Probabilistic Query Rewriting for Efficient and Effective Keyword Search on Graph Data

Lei Zhang
Karlsruhe Institute of Technology
76128 Karlsruhe, Germany
l.zhang@kit.edu

Thanh Tran
Karlsruhe Institute of Technology
76128 Karlsruhe, Germany
duc.tran@kit.edu

Achim Rettinger
Karlsruhe Institute of Technology
76128 Karlsruhe, Germany
rettinger@kit.edu

ABSTRACT

The problem of rewriting keyword search queries on graph data has been studied recently, where the main goal is to clean user queries by rewriting keywords as valid tokens appearing in the data and grouping them into meaningful segments. The main solution to this problem employs heuristics for ranking query rewrites and a dynamic programming algorithm for computing them. Based on a broader set of queries defined by an existing benchmark, we show that the use of these heuristics does not yield good results. We propose a novel probabilistic framework, which enables the optimality of a query rewrite to be estimated in a more principled way. We show that our approach outperforms existing work in terms of effectiveness and efficiency of query rewriting. More importantly, we provide the first results indicating query rewriting can indeed improve overall keyword search runtime performance and result quality.

1. INTRODUCTION

Keyword search on graph data has attracted large interest. It has proven to be an intuitive and effective paradigm for accessing information, helping to circumvent the complexity of structured query languages and to hide the underlying data representation. Using simple keyword queries, users can search for complex structured results, including connected tuples from relational databases, XML data, RDF graphs, and general data graphs [8, 5, 18]. Existing work so far focuses on the efficient *processing of keyword queries* [6, 5], or effective *ranking of results* [12, 14].

In addition, recent work studies the problem of *keyword query cleaning* [16, 4]. The motivation is *keyword queries are dirty*, often containing words not intended to be part of the query, words that are misspelled, or words that do not directly appear but are semantically equivalent to words in the data. Besides dirty queries, keyword search solutions also face the problem of *search space explosion*. Searching results on graph data requires finding matches for the individual keywords as well as considering subgraphs in the data connecting them, which represent final answers covering all query keywords. The space of possible subgraphs is generally exponential in the number of query keywords. Through grouping keywords into larger meaningful units (called *segments*), the number of keywords to be processed and the corresponding search space is reduced.

The two main tasks involved in query cleaning (henceforth also called *query rewriting*) are *token rewriting*, where query keywords are rewritten as tokens appearing in the data, and *query segmentation*, where tokens are grouped together as segments representing compound keywords. Query rewriting helps to improve not only the result quality but also the runtime performance of keyword search. Towards a rewriting solution that enables more effective and efficient keyword search, we provide the following contributions:

Probabilistic Ranking of Query Rewrites and Its Impact on Keyword Search Effectiveness. The optimality of query rewrites has been defined based on heuristics for scoring tokens and segments, including an adoption of TFIDF [16, 4]. However, we show in this work that for ranking query rewrites, existing work based on these heuristics has several conceptual flaws and does not yield high quality results. Instead of using ad-hoc heuristics, we propose a probabilistic framework for keyword query rewriting, which enables the optimality of query rewrites to be studied in a systematic fashion. In particular, optimality is captured in terms of the probability a query rewrite can be observed given the data, and estimated using the principled technique (Maximum Likelihood Estimation). Furthermore, while previous work only considers the textual information but neglects the rather rich graph structure, which might be more crucial for keyword search on graph data, our approach takes both textual and structural information in the data into account. In [16, 4], it has shown that w.r.t. the proposed ad-hoc notion of optimality, computed rewrites are accurate. However, the actual effect of query rewriting on the quality of keyword search results is not clear. Using the recently established benchmark [2] for keyword search, we show that our approach not only yields *better query rewrites* but more importantly, also *better keyword search results*.

Context-based Computation of Query Rewrites and Its Impact on Keyword Search Efficiency. The problem of computing query rewrites has shown to be NP-hard. A solution [16] based on dynamic programming has been proposed for this, which computes optimal query rewrites by considering all possible combinations of optimal sub-query rewrites. There, the optimality of a rewrite is based on the optimality of all its components, while our probabilistic ap-

Proceedings of the VLDB Endowment, Vol. 6, No. 14

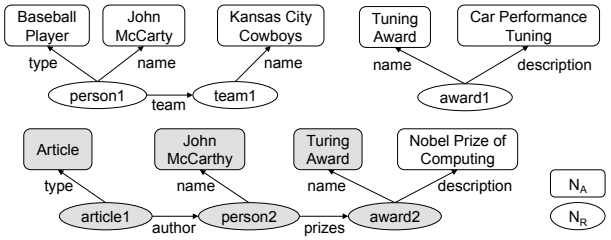


Figure 1: Example data graph

proach enables optimality to be captured merely based on the previously observed context in an incremental rewriting process. We show that this probabilistic model not only produces higher quality results but also can be exploited by a context-based top-k algorithm that is *more efficient* than the previous solution. Moreover, while previous work reported the search space reduction resulting from segmentation, its impact on overall keyword search performance is not clear. In this work, we show that the search space reduction can outweigh the overhead incurred through query rewriting, resulting in *better overall runtime performance*.

Outline. We provide an overview of the problems in Sec. 2. Then, we present our solution for ranking and computing query rewrites along with differences to the most related work in Sec. 3 and Sec. 4, respectively. Experimental results are presented in Sec. 5, followed by more related work in Sec. 6 and conclusions in Sec. 7.

2. OVERVIEW

We firstly provide an overview of the keyword search problem, then discuss the role of keyword query rewriting.

2.1 Keyword Search on Graph Data

Keyword search solutions have been proposed for dealing with different kinds of data, including relational, XML and RDF data. In the general setting, existing approaches treat these different kinds of data as graphs:

DEFINITION 1 (DATA). *Data are captured as a directed labeled graph $D(N, E)$ called* data graph*, where $N = N_R \uplus N_A$ is the disjoint union of* resource *and* attribute value *nodes N_R and N_A, respectively, and $E = E_R \uplus E_A$ is the set of directed* edges*, where E_R are edges between two resources called* relations*, i.e., $e(n_i, n_j) \in E_R$ iff $n_i, n_j \in N_R$, and E_A are edges between a resource and an attribute value called* attributes*, i.e., $e(n_i, n_j) \in E_A$ iff $n_i \in N_R \wedge n_j \in N_A$. Each data element $e \in N \uplus E$ is labeled with some text $L(e)$ called* label *describing e.*

Results in this setting are defined as Steiner trees [8], or *Steiner graphs* in the graph data setting [10, 9]:

DEFINITION 2 (RESULT / STEINER GRAPH). *A* result *to a keyword query Q also called* Steiner graph *is a subgraph of $D(N, E)$ denoted as $D_S = (N_S, E_S)$, which satisfies the following conditions: 1) for every $q \in Q$ there is at least one element $n_q \in N$ (called* keyword element*) that matches q, i.e., the label $L(n_q)$ contains q. The set of keyword elements containing one for every $q \in Q$ is $N_Q \subseteq N_S$; 2) for every possible pair $n_i, n_j \in N_Q$ and $n_i \neq n_j$, there is a path $n_i \rightsquigarrow n_j$, i.e., an edge $e(n_i, n_j) \in E$ or a sequence of edges $e(n_i, n_k) \dots e(n_l, n_j)$ in E, such that every $n_i \in N_Q$ is connected to every other $n_j \in N_Q$. Such a graph is called a d-*length* Steiner graph *when paths that connect keyword elements are of length d or less.*

Keyword Query	Possible Query Rewrites
"*Publication John McCarty Tuning Award*"	Article⊘John⊕McCarthy⊘Turing⊕Award*
	Article⊘John⊕McCarthy⊘Tuning⊕Award
	Article⊘John⊕McCarty⊘Turing⊕Award
	Article⊘John⊕McCarty⊘Tuning⊕Award

Table 1: Possible query rewrites

EXAMPLE 1. *Given the data graph in Fig. 1, for the keyword query shown in Table 1, there is one matching Steiner graph as highlighted in Fig. 1, namely the one connecting the three nodes Article, John McCarthy and Turing Award (assuming that keywords have already been rewritten so that they match the labels of these three nodes, e.g., "Tuning Award" has been rewritten to match the node Turing Award).*

For finding whether some data elements match query keywords, existing solutions typically use an inverted index and treat elements (their labels) as documents (task 1). For finding paths to form Steiner graph from these elements (task 2), they explore the data as an undirected graph, traversing the edges without taking their direction into account. For pragmatic reasons, existing keyword search solutions [5, 18, 10] apply a maximum path length restriction d, such that only paths of length d or less have to be traversed.

2.2 Keyword Query Rewriting

The label $L(e)$ of each data element e and the query Q can be conceived as a sequence of tokens, e.g., the label *Turing Award* consists of two tokens *Turing* and *Award*. Query rewriting firstly maps query keywords (also called query tokens) to tokens appearing in the labels of data elements (*token rewriting*), and then groups the resulting data tokens into segments to form *query rewrites* (*query segmentation*):

DEFINITION 3 (TOKEN REWRITE). *Let* TOKEN^D *be the set of all tokens in the data graph D.* Token rewriting *with factor m is a function* rewrite$_m$*, which maps a query token q to a list of m data tokens $t \in$* TOKEN^D *associated with the respective distance d between q and t. Given a keyword query $Q = \{q_1, q_2, \dots, q_n\}$, a* query token rewrite *is a $m \times n$ matrix M of tokens $t \in$* TOKEN^D*, where the i-th column is obtained through* rewrite$_m(q_i)$.

EXAMPLE 2. *Given the data graph in Fig. 1, we can construct the matrix M for the example query in Table 1 using the rewriting function* rewrite$_2$*:*

$$M = \begin{pmatrix} Article & John & McCarty & Tuning & Award \\ -- & -- & McCarthy & Turing & -- \end{pmatrix}$$

Note that the matrix M might have empty entries when there are less than m candidate data tokens for a query token.

DEFINITION 4 (SEGMENT AND QUERY REWRITE). *Given the query token rewrite M of dimension $m \times n$, a* segment *is a sequence of tokens in M from adjacent columns. A* query rewrite *(also called* segmentation*) is a sequence of continuous and non-overlapping segments $S = s_1 s_2 \dots s_k$ such that for all segments s_i, $1 \leq i \leq k$, the first column of s_{i+1} is next to the last column of s_i, i.e., $start(s_{i+1}) = end(s_i)+1$, where $start(s)$ and $end(s)$ denote the first column and the last column covered by s, respectively. A query rewrite can also be*

seen as a sequence of tokens $t \in M$ and actions α, namely $S = t_1\alpha_1 t_2 \dots t_{n-1}\alpha_{n-1}t_n$, where t_i denotes one token in the i-th column of M and α_i represents a concatenation *action denoted by $\oplus$ or a* splitting *action denoted by $\oslash$. A splitting action $\oslash$ captures the boundary of two segments.*

EXAMPLE 3. *For our example query, Table 1 shows a few query rewrites. The segment-based representation of the rewrite $Article \oslash John \oplus McCarthy \oslash Turing \oplus Award$ is $s_1 = \{Article\}, s_2 = \{John, McCarthy\}, s_3 = \{Turing, Award\}$.*

Note that the first rewrite in the table captures the query we would like to obtain because it yields the Steiner graph presented in the previous example. As opposed to the original query, segments in this rewrite correspond to tokens in the data, thus facilitating the finding of relevant results. Further, because segments stand for compound query keywords, this rewrite contains only three instead of five. Observe that we have three other rewrites, where all constituent segments also correspond to data tokens. However, we can see data elements matching these segments are not connected, i.e., do not form Steiner graphs. We consider a rewrite to be *valid* when it yields Steiner graphs, and *relevant*, when these graphs represent relevant answers. In order to assess the relevance of answers, we use manually defined ground truth provided by the keyword search benchmark [2]. Considering query rewrite optimality under these aspects of validity and relevance makes our work different from the main existing solution [16], which defines optimality based on several heuristics that we will discuss next.

3. PROBABILISTIC QUERY REWRITING

Existing work [16] ranks a query rewrite S based on the sum of all the scores of its segments, where the score of each segment s depends on several heuristics, including the *distance* of tokens in s from the corresponding query keywords and the number of tokens in s. A central heuristic is the one based on an adoption of TFIDF. The TFIDF score of a segment s is defined as $Score_{IR}(s) = max\{tfidf(s,e) : e \in N \uplus E\}$, where $tfidf(s,e)$ is the TFIDF weight of the segment s in the data element e, which is a tuple in previous work. With respect to the two main aspects of query rewriting, namely validity and relevance, we identify the following problems with TFIDF-based ranking:

Relevance. Intuitively, the TFIDF weight of a query term q is high for a document d, when d contains a large number of mentions of q (TF), and q discriminates d well from other documents (IDF). The adoption of TFIDF here computes the weight w.r.t. a tuple. However, query rewrites have to be ranked, not tuples. A query rewrite S may contain several segments corresponding to several tuples. Thus, when S contains a segment s with high TFIDF weight w.r.t. some tuples, it does not mean that S contains a large number of mentions of s and that s discriminate S well from others. In other words, it is not clear why a rewrite S with higher TFIDF weighted segments is more relevant.

Validity. The TFIDF heuristic and others do not consider structural information in the data. Some data elements contain tokens and segments that represent relevant candidates for token rewriting and segmentation. However, these elements only help to generate valid query rewrites, when they are actually parts of some Steiner graphs. Thus, to ensure validity, paths in the data have to be considered.

3.1 Probabilistic Model

Let $Q = \{q_1, q_2, \dots, q_n\}$ be the user query, D be the data, and $S = t_1\alpha_1 t_2 \dots \alpha_{n-1}t_n$ be a query rewrite. The probability $P(S|Q,D)$ can be calculated based on Bayes theorem:

$$P(S|Q,D) = \frac{P(Q|S,D) \cdot P(S|D)}{P(Q|D)} \quad (1)$$

Since $P(Q|D)$ can be considered as a constant, denoted as γ, given the fixed Q and D, we have

$$P(S|Q,D) = \frac{1}{\gamma} \cdot P(Q|S,D) \cdot P(S|D) \quad (2)$$

The term $P(S|D)$ is of particular interest in this work, as it captures the *probability of query rewrites*. For token rewriting, we can focus on $P(Q|S,D)$, which captures the probability of observing (the keywords in) Q given the (tokens in the) intended query rewrite S and the data D.

3.2 Probabilistic Token Rewriting

Since users having the intended token t_i in mind specify the query keyword q_i commonly according to their word usage and spelling habit, we assume that each q_i is only related to the corresponding token rewrite t_i reflecting the user's search intention and the keyword query Q is independent of the data D given the intended query rewrite S, i.e., $P(Q|S,D) = P(Q|S)$. For the purpose of token rewriting, the actions in a query rewrite S can be removed and each q_i is only dependent on t_i. That is,

$$\begin{aligned} P(Q|S) &= P(q_1, q_2, \dots, q_n | t_1\alpha_1 t_2 \dots \alpha_{n-1}t_n) \\ &= P(q_1, q_2, \dots, q_n | t_1, t_2, \dots, t_n) = \prod_{i=1}^{n} P(q_i|t_i) \end{aligned} \quad (3)$$

where $P(q_i|t_i)$ models the likelihood of observing a query keyword q_i, given that the intended token is t_i.

Then, this probability mass is distributed inverse proportionally to the distance $d(q_i, t_i)$, which measures the syntactic and semantic distance between q_i and t_i. In our implementation, $d(q_i, t_i)$ is a combination of edit distance and semantic distance, which is derived from the lexical database WordNet, For each query keyword q_i, we have

$$P(q_i|t_i) = \frac{1}{\varepsilon} \cdot \exp(-\eta \cdot d(q_i, t_i)) \quad (4)$$

where η is a parameter that controls how fast the probability decreases with the distance and ε is a normalization factor.

3.3 Probabilities of Query Rewrites

For query segmentation, S is conceived as a sequence of segments, or a *sequence of token and segmentation action pairs*, such that the probability $P(S|D)$ is estimated based on tokens and actions in S:

$$\begin{aligned} P(S|D) &= P(t_1\alpha_1 t_2 \dots \alpha_{n-1}t_n | D) \\ &= \prod_{i=0}^{n-1} P_{\mathrm{D}}(\alpha_i t_{i+1} | t_1\alpha_1 t_2 \dots \alpha_{i-1}t_i) \end{aligned} \quad (5)$$

where $P_{\mathrm{D}}(\alpha_0 t_1) = P_{\mathrm{D}}(t_1)$ and $P_{\mathrm{D}}(\alpha_i t_{i+1} | t_1\alpha_1 t_2 \dots \alpha_{i-1}t_i)$ stands for $P(\alpha_i t_{i+1} | t_1\alpha_1 t_2 \dots \alpha_{i-1}t_i, D)$. However, for a keyword query Q containing many keywords, computing $P(S|D)$ will incur prohibitive cost when D is large in size. To address this problem, we make the N^{th} order Markov

assumption to approximate that the probability of an action on a token only depends on the N preceding token and action pairs (to be precise, N preceding tokens and $N-1$ actions and $N=2$ in the following examples). That is,

$$P(S|D) \approx \prod_{i=0}^{n-1} P_{\mathrm{D}}(\alpha_i t_{i+1} | t_{i-N+1}\alpha_{i-N+1}\dots\alpha_{i-1}t_i) \quad (6)$$

For computing this, we build upon the idea behind the *n-gram language model*. The n-gram model defines the probability of a sequence of tokens $s = t_1 t_2 \dots t_l$ that appear in the data as the joint probability of observing every token t_{i+1} in s, given the previous tokens $t_{i-N+1}\dots t_i$ (called context), i.e., $P(t_1 t_2 \dots t_l) \approx \prod_{i=0}^{l-1} P(t_{i+1}|t_{i-N+1}\dots t_i)$ (note that instead of n, we use N where $n = N+1$). For various information retrieval and text processing tasks, this approximation based on the Markov assumption has proven to work well. We also rely on this assumption to *focus only on the previously observed context* during the computation of query rewrite probability. Typically, the Maximum Likelihood Estimation is employed, which computes this probability as the count of $t_{i-N+1}\dots t_i t_{i+1}$, divided by the sum of counts of all n-grams that share the same context $t_{i-N+1}\dots t_i$, i.e., $P(t_{i+1}|t_{i-N+1}\dots t_i) = \frac{C(t_{i-N+1}\dots t_i t_{i+1})}{\sum_t C(t_{i-N+1}\dots t_i t)}$, where $C(t_i \dots t_j)$ denotes the count of $t_i \dots t_j$ appearing in the data.

For query segmentation, we need to adopt this idea such that instead of token probability, the *action-token pair* probability specified in Eq. 6 can be derived. First, since query segmentation is order insensitive, i.e., both "John McCarthy" and "McCarthy John" should be grouped into one segment, we consider n-gram as a set of tokens that co-occur in a window of size n instead of a sequence of n tokens that appear contiguously. To facilitate the following discussion, we firstly define the concept of action induced segment:

Definition 5 (Action Induced Segment). *For $Q = \{q_1, q_2, \dots, q_n\}$ and the corresponding query rewrite $S = t_1\alpha_1 t_2 \dots \alpha_{n-1} t_n$, a* segment s_i induced *by action α_{i-1} is the concatenation of the previously induced segment s_{i-1} resulting from α_{i-2} and the token t_i, i.e., $s_i = s_{i-1}t_i$ if $\alpha_{i-1} = \oplus$; otherwise (i.e., $\alpha_{i-1} = \oslash$), $s_i = t_i$. For α_0, we have $s_1 = t_1$. The induced segment $s_i(l)$ is a segment with length (i.e., the number of constituent tokens) no larger than l. For a segment s_i with more than l tokens, $s_i(l)$ is s_i without the first $l(s_i) - l$ tokens, where $l(s_i)$ is the length of s_i.*

While the n-gram model predicts the probability of a token t_{i+1} given the context s_i, the task of query segmentation is to predict the action-token pair $\alpha_i t_{i+1}$, i.e., the probability that t_{i+1} is concatenated with s_i ($\oplus t_{i+1}$) and that t_{i+1} forms a new segment ($\oslash t_{i+1}$). Whereas $\oplus$ depends on the probability t_{i+1} can be observed given s_i, the action $\oslash$ intuitively depends on the probability t_{i+1} has a different context $\neg s_i (\neq s_i)$. To compute the probabilities for both these actions, the entire event space consisting of both contexts s_i and $\neg s_i$ has to be taken into account. Based on these observations, for the case where $i > 0$, we have

$$P_{\mathrm{D}}(\alpha_i t_{i+1} | s_i(N), \neg s_i(N)) = \begin{cases} \frac{C(s_i(N)t_{i+1})}{\sum_t C(s_i(N)t)+C(\neg s_i(N)t)} & \text{if } \alpha_i = \oplus \\ \frac{C(\neg s_i(N)t_{i+1})}{\sum_t C(s_i(N)t)+C(\neg s_i(N)t)} & \text{if } \alpha_i = \oslash \end{cases} \quad (7)$$

where $C(s_i(N)t_{i+1})$ is the count of $s_i(N)t_{i+1}$ as n-gram in the labels of some elements in D. Note that $\sum_t C(s_i(N)t) + C(\neg s_i(N)t) = \sum_t C(t)$. For $i = 0$, the query rewrite probability can be computed by considering only the first token because there is no need to make an action. Thus, we have

$$P_{\mathrm{D}}(\alpha_0 t_1) = P_{\mathrm{D}}(t_1) = \frac{C(t_1)}{\sum_t C(t)} \quad (8)$$

where $C(t)$ is the count of token t in D. The following example shows that while intuitively appealing, using this probability of query rewrite leads to unexpected results.

Example 4. *Suppose that for the partial keyword query Q' = "Publication John McCarty" we have S'=Article $\oslash$ $John \oplus McCarthy$. Given the next query keyword "Tuning", we then have the token rewrites "Tuning" and "Turing", and the counts $C((John \oplus McCarthy)\,Tuning) = 0$ and $C((John \oplus McCarthy)\,Turing) = 0$ because "Tuning" and "Turing" never appear together with "John McCarthy", $C(\neg(John \oplus McCarthy)\,Tuning) = 2$ and $C(\neg(John \oplus McCarthy)\ Turing) = 1$ because "Tuning" and "Turing" appear respectively twice and once in other contexts. Based on Eq. 7, we have $P(\oslash Tuning|John \oplus McCarthy) = \frac{2}{3}$ and $P(\oslash Turing|John \oplus McCarthy) = \frac{1}{3}$. The resulting query rewrites are respectively $Article \oslash John \oplus McCarthy \oslash Tuning$ and $Article \oslash John \oplus McCarthy \oslash Turing$, where the former is more likely than the latter. Continuing with "Award", we obtain 4 final query rewrites where those with "Tuning" still have higher probability than those with "Turing". Looking at the data, we rather expect the contrary, i.e., those with "Turing" should be preferred.*

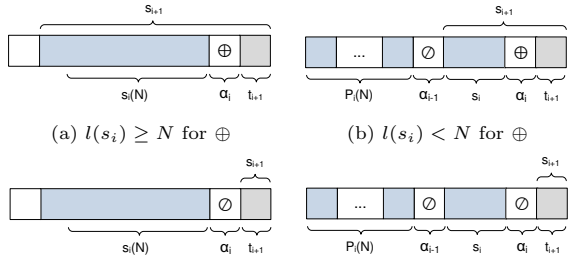
3.4 Probabilities of Valid Query Rewrites

The previous model considers relevance but not validity. The probability of every action-token pair $\alpha_i t_{i+1}$ depends on the count of $\neg s_i(N)t_{i+1}$. This may lead to cases, where query rewrites do not yield Steiner graphs, i.e., the segments match keyword elements that are not connected. In particular, the previous example show that $P(\oslash Tuning|John \oplus McCarthy)$ is relatively high (i.e., relevant) because $Tuning$ matches some data elements. However, $John \oplus McCarthy$ and $Tuning$ match data elements that are not connected and thus the splitting action inducing $John \oplus McCarthy \oslash Tuning$ does not result in any answer (i.e., is not valid).

The above problem arises because the language model is designed to model unstructured data. It might be ineffective when applied to Steiner graphs, which are rich in structural information. Extending this model to take the graph structure into account, we propose to focus on estimating the actions only based on events that actually lead to results. The goal is to produce *valid query rewrites*, which yield non-empty sets of Steiner graphs. Clearly, it follows from Def. 2 that a query rewrite is valid when every possible pair of its segments is connected. More formally, the connectivity of segments is defined as follows:

Definition 6 (Connected Segments). *Let s_i and s_j be two segments, $N_i, N_j \subseteq N$ be the sets of corresponding keyword elements in $D(N, E)$ such that for each $n_i \in N_i$ and $n_j \in N_j$, the labels $L(n_i)$ and $L(n_j)$ contain s_i and s_j, respectively. The segments s_i and s_j are* connected *(denoted as $s_i \leftrightsquigarrow s_j$) when there is at least one $n_i \in N_i$ and one $n_j \in N_j$ and $n_i \neq n_j$ such that $n_i \leftrightsquigarrow n_j$, where the d-length restriction of paths also applies.*

With the N^{th} order Markov assumption, there are two cases to consider for computing valid query rewrites. When



(a) $l(s_i) \geq N$ for $\oplus$ (b) $l(s_i) < N$ for $\oplus$

(c) $l(s_i) \geq N$ for $\oslash$ (d) $l(s_i) < N$ for $\oslash$

Figure 2: Segment s_{i+1} induced by action α_i performed on segment s_i (set of segments $\mathcal{P}_i(N)$) and token t_{i+1}

the previously induced segment s_i has length equal or greater than N, it suffices to focus on $s_i(N)$ to predict the next action α_i on t_{i+1}. Fig. 2(a) and 2(c) illustrate this, showing the induced segment s_{i+1} given the action α_i is $\oplus$ or $\oslash$. As before, the events for $\oplus t_{i+1}$ are $s_i(N)t_{i+1}$ (clearly, these events lead to valid segments because they correspond to cases where elements in the data graph have labels containing $s_i(N)t_{i+1}$). In cases where t_{i+1} does not have context $s_i(N_i)$, $\oslash$ is considered. However, $\oslash$ only yields Steiner graphs when t_{i+1} is connected with $s_i(N)$. That is, instead of all $\neg s_i(N)t_{i+1}$, only the events $s_i(N) \leftrightsquigarrow t_{i+1}$ are relevant in this case. Note that $\neg s_i(N)t_{i+1}$ captures all events where t_{i+1} does not co-occur with $s_i(N)$, which clearly include all events where t_{i+1} appears in the label $L(n_i)$, $s_i(N)$ appears in the label $L(n_j)$ and $n_i \neq n_j$. The set of events denoted by $s_i(N) \leftrightsquigarrow t_{i+1}$ is a subset of events captured by $\neg s_i(N)t_{i+1}$, namely $n_i \leftrightsquigarrow n_j$ instead of $n_i \neq n_j$. We use $s_i(N) \leftrightsquigarrow t_{i+1}$ to focus on valid query rewrites while $\neg s_i(N)t_{i+1}$ stands for all query rewrites. For estimating the probability, we have

$$P_{\mathrm{D}}(\alpha_i t_{i+1} | s_i(N), s_i(N) \leftrightsquigarrow) = \begin{cases} \frac{C(s_i(N)t_{i+1})}{\sum_t C(s_i(N)t)+C(s_i(N)\leftrightsquigarrow t)} & \text{if } \alpha_i = \oplus \\ \frac{C(s_i(N)\leftrightsquigarrow t_{i+1})}{\sum_t C(s_i(N)t)+C(s_i(N)\leftrightsquigarrow t)} & \text{if } \alpha_i = \oslash \end{cases} \quad (9)$$

As opposed to the previous adoption of the n-gram model, focusing on s_i alone when it has length less than N is not enough. This is because the connectivity of segments induced previous to s_i has an impact on the validity of query rewrites. The action α_i on the next token t_{i+1} depends on the set of previously induced segments $\mathcal{P}_i(N)$ and s_i, where $\mathcal{P}_i(N)$ is the set of the induced segments that precede s_i and together with s_i, contains at most N tokens in total, i.e., $\sum_{s_{\rho_i} \in \mathcal{P}_i(N)} l(s_{\rho_i}) + l(s_i) \leq N$. The components to be considered for the probability estimation of $\oplus$ and $\oslash$ are shown in Fig. 2(b) and 2(d), respectively. The segment $s_i t_{i+1}$ resulting from the concatenation action $\oplus$ is valid only when $s_i t_{i+1}$ is connected to all preceding segments in $\mathcal{P}_i(N)$. Similarly, a splitting action $\oslash$ only leads to valid segments when t_{i+1} is connected to all preceding segments in $\mathcal{P}_i(N) \cup \{s_i\}$ (henceforth, simply denoted as $\mathcal{P}_i(N)s_i$). Thus in this case, the probability is estimated as

$$P_{\mathrm{D}}(\alpha_i t_{i+1} | \mathcal{P}_i(N) \leftrightsquigarrow s_i, \mathcal{P}_i(N)s_i \leftrightsquigarrow) = \begin{cases} \frac{C(\mathcal{P}_i(N)\leftrightsquigarrow s_i t_{i+1})}{\sum_t C(\mathcal{P}_i(N)\leftrightsquigarrow s_i t)+C(\mathcal{P}_i(N)s_i\leftrightsquigarrow t)} & \text{if } \alpha_i = \oplus \\ \frac{C(\mathcal{P}_i(N)s_i\leftrightsquigarrow t_{i+1})}{\sum_t C(\mathcal{P}_i(N)\leftrightsquigarrow s_i t)+C(\mathcal{P}_i(N)s_i\leftrightsquigarrow t)} & \text{if } \alpha_i = \oslash \end{cases} \quad (10)$$

where $C(\mathcal{P} \leftrightsquigarrow s)$ denotes the count of segment s that is connected to all segments in the set of segments $\mathcal{P}$.

In addition to these two cases, Eq. 8 also applies for the case $i = 0$, because no actions have to be considered.

Example 5. *Consider the same case as in Example 4, for Q', S' and the next query keyword "Tuning". Due to the same reason, we have $C((John \oplus McCarthy)\, Tuning) = 0$ and $C((John \oplus McCarthy)\, Turing) = 0$. Differently, we observe that $C((John \oplus McCarthy) \leftrightsquigarrow Tuning) = 0$ and $C((John \oplus McCarthy) \leftrightsquigarrow Turing) = 1$, because "Turing" is connected with "John McCarthy" once but "Tuning" never. Based on Eq. 9, we have $P(\oslash Turing | John \oplus McCarthy) = 1$. Accordingly, the only query rewrite with non-zero probability is $Article \oslash John \oplus McCarthy \oslash Turing$. When continuing with the keyword "Award", instead of a total of 4 final query rewrites, only the valid query rewrite $Article \oslash John \oplus McCarthy \oslash Turing \oplus Award$ remains.*

3.5 Reward Maximization Framework

Besides this principled ranking model based on language modeling, additional heuristics that may perform well in specific settings can be added on top using a reward model. A typical assumption in keyword search is that when a result is more compact, it is considered to be more meaningful and relevant [8]. Also, neighboring query keywords should be grouped together to produce longer segments [16].

We propose a reward model to accommodate heuristics. A reward is associated with every action made in the query rewriting process. To give preference to longer segments for instance, we assign a reward for each action α_i as

$$R(\alpha_i) = \exp(\beta \cdot l(s_{i+1})) \quad (11)$$

where s_{i+1} is the segment induced by α_i, β is used to control the importance of this length based heuristic and $R(\alpha_0) = 1$.

The *overall reward* of a query rewrite S is computed from the rewards of all actions made during query rewriting, i.e.,

$$R(S) = \prod_{i=0}^{n-1} R(\alpha_i) \quad (12)$$

The final ranking, which combines the probability of query rewrites $P(S|Q, D)$ with the additional quality criteria $R(S)$, is captured by the *conditional reward* defined as

$$R(S|Q, D) = R(S) \cdot P(S|Q, D) = \frac{1}{\gamma} \cdot R(S) \cdot P(Q|S) \cdot P(S|D) = \frac{1}{\gamma} \cdot \prod_{i=0}^{n-1} R(\alpha_i) \cdot P(q_{i+1}|t_{i+1}) \cdot P_{\mathrm{D}}(\alpha_i t_{i+1}|e) \quad (13)$$

where $e = null$ when $i = 0$, $e = \{s_i(N), s_i(N) \leftrightsquigarrow\}$ when $l(s_i) \geq N$, and $e = \{\mathcal{P}_i(N) \leftrightsquigarrow s_i, \mathcal{P}_i(N)s_i \leftrightsquigarrow\}$ when $l(s_i) < N$. Now, we arrive at our final notion of optimality:

Definition 7. *(*Optimal Query Rewrites*). Given the data D, the query Q and its set of query rewrites $\boldsymbol{S}$, the* optimal query rewrite S^* *is the one with the highest conditional reward, i.e., $S^* = \arg\max_{S \in \boldsymbol{S}} R(S|Q, D)$. The* top-$k$ optimal query rewrites $\boldsymbol{S}^k$ *are the k ones with the highest conditional rewards.*

4. COMPUTING TOP-K QUERY REWRITES

We will briefly revisit existing work on query rewriting and show that our model enables a more efficient algorithm by focusing only on the previously observed *context*. First, we present the indexes and then the top-k rewriting algorithm.

Segment	Segment Length	Count
s_i	$l(s_i) \leq N+1$	$C(s_i)$
$s_i \leftrightsquigarrow S_j$	$l(s_j) + \sum_{s \in S_i} l(s) \leq N+1$	$C(s_i \leftrightsquigarrow S_j)$

Table 2: The extended n-gram index capturing segments containing no more than $N+1$ tokens, and their connections

4.1 Indexing

For token rewriting, tokens are managed separately in a *token index*. It keeps tokens in the data as well as semantically related entries such as synonyms extracted from WordNet. The semantic distance between them is precomputed and stored. This and the edit distance between query and index tokens are used to compute $P(q_{i+1}|t_{i+1})$ in Eq. 13.

For query segmentation, we build an *extended n-gram index* to materialize segments and connections between them. It stores all segments s_i containing no more than $N+1$ tokens and their counts. Further, let s_j denote segments that have length less than $N+1$. For every s_i, the set of all possible combinations of segments connected to s_i that together with s_i have total length no more than $N+1$, denoted as S_j, are stored in the index together with the count of $s_i \leftrightsquigarrow S_j$. The extended n-gram index is illustrated in Table 2. This index is employed to compute $P_{\mathrm{D}}(\alpha_i t_{i+1}|e)$ in Eq. 13.

For efficient extended n-gram indexing, we employ the concept of *connectivity matrix* M_D^d, which is a boolean matrix capturing paths between nodes in the data graph D. An entry m_{ij}^d in M_D^d is 1, iff there is a path between nodes n_i and n_j of length no larger than d; otherwise, m_{ij}^d is 0. The matrix M_D^d is constructed iteratively using the formula $M_D^d = M_D^{d-1} \times M_D^1$. These matrices can be represented by tables of the maximum size n_c containing connected node pairs in D, where n_c denotes the number of node pairs that are connected by paths of length d or less. M_D^d is then generated by performing join on M_D^{d-1} and M_D^1. For further details, we refer the interested readers to [19].

Now we clarify the index costs of our approach. Let n_a, n_r and $n = n_a + n_r$ be the number of attribute value nodes, resource nodes and all nodes in D respectively, and l be the bound of their labels. The time complexity and index size w.r.t. the token index are both $O(n_a \cdot l)$. For constructing the extended n-gram index, nodes in the data graph have to be joined for computing paths between them. In the worst case, a join on $input_i$ and $input_j$ requires $|input_i| \times |input_j|$ time such that the complexity of computing paths with length no larger than d is $O(n_c^2 \cdot d)$. In practice, join operation can be performed more efficiently using special indexes and implementations like hash join. As a result, instead of $|input_i| \times |input_j|$, a join requires only $|input_i| + |input_j|$ such that the complexity is $O(n_c \cdot d)$. Clearly, there are at most $O(n_a \cdot l)$ segments s_i resulting in the time complexity and index size both as $O(n_a \cdot l)$. For each s_i, at most $O((n_{a \leftrightsquigarrow}^{max} \cdot l)^N)$ combinations of connected segments S_j can be found, where $n_{a \leftrightsquigarrow}^{max}$ denotes the maximum number of attribute value nodes that are connected with one and the same attribute value node by paths. As this has to be done for all segments, the complexity for processing them is $O(n_a \cdot l \cdot (n_{a \leftrightsquigarrow}^{max} \cdot l)^N)$. Accordingly, the index size w.r.t. the connected segments also comes to $O(n_a \cdot l \cdot (n_{a \leftrightsquigarrow}^{max} \cdot l)^N)$.

In summary, the total time complexity of constructing the extended n-gram index is $O(n_c \cdot d + n_a \cdot l + n_a \cdot l \cdot (n_{a \leftrightsquigarrow}^{max} \cdot l)^N)$, including time for join processing and time for indexing the individual segments s_i and the connected segments $s_i \leftrightsquigarrow S_j$.

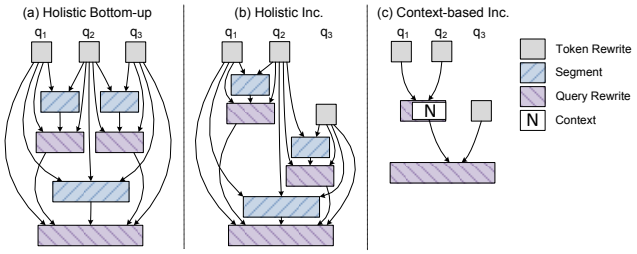


Figure 3: Approaches to query rewriting

The total index size is $O(n_a \cdot l + n_a \cdot l \cdot (n_{a \leftrightsquigarrow}^{max} \cdot l)^N)$, including the indexes of s_i and $s_i \leftrightsquigarrow S_j$. In our experiments, we use $N = 2$ ($n = 3$), which has shown to be sufficient for queries used in the benchmark [2]. Additionally, while $n_{a \leftrightsquigarrow}^{max} = n_a$ and $n_c = n^2$ at the most, in practice they are likely to be relatively small, as one node is not connected to all others but only a limited number of them, especially given the maximum path length d, such that the overall time complexity and index size are much smaller than the worst case. Compared with the indexing of previous work [16], which has the time complexity and index size both as $O(n_a \cdot l)$, our indexing process is still more expensive. However, the additional indexing consumption will become the supplementary to the online query processing, which we will discuss later.

4.2 Holistic Top-k Query Rewriting

Previous work [16] has shown that the problem of computing top-k query rewrites is NP-hard and proposed a dynamic programming solution, which relies on a procedure for computing the top-k segments ($\texttt{find\_}\boldsymbol{s}^k$). The input is the token rewrite matrix M of dimension $m \times n$ (n denotes number of query keywords and m the number of tokens for every keyword). For any given (sub-)query covering keywords from i to j, $\texttt{find\_}\boldsymbol{s}^k$ computes the optimal segments $\boldsymbol{s}^k(i,j)$ that cover the columns from i to j in M. A greedy algorithm is employed for scanning paths in the submatrix of dimension $m \times n'$, $n' = j - i + 1$, which in the worst case, produces $m^{n'}$ possible segments. The complexity of $\texttt{find\_}\boldsymbol{s}^k$ is $\mathcal{O}(m^l)$, when assuming that the lengths of database terms, namely labels, are bounded by l and $l < n'$, otherwise $\mathcal{O}(m^{n'})$.

Clearly, query rewriting solution ($\texttt{find\_}\boldsymbol{S}^k$) covering the columns from i to j may include optimal segments of length n' as well as any combination of smaller segments in subsolutions that spans from i to j for finding the top-k rewrites $\boldsymbol{S}^k(i,j)$, which results in the complexity of $\mathcal{O}(k \cdot n' \cdot m^l)$. For computing rewrites of a query of length n, we need to find the optimal segments of length n, as well as solving (a maximum of n^2) sub-problems of finding and combining query rewriting solutions of (sub-)queries covering keywords from i to j, $1 \leq i \leq j \leq n$, such that the complexity of computing top-k query rewrites is $\mathcal{O}(k \cdot n^3 \cdot m^l)$.

Fig. 3(a) illustrates the *bottom-up approach*, where each box with label *Token Rewrite* denotes a set of tokens in the data for each keyword and each box with label *Segment* and *Query Rewrite* stands for a set of optimal segments and rewriting solutions for a particular pair of (i,j), respectively. For computing the *Query Rewrite* box corresponding to $i = 1$ and $j = 3$, which represents the solution to the final query consisting of three keywords, this approach starts with smaller solutions and iteratively combines them (the combination is illustrated through arrows). The *incremental variant* of this approach is shown in Fig. 3(b), which

involves solving the same (number of) subproblems. The difference is only the order in which the sub-solutions are combined (it incrementally covers more keywords in every iteration). An early return condition is introduced, which can yield $\mathcal{O}(k \cdot n^2 \cdot m^l)$ but because there is no guarantee for this to apply, the worst case complexity is still $\mathcal{O}(k \cdot n^3 \cdot m^l)$.

4.3 Context-based Top-k Query Rewriting

A substantial difference between previous work and ours lies in the notion of optimal query rewrites. The previous algorithm takes all possible segments of a (sub-)query rewrite into account because determining optimality requires computing the score of every (sub-)query rewrite, which is based on the score of all its segments. As opposed to that, our probabilistic model provides a principled way to compute query rewrite scores based on query rewrites probabilities, and to *focus only on the previously observed context.*

We propose an incremental top-k procedure that starts with query rewrites containing one token and then iteratively constructs larger query rewrites by appending more token rewrites. Fig. 3(c) illustrates that query rewrites in each iteration are computed based on the combination of query rewrites obtained in the previous iteration and token rewrites from the current iteration. The main difference to the holistic approach is that in each iteration, instead of considering all combinations of sub-solutions as well as the segments covering the current query, we directly employ the previous query rewrites. In particular, we focus on those ones that vary in the context of a fixed length N (because intuitively speaking, only this context has an impact on the optimality). We introduce the notion of *pattern* to group query rewrites representing the same context.

Definition 8 (Prefix, Suffix and Pattern). *Given a (partial) query rewrite $S = t_1\alpha_1 t_2 \dots \alpha_{n-1} t_n$, a* prefix *of S with length l is a partial query rewrite $\widehat{S} = t_1\alpha_1 t_2 \dots \alpha_{l-1} t_l$ and a* suffix *of S with length l is a partial query rewrite $\hat{S} = t_{n-l+1}\alpha_{n-l+2} t_{n-l+2} \dots \alpha_{n-1} t_n$, where $1 \leq l \leq n$. The* pattern *p of a query rewrite S is the suffix of S with length N, when S has more than N tokens, otherwise p is S.*

When partial query rewrites share the same pattern, the one with higher conditional reward is preferred over one other because it results in final rewrites with higher rewards:

Lemma 1. *Let $Q = Q'Q''$ consisting of two partial queries Q' and Q''. Let S'' be a query rewrite corresponding to Q'', S'_1, S'_2 and $S_1 = S'_1\alpha S'', S_2 = S'_2\alpha S''$ be two particular query rewrites corresponding to Q' and Q, respectively. When S'_1 and S'_2 share the same pattern, i.e., $p(S'_1) = p(S'_2)$, we have*

$$R(S'_1|Q', D) > R(S'_2|Q', D) \Longrightarrow R(S_1|Q, D) > R(S_2|Q, D)$$

Proof Outline: Consider $l(Q'') = 1$. For any rewrite of Q'' denoted by t (i.e., $S'' = t$), we have conditional rewards $R(S_1|Q, D) = \frac{1}{\gamma} \cdot R(S'_1|Q', D) \cdot [R(\alpha) \cdot P(Q''|t) \cdot P_{\text{D}}(\alpha t|e_1)]$ and $R(S_2|Q, D) = \frac{1}{\gamma} \cdot R(S'_2|Q', D) \cdot [R(\alpha) \cdot P(Q''|t) \cdot P_{\text{D}}(\alpha t|e_2)]$ using Eq. 13. When S'_1 and S'_2 have the same pattern p, the events are same, i.e., $e_1 = e_2$ such that $P_{\text{D}}(\alpha t|e_1) = P_{\text{D}}(\alpha t|e_2)$. Hence, if $R(S'_1|Q', D) > R(S'_2|Q', D)$, then we have $R(S_1|Q, D) > R(S_2|Q, D)$. This also generalizes to $l(Q'') > 1$. For $Q'' = \{q_j, \dots, q_n\}$, we have $R(S_1|Q, D) = \frac{1}{\gamma} \cdot R(S'_1|Q', D) \cdot \prod_{i=j-1}^{n-1} [R(\alpha_i) \cdot P(q_{i+1}|t_{i+1}) \cdot P_{\text{D}}(\alpha_i t_{i+1}|e_{i,1})]$ and $R(S_2|Q, D) = \frac{1}{\gamma} \cdot R(S'_2|Q', D) \cdot \prod_{i=j-1}^{n-1} [R(\alpha_i) \cdot P(q_{i+1}|t_{i+1}) \cdot P_{\text{D}}(\alpha_i t_{i+1}|e_{i,2})]$. Because $e_{i,1} = e_{i,2}$ for $j - 1 \leq i < n$. □

Algorithm 1: Finding Top-k Query Rewrites

Input: the user query $Q = \{q_1, q_2, \dots, q_n\}$.
Result: the top-k optimal query rewrites $\boldsymbol{S}^k$.

```
P ← ∅;
foreach i ∈ [1 ... n] do
    (P', P) ← (P, ∅);
    T_i ← TokensRewrites(q_i);
    foreach sp ∈ CommonSubpatterns(P') do
        P'_sp ← PatternsWithSuffix(P', sp);
        foreach t ∈ T_i do
            S^k_{sp⊕t} ← ∪^k_{p'∈P'_sp} (S^k_{p'} ⋈_⊕ t);
            if S^k_{sp⊕t} ≠ ∅ then
                P ← P ∪ {sp ⊕ t};
            end
            S^k_{sp⊘t} ← ∪^k_{p'∈P'_sp} (S^k_{p'} ⋈_⊘ t);
            if S^k_{sp⊘t} ≠ ∅ then
                P ← P ∪ {sp ⊘ t};
            end
        end
    end
end
S^k ← ∪^k_{p∈P} S^k_p ;
return S^k;
```

We not only prefer the ones with higher rewards but, more specifically, we can focus on the k ones with highest rewards. We provide this theorem to capture that it is sufficient to keep track of the top-k rewrites for each distinct pattern:

Theorem 1. *Let $Q = (q_1, \dots, q_n)$ be the query and $Q' = (q_1, \dots, q_i)$ be any partial query s.t. $0 < i < n$. Let $\boldsymbol{S}^k$ be the top-k query rewrites of Q and $\boldsymbol{S}'^k_p$ be those top-k query rewrites of Q' with pattern p. Then for any non-top-k query rewrite $S'_p \notin \boldsymbol{S}'^k_p$ with pattern p, there is no top-k query rewrite $S \in \boldsymbol{S}^k$ such that S'_p is a prefix of S.*

Proof Outline: Assume that there is a top-k query rewrite $S = S'_p\alpha S''$ of Q with a non-top-k S'_p as prefix. Let $\bar{S} = \bar{S}'_p\alpha S''$ be a query rewrite of Q with a top-k $\bar{S}'_p \in \boldsymbol{S}'^k_p$ as prefix. As $R(\bar{S}'_p|Q', D) > R(S'_p|Q', D)$, it follows from Lemma 1 that $R(\bar{S}|Q, D) > R(S|Q, D)$. Thus, there are at least k query rewrites $\bar{S}$ with $R(\bar{S}|Q, D) > R(S|Q, D)$, which contradicts the assumption that S is a top-k rewrite. □

4.4 Algorithm

Based on these results, we propose an algorithm that in every iteration, joins token rewrites with previous partial query rewrites and keeps the top-k results for each pattern.

Definition 9 (Action Induced Join). *Let $\boldsymbol{S}$ be a set of partial query rewrites with non-zero rewards, i.e., $\forall S \in \boldsymbol{S}, R(S|Q, D) > 0$. The join between $\boldsymbol{S}$ and a token t induced by an action α results in a new set of query rewrites*

$$\boldsymbol{S} \bowtie_\alpha t = \{S\alpha t | S \in \boldsymbol{S} \wedge R(S\alpha t|Q, D) > 0\}$$

Performing this join thus requires computing the reward for $S\alpha t$ (via Eq. 13). The $\bowtie_\alpha$ is only successful when adding t to S (through concatenation or splitting) does not render the resulting rewrite invalid, i.e., only when $R(S\alpha t|Q, D) > 0$.

Definition 10 (Top-k Union). *Given sets of rewrites $[\boldsymbol{S}] = \{\boldsymbol{S}_1, \dots, \boldsymbol{S}_m\}$, where every rewrite in $\boldsymbol{S}_i$ is associated with a reward, the* top-k union $\cup^k_{\boldsymbol{S}_i \in [\boldsymbol{S}]} \boldsymbol{S}_i$ *simply returns the k rewrites in $\cup_{\boldsymbol{S}_i \in [\boldsymbol{S}]} \boldsymbol{S}_i$ with the highest rewards.*

Dataset	Size	Rel.	Tuples	I_{Token}	I_{PQR}	I_{PVQR}	I_{BQR}
Mondial	9	28	17,115	0.2/0.04	0.3/0.08	1.8/0.18	2.2/0.05
IMDb	516	6	1,673,074	7.9/8.67	179/17.6	303/40.8	150/9.53
Wikipedia	550	6	206,318	13/2.18	320/3.46	445/8.01	176/2.26

Table 3: Dataset size, number of relations and tuples, index size/indexing time w.r.t. token index I_{Token} (same one used by all approaches) and the additional indexes used by two variants of our approach I_{PQR}, I_{PVQR} and the one used by the state-of-the-art baseline I_{BQR} (all sizes and time are in MB and minutes)

Dataset	$\|Q\|$	$\|q\|$	$\bar{\|q\|}$	$\lfloor\|R\|$	$\bar{\|R\|}$
Mondial	50	1-5	2.04	1-35	5.90
IMDb	50	1-26	3.88	1-35	4.32
Wikipedia	50	1-6	2.66	1-13	3.26

Table 4: Number of queries $|Q|$, range in number of query keywords $|q|$ and relevant results $\lfloor|R|$, average number of query keywords $\bar{|q|}$ and relevant results $\bar{|R|}$ per query

Employing these operators, Alg. 1 starts with the first query keyword ($i = 1$) and iteratively constructs larger rewrites by appending more keywords ($1 < i \leq n$). It uses P' and P to keep track of the patterns of the last and current iteration, and $\boldsymbol{S}_p^k$ to keep track of the top-k rewrites for each pattern p. In every iteration, $sp\alpha t$ are collected (line 10 and 14) and added to P, where sp is a subpattern and t a token rewrite. A subpattern sp of p is simply p when $l(p) < N$, otherwise it is $p(N-1)$ (p without the first token). The grouping of patterns in P' to their subpatterns sp (line 6) yields group containing elements $p' \in P'_{sp}$ that share the same suffix sp. For each q_i, a list T_i of m token rewrites are retrieved from the token index (line 4). For every subpattern sp and $t \in T_i$, the new patterns $sp \oplus t$ and $sp \oslash t$ can be formed. For each new pattern, the top-k query rewrites $\boldsymbol{S}^k_{sp\alpha t}$ are computed and updated by employing $\bowtie_\alpha$ and top-k union (line 8 and 12). The final top-k query rewrites of Q are computed by applying the top-k union on the top-k results $\boldsymbol{S}_p^k$ obtained for each $p \in P$ (line 19).

Complexity. In each iteration, there are at most m token rewrites, which have to be joined with the k results for each pattern. In the worst case, the number of patterns is same as the number of segments of length N, which as discussed, is m^N. As this has to be done for n iterations, the total complexity of Alg. 1 is $O(k \cdot n \cdot m^{N+1})$. With respect to the complexity of the holistic approach, $\mathcal{O}(k \cdot n^3 \cdot m^l)$, using previously obtained query rewrites in every iteration and focusing on the context of length N translate to the changes from n^3 to n and m^l to m^{N+1}. The former can yield a substantial difference in performance because while the other parameters can be fixed to a small number, the number of keywords n cannot be controlled and may be large. The latter effect can also be substantial as it has been shown that n-grams with a relatively small N are indeed sufficient in many information and text processing tasks, while the bound of labels l could be much larger.

5. EXPERIMENTAL EVALUATION

We performed experiments to assess the merits of our approach to query rewriting and its impact on keyword search based on the recently established benchmark [2].

5.1 Evaluation Setting

We compare our approach with an implementation of the state-of-the-art keyword query cleaning solution (BQR) [16]. We use two variants of our approach, one ranks based on the probability of query rewrites (PQR) and the other uses the probability of valid query rewrites ($PVQR$) as discussed in Sec. 3.3 and Sec. 3.4. Both of them integrate the additional heuristics shown in Sec. 3.5. All systems were implemented in Java 1.6 on top of MySQL[1] and Lucene[2]. Experiments were performed on a Linux server with two Intel Xeon 2.8GHz Dual-Core CPUs and 8GB memory. We use all the three sets of data, queries, and relevance assessments available in the benchmark [2]. In the experiments, we use $N = 2$ and $d = 3$, which are sufficient for queries used in the benchmark. We found that the setting of $\eta = 1$, $\beta = 0.33$ and $m = 10$ achieve the best performance. All reported results are based on these values. The effects of these model parameters are discussed in detail in Sec. 5.3.

Data. Table 3 provides the main statistics of the three datasets. IMDb employed in [2] is actually a subset from the original IMDb. Also, a selection of articles from Wikipedia was included in the benchmark, and the PageLinks table was augmented with an additional foreign key to explicitly indicate referenced pages. The Mondial dataset is much smaller, which captures geographical and demographic information from the Web sources such as the CIA World Factbook.

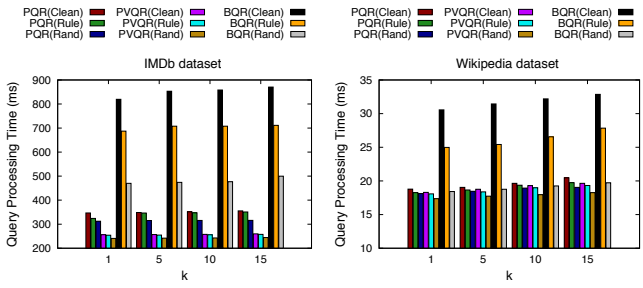
Indexes. Table 3 also reports indexing performance of the three systems w.r.t. index size and indexing time. As shown, the index used by PVQR needs more time and space than the one for PQR, because the former indexes not only n-grams, but also connectivity information. Compared to BQR, PVQR's index is about a factor of 2 larger and the indexing process takes about 4 times longer, which is consistent with our analysis. We also provide a breakdown of the indexing time of PVQR into two parts attributable to join processing and index creation. The elapsed time (join processing + index creation) for indexing Mondial, IMDb and Wikipedia is respectively 0.18(0.06+0.12), 40.8(11.5+29.3) and 8.01(0.47+7.54) minutes. Observe that join processing make up 33%, 28% and 6% of the indexing time for Mondial, IMDb and Wikipedia, respectively. The reason of such difference lies in the graph topology of the datasets, where the structure of Mondial is slightly more complex than that of IMDb, which in turn is much denser than that of Wikipedia.

Queries. For each dataset, 50 queries were proposed [2]. Table 4 provides the statistics of queries and results. Many keywords in these queries can be grouped into segments. While they are suitable for studying the segmentation problem, further token modifications are needed to study token rewriting. From these queries, called *Clean* set, we obtain queries with dirty tokens by rewriting keywords following the same method used in XClean [13], a recent proposal for the token rewriting problem. First, we apply random edit operations, namely insertion, deletion and substitution, to each keyword with length larger than 4 in the Clean queries to obtain the *Rand* set of dirty queries. Second, we make use of the list of common misspellings occurring in Wikipedia[3]. For each Clean query, we replace the keyword that can be found in the list with one of its misspelled forms to obtain the *Rule* set of dirty queries.

[1] http://www.mysql.com

[2] http://lucene.apache.org

[3] http://en.wikipedia.org/wiki/Wikipedia:Lists_of_common_misspellings/For_machines



(a) (b)

Figure 4: Evaluation results for efficiency of query rewriting

5.2 Efficiency of Query Rewriting

Figs. 4(a-b) show the average time for computing top-k query rewrites for IMDb and Wikipedia. Mondial is a very small dataset, where all queries can be rewritten in less than 8 ms on average. For the sake of space, we omit its results because individual times exhibit only minor differences.

Compared to Wikipedia, IMDb contains many more tuples and IMDB queries are longer. This is reflected in the performance results. All systems take substantially more time for IMDb than Wikipedia. The performance of *PVQR is consistently better* than the other two systems for both datasets. PVQR is about *3-4 times* faster than BQR for IMDb and about *2 times* faster for Wikipedia. These differences are primarily due to the pruning capability of PVQR, i.e., PVQR prunes non-valid results. Compared to PQR, the amount of valid sub-query rewrites that have to be kept track of is smaller. The amount of partial rewrites considered by BQR is even much larger than PQR, as it considers all possible combinations of previously obtained segments. It is worth mentioning that Fig. 4(a) excludes the effect of 4 long IMDB queries with length 9, 11, 26, and 11. The reason is that BQR could not finish them within the time limit we set to 1 minute, while PVQR only takes 634 ms, 691 ms, 1657 ms and 746 ms respectively, for Clean queries (and even less for Rule and Rand queries).

We observe that *Clean queries require more time* than Rule queries, which in turn, take more time than Rand queries. This may seem less intuitive for that one would expect processing clean queries should be easier. Clearly, for Clean queries, the list of token rewrites always contains the intended one. These correct token rewrites yield segments, i.e., intermediate results, which have to be processed. For dirty queries, especially Rand, the list of token rewrites may contain no (or not many) correct ones, which cannot be combined to form segments, hence there are no (or fewer) intermediate results to be processed. *More time is needed for segmentation* when there are more intermediate results.

5.3 Effectiveness of Query Rewriting

The ground truth for this experiment can be obtained from the keyword search results captured by the mentioned benchmark. According to the results judged as correct, we add segment boundaries to the Clean queries. These *target queries* and their identified segments constitute the ground truth. This ground truth thus reflects both the quality of token rewriting and query segmentation. We use the standard metric *Mean Reciprocal Rank* (MRR) and an adoption of *Precision at k* ($P@k$). Given a set of keyword queries $\mathcal{Q}$ and the corresponding top-k lists of rewrites, let $\mathcal{Q}^*$ be the queries for which the correct rewrite could be captured by the corresponding top-k list, and for each query $Q_i \in \mathcal{Q}$, let $rank_i$ be the rank of the correct rewrite in the top-k list, then $P@k = \frac{|\mathcal{Q}^*|}{|\mathcal{Q}|}$ and $MRR = \frac{1}{|\mathcal{Q}|}\sum_{Q_i \in \mathcal{Q}} \frac{1}{rank_i}$.

Query Set	$\eta = 0$	1	5	10	15
Mondial(Clean)	0.80	**0.97**	**0.97**	**0.97**	**0.97**
Mondial(Rule)	0.80	**0.97**	**0.97**	**0.97**	**0.97**
Mondial(Rand)	0.86	**0.99**	**0.99**	**0.99**	**0.97**
IMDb(Clean)	0.67	0.82	**0.83**	**0.83**	**0.83**
IMDb(Rule)	0.68	**0.82**	0.81	0.81	0.81
IMDb(Rand)	0.60	**0.77**	0.73	0.72	0.72
Wikipedia(Clean)	0.81	**0.94**	**0.94**	**0.94**	**0.94**
Wikipedia(Rule)	0.79	**0.89**	**0.89**	**0.89**	**0.89**
Wikipedia(Rand)	0.84	**0.93**	0.91	0.91	0.91

Table 5: MRR of PVQR vs. η ($\beta = 0.33$)

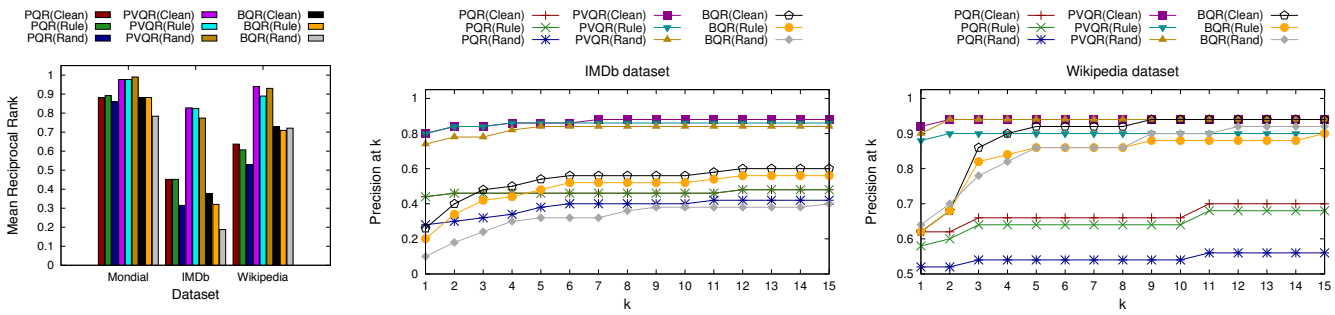
Query Set	$\beta = 0$	0.25	0.33	0.5	1
Mondial(Clean)	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**
Mondial(Rule)	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**
Mondial(Rand)	0.98	**0.99**	**0.99**	**0.99**	**0.99**
IMDb(Clean)	0.74	0.80	**0.82**	**0.82**	0.81
IMDb(Rule)	0.74	0.80	**0.82**	**0.82**	0.81
IMDb(Rand)	0.68	0.74	**0.77**	**0.77**	**0.77**
Wikipedia(Clean)	0.89	**0.94**	**0.94**	**0.94**	0.93
Wikipedia(Rule)	0.85	**0.89**	**0.89**	**0.89**	**0.89**
Wikipedia(Rand)	0.88	0.92	**0.93**	**0.93**	**0.93**

Table 6: MRR of PVQR vs. β ($\eta = 1$)

First, we study the effects of different model parameters on query rewriting. We experimented with different values of η, which reflects the sensitivity to spelling errors and semantic differences (See Eq. 4). The effect of η on MRR values for PVQR is shown in Table 5. The best results are highlighted in bold font. Observe that $\eta = 1$ achieves the best results for almost every query set except Clean queries for IMDb. The MRR values increase quickly from $\eta = 0$ to $\eta = 1$, then reach a plateau. When $\eta > 1$, while the MRRs might increase slightly for clean queries (See IMDb(Clean)), we observe minor decrease for dirty queries (See IMDb(Rule), IMDb(Rand) and WIkipedia(Rand)). This is probably due to the fact that when η is higher, we are stricter with the distance between token rewrites and query keywords. In other words, we prefer the original queries without token rewriting. That has a beneficial effect on clean queries but might bring errors in dirty queries because the misspelled query keywords will be ranked higher. The effect of β, which reflects the sensitivity to the length of segment (See Eq. 11), for PVQR is shown in Table 6. When β is larger, longer segments are preferred. In the experiments, the MRRs improve when β is larger than 0. This means applying this segment length based heuristic yields better results. However, this should not be done too aggressively: the best results are achieved when β reaches 0.33. To study the effect of m, which denotes the number of token rewrites considered for each query keyword, we vary its value from 1 to 15. We observe that the MRR values for all three approaches are highest and most stable when m approaches 10.

Fig. 5(a) illustrates MRR for the three datasets. Similar to the performance results, IMDb constitutes the most difficult case, where MRR is particularly low for PQR and BQR. PVQR achieves the best results for all types of queries over all datasets. On average, *Rand queries yield the lowest* MRR while *Clean queries the highest*. This is expected because in the latter case, it is easier to obtain correct token rewrites, hence more relevant segments can be constructed.

Figs. 5(b-c) illustrate $P@k$ for IMDb and Wikipedia. On average, PVQR also outperforms the other two systems for



(a) (b) (c)

Figure 5: Evaluation results for effectiveness of query rewriting

Query Set	BQR	PVQR/H		PVQR	
	MRR	MRR	Impr.	MRR	Impr.
Mondial(Clean)	0.88	0.97	10.2%	0.97	0.0%
Mondial(Rule)	0.88	0.97	10.2%	0.97	0.0%
Mondial(Rand)	0.78	0.98	25.6%	0.99	1.0%
IMDb(Clean)	0.38	0.74	94.7%	0.82	10.8%
IMDb(Rule)	0.32	0.74	131.3%	0.82	10.8%
IMDb(Rand)	0.19	0.68	257.9%	0.77	13.2%
Wikipedia(Clean)	0.73	0.89	21.9%	0.94	5.6%
Wikipedia(Rule)	0.71	0.85	19.7%	0.89	4.7%
Wikipedia(Rand)	0.72	0.88	22.2%	0.93	5.7%

Table 7: The respective effects of our probabilistic model and additional heuristics on effectiveness of query rewriting

all types of queries. For Wikipedia, BQR achieves good results when k is large, especially for Clean queries. Nevertheless, PVQR is still better than BQR for the same type of queries. Because Mondial is simple and good performance is yielded by all systems ($P@k > 0.7$) and especially PVQR ($P@k > 0.9$), we omit its results for the sake of space.

The best performance achieved by PVQR in all the cases clearly reflects the superiority of PVQR and its usage of the graph data structure. The difference in performance between PVQR and other systems is most evident for IMDb. This is because IMDb contains a much larger data graph than other datasets and thus the graph structure is more crucial for finding the Steiner graphs here.

Furthermore, we investigate the respective contributions of our probabilistic model and the additional heuristics to effectiveness of query rewriting. Table 7 illustrates MRRs for BQR completely based on the ad-hoc heuristics, our probabilistic model (PVQR/H) without the heuristics on top and the default PVQR integrating also the additional heuristics. While the results illustrate a significant improvement achieved by PVQR/H on BQR, especially for IMDb, PVQR improves PVQR/H relatively slightly by adding heuristics. This clearly shows the benefit of using our probabilistic model to effectiveness of query rewriting. In addition, the improvement yielded by the additional heuristics witnesses the adaptability of our approach.

5.4 Impact on Efficiency of Keyword Search

For investigating the impact of query rewriting on keyword search, we employed two keyword search systems: the bidirectional search solution (*BDS*) [8] explores paths between keyword elements online, while the keyword join approach (*KJ*) [9] materializes paths in the index and only join them online. KJ was shown to be faster than BDS but also employs a larger index. Given the three query sets (Clean, Rule, Rand), we use them as they are (*NQR*), rewrite them using PVQR and BQR to obtain 9 types of queries. For queries with rewriting, we use the top-1 as input to the keyword search systems. For reasons of space, we omit the Mondial results and explicitly discuss them in the text only when they are relatively different from the other results.

Figs. 6(a-d) illustrate the average time for processing these 9 types of keyword queries using KJ (Figs. 6(a-b)) and BDS (Figs. 6(c-d)) for IMDb and Wikipedia. Further, the time is decomposed into query rewriting and keyword query processing components, e.g., $QR(Clean)$ is the time needed for rewriting the Clean queries, and $KJ(QR(Clean))$ is the time KJ needs to process these rewritten queries.

The ratio of these two components seems to depend on the complexity of keyword query processing (reflected in the dataset size and query length), and the systems used for that: Clearly, with a slower system (BDS), the fraction of time needed for query rewriting is smaller (compare (c+d) with (a+b)). With higher complexity (IMDb), query rewriting makes up a larger part of the total (compare (b+d) with (a+c)). Meanwhile, we also observe that *with slower systems as well as higher complexity, the positive effect of query rewriting on keyword query processing is also higher* (compare NQR with PVQR and BQR), e.g., the highest reduction in time PVQR and BQR can achieve is for BDS over IMDb. This is because for longer queries, more keywords can be grouped into segments, and with slower system and larger datasets, this *effect of segmentation* is more evident.

Clean queries take more time than Rule queries, which in turn, is more difficult to process than Rand queries. Similar to the effect observed in the query rewriting experiment, this is due to the number of intermediate results, e.g., for Rand query keywords, keyword search systems find fewer matching elements. Accordingly, *query rewriting (PVQR,BQR) leads to reduction in time* especially for Clean and Rule queries, i.e., yields better performance than NQR. In particular, PVQR is about *5-6 times (2-3 times) faster* than NQR for IMDb (Wikipedia). For Rand query, less time is needed in total (compared to Rule and Clean). Hence, there is less room for time reduction through rewriting in this case. Also, the segmentation effect is small here as Rand queries yield fewer correct tokens that can be grouped.

We observe that *PVQR is 2-3 times faster than BQR* for IMDb and is slightly better than or similar to BQR for Wikipedia. Actually, BQR is slightly better than PVQR for many Wikipedia queries. However, this is entirely due to the fact that BQR requires less time for keyword query processing. BQR prefers rare terms, which yield fewer (relevant) keyword elements to be processed. However, the fact that BQR processes fewer (relevant) results is not shown in this experiment, but becomes evident in the following study.

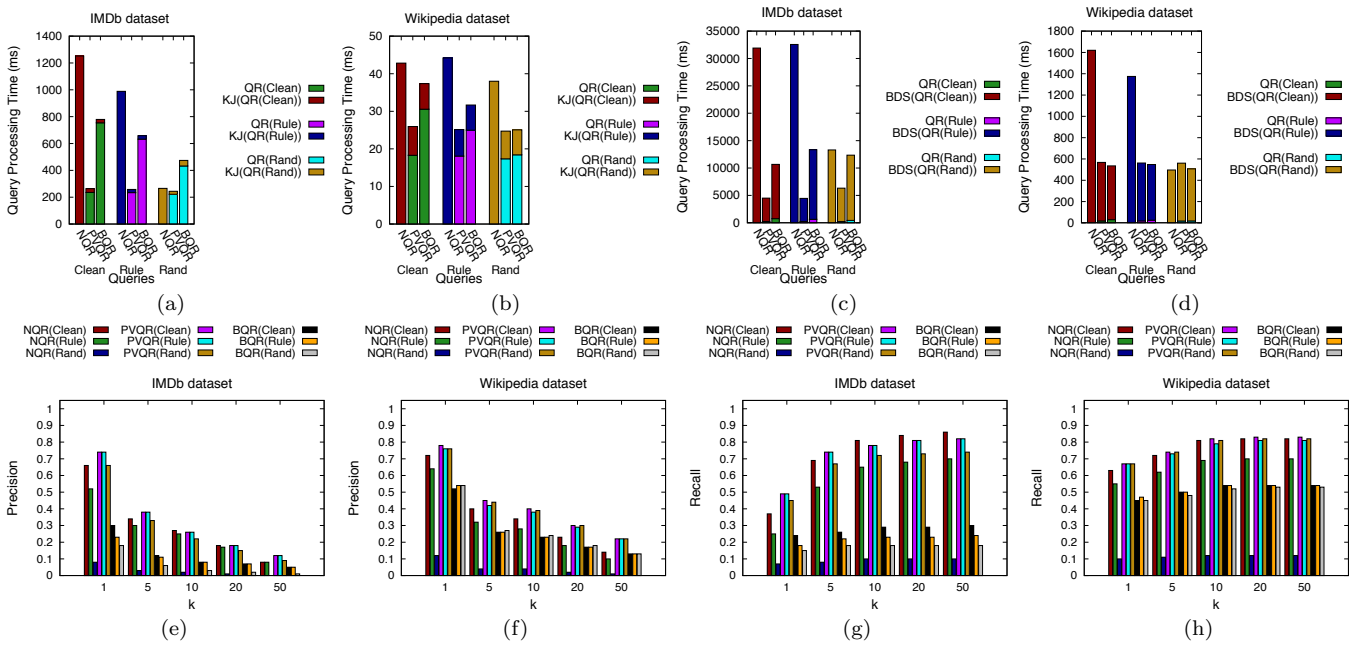


Figure 6: Evaluation results for keyword search

5.5 Impact on Effectiveness of Keyword Search

Both KJ and BSD implement a combination of proximity- and TFIDF-based ranking studied in the benchmark [2]. Since both systems use the same ranking, keyword search answers are very similar, hence we only show results for KJ. We use *Precision* and *Recall* for evaluating keyword search results obtained for the 9 types of queries. Given Q, let R_k be the top-k results and R^* the ground truth results captured by the benchmark. For different values of k, we have $Precision = \frac{|R^* \cap R_k|}{|R_k|}$, and $Recall = \frac{|R^* \cap R_k|}{|R^*|}$.

For different k, Figs. 6(e-f) plot the precision achieved by KJ for the 9 types of queries over IMDb and Wikipedia. As expected, precision consistently decreases with higher k. The queries rewritten by PVQR achieve the best results and the worst results are yielded for BQR queries. Improvements achieved by PVQR over NQR are largest for the dirty queries Rule and Rand (up to *60%* for $k = 1$) and smallest for Clean (up to *10%*). BQR obtains better results than NQR only for Rand queries. Thus, the conclusions are: Higher precision can be obtained for Clean queries compared to dirty queries (with or without rewriting). Rewriting with *PVQR improves precision for all types of queries* while *BQR yields better results only for the most dirty queries (Rand).* Note that these results correspond to the ones from the rewriting experiments, where PVQR produces better rewrites than BQR. Hence, we conclude that *better query rewrites yield higher precision of keyword search results.*

Figs. 6(g-h) show that for recall, similar differences can be observed between the approaches (NQR, PVQR and BQR) and queries (Clean, Rule and Rand) for small values of k. However, while PVQR achieves highest recall for all Wikipedia queries, it performs slightly worst than NQR on Clean IMDb queries when $k \geq 10$. The conclusion is *PVQR improves recall on dirty queries but not on Clean queries* when a large number of results have to be considered.

The relative differences between the approaches and between the queries are the same for the Mondial dataset. However, we note that precision and recall for Mondial are consistently higher than for IMDb and Wikipedia.

5.6 Analysis of Impact of Query Rewriting

In the experiments, we observe that token rewriting helps to find more relevant keyword elements and thus improve the quality of the final keyword search answers for dirty queries. This explains why PVQR achieves significantly higher precision and recall than NQR for Rule and Rand queries. BQR improves NQR only for Rand queries because it yields poor results for retrieving the top-1 query rewrite that we use as input to the keyword search systems.

Existing keyword search systems usually use a threshold to restrict the size of the retrieval list of keyword elements, where relevant ones might be excluded. Here we use the default setting in [9] to retrieve the top-300 matching elements for each keyword. In essence, query segmentation leads to fewer compound keywords. Clearly, due to the higher selectivity of compound keywords yielded by segmentation, it is more likely to have the correct keyword elements. For queries without segmentation, the retrieval list may contain no or fewer correct ones, especially for the common (non-discriminative) keywords. The observation that PVQR obtains better results than NQR even for Clean queries confirms our analysis. The only exception is the recall for Clean IMDb queries when k is large. This is because while query segmentation reduces the search space, it may not preserve all true positives, hence it cannot yield higher recall.

In terms of efficiency, query segmentation has a positive effect because fewer keywords have to be processed. This effect is evident for clean queries, where efficiency improvements can be entirely attributed to query segmentation. While token rewriting improves the quality of the keyword search, it has a negative effect on efficiency. The reason is that clean tokens yield more keyword elements that have to be processed. However, the combined effect of token rewriting and query segmentation on efficiency is still positive, as indicated by improvements obtained for the dirty query sets.

In summary, query rewriting has a clear positive effect on precision of keyword search, while still preserving high recall when the number of results is not too large. Also, it improves efficiency because the positive effect of query segmentation is larger than the negative effect of token rewriting.

6. RELATED WORK

We firstly discuss the previous work that specifically targets token rewriting and query segmentation, and then the related work of query rewriting that tackles both tasks.

Token Rewriting. This problem, a.k.a. spell checking, has attracted interest in the Web context [11, 3]. Syntactic and semantic distances to dictionary words and the context constitute the main feature space. Based on such features, XClean [13] and our approach employ the same error model [15] to estimate the probability of token rewrite. The difference is that while XClean assumes the specific XML type semantics in a semi-structured setting which does not exist in our more general graph setting, our approach takes into account connectivity information to prune token rewrites that do not lead to valid results. Further, XClean only considers the problem of token rewriting (thus, only Sec. 3.2 contains overlaps with XClean).

Query Segmentation. Query segmentation is extensively studied in the Web search setting [17, 1, 7]. In [7], query segmentation is based on mutual information between pairs of query keywords. The work in [1] uses supervised learning to decide whether to create a segment boundary at each keyword position, and [17] proposes an unsupervised method for query segmentation using generative language models. While the use of probabilistic model is not new in the text-centric Web search setting (e.g., [17]), our work is different to the previous work (including [16] for structured data) in that we use connectivity information in the data for focusing on segments that lead to valid results.

Query Rewriting. The most related work [16] first introduces the problem of keyword query rewriting over the relational database. It targets both token rewriting and segmentation based on the ad-hoc heuristics. The subsequent work [4] explores query logs to improve the quality of query rewriting using the same heuristics. In contrast to the existing work, we propose a probabilistic framework to enable the query rewriting problem to be studied in a more principled way. Different from query rewriting, [20] investigates the problem of query reformation to provide totally new queries which are similar or related to the initial one.

7. CONCLUSIONS AND FUTURE WORK

We discuss drawbacks of existing work on query rewriting and propose a principled probabilistic approach to this problem. In the experiments, we show that for query rewriting, our approach is several times faster than the state-of-the-art baseline and also yields higher quality of rewrites especially for large datasets. Most importantly, we show that these improvements also carry over to the actual keyword search. Our approach consistently improves keyword search, i.e., yields several times faster keyword search performance and substantially improves the precision and recall of keyword search results, while the baseline also provides faster performance but compromises on the quality of results, i.e., achieves good results only for very dirty queries.

8. REFERENCES

[1] S. Bergsma and Q. I. Wang. Learning noun phrase query segmentation. In *EMNLP-CoNLL*, pages 819–826, 2007.

[2] J. Coffman and A. C. Weaver. A framework for evaluating database keyword search strategies. In *CIKM*, pages 729–738, 2010.

[3] S. Cucerzan and E. Brill. Spelling correction as an iterative process that exploits the collective knowledge of web users. In *EMNLP*, pages 293–300, 2004.

[4] L. Gao, X. Yu, and Y. Liu. Keyword query cleaning with query logs. In *WAIM*, pages 31–42, 2011.

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

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

[7] R. Jones, B. Rey, O. Madani, and W. Greiner. Generating query substitutions. In *WWW*, pages 387–396, 2006.

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

[9] G. Ladwig and T. Tran. Index structures and top-k join algorithms for native keyword search databases. In *CIKM*, pages 1505–1514, 2011.

[10] G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou. Ease: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *SIGMOD Conference*, pages 903–914, 2008.

[11] M. Li, M. Zhu, Y. Zhang, and M. Zhou. Exploring distributional similarity based models for query spelling correction. In *ACL*, 2006.

[12] F. Liu, C. T. Yu, W. Meng, and A. Chowdhury. Effective keyword search in relational databases. In *SIGMOD Conference*, pages 563–574, 2006.

[13] Y. Lu, W. Wang, J. Li, and C. Liu. Xclean: Providing valid spelling suggestions for xml keyword queries. In *ICDE*, pages 661–672, 2011.

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

[15] E. Mays, F. J. Damerau, and R. L. Mercer. Context based spelling correction. *Inf. Process. Manage.*, 27:517–522, September 1991.

[16] K. Q. Pu and X. Yu. Keyword query cleaning. *PVLDB*, 1(1):909–920, 2008.

[17] B. Tan and F. Peng. Unsupervised query segmentation using generative language models and wikipedia. In *WWW*, pages 347–356, 2008.

[18] T. Tran, H. Wang, S. Rudolph, and P. Cimiano. Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. In *ICDE*, pages 405–416, 2009.

[19] T. Tran and L. Zhang. Keyword query routing. *IEEE Transactions on Knowledge and Data Engineering*, 99(PrePrints):1, 2013.

[20] J. Yao, B. Cui, L. Hua, and Y. Huang. Keyword query reformulation on structured data. In *ICDE*, pages 953–964, 2012.