An Approach to Fuzzy Clustering of Strings

Armen Kostanyan
IT Educational and Research Center
Yerevan State University
Yerevan, Armenia
e-mail: armko@ysu.am

Arevik Harmandayan
American University of Armenia
Yerevan, Armenia
e-mail: arevik.harmandayan@gmail.com

***Abstract*—This paper discusses the problem of finding sequences of adjacent string segments that match a fuzzy pattern represented as a sequence of fuzzy properties of segments. To solve this problem, a modification of the Knuth-Morris-Pratt algorithm is proposed, which, when moving to the next symbol of the pattern, advances by a certain segment of the string. The proposed algorithm is non-deterministically polynomial, which opens up the possibility of constructing various approximation algorithms. One such algorithm based on the greedy approach is proposed.**

This work may find application in the analysis of processes generated by a discrete dynamic system, as well as for the classification of DNA sequences using fuzzy prototypes described in one way or another.

***Keywords*—Fuzzy string matching, fuzzy clustering, KMP variations.**

I. Introduction

In this paper, we discuss a heuristic algorithm for finding sequences of adjacent segments of a given string that match a given pattern. The latter is defined as a sequence of fuzzy properties of segments. We say that a sequence of adjacent segments matches a given pattern if they all satisfy the given length constraints and, in addition, match the symbols of the pattern with a given accuracy. We call this problem the *fuzzy string clustering* problem. Note that many of the approximate pattern matching problems in which meta-characters are allowed in the pattern can be viewed as variations of fuzzy string clustering. A detailed review of the works regarding approximate pattern matching is presented in the monograph [1].

The problem under consideration appears to be NP-hard, due to the efficient procedure for verifying whether a given sequence of segments matches the pattern, on the one hand, and due to the exponential number of sequences of segments to be verified, on the other hand.

We propose a heuristic algorithm that finds a fairly large number of occurrences of a given fuzzy pattern in a string using the prefix function from the Knuth-Morris-Pratt (KMP) string matching algorithm [2] together with a string-advance function to find the next valid segment, if one exists. The string advance function we use is non-deterministic, which makes our heuristic algorithm also non-deterministic. Then we propose an efficient customization of the heuristic algorithm in a greedily manner and estimate its complexity.

In particular, when adjacent segments must have unit length, and the pattern, accordingly, is a sequence of fuzzy properties of characters, the fuzzy string clustering problem is transformed into the problem of finding segments of a given string that match the pattern.

The study of fuzzy string clustering proposed in this paper is consistent with several works by the authors in the field of string matching. Particularly, in [3], the periodicity in the pattern was used to improve the efficiency of the preprocessing phase in the method of string matching with finite automata and in the KMP algorithm. In [4], a nondeterministic transition system was constructed to describe the possibilities of processing a given text to find all occurrences of a fuzzy pattern in it. In [5], an efficient algorithm was proposed for determining all occurrences of a fuzzy pattern in the text, imitating the KMP algorithm with a two-dimensional prefix table. In [6] and [7], efficient solutions were proposed to split an entire string into sufficiently large segments to match the fuzzy pattern as closely as possible.

The paper is organized as follows.

Section 2 introduces the concept of a fuzzy clustering pattern and formulates the problem of fuzzy string clustering. Section 3 presents a heuristic solution to the fuzzy clustering problem and analyzes its complexity. Sections 4 and 5 describe the greedy approach to fuzzy string clustering and its customization for unit-size segments. Finally, the conclusion summarizes the obtained results.

II. Preliminaries

A. Fuzzy sets

Suppose $(L, \leq, 0, 1)$ is a linearly ordered set with the smallest element 0 and the largest element 1. A *fuzzy subset A* of the universal set U is defined by the membership function $\mu_A : U \to L$ that associates with each element x from U the value $\mu_A(x)$ from L, called the *degree of membership* of x in A [8]. We say that an element x certainly belongs to A if $\mu_A(x) = 1$, and it certainly does not belong to A if $\mu_A(x) = 0$. Conversely, if $0 < \mu_A(x) < 1$, then we say that x belongs to A with degree $\mu_A(x)$.

B. Fuzzy clustering problem

Let Σ be an alphabet of characters and Σ^* be the set of all finite length strings in Σ.

https://doi.org/10.51408/csit2025_55

We define a *clustering symbol* as a fuzzy subset of Σ^* that allows strings in Σ to be measured by elements from *L*. Given a clustering symbol α and a string $x \in \Sigma^*$, we say that *x* matches α with degree $\mu_\alpha(x)$. We define the *clustering pattern* to be an array $P[1..m]$ of clustering symbols.

Suppose we are given a cluster verification function as a mapping

$$check : \Sigma^* \to \{true, false\}.$$

Assume that the valid clusterings we are looking for must consist of segments that satisfy the $check$ function. The pair $C = (check, \mu)$, where $\mu \in L$ is the minimum matching degree, let us call a *clustering constraint*.

For an $x \in \Sigma^*$, a clustering symbol α, and a constraint C, we say that x C-matches α and write $x \overset{C}{\sim} \alpha$ if $check(x) = true$ and $\mu_\alpha(x) \geq \mu$.

Accordingly, given a string $T[1..n]$, a segmentation pattern $P[1..m](m \leq n)$, and a constraint C, we define a (P, C) - clustering of T as a sequence

$$s_1 = [low_1, high_1],\ \ldots,\ s_m = [low_m, high_m],$$

$$high_k + 1 = low_{k+1}, 1 \leq k \leq m-1,$$

of adjacent segments of T such that $s_k \overset{C}{\sim} P[k]$ for all $k, 1 \leq k \leq m$.

Finally, we define the (P, C) - *clustering problem* as the problem of finding all valid (P, C) - clusterings of T.

Example 2.1: Let *L* be the segment [0, 1] of ordered reals and Σ = {0, 1} be a two-element alphabet. Consider the clustering symbols α_0 and α_1 such that for all $x \in \Sigma^*$, $\mu_{\alpha_0}(x)$ and $\mu_{\alpha_1}(x)$ are the relative numbers of 0's and 1's in *x*, respectively.

Suppose that $T = 101100011, P = \alpha_1\alpha_0\alpha_1$, and the clustering constraint is $C = (check, \mu)$, where

$$check(x) = true \Leftrightarrow 2 \leq size(x) \leq 3 \text{ and } \mu = 2/3.$$

Then, there are the following valid (P, C) - clusterings of T:

$$T = (101)(100)(011), T = 1(011)(00)(011),$$

$$T = 10(11)(000)(11), \text{etc.}$$

III. String Clustering Heuristic

The proposed heuristic is based on the KMP string matching algorithm [2]. Note that in the KMP algorithm, moving forward in the text is carried out by one position. On the contrary, the proposed algorithm moves forward through the string by the length of a segment that matches the next clustering symbol. The found segment is then fixed as the next element of the clustering to be constructed.

Let us consider the following functions.

- $j = lookAhead(T, i, \alpha, C)$. This function *non-deterministically* returns the rightmost position of any segment x that starts at position i of T and satisfies the $x \overset{C}{\sim} \alpha$ condition, if such exists. Otherwise, the function returns -1.
- $advance(i, j)$. This function returns $i+1$ if $j = -1$, and $j + 1$ otherwise.

For a given a clustering pattern $P = P[1..m]$, define the border of P to be a subarray $P[1..k](k < m)$ such that $P[i] = P[m - k + i]$ for all $1 \leq i \leq k$ (that is, the first k symbols of P coincide with its last k symbols). Let $LB(P)$ denote the longest border of P.

Define the *prefix function* for P as a mapping

$$\pi : \{1, \ldots, m\} \to \{0, \ldots, m-1\}$$

such that for all $i, 1 \leq i \leq m$,

$$\pi(i) = size(LB(P[1..i]).$$

Additionally, suppose that $\pi(0) = 0$. As it is known from the KMP algorithm, the prefix function for $P = P[1..m]$ can be computed in $O(m)$ time.

To represent the current clustering, let us use a *multi-queue* of segments specified by their endpoints, with the following operations:

- $push(x)$: inserts x to the end,
- $mulipop(quantity)$: removes $quantity$ elements from the beginning,
- $size()$: returns the number of segments in the queue,
- $print()$: prints the contents.

Based on these operations, we represent the clustering heuristic as follows:

```
SC-HEURISTIC (T, P, n, m, C)
1   π = COMPUTE-PREFIX-FUNCTION(P, m)
2   mq = ∅ //multi-queue is empty
3   i = 1
4   while i ≤ n
5       j = −1
6       while (k = mq.size()) > 0 and
7             lookAhead(T, i, P[k + 1], C) == −1
8         mq.multipop(k − π(k)
9       if j ≠ −1
10          mq.push(< i, j >)
11      if mq.size() == m //entire pattern matched
12          mq.print(); mq.multipop(m − π(m))
13      i = advance(i, j)
```

Example 3.1: Suppose that the clustering symbols α_0, α_1 and the constraint C are defined as in *Example 2.1*,

$$T[1..20] = 00010001100100110100,$$

$$P[1..4] = \alpha_0\alpha_0\alpha_1\alpha_0.$$

Let us consider the following two options for processing these data using the SC-HEURISTIC algorithm:

Option 1:

- ◇ The segments 00 and 010 can be chosen to match $\alpha_0\alpha_0$.
- ◇ Then, we have a mismatch for α_1 in position $i = 6$.
- ◇ Since $\pi[2] = 1$, we fix the matching α_0 with the segment $T[3..5] = (010)$ and continue processing for $T[6..20]$ and $P[2..4]$.

◇ The segments 100, 11, and 001 can be chosen to match subsequent $\alpha_0\alpha_1\alpha_0$, resulting in the *first* valid clustering

$$T[3..5] = (010) \overset{C}{\sim} \alpha_0, T[6..8] = (100) \overset{C}{\sim} \alpha_0,$$

$$T[9..10] = (11) \overset{C}{\sim} \alpha_1, T[11..13] = (001) \overset{C}{\sim} \alpha_0.$$

◇ Since $\pi[4] = 1$, we fix the matching α_0 with $T[11..13] = (001)$ and continue processing for $T[14..20]$ and $P[2..4]$.

◇ The segments 00, 101, and 00 can be chosen to match next $\alpha_0\alpha_1\alpha_0$, resulting in the *second* valid clustering

$$T[11..13] = (001) \overset{C}{\sim} \alpha_0, T[14..15] = (00) \overset{C}{\sim} \alpha_0,$$

$$T[16..18] = (101) \overset{C}{\sim} \alpha_1, T[19..20] = (00) \overset{C}{\sim} \alpha_0.$$

◇ Thus, in this option, the SC-HEURISTIC algorithm generates the following two overlapping (P, C) - clusterings of T:

$$T = 00(010)(100)(11)(001)0010100, \text{and}$$

$$T = 0001010011(001)(00)(101)(00).$$

For the illustration of this option, see Fig. 1.

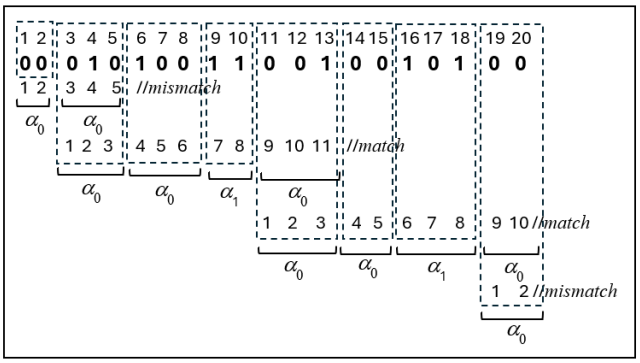

Fig. 1. The first option for executing the SC-HEURISTIC algorithm

Option 2:

◇ Likewise, the first choice $T[1..3] = 000$ to match α_0 results in the only clustering

$$T = 000101001(100)(100)(101)(00).$$

illustrated in Fig.2.

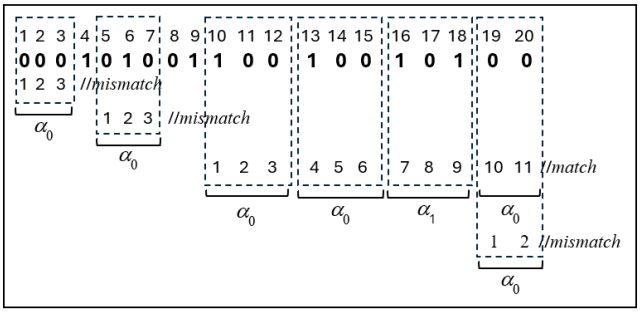

Fig. 2. The second option for executing the SC-HEURISTIC algorithm

Note also that despite its non-deterministic nature, the SC-HEURISTIC algorithm does not generate the following valid (P, C) clastering

$$T = 0(001)(010)(011)(001)0010100.$$

IV. Greedy Approach

A. Description

The use of a non-deterministic $lookAhead$ function makes the SC-HEURISTIC algorithm non-deterministically polynomial. Let us build a greedy approximation of this function based on the following details.

Given constants $\lambda_1 \leq \lambda_2$, suppose that we are looking for clusterings consisting of segments the length of which is between λ_1 and λ_2. This can be achieved by assuming

$$check(x) = true \Leftrightarrow \lambda_1 \leq size(x) \leq \lambda_2 \text{ for all } x \in \Sigma^*.$$

Next, assume that the symbols used in the pattern meet the following conditions:

✓ The $\mu_\alpha(\epsilon)$ value can be computed in $O(1)$ time, and

✓ For any $c \in \Sigma$, the value $\mu_\alpha(xc)$ can be obtained from the value of $\mu_\alpha(x)$ in constant time with appropriate tracking of the process of calculating the value of $\mu_\alpha(x)$.

Clustering symbols that satisfy these conditions let us call *regular*. Note that symbols α_0 and α_1 from *Example 2.1* are regular.

Taking these assumptions into account, let us consider the greedy version of the $lookAhead(T, i, \alpha, C)$ function that returns the rightmost position of the *longest* segment that starts at position i of T and C - matches the clustering symbol α. (As before, this function returns -1 if no such segment exists).

Due to the regularity of α and the (λ_1, λ_2)-boundedness of segments, the value $lookAhead(T, i, \alpha, C)$ can be calculated in a time proportional to the value of λ_2, i.e., in constant time.

The version of the SC-HEURISTIC algorithm with a greedy $lookAhead$ function, let us call GREEDY-SC-HEURISTIC.

Example 4.1: Suppose the clustering symbols α_0, α_1, and the clustering constraint C are defined as in *Example 2.1*; T and P are defined as in *Example 3.1*, i.e.

$$T = 00010001100100110100, P = \alpha_0\alpha_0\alpha_1\alpha_0;$$

$\lambda_1 = 2, \lambda_2 = 3$.

It is easy to check that in this case, the only (P, C) - clustering of T produced by the GREEDY-SC-HEURISTIC algorithm is the same as that produced by the SC-HEURISTIC in option 2, i.e.,

$$T = 000101001(100)(100)(101)(00).$$

B. Analysis

As it was mentioned, the prefix function calculation in line 1 takes $O(m) = O(n)$ time. The assignments in lines $2-3$ obviously take $O(1)$ time.

Let us use the *potential method* ([2]) to estimate the complexity of the outer **while** loop in lines $4-13$. Define

the potential before each execution of the body of this loop as $size(mq)$, which is initially 0 and never becomes negative.

We note that the assignment in line 3 clearly has a constant amortized complexity.

The same holds for the inner **while** loop in lines $6-8$. Indeed, the actual cost of an iteration (including computing of the $lookAhead$ function and performing a $multipop$ operation) is compensated by a decrease in potential with an appropriate scaling of the potential unit.

The **if** statement in lines $9-10$ has an amortized complexity of $O(1)$ due to the actual cost of $O(1)$ and the potential increase by at most one unit. The next **if** statement in lines $11-12$ has an amortized complexity of $O(m)$ due to the $print$ operation in line 12. Finally, the assignment in line 13 obviously has $O(1)$ amortized complexity.

Thus, the amortized cost of the body of the outer **while** loop is $O(m)$. Since it runs $O(n)$ times, we get an $O(mn)$ estimate for the actual cost of the outer loop.

Summarizing the above, we conclude that the GREEDY-SC-HEURISTIC algorithm has a time complexity of $O(mn)$. It also uses $O(m)$ memory to represent the multi-queue. □

V. Unit Size Clusters

Let us consider a special case of the fuzzy string clustering problem, when all segments must have a length of 1, and, accordingly, the clustering pattern is represented as a sequence of fuzzy properties of string characters.

In [5], an $O(mn)$ time algorithm was proposed to find *all* occurrences of a pattern in a string using extra memory of size $O(mn)$. Observe that we can adapt the GREEDY-SC-HEURISTIC algorithm to *partially* solve this problem in $O(n)$ time using $O(m)$ memory.

Indeed,

- The function $lookAhead(T, i, \alpha, C)$ can be simplified to a function that returns $i+1$ if $T[i+1] \overset{C}{\sim} \alpha$, and -1, otherwise,
- mq can be replaced by its size, and
- $print$ can print the $shift$ of the occurrence from the beginning of the string, i. e. the value of $i-m$,
- $advance$ can be simplified to $i = i+1$.

The GREEDY-SC-HEURISTIC algorithm adapted in this way, after renaming it to the U-GREEDY-SC-HEURISTIC algorithm, can be described as follows:

```
U-GREEDY-SC-HEURISTIC (T, P, n, m, C)
1   π = COMPUTE-PREFIX-FUNCTION(P, m)
2   k = 0 // number of characters matched
3   for i = 1 to n
4       while k > 0 and T[i+1] ~C α
5           k = k − π(k))
6       if T[i+1] ~C α
7           k = k + 1
8       if k == m //entire pattern matched
9           output(i − m)
```

VI. Conclusion

This paper considers the problem of finding clusterings of strings according to a given fuzzy pattern. The considered problem is investigated in the following aspects.

- A heuristic non-deterministic algorithm is proposed that moves from one string segment to another based on tracking the history of pattern prefix matching.
- This algorithm is greedily restricted to obtain a deterministic algorithm that finds a certain number of clusterings in $O(mn)$ time using $O(m)$ memory, where m and n are the sizes of the pattern and the string, respectively.
- It is found that in the special case where the clustering segments must have unit length, the greedy algorithm partially solves the clustering problem in $O(n)$ time using $O(m)$ memory, while a more powerful algorithm that finds all clusterings solves this problem in $O(mn)$ time using $O(mn)$ memory.

References

[1] W. Smyth. *Computing regularities in strings: A survey*. European Journal of Combinatorics, vol. 34, no. 1, pp. 3–14, 2013. doi:10.1016/j.ejc.2012.07.010. Combinatorics and Stringology.

[2] T. Cormen, C. Leiserson, R. Rivest, C. Stein. *Introduction to Algorithms*, 4th edition, The MIT Press, 2022.

[3] A. Kostanyan, A. Karapetyan. *String Matching in Case of Periodicity in the Pattern*. In: Dolinina O, Brovko A, Pechenkin V, Lvov A, Zhmud V, Kreinovich V (eds.), Recent Research in Control Engineering and Decision Making. Springer International Publishing, Cham. ISBN 978-3-030-12072-6, pp. 61–66, 2019.

[4] A. Kostanyan. *Fuzzy string matching with finite automata*. In: 2017 Computer Science and Information Technologies (CSIT), pp. 9–11, 2017. doi:10.1109/CSITechnol.2017.8312128.

[5] A. Kostanyan. *Fuzzy String Matching Using a Prefix Table*. Mathematical Problems of Computer Science, vol. 54, pp. 116–121, 2020. doi:10.51408/1963-0065.

[6] A. Kostanyan, A. Harmandayan. *Segmentation of String to Match a Fuzzy Pattern*, In Proc. of Computer Science and Information Technologies (CSIT) Conference, Yerevan, Armenia, pp. 17-19, 2019.

[7] A. Kostanyan, A. Harmandayan. *Mapping a Fuzzy Pattern onto a String*. In Proceedings on 2019 IEEE Conference "2019 Computer Science and Information Technologies" (CSIT), Yerevan, Armenia, 2019, IEEE Press, USA, pp. 5 - 8, 2019.

[8] A. Piegat. *Fuzzy Modeling and Control*. Physica-Verlag Heidelberg, Springer-Verlag, 2001.