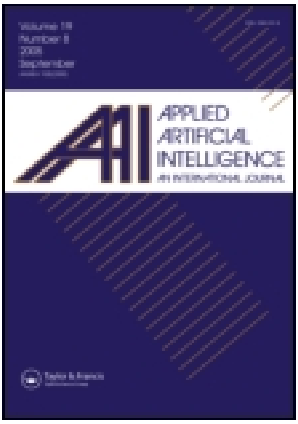

Applied Artificial Intelligence
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/uaai20

A NOTE ON TWO APPLICATIONS OF LOGICAL MATCHING STRATEGY

Sanil Shanker KP [a], Elizabeth Sherly [b] & Jim Austin [c]
[a] Department of Computer Science, University of Kerala, Kerala, India
[b] Indian Institute of Information Technology and Management -- Kerala, Kerala, India
[c] Department of Computer Science, University of York, York, UK

Available online: 20 Sep 2011

Applied Artificial Intelligence, 25:708–720, 2011

ISSN: 0883-9514 print/1087-6545 online
DOI: 10.1080/08839514.2011.606663

A NOTE ON TWO APPLICATIONS OF LOGICAL MATCHING STRATEGY

Sanil Shanker KP[1], Elizabeth Sherly[2], and Jim Austin[3]
[1]*Department of Computer Science, University of Kerala, Kerala, India*
[2]*Indian Institute of Information Technology and Management – Kerala, Kerala, India*
[3]*Department of Computer Science, University of York, York, UK*

□ *This paper proposes Logical Matching Strategy for sequential pattern matching. We show the two real-world applications of the method: (1) locate repeating sequential pattern and (2) alignment-free comparison of sequential pattern of finite length using fuzzy membership values that generate automatically from the number of matches and mismatches. The results show the utility of the method by analyzing DNA sequences taken from the National Center for Biotechnology Information (NCBI) databank. The Logical Matching Strategy can possibly be applied to develop a method of research in sequental pattern matching.*

INTRODUCTION

Sequential pattern matching algorithms devised on the basis of an alphabet set and pattern length are different from one another mainly because of the shifting procedure and the periodicity (Klösgen and Żytkow 2002; Laxman and Sastry 2006). The proposed Logical Matching Strategy for sequential pattern matching is based on the concept of string matching. The following reviews some of these methods. In the Brute Force method, the string matching algorithm compares a pattern character by character in each and every location of the text. Alternatively, it is possible to solve the problems of string matching with the help of finite automata.

For example, in Aho and Corasick (1975), a string matching automation is built from the pattern as a preprocessing step before matching. The text is then scanned through the automation to find occurrences of the pattern in the text. The Knuth-Morris-Pratt algorithm avoids back-tracking on the text

Sanil Shanker KP would like to thank European Research and Educational Collaboration with Asia for financially supporting this research.

Address correspondence to Sanil Shanker KP, Department of Computer Science, University of Kerala, Thiruvananthapuram, Kerala 695581, India. E-mail: sanilshankerkp@gmail.com

when a mismatch occurs by exploiting the knowledge of the matched substring in the text prior to the mismatch (Knuth, Morris, and Pratt 1977). The main peculiarity of the Boyer-Moore algorithm is that some of the characters in the text can be skipped completely without comparing them with the pattern, because it can be shown that they can never contribute to an occurrence of the pattern in the text (Boyer and Moore 1977). The Horspool algorithm is effective when the alphabet size is large and the length of the pattern is small, because the shift value is computed in the preprocessing stage for all the characters in the alphabet set (Horspool 1980).

The designs of the preprocessing phase and searching phase determine the efficiency of the sequential pattern matching algorithm. The characters in the sequence pattern are preprocessed in the preprocessing phase. The information from the preprocessing phase is used in the searching phase in order to reduce the total number of character comparisons.

This paper presents a newly devised algorithm based on logical matching and we show two applications: (1) locate repeating sequential pattern and (2) alignment-free comparison of sequential pattern of finite length using automatically generating fuzzy membership values (Norwich and Turksen 1984; Dombi 1990; Medasani, Kim, and Krishnapuram 1998; Bezdek et al. 1999). The paper points out the fundamental difference of the method with the global alignment using dynamic programming (Eddy 2004).

Preliminaries

Alphabet: Symbols or characters are considered to be the basic elemental building blocks in string matching. An alphabet is a specific set of symbols. It is usually a finite set. For instance, $\Sigma = \{a, b, c, d, e\}$ is an alphabet containing symbols a, b, c, d and e.

String: A string is a sequence of instances of symbols or characters over a finite alphabet Σ. For instance, 'aabaeada' is a string over the alphabet Σ = {a, b, c, d, e}. The length of the string S is a number of instances of the symbols or number of characters in the string. The string S may be represented as $S = s_1\ s_2\ s_3 \ldots\ldots s_m$, where s_i is an instance of a symbol, or character and m is the length of the string S and is represented as $|S|$.

Exact String Matching Problem: Exact string matching is the technique of finding the occurrence of a particular string, called a pattern, in another string called the text. Let $P = p_1\ p_2 \ldots p_m$ and $T = t_1\ t_2 \ldots t_n$ be the pattern and the text of lengths m and n respectively over the same finite alphabet Σ such that $m < n$. We say that the pattern P occurs in text T at the text location k, $1 \leq k \leq n\text{–}m$. For example: Let $T =$ 'cbbababaababacaba' and $P =$ 'baba' be the text and pattern over the finite alphabet $\Sigma = \{$a, b, c$\}$. Here, the pattern 'baba' occurs in text locations 3, 5, and 10, respectively.

Fuzzy Membership Function: If X is a collection of objects, then a fuzzy set A in X is a set of ordered pairs: $A = \{(x, \mu_A(x)): x \in X, \mu_A(x): X \rightarrow [0,1]\}$ where $\mu_A(.)$ is called the membership function of A, and is defined as a function from X into [0,1].

THE LOGICAL MATCHING STRATEGY

Definition: The sequence is arranged so that each character coincides with its corresponding index and then proceeds to match logically the indices of the pattern with those of the text.

APPLICATIONS IN THE CONTEXT OF REAL-WORLD PROBLEMS

We describe the method in the context of the real-world problem, biological sequence pattern searching. This problem remains a computationally difficult problem because the total number of sequences in the underlying databases grows exponentially with the progress of research (Altschul et al. 1994; Pevzner and Waterman 1995; Rigoutsos and Floratos 1998). Algorithms devised for the comparison of molecular sequences are based on the concept of string matching (Gusfield 1997). A substring of a text pattern may exist either as tandem repeating or non-tandem repeating as represented in (i) and (ii), respectively (Benson 1999; Mitra and Acharya 2003): (i) GAAGAAGAA, (ii) GAAGGGAATCATGAA. Here, we proposed two applications of Logical Matching Strategy with sequence datum (see Figure 1).

Input(Sequential Text and Pattern)	
Phase - 1: Pre-processing (Generating Indices of Text and Pattern)	
Phase - 2	
I. Locate Repeating Pattern	II. Alignment-free comparison of Sequential Patterns

FIGURE 1 Two applications of Logical Matching Strategy.

I. Locate Repeating Sequential Pattern

Method

Let $T = t_1\, t_2 \ldots t_n$ and $P = p_1\, p_2 \ldots p_m$ be two strings of lengths n and m, respectively, from the same finite alphabet Σ such that $m < n$.

Phase 1. Generate indices of T and P.

Phase 2. Match the indices of P with the indices of T.

Simulation with Artificial Data

In this simulation, we demonstrate the proposed algorithm with artificial data, where the text is known data, and the pattern is the data to be used as the search query.

Text ⇒ aaaagaagaagaagaagaagaagaagaagaaaataaagaaaagttagccg

Pattern ⇒ gaa   n = 50, m = 3

Phase 1. Generate indices of Text and Pattern.

Shift the Text (see Figure 2) and Pattern (see Figure 3) from right to left so that each character coincides with its corresponding index in its respective column, and arrange the characters with respect to the corresponding indices.

Text ⇒ <A(1, 2, 3, 4, 6, 7, 9, 10, 12, 13, 15, 16, 18, 19, 21, 22, 24, 25, 27, 28, 30, 31, 32, 33, 35, 36, 37, 39, 40, 41, 42, 46); T(34, 44, 45); G(5, 8, 11, 14, 17, 20, 23, 26, 29, 38, 43, 47, 50); C(48, 49)>

Pattern ⇒ <A(2, 3); G(1)>

Phase 2. Match the indices of Pattern with the indices of Text as

Indices of Text ← Characters of Text ← Characters of Pattern ← Indices of Pattern

Characters of Pattern ⇑	*Indices of Pattern* ⇑	*Characters of Text* ⇑	*Indices of Text* ⇑
G	(1)	:G	(**5**, **8**, **11**, **14**, **17**, **20**, **23**, **26**, **29**, **38**, 43, 47, 50)
A	(2)	:A	(1, 2, 3, 4, **6**, 7, **9**, 10, **12**, 13, **15**, 16, **18**, 19, **21**, 22, **24**, 25, **27**, 28, **30**, 31, 32, 33, 35, 36, 37, **39**, 40, 41, 42, 46)
A	(3)	:A	(1, 2, 3, 4, 6, **7**, 9, **10**, 12, **13**, 15, **16**, 18, **19**, 21, **22**, 24, **25**, 27, **28**, 30, **31**, 32, 33, 35, 36, 37, 39, **40**, 41, 42, 46)

Here, the pattern (g, a, a) repeats in the positions (5, 6, 7), (8, 9, 10), (11, 12, 13), (14, 15, 16), (17, 18, 19), (20, 21, 22), (23, 24, 25), (26, 27, 28), (29, 30, 31) in a tandem way and (38, 39, 40) in a non-tandem way with respect to the text (Figure 4).

II. Alignment-Free Sequence Comparison

The alignment-free sequence comparison methods are still in the early stage of development compared to alignment-based methods (Blaisdell 1986; Vinga and Almeida 2003). To compare the sequential patterns, we compute the score using fuzzy membership values that generate automatically from the number of matches and mismatches by Logical Matching Strategy. The alignment-free sequence comparison algorithm using Logical Matching Strategy works as follows:

	A	T	G	C
50			*	
49				*
48				*
47			*	
46	*			
45		*		
44		*		
43			*	
42	*			
41	*			
40	*			
39	*			
38			*	
37	*			
36	*			
35	*			
34		*		
33	*			
32	*			
31	*			
30	*			
29			*	
28	*			
27	*			
26			*	
25	*			
24	*			
23			*	
22	*			
21	*			
20			*	
19	*			
18	*			
17			*	
16	*			
15	*			
14			*	
13	*			
12	*			
11			*	
10	*			
9	*			
8			*	
7	*			
6	*			
5			*	
4	*			
3	*			
2	*			
1	*			

FIGURE 2 Text (Phase 1).

Method

Let $T = t_1\ t_2 \ldots t_n$ and $P = p_1\ p_2 \ldots p_m$ be Text and Pattern of lengths n and m, respectively, where $n \geq m$.

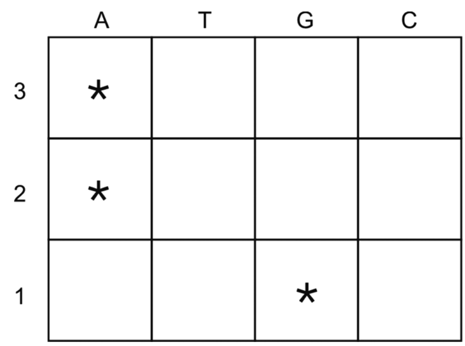

FIGURE 3 Pattern (Phase 1).

Phase 1. Generate Indices of Text and Pattern.

Phase 2. Using Logical Matching Strategy, compute the number of Matches (Text, Pattern) and Mismatches (Text, Pattern).

Compute score, S(Text, Pattern) =

Match in Text $*$ $\mu_{\text{Match (Pattern)}}$[Pattern] $-$ Mismatch in Text $*$ $\mu_{\text{Mismatch (Pattern)}}$[Pattern] where, $\mu_{\text{Match (Pattern)}}$[Pattern] $+$ $\mu_{\text{Mismatch (Pattern)}}$ [Pattern] $= 1$

(See Appendix)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Seq1 ⇒	C	G	A	G	A	A	C	T	G	A	A	T	G	A	T
	|	#	|	|	|	|	#	#	|	|	|	#	|	|	|
Seq2 ⇒	C	T	A	G	A	A	G	A	G	A	A	C	G	A	T

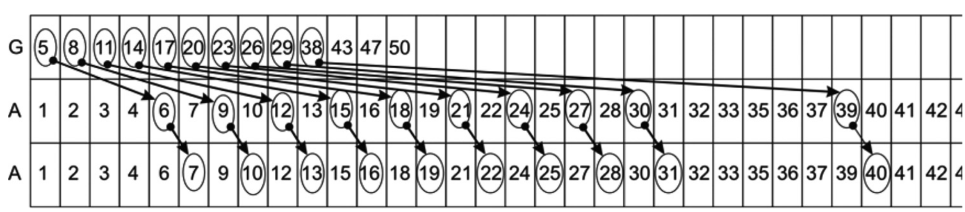

FIGURE 4 Matching regions: patterns in the text.

Simulation with Artificial Data

We demonstrate the proposed method with two artificial sequences: Seq1 <CGAGAACTGAATGAT> and Seq2 <CTAGAAGAGAACGAT>.

Phase 1. Generate indices of Seq1 and Seq2.

Seq1 ⇒	<A(3, 5, 6, 10, 11, 14);	Seq2 ⇒	<A(3, 5, 6, 8, 10, 11, 14);
	T(8, 12, 15);		T(2, 15);
	G(2, 4, 9, 13);		G(4, 7, 9, 13);
	C(1, 7)>		C(1, 12)>

Phase 2. Compute Match(Seq1, Seq2) and Mismatch(Seq1, Seq2) using Logical Matching Strategy.

C:	**1**	7					–	1st Match in Seq1
T:	8	12	15				–	1st Mismatch in Seq1
A:	**3**	5	6	10	11	14	–	2nd Match in Seq1
G:	2	**4**	9	13			–	3rd Match in Seq1
A:	3	**5**	6	10	11	14	–	4th Match in Seq1
A:	3	5	**6**	10	11	14	–	5th Match in Seq1
G:	2	4	9	13			–	2nd Mismatch in Seq1
A:	3	5	6	10	11	14	–	3rd Mismatch in Seq1
G:	2	4	**9**	13			–	6th Match in Seq1
A:	3	5	6	**10**	11	14	–	7th Match in Seq1
A:	3	5	6	10	**11**	14	–	8th Match in Seq1
C:	1	7					–	4th Mismatch in Seq1
G:	2	4	9	**13**			–	9th Match in Seq1
A:	3	5	6	10	11	**14**	–	10th Match in Seq1
T:	8	12	**15**				–	11th Match in Seq1

Score = Match in Seq1 $*$ (Match in Seq 2/Length of Seq2) –
MisMatch in Seq1 $*$ (MisMatch in Seq2/Length of Seq2)
$= \text{Match in Seq1} * \mu_{\text{Match(Seq2)}}$ [Seq2] $-$ MisMatch in Seq1 $* \mu_{\text{Mismatch (Seq2)}}$ [Seq2]
$= 11 * (11/15) - 4 * (4/15) = 11 * 0.733 - 4 * 0.266 \approx 7.$

EXPERIMENTAL RESULTS

For testing the proposed method, the program has been written in **C** language under Linux platform. (See Figures 5 and 6). The results (Table 1) show the utility of the algorithm to locate exact tandem repeat by analyzing DNA sequences of various sizes from real data (www.ncbi.nlm.nih.gov). To evaluate the alignment-free sequence comparison algorithm using Logical Matching Strategy, the method was tested against DNA sequences, the inputs have been taken from the Locus ACU90045 as common text and ACU90045, PAU90054, HSU90049, LPU90051, NAU90053, AVU90046, DCU90047, LEU90050, DPU90048, MGU90052 as patterns of common range 541–560 (20 bp) from the NCBI databank and we demonstrate the difference with the global alignment using dynamic programming (Table 2).

```
begin
initialize T and P with n ← |T |, m ← |P|
/* Input: Sequence T and P */
/* Phase 1 */
for i ← 0 to m-1
generate ( indices of P)
end for
for i ← 0 to n-1
generate ( indices of T)
end for
/* Phase 2 */
for i ← 0 to n-1
match ( indices of P) with (indices of T)
end for
end
```

FIGURE 5 Pseudocode: locate repeating pattern.

```
begin
initialize T and P with n ← |T |, m ← |P|
/* Input: Sequence T and P */
/* Phase 1 */
for i ← 0 to m-1
generate ( indices of P)
end for
for i ← 0 to n-1
generate ( indices of T)
end for
/* Phase 2 */
for i ← 0 to n-1
compute Match (T, P) and Mismatch (T, P)
Score(T, P) ← Match in Text * μ_Match(Pattern)[Pattern] - Mismatch in Text * μ_Mismatch(Pattern)[Pattern]
end for
end
```

FIGURE 6 Pseudocode: alignment-free comparison of sequential pattern.

TABLE 1 Locate the Positions of Repeating Pattern Using Logical Matching Strategy

Disease	Gene designation	Repeat motif	Locus	Repeating positions: Exact tandem repeat	Repeating positions: Non-tandem repeat
Friedreich ataxia	FRDA	GAA	AH003505S1 Region: 2141 . . . 2465 (425 bp)	(145, 146, 147); (148, 149, 150); (151, 152, 153); (154, 155, 156); (157, 158, 159); (160, 161, 162); (163, 164, 165); (166, 167, 168); (169, 170, 171)	(49, 50, 51); (101, 102, 103); (178, 179, 180); (243, 244, 245); (388, 389, 390)
Huntington disease	HD	CAG	NM_002111 Region: 121 . . . 300 (180 bp)	(77, 78, 79); (80, 81, 82); (83, 84, 85); (86, 87, 88); (89, 90, 91); (92, 93, 94); (95, 96, 97); (98, 99, 100); (101, 102, 103); (104, 105, 106); (107, 108, 109); (110, 111, 112); (113, 114, 115); (116, 117, 118); (119, 120, 121); (122, 123, 124); (125, 126, 127); (128, 129, 130); (131, 132, 133); (134, 135, 136); (137, 138, 139)	(143, 144, 145)
Fragile XA Syndrome	FMR1	CGG	HUMFMR1S Region: 13821 . . . 13946(126 bp)	(13, 14, 15); (16, 17, 18); (19, 20, 21); (22, 23, 24); (25, 26, 27); (28, 29, 30); (31, 32, 33); (34, 35, 36); (37, 38, 39); (40, 41, 42); (46, 47, 48); (49, 50, 51); (52, 53, 54); (55, 56, 57); (58, 59, 60); (61, 62, 63); (64, 65, 66); (67, 68, 69); (70, 71, 72)	(5, 6, 7); (103, 104, 105); (109, 110, 111); (117, 118, 119)
Myotonic dystrophy 2	DM2	CCTG	AY329622 Region: 11224 . . . 11520 (297 bp)	(162, 163, 164, 165); (166, 167, 168, 169); (170, 171, 172, 173); (174, 175, 176, 177); (178, 179, 180, 181); (182, 183, 184, 185); (186, 187, 188, 189); (190, 191, 192, 193); (206, 207, 208, 209); (210, 211, 212, 213); (214, 215, 216, 217); (218, 219, 220, 221); (222, 223, 224, 225); (226, 227, 228, 229); (230, 231, 232, 233); (234, 235, 236, 237); (238, 239, 240, 241)	(198, 199, 200, 201)
DRPLA	DRPLA	CAG	HSU23851 Region: 1381 . . . 1747 (367 bp)	(152, 153, 154); (155, 156, 157); (158, 159, 160); (161, 162, 163); (164, 165, 166); (167, 168, 169); (170, 171, 172); (173, 174, 175); (176, 177, 178); (179, 180, 181)	(8, 9, 10); (52, 53, 54); (66, 67, 68); (95, 96, 97); (114, 115, 116); (140, 141, 142); (146, 147, 148); (306, 307, 308); (312, 313, 314); (334, 335, 336); (338, 339, 340); (349, 350, 351); (357, 358, 359)
Kennedy disease	AR	CAG	NM_000044 Region: 1261 . . . 1620 (360 bp)	(27, 28, 29); (30, 31, 32); (33, 34, 35); (36, 37, 38); (39, 40, 41); (42, 43, 44); (45, 46, 47); (48, 49, 50); (51, 52, 53); (54, 55, 56); (57, 58, 59); (60, 61, 62); (63, 64, 65); (66, 67, 68); (69, 70, 71); (72, 73, 74); (75, 76, 77); (78, 79, 80); (81, 82, 83); (84, 85, 86);	(11, 12, 13); (107, 108, 109); (169, 170, 171); (195, 196, 197); (207, 208, 209); (213, 214, 215); (253, 254, 255); (278, 279, 280); (304, 305, 306); (328, 329, 330)

				(87, 88, 89); (90, 91, 92); (111, 112, 113); (114, 115, 116); (117, 118, 119); (120, 121, 122); (123, 124, 125); (126, 127, 128); (294, 295, 296); (297, 298, 299)	
Myotonic dystrophy 1	DM1	CTG	NM_004409 Region: 2342 . . . 2511 (170 bp)	(2, 3, 4); (5, 6, 7); (8, 9, 10); (11, 12, 13); (14, 15, 16); (17, 18, 19); (20, 21, 22); (23, 24, 25); (26, 27, 28); (29, 30, 31); (32, 33, 34); (35, 36, 37)	(73, 74, 75); (81, 82, 83); (99, 100, 101); (107, 108, 109)
SCA7	SCA7	CAG	NM_000333 Region: 481 . . . 720 (240 bp)	(2, 3, 4); (5, 6, 7); (161, 162, 163); (164, 165, 166); (167, 168, 169); (170, 171, 172); (173, 174, 175); (176, 177, 178); (179, 180, 181); (182, 183, 184); (185, 186, 187); (188, 189, 190); (215, 216, 217); (218, 219, 220)	(32, 33, 34); (100, 101, 102); (141, 142, 143); (203, 204, 205); (209, 210, 211)
Oculo-pharyngeal muscular dystrophy	PABPN1	GCG	NM_004643 Region: 1201 . . . 1380 (180 bp)	(58, 59, 60); (61, 62, 63); (77, 78, 79); (80, 81, 82); (86, 87, 88); (89, 90, 91); (92, 93, 94); (95, 96, 97); (98, 99, 100); (111, 112, 113); (148, 149, 150); (151, 152, 153); (154, 155, 156)	(28, 29, 30); (37, 38, 39); (113, 114, 115); (122, 123, 124); (126, 127, 128)
Fragile XE syndrome	FMR2	GCC	HSU48436 Region: 1 . . . 130 (130 bp)	(2, 3, 4); (5, 6, 7); (20, 21, 22); (23, 24, 25); (26, 27, 28); (29, 30, 31); (32, 33, 34); (35, 36, 37); (38, 39, 40); (41, 42, 43); (44, 45, 46); (47, 48, 49); (50, 51, 52); (53, 54, 55); (56, 57, 58); (59, 60, 61); (62, 63, 64); (68, 69, 70); (71, 72, 73); (108, 109, 110); (111, 112, 113); (117, 118, 119); (120, 121, 122)	(14, 15, 16); (80, 81, 82); (85, 86, 87); (90, 91, 92); (96, 97, 98); (102, 103, 104); (126, 127, 128)

TABLE 2 The Results of Alignment–free Comparison Using Logical Matching Strategy and Global Alignment Using Dynamic Programming

Alignment-free comparison using Logical Matching Strategy			Pattern in text using logical matching strategy (%)	Global alignment using dynamic programming
Locus		Region: 541–560 (20 bp)		
Text:	ACU90045	cgacctctggacaggccact	100%	cgacctctggacaggccact
Pattern:	ACU90045	cgacctctggacaggccact		cgacctctggacaggccact
Text:	ACU90045	cgacctctggacaggccact	45%	cga-cctctg-gacaggccact
Pattern:	PAU90054	cgacccactgagaaacctct		cgacccactgaga-a-acctct
Text:	ACU90045	cgacctctggacaggccact	35%	cgacc-tctggac-aggccact
Pattern:	HSU90049	cgaccaactgacaaggctct		cgaccaact-gacaagg-ctct
Text:	ACU90045	cgacctctggacaggccact	30%	cg-acctctggacaggccact
Pattern:	LPU90051	cgtcccactgacaagcctct		cgtcccact-gacaagcctct
Text:	ACU90045	cgacctctggacaggccact	25%	cgacc-tctggac-aggccact
Pattern:	NAU90053	cgcccaactgacaaggctct		cgcccaact-gacaagg-ctct
Text:	ACU90045	cgacctctggacaggccact	20%	cgacctctggac-aggccact
Pattern:	AVU90046	agacctctagacaaagctct		agacctctagacaaag-ctct
Text:	ACU90045	cgacctctggacaggccact	20%	Cgacctctggacaggccact
Pattern:	DCU90047	aggcctttggacaaacctct		aggcctttggacaaacctct
Text:	ACU90045	cgacctctggacaggccact	20%	cgacctc-tggacaggccact
Pattern:	LEU90050	agacccgttgacaagcccct		agacc-cgttgacaagcccct
Text:	ACU90045	cgacctctggacaggccact	10%	cgacc-tctggacaggccact
Pattern:	DPU90048	agaccagttgacaaaccttt		agaccagt-t-gacaaaccttt
Text:	ACU90045	cgacctctggacaggccact	10%	cgacct-ctggacaggccact
Pattern:	MGU90052	agacctactgacaaacctct		agacctact-gacaaacctct

Difference with Global Alignment Using Dynamic Programming

To align two sequences using global dynamic programming, the optimal scores are computed in a two-dimensional matrix, with runtime O(mn). On the other hand, in Logical Matching Strategy, the sequence is arranged so that each character coincides with its corresponding index and then proceeds to match logically the indices of the subsequence with those of the text. In the Phase 1 of the algorithm, the time complexity is $O(m + n)$ and in the Phase 2, the computational time depends on the length of the text.

CONCLUSION

We have presented the method of Logical Matching Strategy and its two applications (1) locate repeating sequential pattern and (2) alignment-free comparison of sequential pattern of finite length. The method provides a solution to the problem of locating the exact tandem repeat and finding alignment-free similarities between two finite sequences by calculating the score using automatically generating fuzzy membership values. The Logical Matching Strategy can be applied to develop a method of research in sequence analysis to locate biologically meaningful segments.

REFERENCES

Aho, A. V., and M. J. Corasick. 1975. Efficient string matching: An aid to bibliographic search. *Communications of the ACM* 18:333–340.

Altschul, S. F., M. S. Boguski, W. Gish, and J. C. Wootton. 1994. Issues in searching molecular sequence databases. *Nature Genetics* 6:119–129.

Benson, G. 1999. Tandem repeats finder: A program to analyze DNA sequences. *Nucleic Acids Research* 27:573–580.

Bezdek, J. C., J. M. Keller, R. Krisnapuram, and N. R. Pal. 1999. *Fuzzy models and algorithm for pattern recognition and image processing*. Massachusetts: Kluwer Academic Publishers.

Blaisdell, B. E. 1986. A measure of the similarity of sets of sequences not requiring sequence alignment. In: *Proceedings of National Academy of Sciences* 83 (14): 5155–5159. USA.

Boyer, R. S., and J. S. Moore. 1977. A fast string searching algorithm. *Communications of the ACM* 20:762–772.

Dombi, J. 1990. Membership function as an evaluation. *Fuzzy Sets and Systems* 35:1–21.

Eddy, S. R. 2004. What is dynamic programming? *Nature Biotechnology*. 22:909–910.

Gusfield, D. 1997. *Algorithms on strings, trees and sequences: Computer science and computational biology*. New York: Cambridge University Press.

Horspool, R. N. 1980. Practical fast searching in strings. *Software: Practice & Experience* 10 (6): 501–506.

Klösgen, W., and J. M. Żytkow. 2002. *Handbook of data mining and knowledge discovery*. New York: Oxford University Press.

Knuth, D. E., J. H. Morris Jr., and V. R. Pratt. 1977. Fast pattern matching in strings. *SIAM Journal on Computing* 6 (1): 323–350.

Laxman, S., and P. S. Sastry. 2006. A survey of temporal data mining. *Sadhana* 31 (Part 2): 173–198.

Medasani, S., J. Kim, and R. Krishnapuram. 1998. An overview of membership function generation techniques for pattern recognition. *International Journal of Approximate Reasoning* 19:391–471.

Mitra, S., and T. Acharya. 2003. *Data mining multimedia, soft computing and bioinformatics*. 143–180. New Jersey: John Wiley & Sons, Inc.

Norwich, A. M., and I. B. Turksen. 1984. A model for the measurement of membership and the consequence of its empirical implementation. *Fuzzy Sets and Systems* 12:1–25.

Pevzner, P. A., and M. S. Waterman. 1995. Open combinatorial problems in computational molecular biology. In: *Proceedings of the third Israel symposium on theory of computing and systems*, 158–173. IEEE Computer Society Press.

Rigoutsos, I., and A. Floratos. 1998. Combinatorial pattern discovery in biological sequences. *Bioinformatics* 14 (1): 55–67.

Vinga, S., and J. Almeida. 2003. Alignment-free sequence comparison—a review. *Bioinformatics* 19:513–523.

APPENDIX

Let $T = t_1\ t_2 \ldots t_n$ and $P = p_1\ p_2 \ldots p_m$ be two strings of lengths n and m respectively from the same finite alphabet Σ such that $n \geq m$.

When P compares (alignment-free) with T gives r matches and s mismatches, $r + s = m$

$\mu_{\text{Match(Pattern)}}$[Pattern] + $\mu_{\text{Mismatch(Pattern)}}$[Pattern] = Match in Pattern/Length of Pattern

+ Mismatch in Pattern/Length of Pattern

$= r/m + s/m = r + s)/m = m/m = 1$

Example: 1

Text⇒	A	T	G	C
	|	|	|	|
Pattern⇒	A	T	G	C

$\mu_{Match(Pattern)}[Pattern] = Match in Pattern/Length of Pattern = 4/4 = 1$,
$\mu_{Mismatch(Pattern)}[Pattern] = Mismatch in Pattern/Length of Pattern = 0/4 = 0$

Score, S(Text, Pattern) =

Match in Text $* \mu_{Match(Pattern)}[Pattern]$ − Mismatch in Text $* \mu_{Mismatch(Pattern)}[Pattern]$

$= 4 * 1 - 0 = 4$

Example: 2

Text⇒	A	T	G	C
	|	|	|	#
Pattern⇒	A	T	G	G

$\mu_{Match(Pattern)}[Pattern] = 3/4 = 0.75$, $\mu_{Mismatch(Pattern)}[Pattern] = 1/4 = 0.25$

$S(Text, Pattern) = 3 * 0.75 - 1 * 0.25 = 2$

Example: 3

Text⇒	A	T	G	C
	|	#	|	#
Pattern⇒	A	C	G	G

$\mu_{Match(Pattern)}[Pattern] = 2/4 = 0.5$, $\mu_{Mismatch(Pattern)}[Pattern] = 2/4 = 0.5$

$S(Text, Pattern) = 2 * 0.5 - 2 * 0.5 = 0$

Example: 4

Text⇒	A	T	G	C
	#	#	|	#
Pattern⇒	T	A	G	G

$\mu_{Match(Pattern)}[Pattern] = 1/4 = 0.25$, $\mu_{Mismatch(Pattern)}[Pattern] = 3/4 = 0.75$

$S(Text, Pattern) = 1 * 0.25 - 3 * 0.75 = -2$