Use of Prefix Trees in Text Error Correction Problem

Dr. Girijamma H A
Professor, CSE
RNS Institute of Technology,
Bangalore, India
girijakasal@gmail.com

Santosh Pattar
8th sem, CSE
RNS Institute of Technology,
Bangalore, India
santoshpattar01@gmail.com

Ashish B T
8th sem, CSE
RNS Institute of Technology,
Bangalore, India

Karthik M N
8th sem, CSE
RNS Institute of Technology,
Bangalore, India

Abstract— A deformed fuzzy automaton can be used to calculate similarity value between strings having a non-limited number of edition errors. In this paper an algorithm is presented that makes use of prefix tree to implement the deformed fuzzy automata. Threshold values for the fuzzy transition functions are also used to calculate the membership values which improve the efficiency of the proposed algorithm.

Keywords- Deformed fuzzy automata, prefix tree, leftmost-child right-sibling representation, t-norm and t-conorm operators.

I. Introduction

In text error correction problems the edition errors are to be dealt with, these include insertion, deletion and substitution errors. In pattern recognition, syntactic and structural methods [4] consist in the representation of patterns as strings of primitive objects (symbols), and each pattern classes a set of such strings. Once such a representation is defined, recognizers for these strings are developed, normally based on the theory of formal languages and automata. Each pattern class constitutes a language generated by a grammar whose members are recognized by an automaton. In practice, a problem to be solved occurs when the string to be classified does not coincide exactly with any one of the given patterns. This fact can be due to errors in the data source (imperfect pattern), errors in the processing previous to the classification (i.e., segmentation errors), or even errors in the previous classification of symbols in the string. Irrespectively of their origin, all these errors are abstracted in the so-called edition errors: substitution, deletion, and insertion of symbols.

A fuzzy automaton allows to compute the similarity between two strings i.e. the *observed string* and the *pattern string* [1].

- *Observed string*: Input string obtained from user.
- *Pattern string*: String found in dictionary.

At first stage a finite deterministic automaton is defined that accepts the pattern string. Then, such an automaton is modified in order to include all the possible edit operations which allow matching every observed string into the pattern string.

The modified automaton is a fuzzy automaton where the states are fuzzy sets defined over a universe of states, and transitions are defined by appropriate fuzzy operations among the fuzzy states.

Let Σ be the finite set of symbols (alphabets) and Σ^* be the set of all strings over Σ. Let $\alpha \in \Sigma^*$ and $\omega \in \Sigma^*$, be two arbitrary strings.

In [1] an algorithm is introduced that computes the fuzzy automata as deformed system. If there are Q states in the fuzzy automata then the membership values of edition errors are computed as follows:

$$\tilde{\mu}(\tilde{P},\tilde{y}) = \{(p,\mu) | \mu \oplus_{q\in Q} (\oplus_{x\in\Sigma}(\mu(q,p,x) \otimes \mu_{\tilde{y}}(x)) \otimes \mu_{\tilde{P}}(q)), p \in Q\},\ \ \tilde{P} \text{ in } F(Q), \tilde{y} \text{ in } F(\Sigma) \qquad (1)$$

$$\mu^{\varepsilon}(\widetilde{P}) = \{(p,\mu) |\ \mu = \mu_{\tilde{P}}(P) \oplus (\oplus_{q\in Q} (\mu_{\tilde{E}_q}(p) \otimes \mu_{\tilde{P}}(q))), p \in Q\} \text{ with } \tilde{P} \text{ in } F(Q) \qquad (2)$$

Where $\tilde{\mu}$ is the transition function, p is the new fuzzy state reached from the initial state $\tilde{P}$ on consuming the symbol x with the fuzzy symbol set $\tilde{y}$. $F(Q)$ represents the possible fuzzy sets over Q. $\widetilde{E}(Q)$ represents fuzzy set in $F(Q)$ representing the set of reachable states from q by repeatedly using transitions by empty string. The operators $\oplus$ and $\otimes$ denote a *t*-conorm and *t*-norm respectively.

In the context of spell checking which makes use of dictionaries, such an algorithm can't be used due to efficiency criteria. In section 2 the adaptation of this algorithm is given that is suitable for classification of words in a dictionary.

A tree can be used to represent a collection of words in a way that makes it quite efficient to check whether a given sequence of characters is a valid word. In this type of tree, called a *trie*[2], each node except the root has an associated letter. The string of characters represented by a node *n* is the sequence of letters along the path from the root to *n*. Given a set of words, the tree consists of nodes for exactly those strings of characters that are prefixes of some word in the set. The label of a node consists of the letter represented by the node and also a boolean indicating whether or not the string from the root to that node forms a complete word.

II. Method

We use a prefix tree to represent the dictionary $D \subseteq \Sigma^+$. The set of prefixes is defined as $\text{Pre}(D) = \{\vartheta \in \Sigma^* | \exists\xi \in \Sigma^*: \vartheta\xi \in D\}$. The *l*-length prefix set of strings set $D \subseteq \Sigma^+$ is defined as $\text{Pre}_l\ (D) = \{\vartheta \in \Sigma^* || \vartheta| = l, \exists\xi \in \Sigma^*: \vartheta\xi \in D\}$. Given a dictionary D a prefix tree is a graph $T_D = (N_D, V_D)$, where N_D is the set of nodes and V_D is the set of arcs. It can be built in the following way.

- $N_D \equiv \text{Pre}(D)$ where $\langle\xi\rangle \in N_D \Leftrightarrow \xi \in \text{Pre}(D)$.
- $(\langle\vartheta\rangle, a, \langle\xi\rangle) \in V_D \Leftrightarrow \vartheta a = \xi, being\ \vartheta, \xi$ being $, \xi \in \text{Pre}(D)$and $a \in \Sigma$. Node$\langle\xi\rangle$is called a *child* of node$\langle\vartheta\rangle$while node $\langle\vartheta\rangle$ is called the *parent* of node$\langle\xi\rangle$.
- $\langle\varepsilon\rangle$is the *root* node.
- For every string $\xi \in D$ there is a node$\langle\xi\rangle \in N_D$ that will be called node *associated* with string ξ (and *vice versa: string associated*).

If we define a total order in Σ, it is possible to univocally represent a prefix tree as a binary tree in Leftmost-Child Right-Sibling representation of a tree [2] by sorting the children of a node as indicated by its total order. We will denote byB_D the binary tree representing the prefix tree $T_D$built from the set of strings D.

Thresholds used:

Thresholds for the symbols and state membership are used to modify the algorithm presented in [1] to improve its efficiency.

- *Threshold for states:* It is possible to define a parameter $\lambda \in [0,1]$ such that:
  1) if $Q(q_i) \leq \lambda$then *C*$_1$:=0 (the computation of $(\oplus_{x\in\Sigma}(\mu_{d^{q_{i_x}}} \otimes \mu_{\tilde{y}_k}(x)))$ is avoided);
  2) if $Q(q_{i-1}) \leq \lambda$then *C*$_2$:=0 (the computation of $(\oplus_{x\in\Sigma}(\mu_{c^{q_{i-1}q^{i}}_{xa}} \otimes \mu_{\tilde{y}_k}(x)))$ is avoided);
- *Threshold for fuzzy symbols*: If fuzzy symbols are provided as lists of symbols sorted by their membership values, it is possible to define a parameter, $h \in N$, that limits the number of symbols to be taken into account. Only the first *h* symbols of each fuzzy symbol (a sorted list) will be used in the computation. We use the notation $\oplus_{x\in\Sigma<h}$ to indicate that only the first *h* symbols of the sorted fuzzy symbol are used.

```
Input: $B_D$, the lists $\mu_{d_a^\beta}$,$\mu_{c_{ax}^\beta}$, $and\ \mu_{i_x^\beta}$, $\forall a,\ x\ \in \Sigma,\ \forall \beta\ \in$ Pre(D)
        $\alpha$ observed string, length m ($x_m$ k-th symbol of $\alpha$).
Output: C and $\omega$, C has the value $\mu_{L(\mathrm{M\,D(\omega)})}(\alpha)$
        Where MD($\omega$) = (Q, $\Sigma, \mu, \sigma, \eta$) is the deformed fuzzy automata.

Algorithm tree_computation( )
{
    tree_initialstate( );
    $\forall k$: 1 ….$m$
    {
        rtransition( $B_D$, 0, $k$ );
        rclosure( nl( $B_D$ ), root( $B_D$ ).val);
    }
    tree_decision( );
}
Procedure tree_initialstate( )
{
    root( $B_D$ ).val = 1;
    rinitiation ( nl( $B_D$ )):
    rclosure( nl( $B_D$ ),root( A ).val);
}
Procedure rinitiation( A )
{
   If not null( A )
   {
       root( A ).val=0;
       rinitiation( nl( A )) ;
       rinitiation( nr( A ));
   }
}
Procedure tree_decision()
{
   $\langle C, \omega\rangle$=$\langle 0, \varepsilon\rangle$;
   rdecision( $B_D$ );
}

Procedure rclosure( A, val )
{
   if not null(A)
   {
       root(A).val = max( root(A).val, val$\otimes\mu_{i_x^\beta}$));
                  where x=root(A).trn,
                  and $\beta$=root(A). $\omega$
       rclosure(nl(A), root(A).val);
       rclosure(nr(A), val);
   }
}
Procedure rtransition( A, val, k )
{
   if not null(A)
   {
       rtransition( nl(A), root(A).val, k );
       rtransition( nr(A), val, k );
      if root(A).val>$\lambda$
      {
          $C_1$= root(A).val $\otimes(\bigoplus_{a\in\Sigma<h}(\mu_{d_a^\beta}\otimes,\mu_{x_k}(a)))$;
```

```
            }
            Else $C_1$=0;
            If val>$\lambda$
            {
                $C_2 = \text{val}\otimes(\oplus_{a\in\Sigma<h}\ (\mu_{c_{az}^{\beta}}\otimes,\mu_{x_k}(a)));$
            }
            Else $C_2$=0;
                where x=root(A).trn , and $\beta = root(A).\omega$
            root(A).val := $C_1 \oplus C_2$;
        }
        Procedure rdecision( A )
        {
            if not null(A)
            {
                rdecision(nl(A));
                rdecision(nr(A));
                if root(A).end
                {
                    If root(A).val> C
                    {
                        C = root(A).val;
                        $\omega$ = root(A). $\omega$;
                    }
                }
            }
        }
```

letT be the subtree of B_D,the function root(T)is defined as follows:

- Root(T). ω: prefix associated with the root node of T;
- Root(T).trn: last symbol of the associated prefix or ε for the root of B_D ;
- Root(T).val: membership value of the root node of T into the computed fuzzy state;
- Root(T).end: true if the prefix associated with the root node of T is a pattern of D.

nl(T) is the left subtree of T.
nr(T) is the right subtree of T
null(T)true if the tree T is null.

`

Figure 1 Algorithm that computes deformed fuzzy automata using binary tree.

Fig. 1 shows the algorithm that computes deformed fuzzy automaton for all patterns of *D*. The main program is given by the algorithm tree_computation( ). Procedure tree_initialstate( ) invokes the recursive procedure rinitiation( ). This procedure traverses the tree, setting to zero the membership value of every node except the root node whose membership value is set to 1. Finally, tree_initialstate( ) computes the ε-closure of the fuzzy state by calling the procedure rclosure( ). Procedure tree_decision( ) traverses the tree using the recursive procedure rdecision( ). This procedure looks for the highest membership value (given by *C*) and its associate string (given by ω) among the nodes associated to strings of *D*. Procedures rtransition( ) and rclosure( ) implement (1) and (2) extended to the prefix tree. Procedure rtransition( ) traverses the tree in *postorder*, so the computation of a new node does not affect to the computation of its children nodes Finally, rclosure( ) traverses the tree in *preorder* so the new value is used to compute the new value for children.

Given a dictionary *D* with different strings, the number of fuzzy transitions needed by the algorithm of the Fig. 1 for processing an observed fuzzy string of *m* symbols is

$$\sum_{j=1}^{L} 2 \text{ x } m \text{ x } |Pre_j(D)| \text{ x } h$$

Where *L* being the length of the largest string of *D* and *h* the threshold for fuzzy symbols. Additionally, the utilization of the threshold for states may lead to a reduction of the average time needed to compute each transition.

III. CONCLUSION

The algorithm presented in this paper is capable of finding, from a dictionary of patterns, the more similar pattern to an input fuzzy string. This algorithm has been defined taking into account some efficiency criteria. The dictionary is represented by means of a binary tree reducing the memory complexity. The computation time maybe reduced considering thresholds when computing the fuzzy states. Also the use of just one automaton for representing all the pattern strings is achieved by using a prefix tree making possible to process all strings at once.

IV. REFERENCES

[1] Gonzalez de Mendivil, Fuzzy automata for imperfect string matching , preceding of ESTYLF 2000.
[2] A. V. Aho and J. D. Ullman, Foundations of Computer Science. New York, Computer Science Press, 1992.
[3] Dr. V. Ramaswamy, Girijamma.H. A., Characterization of Fuzzy Regular Languages, International Journal of Computer Science and Network Security, VOL 8, No. 12, December 2008.
[4] K. S. Fu, Syntactic Methods in Pattern Recognition. New York: Academic, 1974.
[5] J. Echanobe, J. R. González de Mendívil, J. R. Garitagoitia, and C. F. "Deformed systems for contextual postprocessing," Fuzzy Sets Syst., vol. 96, pp. 335–341, 1998.
[6] J. R. González de Mendívil, J. R. Garitagoitia, J. J. Astrain, and J. Echanobe, "Fuzzy automata for imperfect string matching," in Proc. Estylf2000,Sevilla, Spain, Sept. 2000, pp. 141–145.
[7] J. Hopcroft and J. Ullman, Introduction to Automata Theory, Languages and Computation. Reading, MA: Addison-Wesley, 1979.
[8] L. A. Zadeh et al., Fuzzy Sets and Their Applications to Cognitive and Decision Process. New York: Academic, 1975.

V. AUTHORS PROFILE

Dr. Girijamma H A obtained her Ph.D from Kuvempu University, she is professor in Department of Computer Science and Engineering, RNS Institute of Technology Bangalore. Her areas of interest are theory of computation, fuzzy logic and its applications, embedded systems and compiler design.

Santosh pattar is a final year student in the Department of Computer Science and Engineering at RNS Institute of Technology. His areas of interest being data mining, bioinformatics and fuzzy automata and languages.

Ashish B T is a final year student in the Department of Computer Science and Engineering at RNS Institute of Technology. His areas of interest being DBMS, operating systems and formal languages and automata theory.

Karthik M N is a final year student in the Department of Computer Science and Engineering at RNS Institute of Technology. His areas of interest being data structures and fuzzy logic.