A Probabilistic Approach to String Transformation

Ziqi Wang, Gu Xu, Hang Li, and Ming Zhang

Abstract—Many problems in natural language processing, data mining, information retrieval, and bioinformatics can be formalized as string transformation, which is a task as follows. Given an input string, the system generates the k most likely output strings corresponding to the input string. This paper proposes a novel and probabilistic approach to string transformation, which is both accurate and efficient. The approach includes the use of a log linear model, a method for training the model, and an algorithm for generating the top k candidates, whether there is or is not a predefined dictionary. The log linear model is defined as a conditional probability distribution of an output string and a rule set for the transformation conditioned on an input string. The learning method employs maximum likelihood estimation for parameter estimation. The string generation algorithm based on pruning is guaranteed to generate the optimal top k candidates. The proposed method is applied to correction of spelling errors in queries as well as reformulation of queries in web search. Experimental results on large scale data show that the proposed approach is very accurate and efficient improving upon existing methods in terms of accuracy and efficiency in different settings.

Index Terms—String Transformation, Log Linear Model, Spelling Error Correction, Query Reformulation

1 Introduction

This paper addresses string transformation, which is an essential problem, in many applications. In natural language processing, pronunciation generation, spelling error correction, word transliteration, and word stemming can all be formalized as string transformation. String transformation can also be used in query reformulation and query suggestion in search. In data mining, string transformation can be employed in the mining of synonyms and database record matching. As many of the above are online applications, the transformation must be conducted not only accurately but also efficiently.

String transformation can be defined in the following way. Given an input string and a set of operators, we are able to transform the input string to the k most likely output strings by applying a number of operators. Here the strings can be strings of words, characters, or any type of tokens. Each operator is a transformation rule that defines the replacement of a substring with another substring. The likelihood of transformation can represent similarity, relevance, and association between two strings in a specific application. Although certain progress has been made, further investigation of the task is still necessary, particularly from the viewpoint of *enhancing both accuracy and efficiency*, which is precisely the goal of this work.

- *Ziqi Wang and Ming Zhang are with School of EECS, Peking University, China.*
  E-mail: wangziqi@pku.edu.cn, mzhang@net.pku.edu.cn
- *Gu Xu is with Microsoft Bing.*
  E-mail: guxu@microsoft.com
- *Hang Li is with Huawei Noah's Ark Lab.*
  E-mail: hangli.hl@huawei.com

String transformation can be conducted at two different settings, depending on whether or not a dictionary is used. When a dictionary is used, the output strings must exist in the given dictionary, while the size of the dictionary can be very large. Without loss of generality, we specifically study correction of spelling errors in queries as well as reformulation of queries in web search in this paper. In the first task, a string consists of characters. In the second task, a string is comprised of words. The former needs to exploit a dictionary while the latter does not.

Correcting spelling errors in queries usually consists of two steps: candidate generation and candidate selection. Candidate generation is used to find the most likely corrections of a misspelled word from the dictionary. In such a case, a string of characters is input and the operators represent insertion, deletion, and substitution of characters with or without surrounding characters, for example, "a"→"e" and "lly"→"ly". Obviously candidate generation is an example of string transformation. Note that candidate generation is concerned with a single word; after candidate generation, the words in the context (i.e., in the query) can be further leveraged to make the final candidate selection, cf., [1], [2].

Query reformulation in search is aimed at dealing with the term mismatch problem. For example, if the query is "NY Times" and the document only contains "New York Times", then the query and document do not match well and the document will not be ranked high. Query reformulation attempts to transform "NY Times" to "New York Times" and thus make a better matching between the query and document. In the task, given a query (a string of words), one needs to generate all similar queries from the original query (strings of words). The operators are transformations between

words in queries such as "tx"→"texas" and "meaning of"→"definition of", cf., [3].

Previous work on string transformation can be categorized into two groups. Some work mainly considered efficient generation of strings, assuming that the model is given [4]. Other work tried to learn the model with different approaches, such as a generative model [5], a logistic regression model [6], and a discriminative model [7]. However, efficiency is not an important factor taken into consideration in these methods.

In contrast, our work in this paper aims to learn a model for string transformation which can achieve both high accuracy and efficiency. There are three fundamental problems with string transformation: (1) how to define a model which can achieve both high accuracy and efficiency, (2) how to accurately and efficiently train the model from training instances, (3) how to efficiently generate the top k output strings given the input string, with or without using a dictionary.

In this paper, we propose a probabilistic approach to the task. Our method is novel and unique in the following aspects. It employs (1) a log-linear (discriminative) model for string transformation, (2) an effective and accurate algorithm for model learning, and (3) an efficient algorithm for string generation.

The log linear model is defined as a conditional probability distribution of an output string and a rule set for the transformation given an input string. The learning method is based on maximum likelihood estimation. Thus, the model is trained toward the objective of generating strings with the largest likelihood given input strings. The generation algorithm efficiently performs the top k candidates generation using top k pruning. It is guaranteed to find the best k candidates without enumerating all the possibilities. An Aho-Corasick tree is employed to index transformation rules in the model. When a dictionary is used in the transformation, a trie is used to efficiently retrieve the strings in the dictionary.

We empirically evaluated our method in spelling error correction of queries and reformulation of queries in web search. The experimental results on the two problems demonstrate that our method consistently and significantly performs better than the baseline methods of generative model and logistic regression model in terms of accuracy and efficiency. We have also applied our method to the Microsoft Speller Challenge and found that our method can achieve a performance comparable to those of the best performing systems in the challenge.

2 Related Work

String transformation has many applications in data mining, natural language processing, information retrieval, and bioinformatics. String transformation has been studied in different specific tasks such as database record matching, spelling error correction, query reformulation and synonym mining. The major difference between our work and the existing work is that we focus on enhancement of both accuracy and efficiency of string transformation.

2.1 Learning for String Transformation

String transformation is about generating one string from another string, such as "TKDE" from "Transactions on Knowledge and Data Engineering". Studies have been conducted on automated learning of a transformation model from data.

Arasu *et al.* [8] proposed a method which can learn a set of transformation rules that explain most of the given examples. Increasing the coverage of the rule set was the primary focus. Tejada *et al.* [9] proposed an active learning method that can estimate the weights of transformation rules with limited user input. The types of the transformation rules are predefined such as stemming, prefix, suffix and acronym. Okazaki *et al.* [6] incorporated rules into an L_1-regularized logistic regression model and utilized the model for string transformation. Dreyer *et al.* [7] also proposed a log-linear model for string transformation, with features representing latent alignments between the input and output strings. Finite-state transducers are employed to generate the candidates. Efficiency is not their main consideration since it is used for offline application. Our model is different from Dreyer *et al.*'s model in several points. Particularly our model is designed for both accurate and efficient string transformation, with transformation rules as features and non-positive values as feature weights.

Okazaki *et al.*'s model is largely different from the model proposed in this paper, although both are discriminative models. Their model is defined as a logistic regression model (classification model) $P(t \mid s)$, where s and t denote input string and output string respectively, and a feature represents a substitution rule

$$f_k(s,t) = \begin{cases} 1 & \text{rule } r_k \text{ can convert } s \text{ to } t \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Their model utilizes all the rules that can convert s to t and it is assumed only one rule can be applied each time.

2.2 Approximate String Search

There are two possible settings for string transformation. One is to generate strings within a dictionary, and the other is to do so without a dictionary. In the former, string transformation becomes approximate string search, which is the problem of identifying strings in a given dictionary that are similar to an input string [10].

In approximate string search, it is usually assumed that the model (similarity or distance) is fixed and the objective is to efficiently find all the strings in the dictionary. Most existing methods attempt to find all the candidates within a fixed range and employ n-gram based algorithms [4], [11], [12], [13], or trie based

algorithm [14]. There are also methods for finding the top k candidates by using n-grams [15], [16]. Efficiency is the major focus for these methods and the similarity functions in them are predefined. In contrast, our work in this paper aims to learn and utilize a similarity function which can achieve both high accuracy and efficiency.

2.3 Spelling Error Correction

Spelling error correction normally consists of candidate generation and candidate selection. The former task is an example of string transformation. Candidate generation is usually only concerned with a single word.

For single-word candidate generation, a rule-based approach is commonly used. The use of edit distance is a typical approach, which exploits operations of character deletion, insertion and substitution. Some methods generate candidates within a fixed range of edit distance or different ranges for strings with different lengths [1], [17]. Other methods learn weighted edit distance to enhance the representation power [18], [19], [20], [21].

Edit distance does not take context information into consideration. For example, people tend to misspell "c" as "s" or "k" depending on context, and a straightforward use of edit distance cannot deal with the case. To address the challenge, some researchers proposed using a large number of substitution rules containing context information (at character level). For example, Brill and Moore [5] developed a generative model including contextual substitution rules. Toutanova and Moore [22] further improved the model by adding pronunciation factors into the model. Duan and Hsu [23] also proposed a generative approach to spelling correction using a noisy channel model. They also considered efficiently generating candidates by using a trie. In this paper, we propose using a discriminative model. Both approaches have pros and cons, and normally a discriminative model can work better than a generative model because it is trained for enhancing accuracy.

Since users' behavior of misspelling and correction can be frequently observed in web search log data, it has been proposed to mine spelling-error and correction pairs by using search log data. The mined pairs can be directly used in spelling error correction. Methods of selecting spelling and correction pairs with a maximum entropy model [24] and similarity functions [25], [26] have been developed. Only high frequency pairs can be found from log data, however. In this paper, we work on candidate generation *at the character level*, which can be applied to spelling error correction for both high and low frequency words.

2.4 Query Reformulation

Query reformulation involves rewriting the original query with its similar queries and enhancing the effectiveness of search. Most existing methods manage to mine transformation rules from pairs of queries in the search logs. One represents an original query and the other represents a similar query (e.g., hotmail sign-on, hotmail sign-up). For example, the method proposed by Jones *et al.* [27] first identifies phrase-based transformation rules from query pairs, and then segments the input query into phrases, and generates a number of candidates based on substitutions of each phrase using the rules. The weights of the transformation rules are calculated based on log likelihood ratio. A query dictionary is used in this case. Wang and Zhai [28] mined contextual substitution patterns and tried to replace the words in the input query by using the patterns. They created a set of candidates that each differ from the input query in one word. The existing methods mainly focused on how to extract useful patterns and rank the candidates with the patterns, while the models for candidate generation are simple. In this paper, we work on query reformulation as an example of string transformation and we employ a more sophisticated model.

3 Model for String Transformation

We propose a probabilistic approach to string transformation that can achieve both high accuracy and efficiency, and is particularly powerful when the scale is large.

In our method, it is assumed that a large number of input string and output string pairs are given as training data. Furthermore, a set of operators for string transformation is provided. A probabilistic model is then obtained from the training data and the operators, which can assign scores to candidates of output strings given an input string. The best candidates are defined as those having the highest probabilistic scores with respect to the training data.

The overview of our method is shown in Fig. 1. There are two processes, learning and generation. In the learning process, rules are first extracted from training string pairs. Then the model of string transformation is constructed using the learning system, consisting of rules and weights. In the generation process, given a new input string, the generation system produces the top k candidates of output string by referring to the model (rules and weights) stored in the rule index.

In our method, the model is a log linear model representing the rules and weights, the learning is driven by maximum likelihood estimation on the training data, and the generation is efficiently conducted with top k pruning. In the next section, we will describe the details of our method.

3.1 Model

The model consists of rules and weights. A rule is formally represented as $\alpha \rightarrow \beta$ which denotes an operation of replacing substring α in the input string with substring β, where $\alpha, \beta \in \{s | s = t, s = \hat{}t, s = t\$$, or $s = \hat{}t\$\}$ and $t \in \Sigma^*$ is the set of possible strings over the alphabet, and $\hat{}$ and \$ are the start and end symbols respectively. For different applications, we can consider

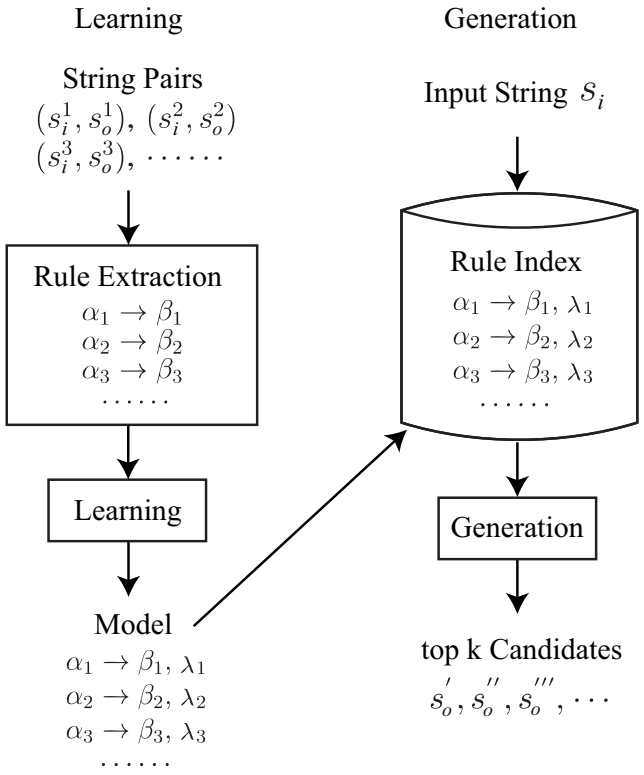

Fig. 1. Overview of Our Method

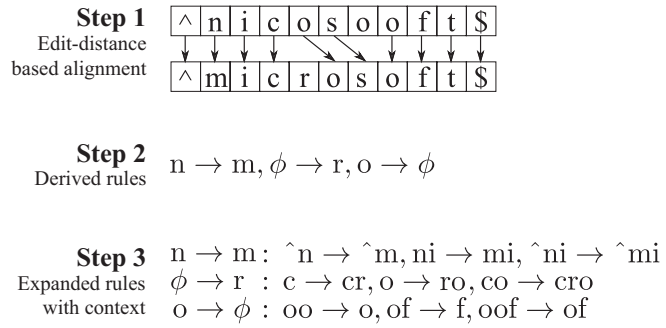

Fig. 2. Example of Rule Extraction

character-level or word-level transformations and thus employ character-level or word-level rules.

All the possible rules are derived from the training data based on string alignment. Fig. 2 shows derivation of character-level rules from character-level alignment. First we align the characters in the input string and the output string based on edit-distance, and then derive rules from the alignment. Next we expand the derived rules with surrounding contexts. Without loss of generality, we only consider expanding $0-2$ characters from left and right sides as contexts in this paper. Derivation of word-level rules can be performed similarly.

If a set of rules can be utilized to transform the input string s_i to the output target string s_o, then the rule set is said to form a "transformation" for the string pair s_i and s_o. Note that for a given string pair, there might be multiple possible transformations. For example, both ("n"→"m","tt"→"t") and ("ni"→"mi","t$"→ $) can transform "nicrosoftt" to "microsoft".

Without loss of generality, we assume that the maximum number of rules applicable to a string pair is predefined. As a result, the number of possible transformations for a string pair is also limited. This is reasonable because the difference between an input string and output string should not be so large. In spelling error correction, for example, the number of possible spelling errors in a word should be rather small.

Let (s_i, s_o) denote a string pair, and $R(s_i, s_o)$ denote a transformation that can rewrite s_i to s_o. We consider that there is a probabilistic mapping between the input string s_i and output string s_o with transformation $R(s_i, s_o)$. We also consider a conditional probability distribution of s_o and $R(s_i, s_o)$ given s_i and take it as model for string transformation. We specifically define the model as the following log linear model:

$$P(s_o, R(s_i, s_o) \mid s_i) \tag{2}$$
$$= \frac{\exp\left(\sum_{r \in R(s_i, s_o)} \lambda_r\right)}{\sum_{(s_t', R(s_i, s_t')) \in \mathcal{Z}(s_i)} \exp\left(\sum_{o \in R(s_i, s_t')} \lambda_o\right)}$$

where r and o denote rules, λ_r and λ_o denote weights, and the normalization is carried over $\mathcal{Z}(s_i)$, all pairs of string s_t' and transformation $R(s_i, s_t')$, such that s_i can be transformed to s_t' by $R(s_i, s_t')$. The log linear model actually uses binary features to indicate whether or not rules are applied.

In general, the weights in Eq. (2) can be any real numbers. To improve generation efficiency, we further assume that all the weights are non-positive, i.e., $\forall \lambda_r \leq 0$. It introduces monotonicity in rule application and implies that applying additional rules cannot lead to generation of better candidates. For example, in spelling error correction, both "office" and "officer" are correct candidates of "ofice". We view "office" a better candidate than "officer" (higher probability), as it needs one less rule. The assumption is reasonable because the chance of making more errors should be lower than that of making fewer errors. Our experimental results have shown that the change in accuracy by making the assumption is negligible, but the gain in efficiency is very significant. This is one of the key ideas of our method.

3.2 Training of Model

Training data is given as a set of pairs $\mathcal{T} = \left\{(s_i^j, s_o^j)\right\}_{j=1}^{N}$, where s_i^j is an input string and s_o^j is an output string. We consider employing maximum likelihood estimation to learn the log linear model.

Ideally we would want to maximize the likelihood $\prod_{j=1}^{N} P(s_o^j, R(s_i^j, s_o^j) \mid s_i^j)$. However, it is often the case that there are multiple transformations $R(s_i^j, s_o^j)$ which can transform the input string to the output string. We do not know which transformation is the "true" one, because it is not given in the training data and it might

also be difficult to derive it automatically from data or have it labeled by humans.

In this paper, we define the likelihood on the basis of conditional probability of output string given input string $\prod_{j=1}^{N} P(s_o^j \mid s_i^j)$. The conditional probability is simply marginalized over all possible transformations.

$$P(s_o \mid s_i) = \sum_{R(s_i,s_o)} P(s_o, R(s_i, s_o) \mid s_i) \tag{3}$$

Therefore, the log likelihood function becomes

$$L(\lambda) = \sum_j \log P(s_o^j \mid s_i^j) \tag{4}$$

This is a common practice in machine learning, in which the likelihood function is defined over observed data (marginalized over hidden data).

Our goal is to find the optimal parameter λ^* of the log linear model by solving the following optimization problem.

$$\begin{aligned} \lambda^* &= \arg\max_{\lambda} L(\lambda) \\ &= \arg\max_{\lambda} \sum_j \log P(s_o^j \mid s_i^j) \end{aligned} \tag{5}$$

where λ denotes the weight parameter.

We employ the Quasi Newton method in the optimization. In this paper, we specifically employ the bounded L-BFGS [29] algorithm as the optimization technique, which works well even when the number of parameters is large.

L-BFGS only needs to compute the gradient of the objective function. The partial derivative with respect to parameter λ_k is calculated as

$$\begin{aligned} \frac{\partial L}{\partial \lambda_k} &= \sum_j \frac{\sum_{R(s_i,s_o)} \exp\left(\sum_{r \in R(s_i,s_o)} \lambda_r\right) \delta(k \in R(s_i, s_o))}{\sum_{R(s_i,s_o)} \exp\left(\sum_{r \in R(s_i,s_o)} \lambda_r\right)} \\ &- \sum_j \frac{\sum_{(s_t', R(s_i,s_t')) \in \mathcal{Z}(s_i)} \exp\left(\sum_{o \in R(s_i,s_t')} \lambda_o\right) \delta(k \in R(s_i, s_t'))}{\sum_{(s_t', R(s_i,s_t')) \in \mathcal{Z}(s_i)} \exp\left(\sum_{o \in R(s_i,s_t')} \lambda_o\right)} \end{aligned} \tag{6}$$

where $\delta(c)$ is the binary function whose value is one if the condition c is true.

3.3 String Generation

In string generation, given an input string s_i, we aim to generate the most likely k output strings s_o that can be transformed from s_i and have the largest probabilities $P(s_o, R(s_i, s_o) \mid s_i)$ assigned by the learned model.

$$\begin{aligned} &\max_{R(s_i,s_o)} P(s_o, R(s_i, s_o) \mid s_i) \\ &= \max_{R(s_i,s_o)} \frac{\exp\left(\sum_{r \in R(s_i,s_o)} \lambda_r\right)}{\sum_{(s_t', R(s_i,s_t')) \in \mathcal{Z}(s_i)} \exp\left(\sum_{o \in R(s_i,s_t')} \lambda_o\right)} \end{aligned} \tag{7}$$

Here we take the maximum of the conditional probabilities $P(s_o, R(s_i, s_o) \mid s_i)$ instead of taking max of the conditional probabilities $P(s_o \mid s_i)$, because in this way we can make the generation process very efficient.

In fact we only need to utilize the following scoring function to rank candidates of output strings s_o given an input string s_i.

$$\text{rank}(s_o \mid s_i) = \max_{R(s_i,s_o)} \left(\sum_{r \in R(s_i,s_o)} \lambda_r \right) \tag{8}$$

For each possible transformation, we simply take summation of the weights of the rules used in the transformation.

4 String Generation Algorithm

In this section, we introduce how to efficiently generate the top k output strings. We employ top k pruning, which can guarantee to find the optimal k output strings. We also exploit two special data structures to facilitate efficient generation. We index the rules with an Aho-Corasick tree. When a dictionary is utilized, we index the dictionary with a trie.

4.1 Rule Index

The rule index stores all the rules and their weights using an Aho-Corasick tree (AC tree) [30], which can make the references of rules very efficient.

The AC tree is a trie with "failure links", on which the Aho-Corasick string matching algorithm can be executed. The Aho-Corasick algorithm is a well-known dictionary-matching algorithm which can quickly locate the elements of a finite set of strings (here strings are the αs in the rules $\alpha \to \beta$) within an input string. The time complexity of the algorithm is of linear order in the length of input string plus the number of matched entries.

We construct an AC tree using all the αs in the rules. Each leaf node corresponds to an α, and the corresponding βs are stored in an associated list in decreasing order of rule weights λ, as illustrated in Fig. 3. One may want to further improve the efficiency by using a trie rather than a ranking list to store the βs associated with the same α. However the improvement would not be significant because the number of βs associated with each α is usually small.

In string generation, given an input string, we first retrieve all the applicable rules and their weights from the AC tree in time complexity of input string length plus number of matched entries.

4.2 Top k Pruning

The string generation problem amounts to that of finding the top k output strings given the input string. Fig. 4 illustrates the lattice structure representing the input and output strings. We assume that the input string s_i

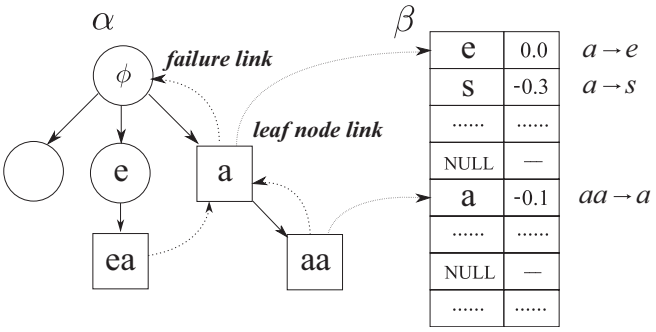

Fig. 3. Rule Index based on Aho Corasick Tree

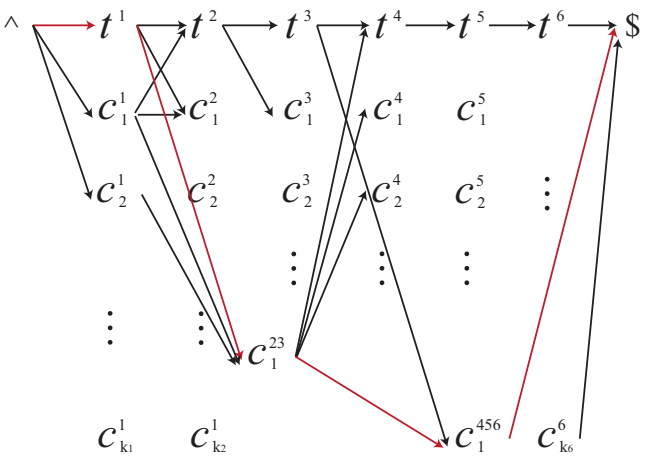

Fig. 4. Lattice Structure for String Generation

Algorithm 1: Top k Pruning

Input: rule index I_r, input string s, candidate number k
Output: top k output strings (candidates) in S_{topk}

1 **begin**
2 Find all rules applicable to s from I_r with Aho-Corasick algorithm
3 $minscore = -\infty$
4 $Q_{path} = S_{topk} = \{\}$
5 Add $(1, \hat{}, 0)$ into Q_{path}
6 **while** Q_{path} *is not empty* **do**
7 Pickup a path $(pos, string, score)$ from Q_{path} with heuristics
8 **if** $score \leq minscore$ **then**
9 continue
10 **if** $pos == |s|$ **AND** $string$ *reaches* \$ **then**
11 **if** $|S_k| \geq k$ **then**
12 Remove candidate with minimum score from S_{topk}
13 Add candidate $(string, score)$ into S_{topk}
14 Update $minscore$ with minimum score in S_{topk}
15 **foreach** *next substring* c *at* pos **do**
16 $\alpha \rightarrow \beta$ = corresponding rule of c
17 $pos' = pos + |\alpha|$
18 $string' = string + \beta$
19 $score' = score + \lambda_{\alpha \rightarrow \beta}$
20 Add $(pos', string', score')$ into Q_{path}
21 **if** $(pos', string', score'')$ in Q_{path} **then**
22 Drop the path with smaller score
23 **return** S_k

consists of tokens $\hat{}t^1t^2 \dots t^n\$$. In the figure, for example, c_1^1 is the token transformed from t^1, and c_1^{456} is the token transformed from $t^4t^5t^6$. In this case, the top k candidates correspond with the top k paths in the lattice. Note that the length of the candidate string can be shorter or longer than the input string.

One could employ a naive algorithm that tries to find the top k candidates from all the possible paths between $\hat{}$ and \$, using the scoring function in Eq. (8). Obviously, this is not tractable when the paths become long.

We employ the top k pruning techniques to efficiently conduct the string generation task. Algo. 1 gives the details. We use triple $(pos, string, score)$ to denote each path generated so far, corresponding to the position, the content and the score of the path. Q_{path} is a priority queue storing paths, and it is initialized with path $(1, \hat{}, 0)$. S_{topk} is a set storing the best k candidates and their scores $(string, score)$ found so far and it is empty at the beginning.

The algorithm picks up one path from the queue Q_{path} each time. It expands the path by following the path from its current position (line 15-20). After one path is processed, another path is popped up from the priority queue with heuristics (line 7).

The algorithm uses the top k pruning strategy to eliminate unlikely paths and thus improve efficiency (line 8-9). If the score of a path is smaller than the minimum score of the top k list S_{topk}, then the path will be discarded and not be used further. This pruning strategy works, because the weights of rules are all non-positive and applying additional rules cannot generate a candidate with higher probability. Therefore, it is not difficult to prove that the best k candidates in terms of the scores can be guaranteed to be found.

The algorithm further discards unlikely paths locally. If two paths have the same pos and $string$, then only the path with a larger score needs to be kept (line 21-22).

The heuristics for picking up a "promising" path from the queue Q_{path} also improve the efficiency (line 7). If the top k list S_{topk} can be constructed more quickly, the pruning strategy can start to work earlier. In this paper, we adopt the following heuristics, which are intuitively reasonable and work well in practice:

1) A path is preferred if no rule is applicable at $pos - 1$ of it.
2) A path is preferred if its pos is larger.
3) A path is preferred if its $score$ is higher.

The heuristics are utilized in their ranked order. Specifically, if there is a rule in Q_{path} satisfying the first heuristic, we will pick it up as the current path. If there are more than one path or there is no path satisfied, we will use the next heuristic, and repeat this process.

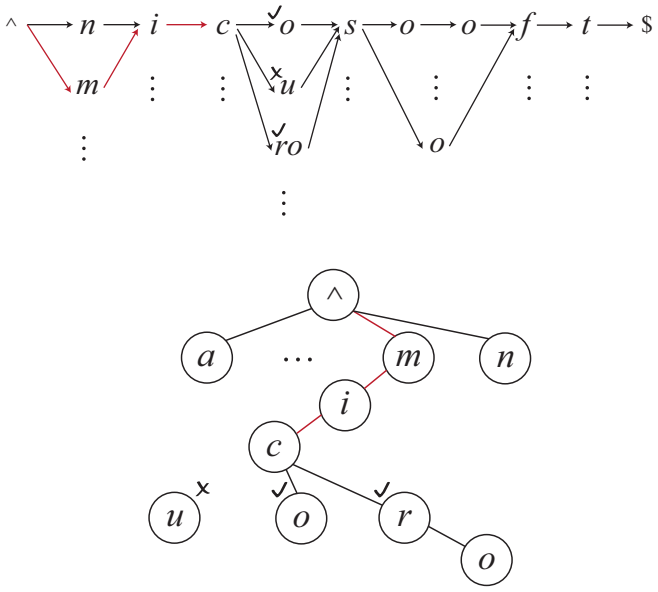

Fig. 5. Dictionary Trie Matching

4.3 Efficient Dictionary Matching Algorithm

Sometimes a dictionary is utilized in string transformation in which the output strings must exist in the dictionary, such as spelling error correction, database record matching, and synonym mining. In the setting of using a dictionary, we can further enhance the efficiency. Specifically, we index the dictionary in a trie, such that each string in the dictionary corresponds to the path from the root node to a leaf node. When we expand a path (substring) in candidate generation, we match it against the trie, and see whether the expansions from it are legitimate paths. If not, we discard the expansions and avoid generating unlikely candidates. In other words, candidate generation is guided by the traversal of the trie.

Fig. 5 gives an example. Suppose that the current path represents string $\hat{}mic$. There are three possible ways to expand it by either continuously matching to o or applying the transformation rules $o \rightarrow u$ and $o \rightarrow ro$. However, node c in the dictionary trie does not have node u as a child node, which means that no string in the dictionary has $\hat{}micu$ as prefix. In such case, the path will not be considered in candidate generation.

5 Experimental Results

We have experimentally evaluated our method to solve two problems, spelling error correction of queries and reformulation of queries in web search. The difference between the two problems is that string transformation is performed at a character level in the former task and at a word level in the latter task. A dictionary is used in the former problem.

5.1 Spelling Error Correction

Efficiency is vital for this task due to the following reasons. (1) The dictionary is extremely large and (2) The response time must be very short.

TABLE 1
Examples of Word Pairs

Misspelled	Correct	Misspelled	Correct
aacoustic	acoustic	chevorle	chevrolet
liyerature	literature	tournemen	tournament
shinngle	shingle	newpape	newspaper
finlad	finland	ccomponet	component
reteive	retrieve	olimpick	olympic

5.1.1 Word Pair Mining

A search session in web search is comprised of a sequence of queries from the same user within a short time period. Many of search sessions in our data consist of misspelled queries and their corrections. We employed heuristics to automatically mine training pairs from search session data at Bing.

First, we segmented the query stream from each user into sessions. If the time period between two queries was more than 5 minutes, then we put a session boundary between them. We used short sessions here because we observed that search users usually correct their misspelled queries very quickly after they find the misspellings. Then the following heuristics were employed to identify pairs of misspelled words and their corrections from two consecutive queries within a session:

1) The two queries have the same number of words.
2) There is only one word difference between the two queries.
3) For the two distinct words, the word in the first query is considered misspelled and the second one its correction.

Finally, we aggregated the identified word pairs across sessions and users and discarded the pairs with low frequencies. Table 1 shows some examples of the mined word pairs.

5.1.2 Experiments on Accuracy

Two representative methods were used as baselines: the generative model proposed by Brill and Moore [5] referred to as *generative* and the logistic regression model (classification model) proposed by Okazaki *et al.* [6] referred to as *logistic*.

We compared our method with the two baselines in terms of top k accuracy, which is the ratio of the true corrections among the top k candidates generated by a method. For each case,

$$Accuracy@k = \begin{cases} 1 & \text{true correction is in the top } k \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

All the methods shared the same default settings: 973,902 words in the dictionary, 10,597 rules for correction, and up to two rules used in one transformation. We made use of 100,000 word pairs mined from query

TABLE 2
Summary of Experiments on Spelling Error Correction

Experiment Settings	Accuracy	Efficiency
Default Settings	Fig. 6	Fig. 12
Dictionary Sizes *(small v.s. large)*	Fig. 7	Fig. 13
Applicable Rules *(2 v.s. 3)*	Fig. 8	Fig. 14
Size of Rule Set *(small v.s. large)*	Fig. 9	Fig. 15

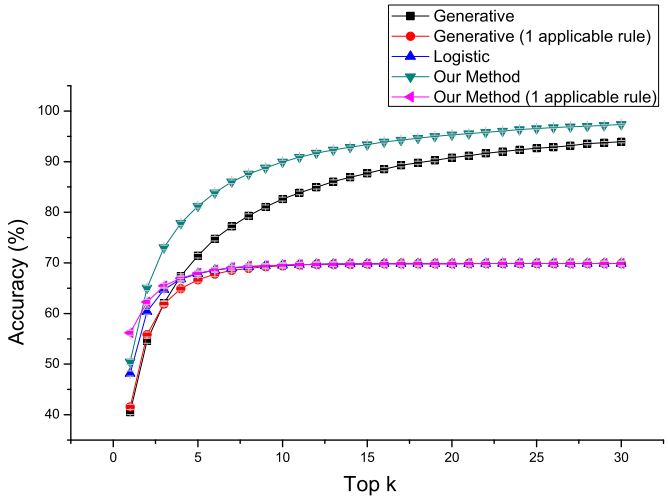

Fig. 6. Accuracy Comparison between Baselines and Our Method with Default Settings

sessions for training, and 10,000 word pairs for testing. A summary of following experiments is shown in Table 2.

The experimental results are shown in Fig. 6. We can see that our method always performs better when compared with the baselines and the improvements are statistically significant ($p < 0.01$ in t-test). The performance of *logistic* becomes saturated when k increases, because the method only allows the use of one rule each time. We observe that there are many word pairs in the data that need to be transformed with multiple rules. We further compared the three methods by only allowing application of one rule. Our method still works better than the baselines, especially when k is small.

Next, we conducted experiments to investigate how the top k accuracy changes with different sizes of dictionary, maximum numbers of applicable rules, and sizes of rule set for the three methods. The experimental results are shown in Fig. 7, Fig. 8 and Fig. 9. For the experiment in Fig. 7, we enlarged the dictionary size from 973,902 (small dict) to 2,206,948 (large dict) and kept the other settings the same as in the previous experiment. The performances of all the methods decline, because more candidates can be generated with a larger dictionary. However, the drop of accuracy by our method is smaller than that by *generative*, which means our method is more powerful when the dictionary is large. For the experiment in Fig. 8, we changed the maximum number of rules that can be applied to a transformation from 2 to 3. *Logistic* is not included in this experiment, because

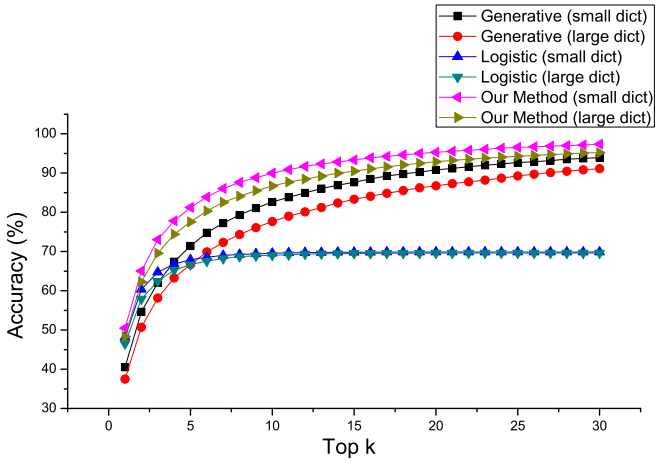

Fig. 7. Accuracy Comparison between Baselines and Our Method with Different Dictionary Sizes

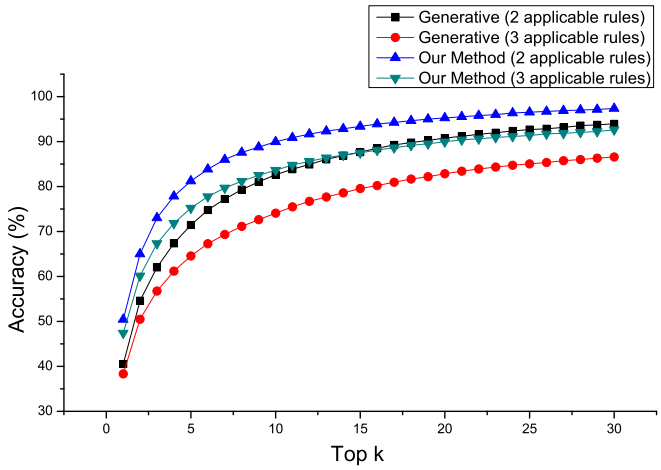

Fig. 8. Accuracy Comparison between Generative and Our Method with Different Maximum Numbers of Applicable Rules

it can only use one rule at a time. When there are more applicable rules, more candidates can be generated and thus ranking of them becomes more challenging. The accuracies of both methods drop, but our method is constantly better than *generative*. Moreover, the decrease in accuracy by our method is clearly less than that by *generative*. For the experiment in Fig. 9, we enlarged the size of the rule set from 10,497 (small rule set) to 24,054 (large rule set). The performance of our method and those of the two baselines do not change so much, and our method still visibly outperforms the baselines when more rules are exploited.

5.1.3 Experiments on Model Constraint

We introduce the non-positive constraint on the parameters, i.e., $\forall \lambda_r \leq 0$, to facilitate our pruning strategy for efficient top k candidate generation. This is also one of the major differences between our method and that of Dreyer *et al.*'s. We experimentally verified the

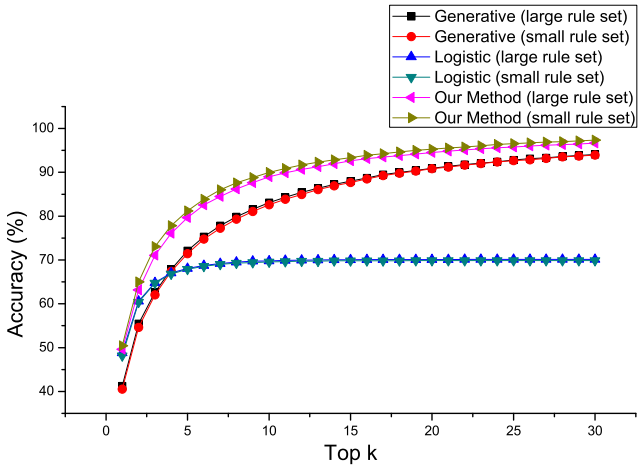

Fig. 9. Accuracy Comparison between Baselines and Our Method with Different Sizes of Rule Set

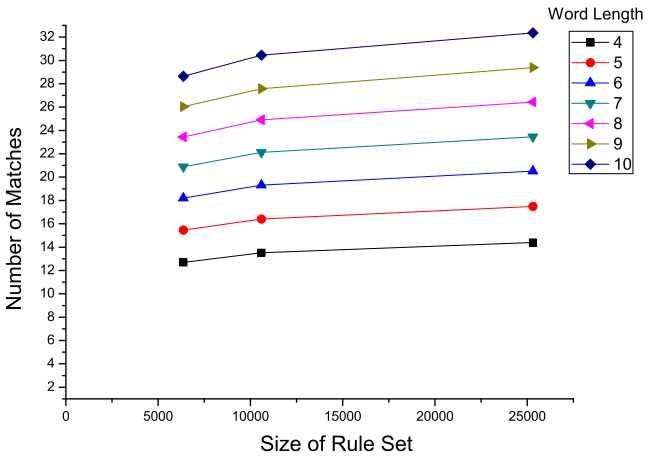

Fig. 11. Efficiency Evaluation on Rule Index

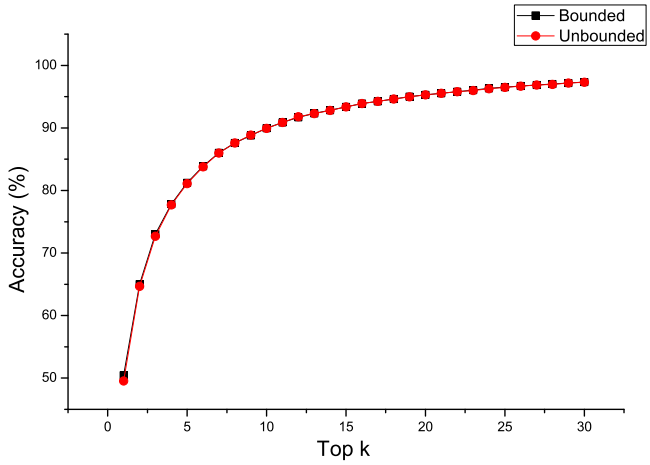

Fig. 10. Accuracy Comparison between Bounded and Unbounded Models

impact of the constraint on the accuracy. For ease of reference, we name the model with the non-positive constraint *bounded*, and the original model *unbounded*. The experimental results on the accuracy are shown in Fig. 10. The experiment was conducted based on the typical setting of our experiments: 973,902 words in the dictionary, 10,597 rules, and up to two rules in one transformation. We can see that the difference between *bounded* and *unbounded* in terms of accuracy is negligible, and we can draw a conclusion that adding the constraint does not hurt the accuracy. Furthermore, as will be discussed in Section 5.1.4, *bounded* is much faster than *unbounded* because of the application of the pruning strategy.

5.1.4 Experiments on Efficiency

We also experimentally evaluated the efficiency of our method, Brill and Moore's method (*generative*), and Okazaki et al.'s method (*logistic*).

First, we tested the effect of using the Aho-Corasick algorithm (the rule index). The time complexity of Aho-Corasick algorithm is determined by the length of query word plus the number of matches. We examined how the number of matches on query word changes when the size of the rule set increases. The experimental result is shown in Fig. 11. We can see that the number of matches is not largely affected when the size of the rule set increases in the rule index. It implies that the time for searching applicable rules is close to a constant and does not change much with different numbers of rules.

Next, we evaluated the efficiency of our method and the baselines in terms of running time. We implemented their methods according to the papers. All the experiments are run on an AMD 2.1GHz CPU with 96GB memory running Windows Server.

The results reported here are those when k is 30, as in Fig. 12. The running time of our method is remarkable less than that of *logistic* and *generative*, especially when the word length is short. When only one rule is applied, the running times of all the three methods are very small. We can conclude that our method is more efficient than the baselines, and the pruning strategy of our method works well.

Next, we tested how the running time of our method changes according to three factors: dictionary size, maximum number of applicable rules in a transformation and rule set size. The experimental results are shown in Fig. 13, Fig. 14 and Fig. 15 respectively. In Fig. 13, with increasing dictionary size, the running time is almost stable, which means our method performs well when the dictionary is large. In Fig. 14, with increasing maximum number of applicable rules in a transformation, the running time increases first and then stabilizes, especially when the word is long. In Fig. 15, the running time keeps growing when the length of words gets longer. However, the running time is still very small, which can meet the requirement of an online application. From all the figures, we can conclude that our pruning strategy is very effective and our method is always efficient especially when the length of query is short.

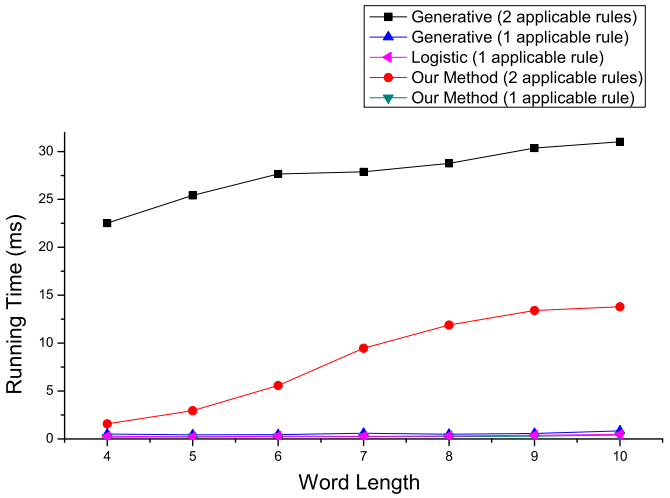

Fig. 12. Efficiency Comparison between Baselines and Our Method with Default Settings

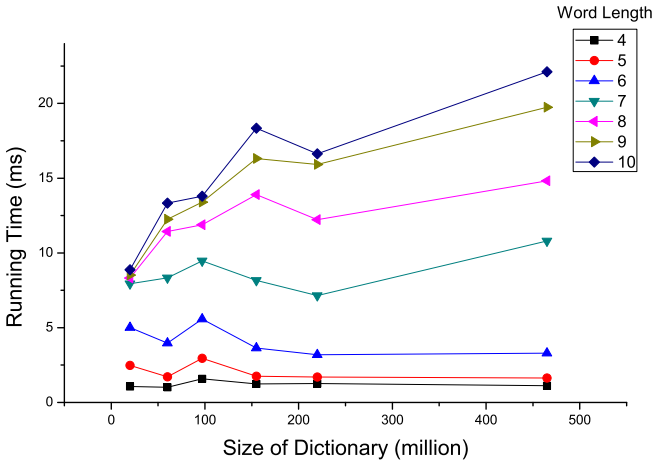

Fig. 13. Efficiency Evaluation with Different Sizes of Dictionary

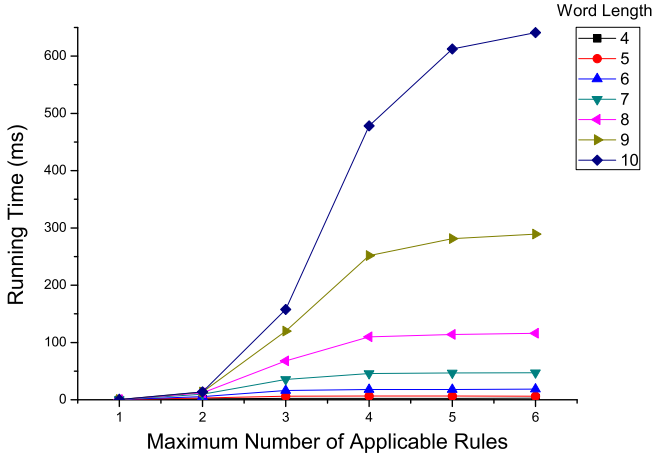

Fig. 14. Efficiency Evaluation with Different Maximum Numbers of Applicable Rules

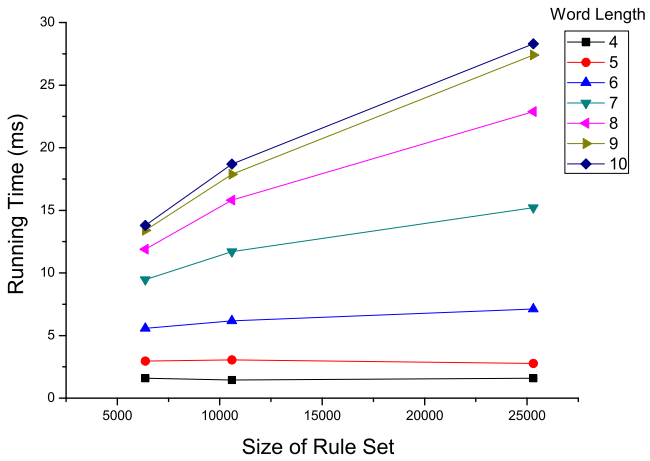

Fig. 15. Efficiency Evaluation with Different Sizes of Rule Set

TABLE 3
Results of Microsoft Speller Challenge

Team with Rank	EF1%
Team 1	86.62
Our Method	85.89
Team 2	85.81
Team 3	85.41
Team 4	85.40
Team 5	85.13

5.2 Microsoft Speller Challenge

The task in the Microsoft Speller Challenge is to build a spelling error correction system that can generate the most plausible candidates for each search query. Submissions to the challenge are evaluated based on the Expected F1 (EF1) measure against a test set created at Bing[1]. We applied our method to generate word candidates and then simply utilized the Microsoft Web N-gram information[2] to rank the final query candidates. The result of our method is shown in Table 3. We also list the results of the best performing systems[3]. The results demonstrate that the performance of our method is comparable to that of the best performing systems. Although our method is a generic algorithm and we did not conduct special tuning, our method still performed well.

5.3 Query Reformulation

In spelling error correction, heuristic is used to mine word pairs. In query reformulation, similar query pairs are used as training. Similar queries can be found from a click-through bipartite graph if they share the same clicked URLs (cf., [31]). Specifically, the Pearson correlation coefficient can be calculated between any two queries on the basis of co-clicks in the bipartite graph.

1. http://web-ngram.research.microsoft.com/spellerchallenge
2. http://web-ngram.research.microsoft.com
3. Information was obtained from the organizers of the challenge.

TABLE 4
Examples of Similar Query Pairs

Similar Query Pairs	
jobs hiring in cedar hill tx	jobs in cedar hill
define load	meaning of load
cheap cars in katy	used cars in katy
city of st francis ks	st francis kansas
1978 boston red sox roster	1978 red sox players
a map of berlin wall	map berlin wall
san francisco ca halloween events	san francisco halloween
word start with t	words that start with a t

TABLE 5
Summary of Experiments on Query Reformulation

Experiment Settings	Accuracy	Efficiency
Default Settings	Fig. 16	Fig. 21
Applicable Rules *(2 v.s. 3)*	Fig. 17	Fig. 22
Size of Rule Set *(small v.s. large)*	Fig. 18	Fig. 23

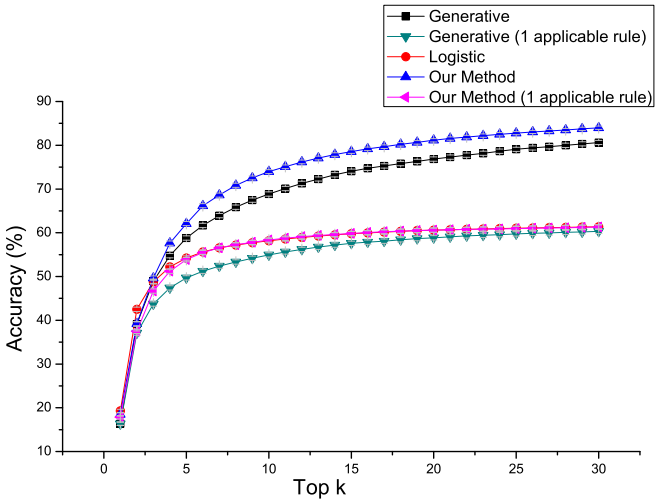

Fig. 16. Accuracy Comparison between Baselines and Our Method with Default Settings

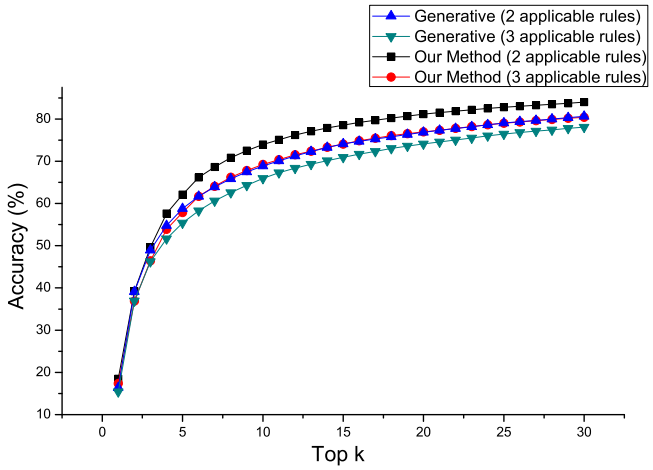

Fig. 17. Accuracy Comparison between Generative and Our Method with Different Maximum Numbers of Applicable Rules

If the coefficient is larger than a threshold, then the two queries are considered similar. This method only works well for head (high frequency) queries. We consider learning a string transformation model trained from head queries and applying the model to tail queries, i.e., conducting top k similar query finding on tail queries. Note that in this case, no dictionary can be used in string transformation, because it is impossible to construct a dictionary of queries in advance in web.

We still take *generative* and *logistic* as baselines, which are extensions of Brill and Moore's model and Okasaki *et al.*'s model to query reformulation. Similar query pairs were mined from search log data at Bing. We made use of 100,000 similar query pairs for training, and 10,000 similar query pairs for testing. Table 4 shows examples of similar query pairs.

Transformation rules were automatically extracted from the training data and there are 55,255 transformation rules. The rules were used for both our method and the baselines, and it was assumed that up to two rules can be used in one transformation. A summary of the experiments is shown in Table 5.

5.3.1 Experiments on Accuracy

We compared our method with the baselines, in terms of top k accuracy. Fig. 16 shows the experimental results. It is clear that our method always outperforms the baselines and the improvements are statistically significant ($p < 0.01$ in t-test).

Fig. 17 and Fig. 18 show how the performances of our method and the baselines change with different maximum numbers of applicable rules and sizes of rule set. In Fig. 17, the maximum number of rules that can be applied to a transformation is changed from 2 to 3. More candidates can be generated in such case, and the generation task becomes harder. The accuracies of both methods drop, but our method is consistentlybetter than *generative*. Since the input query and similar query cannot be so different, the number of applicable rules may not be so large in practice. In Fig. 18, the size of the rule set is increased from 55,255 (small rule set) to 105,609 (large rule set). When there are more rules available, a query can be transformed into more candidates. As a result, the performances of all the methods decrease. Nonetheless our method still consistently and significantly outperforms the baselines.

5.3.2 Experiments on Model Constraint

Again, we tested the accuracy of the model with or without the positivity constraint in our method. We use *bounded* to denote the model with the constraint and *unbounded* to denote the model without the constraint. The experimental result is shown in Fig. 19, while the experiment was conducted in the same setting as before: 55,255 transformation rules and up to two rules used in one transformation. The result indicates

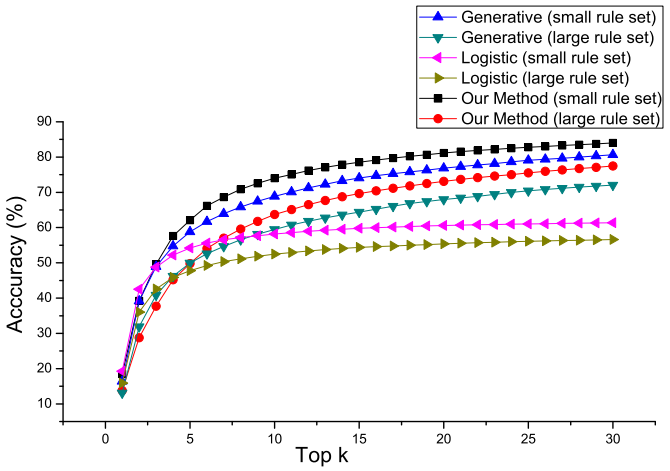

Fig. 18. Accuracy Comparison between Baselines and Our Method with Different Sizes of Rule Set

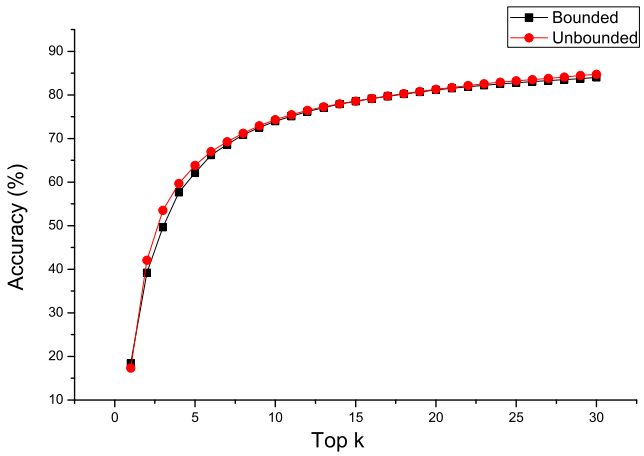

Fig. 19. Accuracy Comparison between Bounded and Unbounded Models

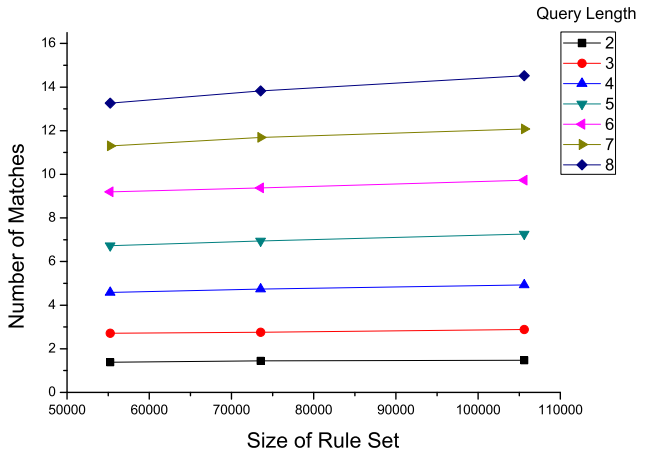

Fig. 20. Efficiency Evaluation on Rule Index

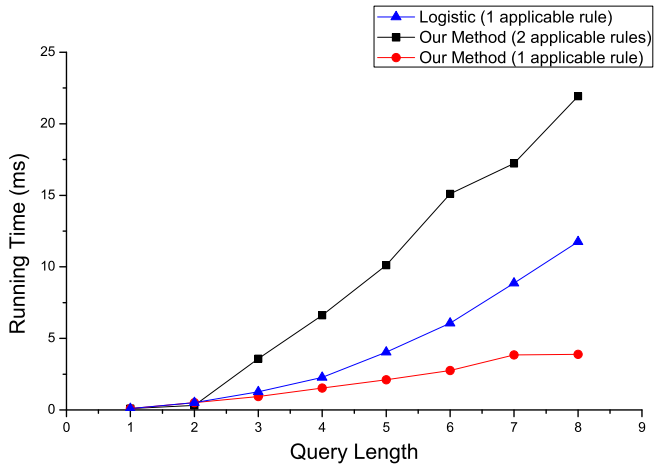

Fig. 21. Efficiency Evaluation between Logistic and Our Method with Default Settings

that the difference between *bounded* and *unbounded* in terms of accuracy is small, which means that adding the constraint does not hurt the accuracy. As will be seen, however, the gain in efficiency is significant.

5.3.3 Experiments on Efficiency

In this experiment, we examined the efficiency of our method and Okasaki *et al.*'s method *logistic* in query reformulation. The retrieval part of Brill and Moore's method is based on a hierarchy of tries, which are constructed using dictionary. However, there is no dictionary in the query reformulation task. Although we can compute the score for candidates, we cannot retrieve candidates using the data structure proposed by Brill and Moore. Thus we did not make comparison with their method in terms of efficiency.

First, we tested the effect of using Aho-Corasick algorithm for rule index, which is shown in Fig. 20. We can observe the advantage of using the Aho-Corasick algorithm, because the number of matches and thus the computation time are basically same with different sizes of rule set.

The running time of our method and *logistic* are shown in Fig. 21, in which k is 30 and only one rule is applied. The running time of *logistic* increases rapidly, while that of our method grows slowly. It indicates that our method which has pruning is very efficient compared to *logistic* which does not have pruning.

Finally, we investigated how the running time of our method changes with different maximum numbers of applicable rules in a transformation and different sizes of rule set. The experimental results are shown in Fig. 22 and Fig. 23 respectively. From Fig. 22, we can see that with increasing maximum numbers of applicable rules, first the running time increases and then stabilizes, especially when the query is long. Since the method's running time is proportional to the number of paths, more paths will be created when more rules are applied. In the meantime, the paths are likely to be pruned because the scores of the paths will also decrease. In Fig. 22, the running times are almost steady for short

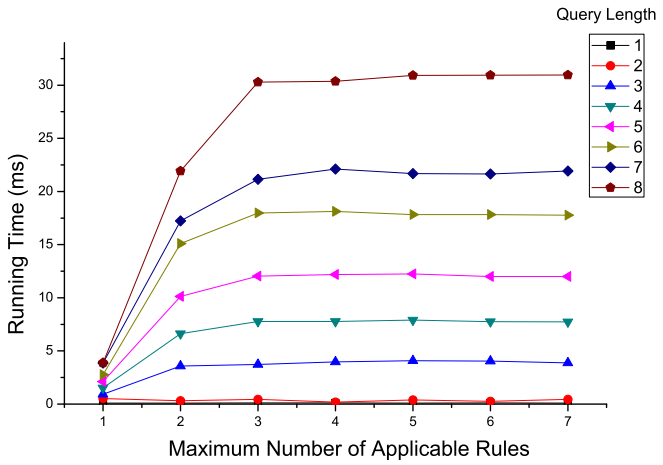

Fig. 22. Efficiency Evaluation with Different Maximum Numbers of Applicable Rules

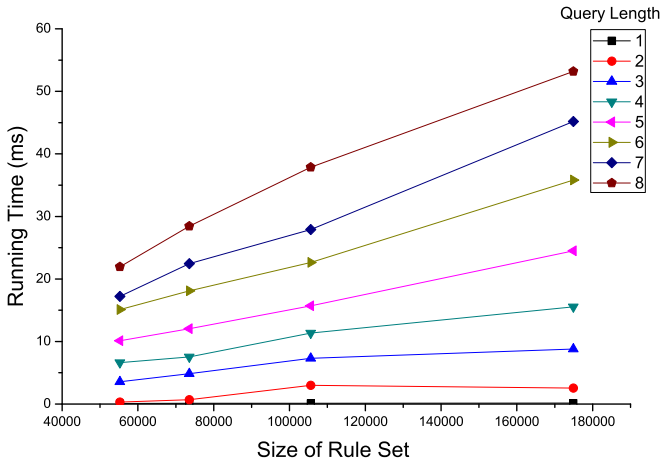

Fig. 23. Efficiency Evaluation with Different Sizes of Rule Set

queries, and the numbers of paths gradually increase for long queries. The effect of pruning in our method is very noticeable, because its running time is significantly smaller.

5.4 Summary of Results

Our method has been applied to two applications, spelling error correction of queries and reformulation of queries in web search. Experiments have been conducted between our method and the baselines including Brill and Moore's method and Okazaki *et al.*'s method. The results show that our method performs consistently better than the baselines in terms of accuracy. Moreover, the accuracy of our method is constantly better than the baselines in different experiment settings, such as size of the rule set, maximum number of applicable rules, and dictionary size. Experiments on efficiency have been conducted with running time as the measure. The running times of our method are smaller than those of the baselines, which indicates that the pruning strategy in our method is very effective. Moreover, the efficiency of our method remains high when the scale becomes large, i.e., larger maximum number of applicable rules, size of rule set, and size of dictionary. In addition, we evaluated our method on the Microsoft Speller Challenge, and the results show that our method is comparable to the best performing systems.

6 Conclusion

In this paper, we have proposed a new statistical learning approach to string transformation. Our method is novel and unique in its model, learning algorithm, and string generation algorithm. Two specific applications are addressed with our method, namely spelling error correction of queries and query reformulation in web search. Experimental results on two large data sets and Microsoft Speller Challenge show that our method improves upon the baselines in terms of accuracy and efficiency. Our method is particularly useful when the -problem occurs on a large scale.

References

[1] M. Li, Y. Zhang, M. Zhu, and M. Zhou, "Exploring distributional similarity based models for query spelling correction," in *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, ser. ACL '06. Morristown, NJ, USA: Association for Computational Linguistics, 2006, pp. 1025–1032.

[2] A. R. Golding and D. Roth, "A winnow-based approach to context-sensitive spelling correction," *Mach. Learn.*, vol. 34, pp. 107–130, February 1999.

[3] J. Guo, G. Xu, H. Li, and X. Cheng, "A unified and discriminative model for query refinement," in *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, ser. SIGIR '08. New York, NY, USA: ACM, 2008, pp. 379–386.

[4] A. Behm, S. Ji, C. Li, and J. Lu, "Space-constrained gram-based indexing for efficient approximate string search," in *Proceedings of the 2009 IEEE International Conference on Data Engineering*, ser. ICDE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 604–615.

[5] E. Brill and R. C. Moore, "An improved error model for noisy channel spelling correction," in *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, ser. ACL '00. Morristown, NJ, USA: Association for Computational Linguistics, 2000, pp. 286–293.

[6] N. Okazaki, Y. Tsuruoka, S. Ananiadou, and J. Tsujii, "A discriminative candidate generator for string transformations," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, ser. EMNLP '08. Morristown, NJ, USA: Association for Computational Linguistics, 2008, pp. 447–456.

[7] M. Dreyer, J. R. Smith, and J. Eisner, "Latent-variable modeling of string transductions with finite-state methods," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, ser. EMNLP '08. Stroudsburg, PA, USA: Association for Computational Linguistics, 2008, pp. 1080–1089.

[8] A. Arasu, S. Chaudhuri, and R. Kaushik, "Learning string transformations from examples," *Proc. VLDB Endow.*, vol. 2, pp. 514–525, August 2009.

[9] S. Tejada, C. A. Knoblock, and S. Minton, "Learning domain-independent string transformation weights for high accuracy object identification," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '02. New York, NY, USA: ACM, 2002, pp. 350–359.

[10] M. Hadjieleftheriou and C. Li, "Efficient approximate search on string collections," *Proc. VLDB Endow.*, vol. 2, pp. 1660–1661, August 2009.

[11] C. Li, B. Wang, and X. Yang, "Vgram: improving performance of approximate queries on string collections using variable-length grams," in *Proceedings of the 33rd international conference on Very large data bases*, ser. VLDB '07. VLDB Endowment, 2007, pp. 303–314.

[12] X. Yang, B. Wang, and C. Li, "Cost-based variable-length-gram selection for string collections to support approximate queries efficiently," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '08. New York, NY, USA: ACM, 2008, pp. 353–364.

[13] C. Li, J. Lu, and Y. Lu, "Efficient merging and filtering algorithms for approximate string searches," in *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, ser. ICDE '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 257–266.

[14] S. Ji, G. Li, C. Li, and J. Feng, "Efficient interactive fuzzy keyword search," in *Proceedings of the 18th international conference on World wide web*, ser. WWW '09. New York, NY, USA: ACM, 2009, pp. 371–380.

[15] R. Vernica and C. Li, "Efficient top-k algorithms for fuzzy search in string collections," in *Proceedings of the First International Workshop on Keyword Search on Structured Data*, ser. KEYS '09. New York, NY, USA: ACM, 2009, pp. 9–14.

[16] Z. Yang, J. Yu, and M. Kitsuregawa, "Fast algorithms for top-k approximate string matching," in *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, ser. AAAI '10, 2010, pp. 1467–1473.

[17] C. Whitelaw, B. Hutchinson, G. Y. Chung, and G. Ellis, "Using the web for language independent spellchecking and autocorrection," in *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, ser. EMNLP '09. Morristown, NJ, USA: Association for Computational Linguistics, 2009, pp. 890–899.

[18] E. S. Ristad and P. N. Yianilos, "Learning string-edit distance," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, pp. 522–532, May 1998.

[19] J. Oncina and M. Sebban, "Learning unbiased stochastic edit distance in the form of a memoryless finite-state transducer," in *In Workshop on Grammatical Inference Applications: Successes and Future Challenges*, 2005.

[20] A. McCallum, K. Bellare, and F. Pereira, "A conditional random field for discriminatively-trained finite-state string edit distance," in *Proceedings of the 21st Conference on Uncertainty in Artifical Intelligence*, ser. UAI '05, 2005, pp. 388–395.

[21] F. Ahmad and G. Kondrak, "Learning a spelling error model from search query logs," in *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, ser. HLT '05. Morristown, NJ, USA: Association for Computational Linguistics, 2005, pp. 955–962.

[22] K. Toutanova and R. C. Moore, "Pronunciation modeling for improved spelling correction," in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ser. ACL '02. Morristown, NJ, USA: Association for Computational Linguistics, 2002, pp. 144–151.

[23] H. Duan and B.-J. P. Hsu, "Online spelling correction for query completion," in *Proceedings of the 20th international conference on World wide web*, ser. WWW '11. New York, NY, USA: ACM, 2011, pp. 117–126.

[24] Q. Chen, M. Li, and M. Zhou, "Improving query spelling correction using web search results," in *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, ser. EMNLP '07, 2007, pp. 181–189.

[25] A. Islam and D. Inkpen, "Real-word spelling correction using google web it 3-grams," in *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, ser. EMNLP '09. Morristown, NJ, USA: Association for Computational Linguistics, 2009, pp. 1241–1249.

[26] ——, "Correcting different types of errors in texts," in *Proceedings of the 24th Canadian conference on Advances in artificial intelligence*, ser. Canadian AI '11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 192–203.

[27] R. Jones, B. Rey, O. Madani, and W. Greiner, "Generating query substitutions," in *Proceedings of the 15th international conference on World Wide Web*, ser. WWW '06. New York, NY, USA: ACM, 2006, pp. 387–396.

[28] X. Wang and C. Zhai, "Mining term association patterns from search logs for effective query reformulation," in *Proceeding of the 17th ACM conference on Information and knowledge management*, ser. CIKM '08. New York, NY, USA: ACM, 2008, pp. 479–488.

[29] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," *SIAM J. Sci. Comput.*, vol. 16, pp. 1190–1208, September 1995.

[30] A. V. Aho and M. J. Corasick, "Efficient string matching: an aid to bibliographic search," *Commun. ACM*, vol. 18, pp. 333–340, June 1975.

[31] J. Xu and G. Xu, "Learning similarity function for rare queries," in *Proceedings of the fourth ACM international conference on Web search and data mining*, ser. WSDM '11. New York, NY, USA: ACM, 2011, pp. 615–624.

Ziqi Wang Ziqi Wang is currently working toward the PhD degree in the Institute of Network Computing and Information, Peking University. She received the BS degree from the Department of Computer Science, Peking University, in 2010. Her research interests include information retrieval, natural language processing and recommender system.

Gu Xu Gu Xu is a applied researcher in Microsoft Bing. Prior to this, he was a researcher in the Information Retrieval and Mining Group at Microsoft Research Asia. He joined Microsoft in 2003 and formerly worked in Multimedia Computing Group. His research interests include web information retrieval, mining and statistical machine learning. He graduated from Tsinghua University in 2003. Gu's recent research is on web query analysis, including query reformulation, query similarity learning and query recommendation. The results of this research were published in major academic conferences including SIGIR, WSDM and CIKM. His recent academic activities include serving as a program committee member of SIGIR'11, WSDM'11, ACL'12 and AAAI'12.

Hang Li Hang Li is chief scientist of the Noahs Ark Lab at Huawei. He is also adjunct professor of Peking University and Nanjing University. His research areas include information retrieval, natural language processing, statistical machine learning, and data mining. He graduated from Kyoto University in 1988 and earned his PhD from the University of Tokyo in 1998. He worked at the NEC lab in Japan during 1991 and 2001, and Microsoft Research Asia during 2001 and 2012. He joined Huawei Technologies in 2012. Hang has about 100 publications at top international journals and conferences, including SIGIR, WWW, WSDM, ACL, EMNLP, ICML, NIPS, and SIGKDD.

Ming Zhang Ming Zhang received the Bachelor and PhD degrees in Computer Science from Peking University in 1988 and 2005, respectively. She is a full professor at the School of Electronics Engineering and Computer Science, Peking University. Prof. Zhang is a member of the Advisory committee of Computing Education, the Ministry of Education in China. Her research interests include digital librari es, text mining and social computing. Prof. Zhang is the leading author of several textbooks on Data Structures and Algorithms in Chinese, and the corresponding course is awarded as the National Elaborate Course by MOE China.