Approximate String Matching by Fuzzy Automata

Václav Snášel[1], Aleš Keprt[2], Ajith Abraham[3], and Aboul Ella Hassanien[4]

[1] Department of Computer Science, Faculty of Electrical Engineering
and Computer Science, VŠB – Technical University of Ostrava
17. listopadu 15, 708 33 Ostrava–Poruba, Czech Republic
Vaclav.Snasel@vsb.cz
[2] Department of Computer Science, Faculty of Sciences, Palacký University
Tomkova 40, 779 00 Olomouc, Czech Republic
Ales.Keprt@upol.cz
[3] Center of Excellence for Quantifiable Quality of Service
Norwegian University of Science and Technology, Norway
ajith.abraham@ieee.org
[4] Cairo University
Faculty of Computer and Information,
Giza, Egypt
aboitcairo@gmail.com

Abstract. We explain new ways of constructing search algorithms using fuzzy sets and fuzzy automata. This technique can be used to search or match strings in special cases when some pairs of symbols are more similar to each other than the others. This kind of similarity can't be handled by usual searching algorithms. We present sample situations, which would use this kind of searching. Then we define a fuzzy automaton, and some basic constructions we need for our purposes. We continue with definition of our fuzzy automaton based approximate string matching algorithm, and add some notes to fuzzy–trellis construction which can be used for approximate searching.

1 Introduction

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

Approximate string matching and searching isn't a new problem, it has been faced and 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.

In this paper, we present a technique which can be used to search or match strings in special cases when some pairs of symbols are more similar to each other than other pairs. This kind of similarity can't be handled by usual searching algorithms. (Also note that even so called "fuzzy string matching" doesn't

distinguish between more or less similar symbols, so it's not related to fuzzy math at all.) We start our paper with some motivational examples, showing that there are several situations, which would use this kind of searching. Then we define a fuzzy automaton, and some basic constructions we need for our purposes. We continue with definition of our fuzzy automata based approximate matching algorithm, and add some notes to fuzzy-trellis construction which can be used for approximate searching.

2 Motivational examples

Let us start with some motivational examples.

2.1 DNA Strings

We can understand 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 [4] 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 can't be simply expressed as the number of different symbols in them. We need to look at the particular symbol pairs. Obviously, the classic algorithm of approximate string searching doesn't cover this situation.

2.2 Spell checker

A spell checker based on a dictionary of correct words and abbreviations is a common way of doing basic check of a text document. We go through document and search each of its words 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 doesn't 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 can't be simply implemented by standard string search models.

2.3 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 – you can find a list of other works in Asveld's survey [1].

3 Approximate String Matching by Fuzzy Automata

3.1 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, see Nguyen & Walker [8] or Bělohlávek [2] for more details. Also, an infinite B could possibly be a base of an infinite fuzzy set, but we do not need this kind of generalization here.

3.2 Fuzzy automaton

Fuzzy automata are generalization of nondeterministic finite automata (see Gruska [3]) in that they can be in each of its states with a degree in 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 $Q \to L$).

3.3 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 [3]). We can see that the notation is similar, but we must be aware that the principles of fuzzy automata are quite different and more generic compared to the quite old-fashioned probabilistic automata.

3.4 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.

3.5 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 $\bigwedge L = 0$.

3.6 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.

3.7 Epsilon transitions

In section 3.5 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.

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

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

4 Minimization of fuzzy automata

4.1 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 us 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

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

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

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 $w = a_1 a_2 \dots a_m$. Then $|f_M(w) - f_{M_\lambda}(w)| < \lambda(m+2)$.

`Proof.` 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 this way:

$$\lambda = \frac{\lambda_0}{m_0 + 2} \tag{3}$$

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

4.2 An example

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

5 Approximate string matching based on distance functions

5.1 Hamming and Levenshtein distance

In this chapter we present constructions $R(M, k)$ and $DIR(M, k)$, originally introduced by Melichar and Holub [5].

These constructions correspond to the creation of a nondeterministic automaton M' from the automaton M accepting string $P = p_1 p_2 \dots p_n$. Automaton M' accepts those strings P', whose value of P to P' distance is equivalent or lower than the given k while using distance functions R or DIR.

Distance $R(P, P')$ is called **Hamming distance** and is defined as the minimum number of symbol replacement operations in string P required for the conversion of string P into string P' (or vice versa).

Distance $DIR(P, P')$ is called **Levenshtein distance** and is defined as the minimum number of operations of symbol deletion (D), insertion (I) or replacement (R) in P required for the conversion of string P into string P' (or vice versa).

Automaton M' is called $R-$trellis or $DIR-$trellis as the case may be. The construction of these automata is described, for example, in the paper [5]. The trellis construction is a crucial part in the approximate string matching, so we need to generalize the trellis construction to our fuzzy automata. We will do in the following paragraphs.

5.2 Construction of fuzzy trellis from a non-fuzzy one

Let us have **similarity function** s, which defines similarity level between each pair of input symbols.

$$s : \Sigma \times \Sigma \to L \tag{4}$$

$$s(a_i, a_j) = \begin{cases} \bigwedge L \text{ if } a_i \text{ and } a_j \text{ are fully different} \\ \bigvee L \text{ if } a_i \text{ and } a_j \text{ are fully equal} \\ \text{other value} \in L \text{ if } a_i \text{ and } a_j \text{ are similar} \end{cases} \tag{5}$$

We usually also define $s(a_i, a_j) = s(a_j, a_i)$, but it is generally not required. If we don't obey this rule, we get an asymmetric relation, which is less common in real applications, but still mathematically correct.

Now we can define fuzzy transition function δ' as

$$\delta'_a(q_i, q_j) = \bigvee_{b \in \Sigma} (s(a,b) \cdot \delta_b(q_i, q_j)) \tag{6}$$

$\forall q_i, q_j \in Q, a \in \Sigma$.

We use formula 6 to construct a fuzzy trellis $M' = (\Sigma, Q, \delta', S, F)$ from the given similarity function s and a given non-fuzzy trellis $M' = (\Sigma, Q, \delta, S, F)$. This construction is generic and not limited to just R-trellis and DIR-trellis presented above.

5.3 An example

Let us show an example of fuzzy trellis construction for Hamming distance.

Let us have $\Sigma = \{$'a','b','c'$\}$, and automaton $M' = (\Sigma, Q, \delta, S, F)$ is a R-trellis based on word $w_0 = \{$"ab"$\}$.

$$\delta_a = \begin{pmatrix} 1\,1\,0\,0\,0 \\ 0\,0\,0\,0\,1 \\ 0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0 \end{pmatrix} \quad \delta_b = \begin{pmatrix} 1\,0\,0\,1\,0 \\ 0\,0\,1\,0\,0 \\ 0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,1 \\ 0\,0\,0\,0\,0 \end{pmatrix} \quad \delta_c = \begin{pmatrix} 1\,0\,0\,1\,0 \\ 0\,0\,0\,0\,1 \\ 0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0 \end{pmatrix}$$

Now we are going to construct fuzzy R-trellis of this R-trellis. Let us say symbols 'a' and 'c' are a bit similar, while the other ones are pairwise different.

$$s = \begin{pmatrix} 1\,0\,* \\ 0\,1\,0 \\ *\,0\,1 \end{pmatrix}, * \in L$$

We construct fuzzy transition function δ'.

$$\delta_a = \begin{pmatrix} 1\,1\,0\,*\,0 \\ 0\,0\,0\,0\,1 \\ 0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0 \end{pmatrix} \quad \delta_b = \begin{pmatrix} 1\,0\,0\,1\,0 \\ 0\,0\,1\,0\,0 \\ 0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,1 \\ 0\,0\,0\,0\,0 \end{pmatrix} \quad \delta_c = \begin{pmatrix} 1\,*\,0\,1\,0 \\ 0\,0\,0\,0\,1 \\ 0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,0 \end{pmatrix}$$

Now we can compare words $w_1 = \{$"bb"$\}$ and $w_1 = \{$"cb"$\}$ to the original word w_0.

If we use the original Hamming function distance R, we get $R(w_0, w_1) = 1$ and $R(w_0, w_2) = 1$. In this case the words w_1 and w_2 have got the same level of similarity to the w_0.

If we say that symbols 'a' and 'c' are a bit similar, we can define for example $* = s($'a','c'$) = 0.3$. Now $f_{M'}(w_1) = 0$, while $f_{M'}(w_2) = 0.3$, and we can see that w_0 is more similar to w_2 than w_1. This is how a fuzzy trellis works.

5.4 Another variants of similarity function

It may be better to use a different approach to similarity function than the one shown in formula 5. Formally, we stay at the same definition (see formula 4), but use a non-zero values for fully different symbols. This time, we define a minimum value s_{min} which is assigned to $s(a_i, a_j)$ whenever i and j and fully different, and use greater values than s_{min} when i and j are similar. This modified approach performed better in our experiments.

6 Conclusion

We described the construction of a fuzzy automaton for string matching and searching for patterns in strings. Our approach allows defining different similarity levels for particular pairs of symbols, which can be very useful in several applications. Specifically, we expect immediate applicability in the field of DNA string matching. We also did some research in the field of fuzzy Aho-Corasick automata (ACA), proving that the classic ACA can be generalized to fuzzy ACA. This generalization brings our fuzzy based approximate string matching to another set of classic pattern searching applications.

In the future we want to focus on the minimization of fuzzy automata and compare fuzzy automata with other related techniques, e.g. constructions with weight function as described by Muthukrishnan [7].

References

1. 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.
2. Bělohlávek R. *Fuzzy Relational Systems: Foundations and Principles.* Kluwer Academic Publishers, **2002**, 382 pp., ISBN 0-306-46777-1.
3. Gruska J. *Foundations of Computing.* International Thomson Computer Press, **1997**, 197 pp., ISBN: 1-85032-243-0.
4. 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.
5. Melichar B., Holub J. 6D Classification of Pattern Matching Problem. In: *Proceedings of the Prague Stringology Club Workshop 97*, **1997**, pp. 24–32.
6. Močkoř J. Minimization of Fuzzy Automata. textitRSV 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.