OE Matching Algorithm for Searching Biological Sequences

Ahmad Fadel Klaib and Hugh Osborne
Informatics Department, University of Huddersfield, Queensgate, Huddersfield, HD1 3DH, UK
E-mail: {a.klaib/ h.r.osborne} @hud.ac.uk

Abstract

String matching algorithms play a key role in many computer science problems, and in the implementation of computer software. This problem has received, and continues to receive a great deal of attention due to various applications in text manipulation, information retrieval, speech recognition, image and signal processing and computational biology. In this study, we propose a new algorithm called the Odd and Even algorithm (OE). OE combines an enhanced preprocessing phase from the Berry Ravindran algorithm with our proposed new searching phase procedure. This variety of searching order allows our proposed algorithm to reduce the number of comparison characters and enhances the searching response time. Experimental results show that OE algorithm offers a smaller number of comparisons and offers improved elapsed searching time when compared to other well-known algorithms for searching any length of alphabets and patterns. The proposed algorithm is applicable to searching protein sequence databases as well as any other string searching applications.

1. Introduction

Protein data can be found in many different forms such as sequences data, structure data, microarray data, and image data. Proteins are fundamental to the structure and the function of all living cells and viruses. Protein compounds are made of 20 different amino acids arranged in a linear chain. They are "complex organic compounds that consist of amino acids joined by peptide bonds" [1]. Computational biology and chemistry that use computational methods handle large amount of data. Protein sequence technologies have produced many extremely large sets of biological data which need faster techniques to process them.

The Swiss-Prot database is one of the main protein sequence databases containing descriptions of protein functions, domain structures, post-translational modifications and variants with a low level of redundancy and a high level of integration with other databases [2].

String-matching algorithms aim to find all occurrences of a given pattern $P= p_1p_2\ldots p_m$ in a text $T=t_1t_2\ldots t_n$. They work as follows: they first align the left ends of the pattern and the text, then compare text characters with pattern characters and after a mismatch between the pattern and the text or a whole match between them they shift the pattern to the right. This procedure is repeated until the right end of the pattern reaches the right end of the text.

Most string-matching algorithms consist of a preprocessing phase and a searching phase to search for the pattern in the given text. The preprocessing phase analyses the characters in the pattern in order to use this information to determine the pattern shift in case of a mismatch or a whole match, with the aim of reducing the total number of character comparisons, while the searching phase defines the order of comparison of characters in each attempt between the pattern and the text. The main aim in algorithm development is to decrease the searching phase during each attempt and to increase the shifting value of the pattern.

String matching algorithms can be classified into seven categories according to the preprocessing function in the algorithm [3]. The first category, e.g. the Brute Force algorithm (BF) [4], shifts the pattern only one position at each attempt. The second category, which includes the Boyer-Moore algorithm (BM) [5]-[7] and the Fast Search algorithm (FS) [8], uses two preprocessing functions. The third category, a good example of which is the Boyer Moore Horspool algorithm (BMH) [9]-[11], uses one preprocessing function based on the rightmost character in the current window. The fourth category, e.g. the Quick Search algorithm (QS) [12], uses one preprocessing function based on the character next to the current window. The fifth category, such as the Berry–Ravindran algorithm (BR) [13], uses one preprocessing function based on the next two characters to the current window. The sixth category, e.g. the Karp-Rabin algorithm (KR) [14] and the Zhu Takaoka algorithm (ZT) [15], uses a preprocessing hashing function. The final category uses hybrid algorithms and includes the SSABS [16], TVSBS [17], ZTMBH [18], BRFS [19], BRBMH [20] and the BRQS [3] algorithms.

The paper is organized as follows: section II includes a survey of the main string-matching algorithms. Section III describes the proposed algorithm and its two main phases. Section IV provides a working example. Section V includes the experimental results with an evaluation of our new algorithm comparing it to other common string-

matching algorithms. Finally the conclusion is presented in section VI.

2. Related works

This section includes a survey of the main string-matching algorithms: the BR, KMP, BM, BMH, KP, ZT, QS, BR, FS, SSABS, TVSBS, ZTMBH, BRFS, BRBMH and the BRQS algorithms [21], [22]. Table 1 summarizes and compares these algorithms:

Algorithm Name	Year	Comparison Order	Preprocessing Time	Searching Time	Main Characteristics
Brute Force Algorithm	Very Old	From left to right	N/A	O(mn)	Shifts the pattern only a position each attempt. Not an optimal algorithm since it does not use the information that could be gained from the last comparison made
Knuth-Morris-Pratt Algorithm	1974	From left to right	O(m)	O(mn)	Uses the notion of the border of the string. It increases performance, decreases delay, and decreases searching time compared to the Brute Force algorithm. It is efficient for large alphabets
Boyer-Moore Algorithm	1977	From right to left	O(m+σ)	O(mn)	Uses two preprocessing functions; the good-suffix shift and the bad-character shift. It is not very efficient for small alphabets
Horspool Algorithm	1980	From left to right	O(m+σ)	O(mn)	Uses the Horspool bad-character preprocessing function based on the rightmost character in the current window. It is a simplification of the Boyer-Moore algorithm. It is faster than, and easier to implement than the Boyer-Moore algorithm
Karp-Rabin Algorithm	1984	From left to right	O(m)	O(mn)	Uses the Karp-Rabin preprocessing hashing function. It is very effective for multiple pattern matching in one-dimensional string matching
Zhu-Takaoka Algorithm	1989	From right to left	$O(m+\sigma^2)$	O(mn)	Uses the Zhu-Takaoka preprocessing hashing function. It is very effective for multiple pattern matching in two-dimensional string matching
Quick-Search Algorithm	1990	From right to left	O(m+σ)	O(mn)	Uses the Quick-Search bad-character preprocessing function based on the next character to the current window. It is especially fast for short patterns
Berry–Ravindran Algorithm	1999	From left to right	$O(m+\sigma^2)$	O(mn)	Uses the Berry-Ravindran preprocessing function based on the next two characters after the current window in order to increase the shifting value of the pattern
Fast Search Algorithm	2003	From right to left	$O(m+\sigma^2)$	O(mn)	Uses the bad-character function only if the mismatching character is the last character of the pattern, otherwise the good-suffix function is to be used. It is efficient in very short patterns
SSABS Algorithm	2004	From right to left	O(m+σ)	O(m(n-m+1))	Uses the Quick-Search bad-character preprocessing function. It scans the pattern with the text starting from the

					right most then the left most then it scans the next last character and goes backward to the left.
TVSBS Algorithm	2006	From right to left	$O(m+\sigma^2)$	$O(m(n - m + 1))$	A combination of the Berry-Ravindran and the SSBAS algorithms. It scans the pattern with the text using the searching phase of the SSABS algorithm. Uses the Berry-Ravindran preprocessing function
ZTMBH Algorithm	2008	From left to right	$O(m+\sigma^2)$	$O(mn)$	A combination of the Zhu Takaoka and the Boyer-Moore Horspool algorithms. It scans the pattern using the searching phase of the BMH algorithm. Uses the Zhu-Takaoka preprocessing hashing function.
BRFS Algorithm	2008	From right to left	$O(m+\sigma^2)$	$O(mn)$	A combination of the Berry-Ravindran and the Fast Search algorithms. It scans the pattern using the searching phase of the FS algorithm. Uses the Berry-Ravindran Preprocessing Function.
BRBMH Algorithm	2008	From left to right	$O(m+\sigma)$	$O(mn)$	Enhances the preprocessing Berry-Ravindran algorithm and combines it with the BMH algorithm. It scans the pattern using the searching phase of the BMH algorithm. Uses the enhanced Berry-Ravindran Preprocessing Function.
BRQS Algorithm	2008	From right to left	$O(m+\sigma)$	$O(mn)$	Uses the enhanced Berry-Ravindran Preprocessing Function and combines it with the QS algorithm. It scans the pattern using the searching phase of the QS algorithm.

Table 1: Summary of common string matching algorithms

3. Proposed algorithm

3.1. Preprocessing phase

In this phase, the proposed algorithm uses our enhanced brBc preprocessing function by counting the shifting values for each character in the pattern and storing them in the one-dimensional brBc array [20]. Fig. 1 shows the pseudo code for the pre-processing phase.

3.2. Searching phase

After implementing several algorithms, we found out that the best order in the searching phase is to compare the pattern and the text window characters from right to left.

Our proposed algorithm searches the pattern from right to left with new order. It starts with the last character of the text window and the pattern, and after a match, it moves backward to compare the odd index positions of pattern and text window characters. If all these characters match, it will return and compare whole even index pattern and text window characters. In case of a mismatch or whole match during the comparison in odd or even positions it uses our enhanced brBc pre-processing function to shift the pattern. Fig. 2 shows the pseudo code for the searching process phase.

```
/*Pre-Processing Phase*/
FOR i=0  TO m-2
SET brBcShiftArray[i] TO m-i
END FOR
IF  t[end]+1 = p[end] THEN
SET shiftvalue to 1
ELSE IF t[end]+2 = p[start] THEN
COMPUTE shiftvalue AS m+1
ELSE
COMPUTE shiftValue AS  m+2
END IF
```

Fig. 1 OE Pre-processing Phase

```
/*Searching Phase*/
WHILE odd>=t[start] AND p[odd]=t[odd]
DECREMENT odd - 2
ENDWHILE
IF odd > t[start] THEN
WHILE even>=t[start] AND p[even]=t[even]
DECREMENT even – 2
ENDWHILE
ELSE
SET notMatch TO true
SET textPortion TO( t[end] +1) + ( t[end] +2)
CALL brBcShiftArray WITH textPortion
END IF
IF wholeMatch = true THEN
CALL brBcShiftArray WITH textPortion
END IF
END WHILE

/*Searching in the brBcShiftArray*/
FOR i=0 TO m
IF p[i] = textPortion THEN
SET shiftValue TO brBcShiftValue
END IF
END FOR
```

Fig. 2 OE Searching Phase

4. Working example

A sample file has been taken from the Swiss-Prot database which consists of 8740 proteins [23]. The following example illustrates our proposed algorithm:

Given:

Pattern(p)="LAVKLATAIVLA", length (m) =12

Text(n)="KRFDSLYKQILAMGIFSIANQHIVLAV KLATAIVLATHTSPVVPVTTPGTKPDLNASFVSAN AE", length(n)=64

4.1. Preprocessing phase

The shift values for the pattern characters are calculated according to Fig. 1. Table 2 shows the brBc one-dimensional array for the pattern characters.

4.2. Searching phase

The searching phase in this example is implemented according to Fig. 2. The following tables illustrate the searching phase for the given pattern (p) in the sample text (t).

4.2.1. Attempt 1: in this attempt, Table 3 shows that $t_0 – t_{11}$ is the current text that is compared with the pattern $p_0 – p_{11}$. The t_{11} comparison with p_{11} has matched, so the algorithm will move backward to the next odd index which compares t_9 to p_9 which causes a mismatch. The pattern will be shifted to the right according to the pre-counted shifting value for the next two characters of the current window which are t_{12} and t_{13} (MG) and in this attempt will shift by 14 positions.

4.2.2. Attempt 2: in this attempt, Table 4 shows that $t_{14} – t_{25}$ is the current text which is compared with the $p_0 – p_{11}$. The t_{25} comparison with p_{11} has matched, so the algorithm will move backward to the next odd indices which are t_{23} and p_9 which match. The next comparison is between t_{21} and p_7 which causes a mismatch. The pattern will be shifted to the right according to the pre-counted shifting value for the next two characters of the current window which are t_{26} and t_{27} (VK) giving a shift of 10 positions.

4.2.3. Attempt 3: in this attempt, Table 5 shows that $t_{14} – t_{25}$ is the current text that is compared with the $p_0 – p_{11}$. The first comparison between t_{35} and p_{11} produces a match, so the algorithm will move backward to the next odd indexes which are t_{33} and p_9 which produces a match again. The next comparison is between t_{31} and p_7 which also produces a match. Then the same procedure is repeated until the all the odd indices match. It will then go back to the first even indexes (from the right) which they are t_{34} and p_8. This produces a match also and it will proceed to move back to compare further even indices. After a whole match between pattern and text it shifts the pattern to the right according to the pre-counted shifting value for the next two characters to the current window which are t_{36} and t_{37} (TH). In this attempt it will be 14 positions.

4.2.4. Attempt 4: in this attempt, Table 6 shows that $t_{38} – t_{49}$ is the current text that is compared with the $p_0 – p_{11}$. The comparison of t_{49} with p_{11} causes a mismatch. The pattern will be shifted to the right according to the pre-counted shifting value for the two characters next to the current window which are t_{50} and t_{51} (KP) and in this attempt it will be 14 positions. But in this case the algorithm will cancel the pattern shifting since the length of the remaining text is 13 which is less than the pattern length.

LA	AV	VK	KL	LA	AT	TA	AI	IV	VL	LA
12	11	10	9	8	7	6	5	4	3	2

Table 2: Preprocessing phase

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
K	R	F	D	S	L	Y	K	Q	I	L	A	M	G	I	F	…	…
									2		1						
L	A	V	K	L	A	T	A	I	V	L	A						

Table 3: Attempt 1 in searching phase

14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
I	F	S	I	A	N	Q	H	I	V	F	A	V	K	L	A	…	…
							3		2		1						
L	A	V	K	L	A	T	A	I	V	L	A						

Table 4: Attempt 2 in searching phase

24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
L	A	V	K	L	A	T	A	I	V	L	A	T	H	T	S	…	…
12	6	11	5	10	4	9	3	8	2	7	1						
L	A	V	K	L	A	T	A	I	V	L	A						

Table 5: Attempt 3 in searching phase

38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55
T	S	P	V	V	P	V	T	T	P	G	T	K	P	D	L	…	…
											1						
L	A	V	K	L	A	T	A	I	V	L	A						

Table 6: Attempt 4 in searching phase

5. Implementation

In this research, local machine (Dell Intel(R) Core™ 2 Duo Processors, CPU (1.80 GHz), 2 GB RAM and Windows Vista 32-bit Operating System) were used with Java SE 6 (JDK) software with update 10 including the Java Runtime Environment (JRE) and command-line development tools for producing applets and applications software.

6. Experimental results

To evaluate our new algorithm, we implemented it based on the code in Fig.1 and Fig.2. Additionally, we implemented six other algorithms; two of them are our previous algorithms using the enhanced brBc preprocessing phase; two developed in 2008 using the original BR as the fastest algorithms to search the proteins; one of them using the rightmost character in the current window and the last one using only one character next to the current text window.

The performance of the algorithm proposed in this research is evaluated using the number of comparison between the pattern and the text and the elapsed time of searching.

A sample file has been taken from the Swiss-Prot database which consists of 8740 proteins to test the efficiency of our algorithm compared to other algorithms. Table 7 below shows the number of comparison and Fig. 3 below shows the average elapsed time (s.) for searching different length of patterns in the protein sample file.

Pattern Length	OE	BRQS	BRMH	BRFS	TVSBS	QS	BMH
32	**95384**	95498	95595	96356	95682	172936	161089
64	**50973**	51171	51202	52101	51258	133723	113597
128	**26985**	27099	27180	27388	27214	87426	59229
256	**10012**	10040	10058	11925	10075	45394	38005
512	**2950**	2978	2987	3186	2997	16120	8502
1024	**1233**	1235	1239	1282	1243	2647	2186

Table 7: Number of comparison

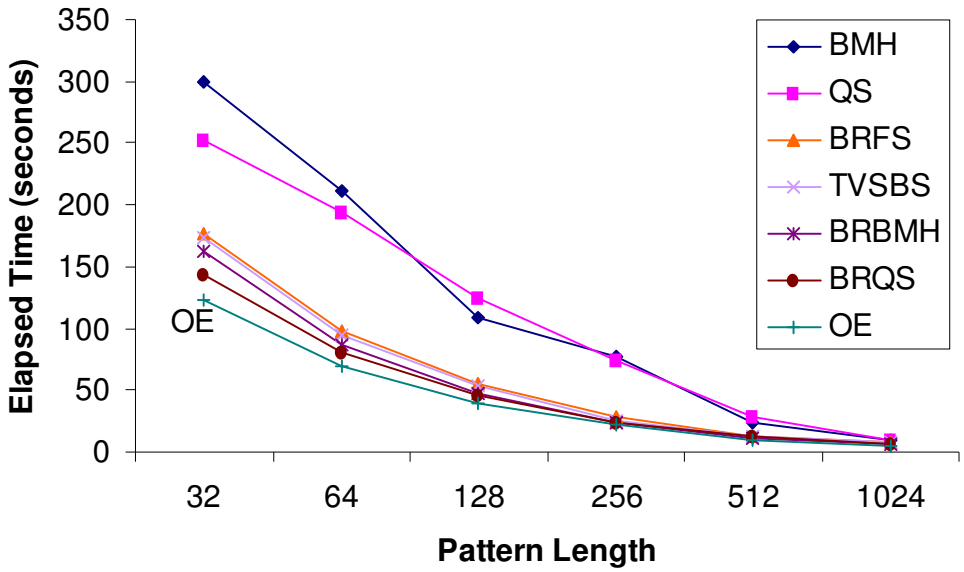

Fig. 3 average elapsed time (s.) for searching different length of patterns

Table I and Fig. I show that the number of comparisons and the elapsed searching time between the pattern and the text using our proposed algorithm is better in all cases than other algorithms.

6. Conclusion

In this paper, we have presented a new algorithm. The OE algorithm is a fast string matching algorithm. It combines our enhanced preprocessing phase from the Berry Ravindran algorithm with our new searching phase procedure. Experimental results show that our algorithm uses fewer comparisons to perform searches and has a shorter elapsed searching time. Our proposed algorithm is therefore suitable for searching the protein sequences in the Swiss-Prot database as well as in any other string searching applications.

References

[1] V. Kurt, "Protein Structure Prediction using Decision Lists," M.S thesis, Sch. Sci. Eng., Koç Univ., Istanbul, Turkey, 2005.

[2] A. Bairoch and R. Apweiler, "The SWISS-PROT protein sequence database and its supplement TrEMBL in 2000," Nucl. Acids Res, vol. 28, no. 1, pp. 45-48, Jan. 2000.

[3] J. Mettetal. (2004, September 16). Brute Force Algorithms: Motif Finding [Online]. Available: http://ocw.mit.edu/NR/rdonlyres/Mathematics/18-417Fall-2004/8BA 92AB3-A9CD-4719-A4AC-1AAFDB8AE5A0/0/lecture_03.pdf

[4] A. F. Klaib and H. Osborne, “BRQS Matching Algorithm for searching Protein Sequence Databases,” unpublished.
[5] G. Plaxton, (Fall 2005). String Matching: Boyer-Moore Algorithm. [Online]. Available: http://www.cs.utexas.edu/users/plaxton/c/337/05f/slides/StringMatching-4.pdf
[6] O. Danvy and H. K. Rohde, “Obtaining the Boyer-Moore String-Matching Algorithm by Partial Evaluation,” Information Processing Letters, vol. 99, pp. 158–162, 2006.
[7] R. S. Boyer, J. S. Moore, “A fast string searching algorithm,” Communications of ACM, vol. 20, no. 10, 1977, pp.762–772.
[8] D. Cantone, S. Faro, “Fast-search: A new efficient variant of the Boyer–Moore string matching algorithm”, Lecture Notes in Computer Science, vol. 2647, pp. 47–58, 2003.
[9] M. Régnier and W. Szpankowski, “Complexity of Sequential Pattern Matching Algorithms” Lecture Notes in Computer Science, vol. 1518, pp. 187-199, 2004.
[10] T. RAITA, “Tuning the Boyer–Moore–Horspool String Searching Algorithm,” Software-Practice and Experience, vol. 22, pp. 879-884, 1992.
[11] R. N. Horspool, “Practical fast searching in strings,” Software-Practice and Experience, vol. 10, no. 6, pp. 501-506, 1980.
[12] Sunday, "A very fast substring search algorithm," CommonACM, no. 33, pp. 132–142, 1990.
[13] Berry and Ravindran, "Fast string matching algorithm and experimental results,", Proceedings of the Prague Stringology Club, pp. 16–26, 2001.
[14] C. Charras, T. Lecroq, Handbook of exact string matching Algorithms. [Online]. Available: http://www-igm.univ-lv.fr/~lecroq/string/.
[15] R.F. Zhu, T. Takaoka, “On improving the average case of the Boyer-Moore string matching algorithm,” Journal of Information Processing, vol. 10, no. 3, pp. 173–177, 1987.
[16] S. S. Sheik, S. K. Aggarwal, A. Poddar, N. Balakrishnan and K. Sekar, “A fast pattern matching algorithm,” Journal of Chemical Information and Computer Sciences, no. 44, pp. 1251–1256, 2004.
[17] R. Thathoo, A. Virmani, S. S. Lakshmi, N. Balakrishnan and K. Sekar, “TVSBS: A fast exact pattern matching algorithm for biological sequences,” Current Science, vol. 91, no. 1, pp. 47–53, 2006.
[18] Y. Huang, X. Pan, Y. Gao, and G. Cai, "A Fast Pattern Matching Algorithm for Biological Sequences," IEEE, pp. 608 – 611, 2008.
[19] Y. Huang, L. Ping, X. Pan, and G. Cai, "A Fast Exact Pattern Matching Algorithm for Biological Sequences,"International Conference on BioMedical Engineering and Informatics, pp.8-12, 2008.
[20] A. F. Klaib and H. Osborne. “Searching Protein Sequence Database Using BRBMH Matching Algorithm,” International Journal of Computer Science and Network Security (IJCSNS), vol. 8, no. 12, pp. 410-414, 2008.
[21] A. F. Klaib, Z. Zainol, N. H. Ahamed, R. Ahmad, and W. Husain, “Application of Exact String Matching Algorithms towards SMILES Representation of Chemical Structures,” International Journal of Computer and Information Science and Engineering, vol. 1, pp.235-239, 2007.
[22] A. F. Klaib, W. Husain and Z. Zainol, “Searching Similar Antimicrobial Structures Using Quick Search and Horspool Algorithms” International Journal: System and Information Sciences Notes, vol. 3, pp. 95-101, 2008.
[23] Swiss Institute for Bioinformatics (SIB) and the European Bioinformatics Institute (EBI). (2008, November 25). UniProtKB/Swiss-Prot [Online]. Available: http://www.ebi.ac.uk/swissprot/