Fuzzified Aho-Corasick Search Automata

Zdeněk Horák, Václav Snášel, Ajith Abraham
Department of Computer Science, FEECS
VŠB - Technical University of Ostrava
Ostrava, Czech Republic
zdenek.horak@vsb.cz, vaclav.snasel@vsb.cz, ajith.abraham@ieee.org

Aboul Ella Hassanien
Cairo University
Faculty of Computer and Information Technology,
Giza, Egypt
aboitcairo@gmail.com

***Abstract*—In this paper, we discuss the need for efficient approximate string matching. We present the well-known Aho-Corasick automaton for locating multiple patterns and discuss an approach for fuzzification of this automaton. Along with some motivational examples, we propose and illustrate a novel algorithm for automaton construction.**

I. Introduction

When constructing search algorithms, we often need to solve the problem of approximate searching. These constructions can also be extended by a weighted function, as described by Muthukrishnan [7].

Approximate string matching and search is not a new problem and has been solved many times. It is usually based on Aho-Corasick automata and trellis constructions, and is often used when working with text documents or databases, or antivirus software.

We begin our paper with some motivational examples, showing that there are several situations, which would use this kind of search procedures. Then we define a fuzzy automaton, and some basic constructions. We continue with the construction of Fuzzy Aho-Corasick automaton and further present a detailed construction algorithm and an example of the constructed automaton.

II. Motivational examples

A. DNA Strings

We can understand the DNA as a string in alphabet $\Sigma = \{A, C, G, T\}$. Bases A and G are called purine, and bases C and T are called pyrimidine.

Kurtz [5] writes: "The transversion/transition weight function reflect the biological fact that a purine→purine and pyrimidine→pyrimidine replacement is much more likely to occur than a purine⇄pyrimidine replacement. Moreover, it takes into account that a deletion or insertion of a base occurs more seldom."

In the other words, we have to take into account that the level of similarity or difference of two particular DNA strings cannot be simply expressed as the number of different symbols in them. We need to look at the particular symbol pairs. Obviously, the classical algorithm of approximate string searching does not cover this situation. Our previous research in this field is illustrated in [9] and [10].

B. Spell checker

A spell checker based on a dictionary of correct words and abbreviations is a common way of doing a basic check of a text document. We go through the document and search each word in our dictionary. The words not found in there are highlighted and a correction is suggested. The suggested words are those ones, which are present in the dictionary and are the most similar to the unknown one is sense of addition, deletion and replacement of symbols.

This common model is simple to implement, but it does not cover the fact that some pairs of symbols are more similar than others. This is also very language–specific. For example in Latin alphabet 'a' and 'e' or 'i' and 'y' are somewhat related hence more similar than for example 'w' and 'b'. In many European languages we can found some letters of extended Latin alphabet, whose similarity solely depends on the nature of a national language, e.g. in some languages 'ä' is similar or even identical to 'ae' so their exchange should be favored over other string operations. The primary problem here is that it cannot be simply implemented by standard string search models.

C. Summary

A fuzzy automaton allows us to define individual levels of similarity for particular pairs of symbols or sequences of symbols, so it can be used as a base for a better string search in the sense of presented examples. There are extensive research materials discussing fuzzy automata [2].

III. Fuzzy Automata

A. Fuzzy set

For completeness, we start with a very short definition of a fuzzy set. We define set L as the interval $L = [0, 1]$, $\bigvee L = \sup L = 1$, $\bigwedge L = \inf L = 0$. Let B is a finite set. Then function $A : B \to L$ is called **fuzzy set** A of B.

Whenever $A \subseteq B$, we can also take A as a fuzzy set $A : B \to L$.

$$\forall b \in B : A(b) = \begin{cases} \bigvee L \text{ if } b \in A \\ \bigwedge L \text{ if } b \notin A \end{cases}$$

Note: Definition of L and related stuff can also be more generalized. The reader may consult Nguyen & Walker [8] or Bělohlávek [3] for more details. Also, an infinite B could possibly be a base of an infinite fuzzy set, but we do not require this kind of generalization.

B. *Fuzzy automaton*

Fuzzy automata are generalization of nondeterministic finite automata (see Gruska [4]) in that they can be in each of its states with a degree within the range L.

Fuzzy automaton is a system

$$M = (\Sigma, Q, \delta, S, F)$$

where
Σ is a finite input alphabet
Q is a finite set of states
$S \subseteq Q$ is the set of start states
$F \subseteq Q$ is the set of final (accepting) states
$\delta = \{\delta_a : a \in \Sigma\}$ is a fuzzy transition function
$\delta_a : Q \times Q \to L$ is a fuzzy transition matrix of order $|Q|$, i.e. a fuzzy relation

Note: Fuzzy Automaton recognizes (accepts) a fuzzy language, i.e. language to which words belong in membership/truth degrees not necessarily equal to 0 or 1. Instead, each state of fuzzy automaton is a vector of values in range $[0, 1]$ (i.e. each state maps $Q \to L$).

C. *The transition function*

Fuzzy transition function δ is actually the set of fuzzy relation matrices mentioned above, i.e. a fuzzy set of $Q \times \Sigma \times Q$.

$$\delta : Q \times \Sigma \times Q \to L$$

For a given $s, t \in Q$ and $a \in \Sigma$, value of $\delta(s, a, t) = \delta_a(s, t)$ is the degree of transition from state s to state t for input symbol a.

Every fuzzy set A of Q is called a **fuzzy state** of automaton M. If an input $a \in \Sigma$ is accepted by M, the present fuzzy state A will be changed to the state $B = A \circ \delta_a$, where $\circ$ is a composition rule of fuzzy relations (e.g. minimax product).

Note: This definition is very similar to the one of the probabilistic finite automaton, including the set of transition matrices (see Gruska [4]). Even though the notation is similar, we must be aware that the principles of fuzzy automata are quite different and more generic compared to the quite old-fashioned probabilistic automata.

D. *Minimax product*

Minimax product is defined as follows:
Let $P = [p_{ij}]$, $Q = [q_{jk}]$ and $R = [r_{ik}]$ be matrix representations of fuzzy relations for which $P \circ Q = R$. Then, by using matrix notation, we can write $[p_{ij}] \circ [q_{jk}] = [r_{ik}]$, where

$$r_{ik} = \bigvee_{\forall j} (p_{ij} \wedge q_{jk})$$

Note: This is equivalent to the classic matrix multiplication with operators $\vee$ (join) and $\wedge$ (meet) used as a substitute for classic operators $+$ (plus) and $\cdot$ (times) respectively. We express this analogy since it can be useful when implementing the fuzzy automata on a computer.

E. *Extension to words*

The fuzzy transition function δ can be extended to the word-based **extended fuzzy transition function** δ^*.

$$\delta^* : Q \times \Sigma^* \times Q \to L$$

For $w = a_1 a_2 \dots a_n \in \Sigma^*$ the fuzzy transition matrix is defined as a composition of fuzzy relations: $\delta^*(w) = \delta_{a_1} \circ \delta_{a_2} \circ \dots \circ \delta_{a_n}$ (from left to right).

For empty word ε we define

$$\delta^*(q_1, \varepsilon, q_2) = \begin{cases} \bigvee L & \text{if } q_1 = q_2 \\ \bigwedge L & \text{if } q_1 \neq q_2 \end{cases}$$

Note that if $L = [0, 1]$, then $\bigvee L = 1$ and $\bigwedge L = 0$.

F. *Final (accepting) states*

Function f_M is the **membership degree of word** $w = a_1 \dots a_n$ to the fuzzy set F of final states.

$$f_M : \Sigma^* \to L$$

$$f_M(w) = f_M(a_1 \dots a_n) = S \circ \delta_{a_1} \circ \dots \circ \delta_{a_n} \circ F$$

Note that f_M is a fuzzy set of Σ^*, but we don't use this terminology here. Instead, we use f_M to determine membership degree of a particular word w.

G. *Epsilon transitions*

In section III-E we defined ε-transitions for extended fuzzy transition function. We can generalize that definition to generic ε-transitions, i.e. we define a fuzzy relation δ_ε.

$$\delta_\varepsilon : Q \times Q \to L$$

$$\delta_\varepsilon(q_1, q_2) = \delta^*(q_1, \varepsilon, q_2)$$

IV. Minimization of fuzzy automata

A. *The minimization of an automaton*

One of the most important problems is the minimization of a given fuzzy automaton, i.e. how to decrease the number of states without the loss of the automaton functionality.

For a given $\lambda \in L$, let's have a partition (factor set) $Q_\lambda = \{\bar{q}_1, \dots, \bar{q}_n\}$ of set Q, such that $\forall\ \bar{q}_i \in Q_\lambda$, $q_r, q_t \in \bar{q}_i$, $q \in Q$, and $a \in \Sigma$ holds

$$\begin{array}{rcl} |\delta_a(q_r, q) - \delta_a(q_t, q)| & < & \lambda \\ |\delta_a(q, q_r) - \delta_a(q, q_t)| & < & \lambda \\ |S(q_r) - S(q_t)| & < & \lambda \end{array} \tag{1}$$

$$\bar{q}_i \subseteq F \text{ or } \bar{q}_i \cap F = \emptyset \tag{2}$$

We construct fuzzy automaton $M_\lambda = (\Sigma, Q_\lambda, \delta_\lambda, S_\lambda, F_\lambda)$ where

$$\delta_\lambda(\bar{q},u,\bar{r}) = \delta_{\lambda u}(\bar{q},\bar{r}) = \frac{\sum_{q_i\in\bar{q}}\sum_{r_j\in\bar{r}}\delta_u(q_i,r_j)}{|\bar{q}|\cdot|\bar{r}|}$$

$$S_\lambda(q) = \frac{\sum_{q_j\in\bar{q}} S(q_j)}{|\bar{q}|}$$

$$F_\lambda = \{\bar{q} : \bar{q} \subseteq F\}$$

and $\bar{q},\bar{r} \in Q_\lambda$

Theorem 1. Let $w = a_1a_2\ldots a_m$. Then $|f_M(w) - f_{M_\lambda}(w)| < \lambda(m+2)$.

P r o o f. See Močkoř [6].

Let us describe how to use these equations: We must define the maximum word length m_0, and the maximum acceptable difference λ_0 for the words of this maximum size. Then we can compute λ as follows:

$$\lambda = \frac{\lambda_0}{m_0+2} \qquad (3)$$

Having the λ value, we can perform desired automaton minimization.

B. An example

Let's have fuzzy automaton $M = (\Sigma, Q, \delta, S, F)$.
$\Sigma = \{0,1\}$
$Q = \{q_1,q_2,q_3,q_4,q_5\}$

$$\delta_0 = \begin{pmatrix} 0.45 & 0.50 & 0.80 & 0.31 & 0.35 \\ 0.47 & 0.46 & 0.78 & 0.34 & 0.30 \\ 0.10 & 0.15 & 0.51 & 0.83 & 0.78 \\ 0.70 & 0.67 & 0.42 & 1.00 & 0.94 \\ 0.71 & 0.68 & 0.37 & 0.95 & 1.00 \end{pmatrix}$$

$$\delta_1 = \begin{pmatrix} 0.78 & 0.74 & 1.00 & 1.00 & 0.96 \\ 0.73 & 0.77 & 0.96 & 0.96 & 0.96 \\ 1.00 & 0.96 & 0.00 & 0.00 & 0.05 \\ 0.10 & 0.12 & 0.80 & 1.00 & 0.97 \\ 0.14 & 0.12 & 0.76 & 0.99 & 0.95 \end{pmatrix}$$

$S = \begin{pmatrix} 1.00 & 0.15 & 1.00 & 0.85 & 0.90 \end{pmatrix}$
$F = \{q_3\}$

We want to minimize this fuzzy automaton in such way that the outgoing transition function will differ by less than 0.25 for words long 2 symbols at most.

According to (3), $\lambda_0 = 0.25$, $m_0 = 2$, so we compute $\lambda = \frac{0.25}{2+2} = 0.0625$.

Now we make the fuzzy automaton M_λ from this analysis according to formulas (1) and (2):
$Q_\lambda = \{\bar{q}_1,\bar{q}_2,\bar{q}_3\}$
$\bar{q}_1 = \{q_1,q_2\}$, $\bar{q}_2 = \{q_3\}$, $\bar{q}_3 = \{q_4,q_5\}$
$S_\lambda = \begin{pmatrix} 0.125 & 1.000 & 0.875 \end{pmatrix}$

Then, for example, we get

$$\delta_{\lambda 0}(\bar{q}_1,\bar{q}_1) = \delta_\lambda(\bar{q}_1,0,\bar{q}_1) = \frac{1}{4}(0.45+0.50+0.47+0.46) = 0.47$$

$$f_M(01) = S \circ \delta_0 \circ \delta_1 \circ F = 0.8$$

$$f_{M_\lambda}(01) = S_\lambda \circ \delta_{\lambda 0} \circ \delta_{\lambda 1} \circ F_\lambda = 0.8$$

As evident from the example, we reduced the number of states from 5 to 3, and still $f_M(01) = f_{M_\lambda}(01)$. Generally, according to the above formulas, $|f_M(w) - f_{M_\lambda}(w)| < 0.25$.

V. Aho-Corasick search automaton

A. Classical variant

Aho-Corasick search automaton (see [1]) is well grounded and widely used method for locating patterns in source data. It differs from trivial search methods, because this methods locates multiple patterns at once. Time complexity of such search is linear in the length of source data plus the length of patterns.

Roughly speaking, the automaton is based on a trie constructed from the dictionary of search patterns. Trie is extended using so-called fail function, which allows efficient string matching.

B. Fuzzified variant

The following section is based on a work of one of Vaclav Snasel's Msc. students (see [11]). Contrary to the classical deterministic Aho-Corasick search automata, the fuzzified variant has to be (according to the previous definition of fuzzy automaton) non-deterministic.

Definition 1. Fuzzy Aho-Corasick search automaton is finite fuzzy automaton G constructed using alphabet Σ and dictionary $X \subseteq \Sigma^*$. This automaton recognizes fuzzy language of all strings from Σ^* containing a word from dictionary X as a suffix. Automaton recognizes fuzzy language L, such that $L(l) = X(x)$ for $l = \Sigma^* x$.

If we select Boolean lattice as our basic structure, we want our Fuzzy Aho-Corasick automaton to accept the same language as the classical one.

The construction of fuzzified Aho-Corasick automaton can be – divided into three separate phases. In the first phase, we create a fuzzy trie from the fuzzy dictionary of words. The second phase constructs a fail function. In the last phase we transform fuzzy trie and fail function into fuzzy transition function of constructed automaton.

C. Fuzzy trie construction

Fuzzy trie may be seen as a classical trie, where each state is enhanced with a fuzzy degree. In the construction process we will also need so-called prefix membership function PMF(weight, wordLength, prefixLength) ($PMF : L \times N \times N \rightarrow L$), which returns the membership degree of current trie state with respect to the fuzzy dictionary. It must hold, that the PMF function is non-increasing with respect to the prefix length. Technically speaking, for $weight \in L, wordLength, p_1, p_2 \in$

$L, p_1 < p_2$ it must hold $PMF(weight, wordLength, p_1) \geq PMF(weight, wordLength, p_2)$.

```
   Input: fuzzy dictionary of words $X$
   Output: relation parent : $Q \to Q$, relation symbol : $Q \to \Sigma$, transition
           function $\delta$
   Data: set of states $Q$, fuzzy set of final states $F$
1  $F = \emptyset$
2  $Q = \{q_0\}$
3  $w(q_0) = 0$
4  $\delta(q, q', c) = 0, \forall q, q' \in Q, c \in \Sigma$
5  foreach word $x \in X$ do
6      $q = q_0$
7      $w(q_0) = \sup(w(q_0), X(x))$
8      for $j = 1 \ldots |x|$
9      do
10         if $\forall q' : \delta(q, q', x_j) = 0$ then
11             $Q = Q \cup q_{new}$
12             $\delta(q, q_{new}, x_j) = 1$
13             $w(q_{new}) = 0$
14             $symbol(q_{new}) = x_j$
15             $parent(q_{new}) = q$
16         $q = q' \mid \delta(q, q', x_j) = 1$
17         $w(q) = \sup(w(q), PMF(X(x), |x|, j))$
18     end
19     $F(q) = \sup(F(q), X(x))$
20 end
```

The algorithm is initiated on lines 1–4, lines 6–19 iterates through the whole dictionary and for each of its words creates new trie states (lines 11–15).

D. Fail function construction

This part of algorithm is very similar to the original Aho-Corasick construction.

```
   Input: function $\delta$ and relation $parent$ from the previous step, function depth
          returning depth of given state in the trie
   Output: function fail : $Q \to Q$
1  $fail(q_0) = \texttt{fail}$
2  foreach $q' \mid parent(q') = q_0$ do
3      $fail(q') = q_0$
4  end
5  foreach $q' \mid depth(q') > 1$, in the BFS order do
6      $q = parent(q')$
7      select $c \in \Sigma$, such that $\delta(q, q', c) \neq 0$
8      $r = fail(q)$
9      while $(r \neq \texttt{fail})$ AND $(\forall r' : \delta(r, r', c) = 0)$ do
10         $r = fail(r)$
11     end
12     if $r = \texttt{fail}$ then
13         $fail(q') = temp \mid \delta(q_0, temp, c) \neq 0$
14     else
15         $fail(q') = temp \mid \delta(r, temp, c) \neq 0$
16         if $F(q') \neq \sup(F(q'), F(fail(q')))$ then
17             $F(q') = \sup(F(q'), F(fail(q')))$
18             $dst = q'$
19             $src = fail(q')$
20             while $src \neq q_0$ do
21                 $w(dst) = \sup(w(dst), w(src))$
22                 $src = parent(src)$
23                 $dst = parent(dst)$
24             end
25
26
27 end
```

The fail function is initiated for the starting symbol q_0 on lines 1–4. The algorithm than iterates over the trie in the breadth-first search (lines 6–24) and modifies the weight function of individual states.

E. Transition function completion

In this phase, we merge the constructed functions and create Fuzzy Aho-Corasick automaton.

```
   Input: functions $\delta$, $fail$ and relation parent, symbol from previous steps and
          weight function w
1  foreach $q \in Q$ in the BFS order do
2      foreach $c \in \Sigma$ do
3          if $\delta(q, q', c) = 0, \forall q'$ then
4              $symbol = c$
5              $dst = \texttt{fail}(q)$
6              while $(dst \neq \texttt{fail})$ AND $(\forall q' : \delta(dst, q', symbol) = 0)$
               do
7                  $dst = fail(dst)$
8              end
9              if $dst = \texttt{fail}$ then
10                 $dst = q_0$
11             else
12                 $dst = temp \mid \delta(dst, temp, symbol) \neq 0$
13             $dstWeight = w(dst)$
14             $src = q$
15             if $w(src) = dstWeight$ then
16                 $dstWeight = 1$
17             else
18                 while $(src \neq q_0)$ AND $(dst \neq q_0)$ AND $(w(src) <$
                   $dstWeight)$ do
19                     $symbol = symbol(src)$
20                     $src = parent(src)$
21                     $dst = parent(dst)$
22                     $dstWeight = w(dst)$
23                 end
24             $\delta(src, dst, symbol) = dstWeight$
25         else
26             foreach $q' \mid \delta(q, q', c) \neq 0$ do
27                 $\delta(q, q', c) = w(q')$
28             end
29
30     end
31     if $F(q) = w(q)$ then
32         $F(q) = 1$
33
34 end
```

This part of algorithm explores all nodes of fuzzy trie in the BFS order and integrates transitions from fail function for all $q \in Q$. Line 3 checks, whether the relation δ is for state q and alphabet symbol c defined. If yes, only calculated weights are incorporated into the transition function δ (lines 26–28). If not, on lines 4–24 the transitions based on fail function are added.

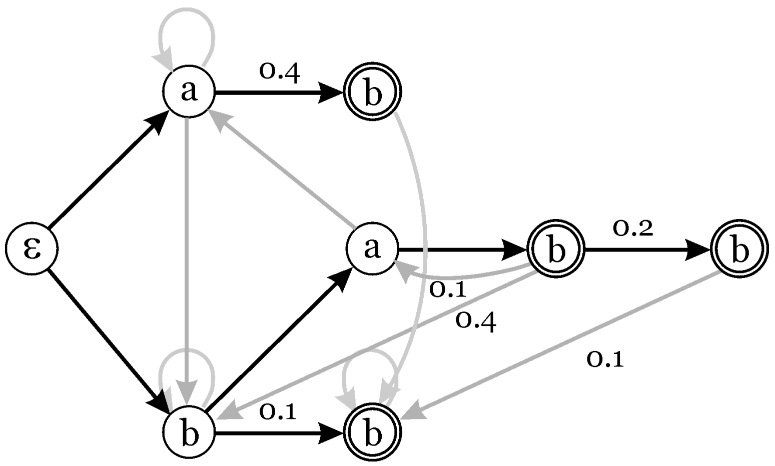

Fig. 1. Example of fuzzy Aho-Corasick automaton

Figure 1 illustrates the fuzzy Aho-Corasick automaton constructed for dictionary $X = \{ab_{0.4}, babb_{0.2}, bb_{0.1}\}$. Black transitions comes from the original trie, gray transitions comes from fail function. Double bordered nodes represent final states.

VI. Conclusions

We described the proposed approach for the construction of a fuzzy Aho-Corasick automaton for string matching and searching for patterns in strings.

In the future we wish to focus on the implementation of the presented algorithms in a parallel computing environment using GPUs (such as NVIDIA CUDA, OpenCl or Microsoft's DirectCompute) and complexity aspects of this automaton.

References

[1] Aho, A.V., Corasick, M.J. *Efficient string matching: an aid to bibliographic search*, Communications of the ACM, vol. 18, **1975**, pp. 333–340.

[2] Asveld P.R.J. A bibliography on Fuzzy Automata, Grammars and Language. In: *Bulletin of the European Association for Theoretical Computer Science.* No.58, **1996**, pp. 187–196.

[3] Bělohlávek R. *Fuzzy Relational Systems: Foundations and Principles.* Kluwer Academic Publishers, **2002**, 382 pp., ISBN 0-306-46777-1.

[4] Gruska J. *Foundations of Computing.* International Thomson Computer Press, **1997**, 197 pp., ISBN: 1-85032-243-0.

[5] Kurtz S. Approximate String Searching under Weighted Edit Distance. In: *Proceedings of 3rd South American Workshop oh String Processing*, Carton University Press, **1996**, pp. 156–170.

[6] Močkoř J. Minimization of Fuzzy Automata. *RSV FMPE*, Havířov, Czech Republic, **1982**. (in Czech)

[7] Muthukrishnan S. New Results and Open Problems Related to Non-Standard Stringology. In: *Proceedings of 6th Combinatorial Pattern Matching Conference CMP 95*, Springer Verlag, LNCS 937, Espoo, Finland, **1995**, pp. 298–317.

[8] Nguyen H.T., Walker E.A. *A First Course in Fuzzy Logic.* (2nd edition) Chapman & Hall/CRC, **2000**, 372 pages, ISBN: 0-8493-1659-6.

[9] Novosad, T., Snasel, V., Abraham, A., Yang, J.Y. *YAPS: Yet Another Protein Similarity*, International Conference on Soft Computing and Pattern Recognition, **2009**, pp.497–504.

[10] Novosad, T., Snasel, V., Abraham, A., Yang, J.Y. *Prosima: Protein similarity algorithm*, World Congress on Nature & Biologically Inspired Computing, **2009**, pp.84–91.

[11] Tejkl J. *Aho-Corasick automata fuzzyfication*, diploma thesis, **2000**.