. Traditionally, the observed output is delimited into individual output symbols $y_1, y_2, \ldots, y_n$. However, in our case, the output symbols are subqueries which are concatenated without explicit delimiters. Fortunately, as we demonstrate, the lack of output delimiters requires only minor modifications to the Viterbi algorithm.

Keyword query cleaning of a user query Q_{keyword} is performed by computing the optimal state path $\mathbf{x}^*$ that generates Q_{keyword} using the Viterbi algorithm. Since each state x is in fact a database item, the state path $\mathbf{x}^*$ is precisely the most probable data query Q_{data}.

In theory, it looks great, but in practice, there are two challenges one must overcome.

- The state space $X = \mathbf{U}$ is much too large in cases of practical databases. We need one state for each database item. For a relational database, the number of database items is linear to the number of tuples in the database.
- The model must be completed with the model parameters (A, B, π). While it is true that these parameters can be estimated from training data, in practice, however, we simply cannot collection the training data from scratch. With the size of the state space, one would need to collect huge collections of training data.

2.2 Practical modeling using HMM

The solution to the shortcomings of the naive model is of two folds.

- Reduce the state space by aggressively pruning irrelevant states using information retrieval style text indexes.
- Boostrap the model parameters using the structures of the database graph $\mathcal{D}$, so that even without training data, one can still perform statistical keyword cleaning.

State space pruning

We only need to consider states that are relevant to the user query Q_{keyword}. Given a subquery $Q_{\text{keyword}}(i, j)$, the relevant database items are those whose textual representation is similar to "$q_i q_{i+1} \ldots q_j$". Let similarity : $\text{TEXT} \times \text{TEXT} \to \mathbb{R}^+$ be a string distance measure; similarity$(s, t) \in [0, 1]$ is the similarity between two strings s and t. Define

$$\mathbf{U}(Q_{\text{keyword}}(i, j)) = \{u \in U : \text{similarity}(u, Q_{\text{keyword}}(i, j)) \ge c\}$$

where we treat $Q_{\text{keyword}}(i, j)$ as a string. So $\mathbf{U}(Q_{\text{keyword}}(i, j))$ is all the database items that are sufficiently similar to subquery $Q_{\text{keyword}}(i, j)$.

Since the keyword query is fixed, without loss of generality, we write $\mathbf{U}(Q_{\text{keyword}}(i, j))$ as $\mathbf{U}(i, j)$. Using standard text indexes [1], one can very quickly compute $\mathbf{U}(i, j)$.

We construct the state space as follows.

$$X = \bigcup_{i,j} \mathbf{U}(i, j)$$

As before $Y = \{Q_{\text{keyword}}(i, j) : 1 \le i \le j \le n\}$.

Bootstrapping the model parameters

We initialize the model parameters (A, B, π) using the database structure $\mathcal{D}$ so that keyword query cleaning can be performed even without any training data.

$$A(x, x') = a_1 \exp(-b_1 \cdot \text{distance}(x, x') + b_2 \cdot \text{popular}(x')) \quad (3)$$

$$B(x, y) = \begin{cases} a_2 \exp(-b_3 \cdot \text{Levenstein}(x, y)) \\ \qquad \text{if } x \in \mathbf{U}(i, j) \text{ and } y = Q(i, j) \\ 0 \qquad \text{else} \end{cases} \quad (4)$$

$$\pi(x) = a_3 \exp(b_4 \cdot \text{popular}(x)) \quad (5)$$

Here, the constants $b_k > 0$ are the new model parameters and $a_k > 0$ are to normalize the distributions so the sum adds up to 1.

The distance function distance : $\mathbf{U} \times \mathbf{U} \to \mathbb{R}^+$ measure the distance between two database items. One can

define distance(u_1, u_2) to be the length of the shortest path connecting u_1 and u_2 in the database graph $\mathcal{D}$. However, due to the size of $\mathcal{D}$, computing the exact distance is intractable in practice. For relational databases, we approximate distance(u_1, u_2) as follows.

- We materialize a set of join indices [16, 12] on a user specified join path along the existing foreign-key constraints.
- If u_1 and u_2 are both tuple values for tuples T_1 and T_2 respectively, then we check the connectivity of T_1 and T_2 along the materialized join indices. If they are connected, that distance(u_1, u_2) is the length of the join path. Otherwise distance$(u_1, u_2) = \infty$.
- If u_1 is a schema item, say a column name, and u_2 is a tuple value for a tuple T_2, then we compute the schema distance between u_1 and T_2.
- If u_1 and u_2 are schema items, then we simply compute the schema distance between u_1 and u_2 as distance(u_1, u_2).

The popularity function popular : $\mathbf{U} \times \mathbb{R}^+$ assigns the popularity of a database item. The value popular(u) can be derived from both the database instance as well as query logs.

We define $\text{popular}_{\text{data}}(u) = |\{e \in \mathbf{E} : u \in e\}|$, i.e. the number of times that u is related to other items in the database. Again, computing the exact value of $\text{popular}_{\text{data}}(u)$ is costly for large databases, but it can be approximated using the prescribed join indices used for the approximation of distance$(-, -)$.

Given a query log L, we can define $\text{popular}_{\text{user}}(u)$ to be the number of times that u has been selected by the keyword queries in L.

Finally, the function Levenstein(x, y) is the string edit distance between the text presentation of the database item x and the subquery y.

This completes the model parameters of $\mathbf{H}$.

3. ADAPTIVE STATISTICAL KEYWORD CLEANING USING HMM

With the complete HMM $\mathbf{H}$ constructed in Section 2, we are able to perform statistical keyword cleaning as outlined in Section 1.

Keyword cleaning is performed by computing the most likely path $\mathbf{x}$ of $\mathbf{H}$ given the observed keyword query Q_{keyword}. It is well known that the dynamic programming (the Viterbi algorithm) can be used to compute optimal path of HMM. Our model has a minor difference: the output is not a sequence of delimited output symbols, but rather the symbols are strings which are concatenated without explicit delimiters. Fortunately, dynamic programming still applies with little modifications. Furthermore, it is quite straightforward to compute a stream of paths ranked in the order of decreasing likelihood.

Optimal keyword cleaning

Given a path $\mathbf{x}$, denote $\mathbf{x}[i]$ to be the i-th state in the path, and $\mathbf{x}[-1]$ to be the last state in the path. Let $\mathbf{x} \oplus \mathbf{x}'$ denote concatenation of two paths. Define

$$\mathbf{x_j}^* = \arg\max_{\mathbf{x}} p_\theta(\mathbf{x}, Q_{\text{keyword}}(0, j))$$

It can be computed as,

$$\mathbf{x_0}^* = \arg\max_{x \in \mathbf{U}} \pi(x) B(x, Q_{\text{keyword}}(0, 0))$$

For $j > 0$, $\mathbf{x_j}^*$ is computed as,

$$\begin{aligned} \mathbf{X_j} &= \{\mathbf{x_i}^* : i < j\} \\ (\mathbf{x}', x') &= \arg\max_{(\mathbf{x}, x) \in \mathbf{X_j} \times \mathbf{U}} A(\mathbf{x}[-1], x) B(x, Q_{\text{keyword}}(i, j)) \\ \mathbf{x_j}^* &= \mathbf{x}' \oplus x \end{aligned}$$

Since each state is a database item, the path $\mathbf{x}_n^*$ is precisely the (statistically) optimal keyword cleaning of the keyword query Q_{keyword}, where $n = |Q_{\text{keyword}}|$. Therefore,

$$Q_{\text{data}} = \mathbf{x}_n$$

The complexity of the keyword cleaning is $\mathcal{O}(|X| \times |Q_{\text{keyword}}|^2)$. The size of the state space $|X|$ depends on the pruning threshold used to compute $\mathbf{U}(i, j)$.

Top-k keyword cleaning

We are interested to generalize the computation of $\mathbf{x}^*$ to computing the top-k likely paths. In fact, as it turns out, the special state structure of $\mathbf{H}$ constructed in Section 2 allows one to compute top-∞, i.e. one can compute the most likely path, then the second most likely, the third likely, etc., without limits. It is made possible because of the special state structure: the state structure X forms a directed acyclic graph partitioned by sets of states $\{\mathbf{U}(i, j) : 0 \le i \le j \le n\}$.

To describe the top-∞ keyword cleaning, we first need some definitions.

By a stream of state paths, we mean scored sequence of state paths: $S = \langle (\mathbf{x}_0, p_0), (\mathbf{x}_1, p_1), \dots \rangle$, which can be infinite. The stream S is *ranked* if the scores are monotonically decreasing. In our case, the scores p_i are the probabilities of the state path $\mathbf{x_i}$ given the observed keyword query.

A transformed stream $f(S)$ is defined as a stream obtained by applying the function f to each path and its probability in S. For for $S = \langle (\mathbf{x}_0, p_0), (\mathbf{x}_1, p_1), \dots \rangle$,

$$f(S) = \langle (f(\mathbf{x}_0), f(p_0)), (f(\mathbf{x}_1), f(p_1)), \dots \rangle$$

Note f must map state paths to state paths and probabilities to probabilities.

We now define a special transformation. Given two states x and x', define $f_{x,x'}$ as:

- For path inputs, $f_{x,x'}(\mathbf{x}) = \mathbf{x} \oplus x'$.
- For probability inputs,

$$f_{x,x'}(p) = p \cdot A(x, x') \cdot B(x', Q_{\text{keyword}}(i, j)$$

where (i, j) is such that $x' \in \mathbf{U}(i, j)$.

Finally, let MERGE be the stream join operator which merges a collection of ranked input streams and produces a ranked output stream.

Now, we can describe the top-∞ keyword cleaning computation. Given a state $x \in \mathbf{U}(i, j)$, define a stream S_x such that S_x is the set of all paths $\mathbf{x}$ such that $\mathbf{x}[-1] = x$

Q_{keyword}	Q_{data}	Time (sec)
K_1 = jacknickson	"Nic Jackson" "Jack Nicholson" "Nickie Jackson" ⋮	0.11
K_2 = tom cruis jack nikelson	"Tom Cruise" "Jack Nicholson" "Tom Cruiso" "Jack Nicholson" ⋮	0.23
K_3 = tom cruis jack nikelson few good menn	"" Cruise" "Jack Nicholson" "Few Good Men, A (1992)" "Tom Cruise" "Jack Nicholson" "Few Good Men, A (1983)" ⋮	0.8

Figure 2: Some sample queries and their cleaning

and ranked in the order of decreasing likelihood given the observed user keyword query Q_{keyword}. It can be computed recursively as follows.

Let $\mathbf{U}_i = \bigcup_{k \le i} \mathbf{U}(k, i)$, then

$$S_x = \underset{x' \in \mathbf{U}_i}{\text{MERGE}} f_{x',x}(S_{x'})$$

This allows a dynamic programming solution to compute the stream of state paths ranked from the most likely state path to the least likely state path. Naturally, each of the state paths is a different way of cleaning the keyword query. The top-∞ keyword cleaning is a generalization of the optimal keyword query cleaning. Effectively, it computes the dynamic programming from a top-down approach.

Adaptive to user feedback

User may not pick the top 1 choice every time. The final data query selected by the user can be feed back to the statistical model for the purpose of adaptation. It is possible for the HMM model to adapt based feedbacks so that accuracy can be improved for future keyword queries. The model adapts to user feedback in two ways:

Changes in popularity measure affects the transition probabilities, thus as a database item becomes more popular, it will be more likely to be visited by the HMM.

Changes to the model parameters $\theta = (b_1, b_2, b_3, b_4)$ can be made. We utilize a gradient ascend algorithm to heuristically modify θ so that the likelihood of the query feedback is maximized. Let the query log be $L = \{(Q_{\text{keyword}_i}, Q_{\text{data}_i})\}_{i \in I}$ where Q_{keyword_i} is issued and the user picked Q_{data_i}. Define the likelihood of the query log:

$$H(\theta) = p_\theta(L) = \prod_{i \in I} p_\theta(Q_{\text{data}_i} | Q_{\text{keyword}_i})$$

Here, we assume that the queries are independent identically distributed. We simply perturb θ so that $p_\theta(L)$ improves:

$$\theta' = \theta + \epsilon \cdot \nabla H$$

where the partial derivatives $\nabla H = \begin{bmatrix} \partial H/\partial b_1 \\ \partial H/\partial b_2 \\ \vdots \end{bmatrix}$ can be approximated numerically. The constant ϵ is a learning rate.

4. IMPLEMENTATION AND PRELIMINARY RESULTS

We have implemented the proposed HMM-based method for statistical keyword query cleaning. Our system is implemented using the Java programming language and runs on a Linux workstation with 500GB disk space and 4GB memory. We utilize Apache Lucene [3] as an information retrieval search engine to compute $\mathbf{U}(i, j)$. Lucene supports fuzzy queries up to a similarity measure.

In order to test the performance and accuracy of the proposed algorithms, we have indexed the IMDB [4] dataset. We have selected actors, movies and directors for our testing. We have built the complete relations between actors/directors and movies. The database consists of over 2 million entities and 4 million relational tuples. Database items $\mathbf{U}$ are person names, movie names, actor biography, movie taglines, and *all* entity and attribute names. There are over 1 million database items. In order to speed up the computation of the distance function distance : $\mathbf{U} \times \mathbf{U} \to \mathbb{N}$, we materialize the actor-movie relation using a highly indexed relational table. This allows us to efficiently compute distance(actor, movie) as well as distance(actor, actor). Finally, statistics are gathered for fast computation of the popularity function, popular : $\mathrm{U} \to \mathbb{N}$.

The HMM based segmentation algorithm bootstraps its model parameters (A, B, π) using Equation 3–5.

Our preliminary results are promising. Without any user feedback, the system is able to achieve the keyword cleaning with good accuracy. Some sample queries are shown in Figure 2. For keyword query K_1, "Nic Jackson" is preferred due to the q-gram string similarity used by the IR search engine. For K_2, "Tom Cruise" is preferred over "Tom Cruiso" because "Tom Cruise" is closely related to "Jack Nicholson", thus boosting the transition probability. Furthermore, "Tom Cruise" has a higher popularity measure. Finally, for K_3, The 1992 release of "A Few Good Men" is correctly placed as the top match due to its close distance to "Jack Nicholson", who is in turn, related to "Tom Cruise". Despite the same IR score for the 1983 version of "A Few Good Men", it is placed second due to the lower distance and popularity measures. In all cases, the performance remains in sub-second range.

Our preliminary evaluation using random queries have shown overall accuracy exceeding 95% for the IMDB dataset for top-3 cleaned queries.

In the extended version of this work, we will present the full experimental evaluation of the HMM by varying the query length, top-k results, spelling noise, etc. and observing their effects on accuracy and performance. We also wish to investigate the trade off between aggressive pruning and relaxed pruning on the performance of the system.

5. CONCLUSION AND FUTURE WORK

We have presented a statistical solution to the keyword cleaning problem using hidden Markov models. We have demonstrated that HMM modeling is quite natural and robust. It offers additional flexibility of being adaptive to user feedback and query logs. We have presented the algorithms for optimal and top-k statistical keyword query cleaning. Our preliminary implementation shows that the HMM-based keyword cleaning performs with high degree of accuracy while maintaining sub-second performance.

Much is to be done in statistical keyword query cleaning. We are actively working on several aspects of our system.

- Online statistical keyword query cleaning: it is possible to perform keyword query cleaning, i.e. HMM-based inference, without examining the entire input keyword query. Literature on online HMM inference and online HMM training offer great insight into high performance online statistical keyword query cleaning algorithms.

- Modeling using general graph models: HMM is a special model belonging to the more general graph models. Is it possible to use more general models? While more general models offer better modeling accuracy, it often comes with expensive computational cost.

- Adaptation using Baum-Welsh algorithm: currently we utilize a gradient based optimization algorithm to adapt to user feedback. However, it may be much more efficient to perform an EM training using the Baum-Welsh algorithm, especially when there is an existing user query log available.

- Schema-aware optimization: we are in the process of designing a specialized HMM-based keyword cleaning algorithm for relational databases. By fixing the data model, one can build the concepts of table space, tables, columns, and attributes into the HMM modeling.

6. REFERENCES

[1] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval, 2nd Ed.* Pearson, 2007.

[2] L. Fissore, F. Ravera, and P. Laface. Using word temporal structure in HMM speech recognition. In *ICASSP '97: Proceedings of the 1997 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '97)-Volume 2*, page 975, Washington, DC, USA, 1997. IEEE Computer Society.

[3] Apache Software Foundation. Apache lucene search engine. Available at http://lucene.apache.org.

[4] http://imdb.com. Internet movie database.

[5] Dan Klein and Christopher D. Manning. Conditional structure versus conditional estimation in nlp models. In *EMNLP '02: Proceedings of the ACL-02 conference on Empirical methods in natural language processing*, pages 9–16, Morristown, NJ, USA, 2002. Association for Computational Linguistics.

[6] R. Kuhn, R. De Mori, and E. Millien. Learning consistent semantics from training data. In *In ICASSP [ICA94]*, pages 37–40, 1994.

[7] Peter M. Lee. *Bayesian Statistics: An Introduction, 3rd Ed.* Arnold, 2004.

[8] Christopher D. Manning and Hinrich Schutze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1999.

[9] Antonio Molina and Ferran Pla. Shallow parsing using specialized HMMs. *J. Mach. Learn. Res.*, 2:595–613, 2002.

[10] Peter Morguet and Manfred Lang. A universal HMM-based approach to image sequence classification. In *ICIP '97: Proceedings of the 1997 International Conference on Image Processing (ICIP '97) 3-Volume Set-Volume 3*, page 146, Washington, DC, USA, 1997. IEEE Computer Society.

[11] Peter Morguet and Manfred Lang. A universal HMM-based approach to image sequence classification. In *ICIP '97: Proceedings of the 1997 International Conference on Image Processing (ICIP '97) 3-Volume Set-Volume 3*, page 146, Washington, DC, USA, 1997. IEEE Computer Society.

[12] Patrick O'Neil and Goetz Graefe. Multi-table joins through bitmapped join indices. *SIGMOD Rec.*, 24(3):8–11, 1995.

[13] Ken Q Pu and Xiaohui Yu. Keyword query cleaning. In *VLDB'08: Proceedings of the 34th International Conference on Very Large Databases*, 2008.

[14] Charles Sutton, Andrew McCallum, and Khashayar Rohanimanesh. Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. *J. Mach. Learn. Res.*, 8:693–723, 2007.

[15] Stephan Vogel, Hermann Ney, and Christoph Tillmann. HMM-based word alignment in statistical translation. In *Proceedings of the 16th conference on Computational linguistics*, pages 836–841, Morristown, NJ, USA, 1996. Association for Computational Linguistics.

[16] Zhaohui Xie and Jiawei Han. Join index hierarchies for supporting efficient navigations on object-oriented databases. In *VLDB'94*, pages 522–533, 1994.