Jeyalakshmi.S *et al*, International Journal of Computer Science and Mobile Applications,
Vol.2 Issue. 3, March- 2014, pg. 55-65 **ISSN: 2321-8363**

IMPROVING EFFICIENCY AND ACCURACY IN STRING TRANSFORMATION ON LARGE DATA SETS

Jeyalakshmi.S[1], Rathika.T[2]
[1]M.Tech final year, Dept. of CSE, SRM University, Ramapuram, jeyalakshmibetch@gmail.com
[2]A.P (O.G), Dept. of CSE, SRM University, Ramapuram, rathika.t@rmp.srmuniv.ac.in

Abstract

This paper discusses the problems in information processing on data mining, information retrieval, and bioinformatics can be put forwarded to string transformation. The *k* most likely output strings are generated corresponding to the given input string for string transformation. It proposes a probabilistic approach such as log linear model-a training method and algorithm for generating top k candidates to string transformation. 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 maximum likelihood parameter estimation is employed for learning method. The optimal top *k* candidates are generated using this string generation algorithm and commentz walter algorithm. Correction of spelling errors in queries as well as reformulation of queries in web search is made using our proposed method. 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.

Keywords: Log Linear Model, Parameter Estimation, Query Reformulation, Spelling Error Correction, String Transformation, commentz walter algorithm

1. Introduction

String transformation can be formulated to natural language processing, pronunciation generation, spelling error correction, word transliteration, and word stemming. String transformation can be defined as given an input string and a set of operators, one can 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.
String transformation can be performed in two different ways, depending on whether or not a dictionary is used.
In the first method, a string consists of characters. In the second method, a string is comprised of words. The former needs the help of a dictionary while the latter does not. Spelling errors is of two steps. 1) Non word errors and 2) Real word errors. Non word errors are those words not in dictionary. For ex: graffe→giraffe, acress. Fig 2. Shows candidate correction through edit distance for the word acress. Real word errors are those that are in dictionary. For ex: three→there, piece→peace, two→too. Spelling errors in queries can be corrected in two steps: (1) Candidate generation and (2) Candidate selection. Fig 1.shows the spelling error correction in word processing. Candidate generation is used to find the words with similar spelling, from the dictionary. In a case, a string of characters is input and the operators represent insertion, deletion, and substitution of characters with or without surrounding characters. They are done by using small edit distance to error. Candidate generation is concerned with a single word; after candidate generation, the words in the context can be further leveraged to make the final candidate selection,[1], [2]. Query reformulation in search is aimed at dealing with the term mismatch problem. For example, if the query is "MCH" and the document only contains "MC Hospital", then the query and document do not match well and the document will not be ranked high.

Query reformulation attempts to transform "MCH" to "Medical College Hospital" and thus make a better matching between the query and document. In the task, a query has been (string of words), system needs to generate all similar queries from the original query (strings of words). The operators are transformations between words in queries such as "ex"→"example" and "carrying"→"holding" [3].

Previous work on string transformation can be categorized into two groups. Some task 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]. 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 train the model accurately and efficiently 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 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. To find the best *k* candidates pruning is guaranteed without enumerating all the possibilities.

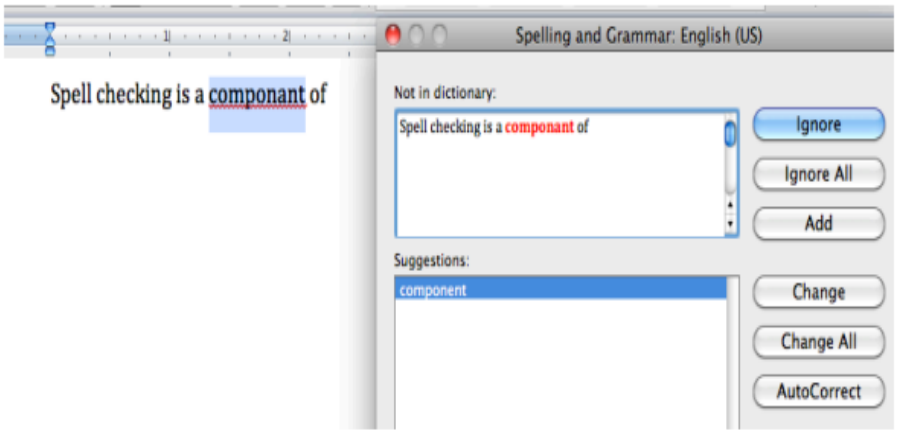


Fig 1.Spell checking in word processing

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 improve efficient to retrieve the strings from the dictionary. We empirically evaluated our method in spelling error correction of queries and reformulation of queries in web search.

Error	Candidate Correction	Correct Letter	Error Letter	Type
acress	actress	t	–	deletion
acress	cress	–	a	insertion
acress	caress	ca	ac	transposition
acress	access	c	r	substitution
acress	across	o	e	substitution
acress	acres	–	s	insertion
acress	acres	–	s	insertion

Fig 2.Words within one of acress

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.

2. Related Work

There are several papers dealing with the information processing. But the major difference between our work and the existing work is that we focus on enhancement of both accuracy and efficiency of string transformation.

Dreyer [7] also proposed a loglinear model for string transformation, with features representing latent alignments between the input and output strings. Tejada [9] proposed an active learning method that can estimate the weights of transformation rules with limited user input. Arasu [8] proposed a method which can learn a set of transformation rules that explain most of the given examples. There are also methods for finding the top *k* candidates by using *n*-g rams [10], [11]. Wang and Zhai [14] mined contextual substitution patterns and tried to replace the words in the input query by using the patterns. Brill and Moore [5] developed a generative model including contextual substitution rules. Toutanova and Moore [12] further improved the model by adding pronunciation factors into the model. Duan and Hsu [13] also proposed a generative approach to spelling correction using a noisy channel model.

3. String Transformation Model

The overview of our method is shown in Fig. 3. There are two processes, they are learning and generation. In the learning process, rules are first extracted from training pairs of string. Then the model of string transformation is constructed using the learning system, deals with rules and weights. In the generation process, given a new input string, produces the top *k* candidates of output string by referring to the model (rules and weights) stored in the rule index.

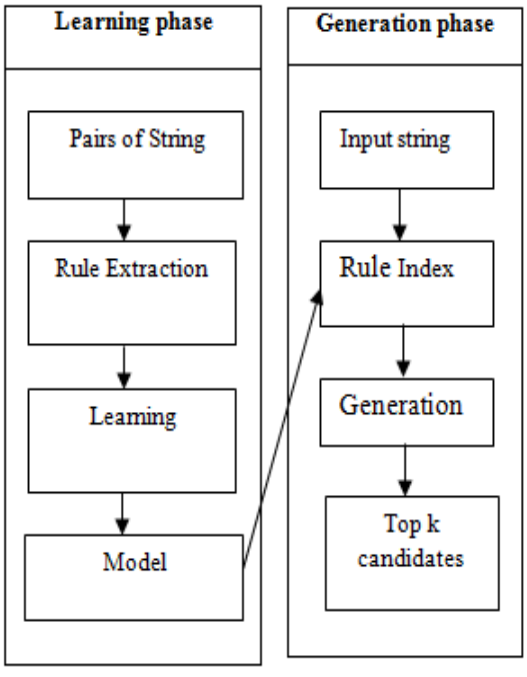
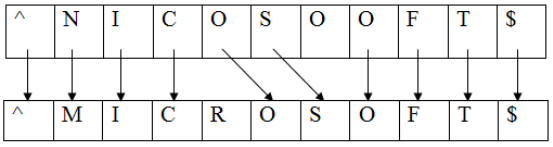


Fig 3.Overview of string transformation

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 $\alpha, \beta \in \{ s|s = t, s = ^t, s = t\$, or s =$

^t$ } and t € Σ^* is the set of possible strings over the alphabet, and ^ and $ are the start and end symbols respectively.

Step 1: Edit-distance based alignment



Step 2: Rules derived

N → M , φ→R , O→ φ

Step 3: Context expanded rules

N→M: ^N→^M, NI→MI , ^NI→ ^MI

φ→R, C→CR, O→RO, CO→CRO

O→ φ, OO→O, OF→F, OOF→OF

Fig 4. Rule Extraction example

All the possible rules are derived from the training data based on string alignment. Fig. 4 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 derives rules from the alignment.

4. Log Linear Model

A log-linear model consists of the following components:

- A set X of possible inputs.
- A set Y of possible labels. The set Y is assumed to be finite.
- A positive integer d specifies the number of features and parameters in the model.
- A function f : X * Y → R^d that maps any (x, y) pair to a feature-vector f(x, y).
- A parameter vector v € R^d.

For any x € X, y € Y, the model defines a conditional probability

$$p(y|x;v) = \frac{\exp\ (v.f(x,y))}{\Sigma_{y' \in y} \exp\ (v.f(x,y'))}$$

Here $\exp\ (x) = e^x$, and $v.f(x,y) = \Sigma_{k=1}^{d} v_k f_k\,(x,y)$

is the inner product between $f(x,y)$ and v. The term $p(y|x;v)$ is intended to be read as "the probability of y conditioned on x, in the parameter values v".

We now describe the components of the model in more detail, first focusing on the feature-vector definitions $f(x,y)$, then giving intuition behind the model form

$$p(y|x;v) = \frac{\exp\ (v.f(x,y))}{\Sigma_{y' \in y} \exp\ (v.f(x,y'))}$$

The models can be represented by a set of expected frequencies that may or may not resemble the observed frequencies. The following model is referring to the traditional chi-square test where two variables, each with two levels (2 x 2 table), to be evaluated to see if an association exists between the variables.

$$Ln(F_{ij}) = \mu + \lambda_i^A + \lambda_j^B + \lambda_{ij}^{AB}$$

$Ln(F_{ij})$ = is the log, expected cell frequencies of the cases for cell ij in the contingency table.
μ = is the whole mean of the natural log of the expected frequencies
λ = terms each represent "effects" which the variables have on the cell frequencies
A and B = the variables
i and j = refer to the categories within the variables
Therefore:
λ_i^A = the main effect for variable A
λ_j^B = the main effect for variable B
λ_{ij}^{AB} = is the interaction effects for variables B and A

The above model is considered a Saturated Model because it includes all possible one way and two-way effects. The saturated model has the same amount of cells in the contingency table as it does effects as given, the expected cell frequencies will always exactly match the observed frequencies, with no degrees of freedom remaining. For example, in a 2 x 2 table there are four cells and in a saturated model involving two variables there are four effects, $\mu, \lambda_i^A, \lambda_j^B, \lambda_{ij}^{AB}$, therefore the expected cell frequencies will exactly match the observed frequencies. In order to find a more parsimonious model that will isolate the effects best demonstrating the data patterns, must be sought as an a non-saturated model. This can be achieved by setting the parameters as zero to some of the effect parameters . For instance, if we set the effects parameter λij AB to zero (i.e. we assume that variable A has no effect on variable B and also B has no effect on variable A. we are left with the unsaturated model.

$$Ln(F_{ij}) = \mu + \lambda_i^A + \lambda_j^B$$

This particular unsaturated model is titled as the Independence Model because it lacks an interaction effect parameter between A and B. This model holds that the variables are unassociated, implicity. Note that the independence model is analogous to the chi-square analysis and testing should be the hypothesis of independence.

5.String Generation Algorithm

Top k pruning algorithm is used to efficiently generate the optimal k output string. Aho–Corasick string matching algorithm is a string searching algorithms. It is otherwise known as dictionary-matching algorithm that locates the elements of a finite set of strings (the "dictionary") within an input text. Which matches all the patterns simultaneously and the complexity of the algorithm is linear in the length of the patterns plus the length of the searched text plus the number of output matches. The string generation problem amounts to that of finding the top k output strings given the input string.

Algorithm1:(For Reference)
Top k Pruning

Input: Input string s, Rule index I_r, candidate key k
Output: top k output strings in s_{topk}
1. **begin**
2. Find all rules applicable to s from L_r with Aho-Corasick algorithm
3. minscore = -∞
4. Q_{path}=S_{topk}={}
5. Add (1, ^, 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 ≤ 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. for each next substring c at pos do
16. $\alpha \rightarrow \beta$ = corresponding rule of c
17. pos' = pos + $|\alpha|$
18. string' = string + β
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

We employ the top *k* pruning techniques to efficiently conduct the string generation task. Alg.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, ^, 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 nonpositive 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).

6. String Matching Algorithm

A) Knuth-Morris-Pratt Algorithm

The Knuth-Morris-Pratt algorithm is based on finite automata but uses a simpler method of handling the situation of when the characters don't match. In the Knuth-Morris-Pratt algorithm, we label the states with the symbol that should match at that point. We then only need two links from each state, one link for a successful match and the other link for a failure. The success link will take us to the next node in the chain, and the failure link will take us back to a previous node based on the word pattern. Which provide each success link of a Knuth-Morris-Pratt automata causes the "fetch" of a new character from the text. Failure links do not get a new character but reuse the last character that can be fetched. If we reach the final state, we know that we found the substring.

B) Boyer-Moore Algorithm

The Boyer-Moore algorithm is different from the previous two algorithms in that it matches the pattern from the right end instead of left . For example, in the following example, we first compare the y with the r and find a mismatch character. Because r doesn't appear in the pattern at all, we know the pattern can be moved to the right a full four characters (the size of the pattern). We next compare the y with the h and find a mismatch. This time because the h does appear in the pattern, we have to move the pattern only two characters to the right so that the h characters line up and then we begin the match from the right side and find a complete match for the pattern.

		Comparisons
Text 1:	there they are	
Pass 1:	they	1
Text 2:	there they are	
Pass 2:	they	1
Text 3:	there they are	
Pass 3:	they	4

In the Boyer-Moore algorithm, we have done 6 character comparisons versus 13 in the standard algorithm.

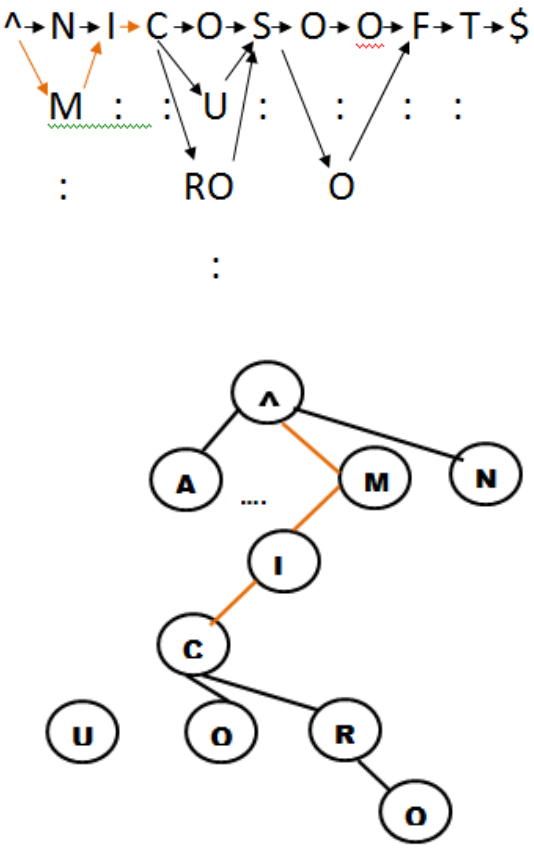


Fig 5. Dictionary Trie Matching

C) 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 that should

present in dictionary . In the setting of using a dictionary, we can further enhance the efficiency. Specifically, we index that the dictionary is 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 and candidate generation is guided by the traversal of the trie. Fig. 5 gives an example. Suppose that the current path represents string ^MIC. There are three possible ways to expand it by either continuously matching to O or applying the transformation rules O → U and O → 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 ^MICU as prefix. In such case, the path will not be considered in candidate generation.

D) Commentz – Walter String Matching Algorithm

Commentz-Walter algorithm is the combination of the Boyer- Moore technique with the Aho-Corasick algorithm,So this algorithm provide more accuracy and efficiency in string transformation. In pre-processing stage, it differs from Aho-Corasick algorithm, Commentz-Walter algorithm builds a converse state machine from the patterns to be matched. Each pattern to be matched and adds states to the machine,starting from right side and going to the first character of the pattern, and combining the same node. In searching stage, Commentz-Walter algorithm Which uses the idea of BM algorithm. The length of matching window is the minimum pattern length. In matching window, Commentz-Walter that scans the characters of the pattern from right to left beginning with the rightmost one. In the case of a mismatch (or a complete match of the whole pattern) it uses a pre-computed shift table to shift the window to the right.

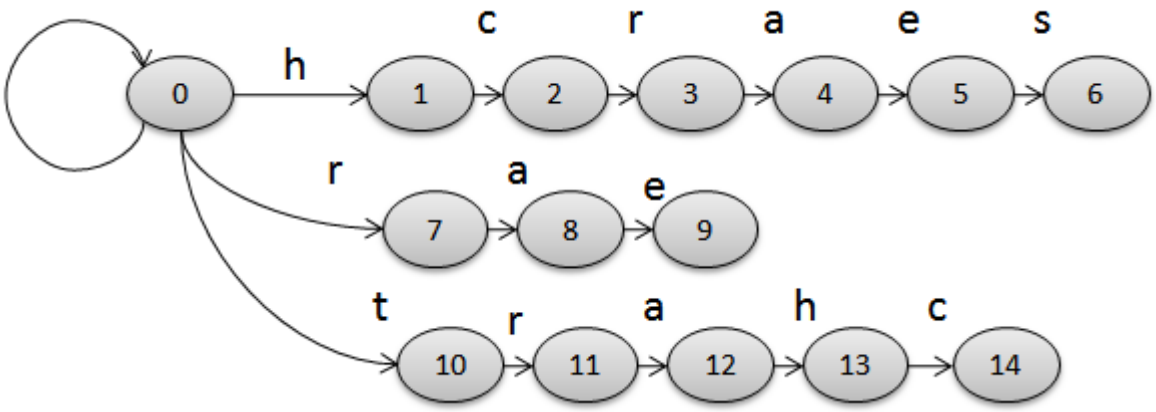


Fig 6.Commentz Walter Example For pattern set { search, ear, arch, chart },

Commentz-Walter Algorithm for search phase

Initial phase:
B ← root r(b is the "present" node of T)
j ← wmin(j points of the document letter above the node of depth)
k ← (k indicates the depth of the present node b)
while j ≤ length document do

Scan phase:
Begin
While there is some son B of B
Labeled by d_{j-k} do
Begin
B ← B^1
K ←k+ 1
Output:(w,j) for each w of out(B)
End

Shift phase:
Begin
j ← j +s(B, d_{j-k})
k ← 0
end
end.
Where s(B, d_{j-k}) is the length of the shift defined by
S(B, d_{j-k})=min(max(shift1(B),
Char($d_{j\,k}$)-k-1),
Shift 2 (B)),

7. Experimental Results

Next, we tested how to reduce 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.

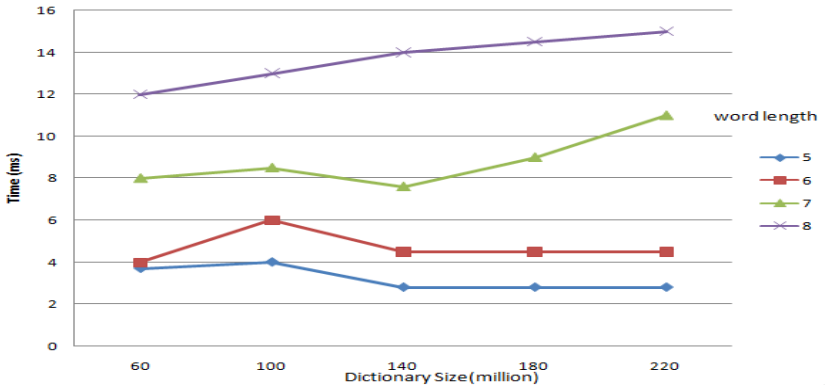


Fig 7. Efficiency evaluation with different sizes of dictionary

In Fig.7, with increasing dictionary size, the running time is almost stable, which means our method performs well when the dictionary is large. In Fig. 8, with increasing maximum number of applicable rules in a transformation, the running time increases first and then stabilizes, especially when the word is long.

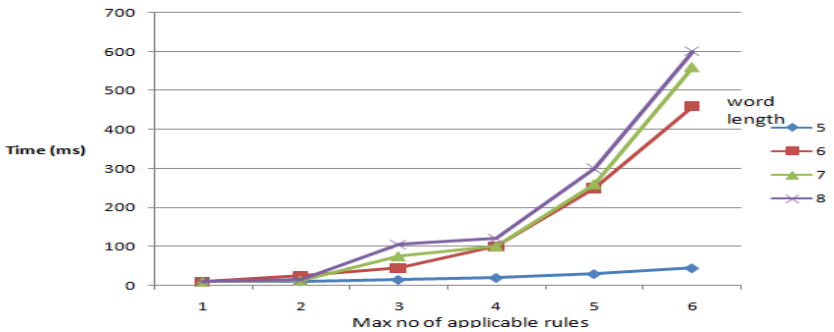


Fig 8. Efficiency evaluation with maximum no of applicable rules

In Fig. 9, 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.

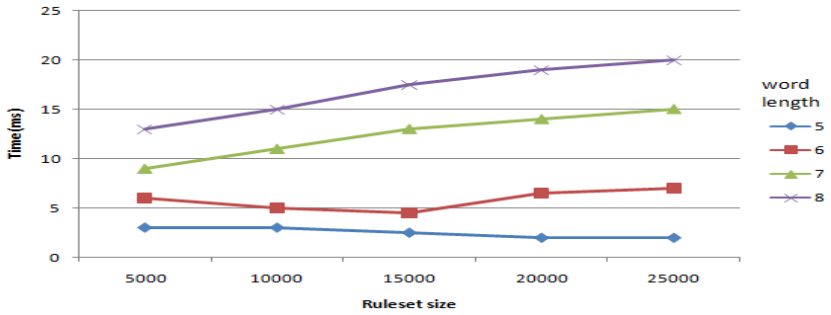


Fig.9 Efficiency evaluation with different sizes of rule set

8. Conclusion

Thus our work reduces the problem with information processing by making use of a new statistical learning approach to string transformation. This method is novel and unique in its model, learning algorithm, string generation algorithm and commentz walter algorithm. The commentz walter algorithm provides more accuracy and efficiency in Specific applications such as spelling error correction and query reformulation in web queries were addressed with this method. Experimental results on two large data sets show that our method improves upon the baselines in terms of accuracy and efficiency in string transformation. Our method is particularly more useful when the problem occurs on a large scale datasets.

REFERENCES

[1] M. Li, Y. Zhang, M. Zhu, and M. Zhou, "Exploring distributional similarity based models for query spelling correction," inProceedings 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," inProceedings 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. 604–610.

[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. 146–293.

[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," inProceedings 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 domainindependent string transformation weights for high accuracy object identification," inProceedings 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] 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.

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

[12] 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. 144–101.

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

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

