Approximate String Matching for Searching DNA Sequences

Jolanta Kawulok

Abstract—**This paper presents a new algorithm for searching short fragments of sequences in long DNA sequences. A short sequence (pattern) is searched in both DNA strands with a given maximal value of errors. Each DNA sequence (T) is preprocessed by compressing it using Burrows-Wheeler transform and wavelet tree. First, the pattern is divided into short words which overlap themselves, and then their positions in T are determined using FM-index. Connections between the words are searched under the assumption of an acceptable maximal error allowed. Experimental results indicate that the algorithm is highly effective and it outperforms a popular Basic Local Alignment Search Tool (BLAST) in case of searching for short sequences.**

Index Terms—**Approximate string matching, compressed index, wavelet tree**

I. Introduction

Existing DNA sequencing methods are becoming faster and cheaper, hence the number of known sequences is rapidly increasing. Acquired DNA sequences are stored in a number of gene banks, such as the well-known GenBank [1], which contains billions of base pairs (i.e. nucleotides) from human DNA sequences. Acquisition of DNA information is no longer a bottleneck in genetics, and there is a need for more effective algorithms for DNA sequence processing, such as searching for selected parts of a sequence, analysis of similarities, differences, or repetitive fragments. Therefore, sequence alignment methods are of a great relevance for the research in biology and medicine. In many cases, the sequences are expected to be matched despite some small differences between them, as they may be caused by the acquisition errors, hence the searching procedure should accept a controlled number of mismatches.

One of the first methods for comparing whole DNA sequences, based on analysis of dot-matrix plots, was introduced in 1969, and it was later used in a popular Dotter program [2]. However, the Needleman-Wunsch algorithm [3], which uses dynamic programming, was found to be more effective and currently it is the preferred approach for sequence alignment. Searching sequence fragments in large databases can be handled using heuristic methods, implemented in Basic Local Alignment Search Tool (BLAST), which is available at http://blast.ncbi.nlm.nih.gov website. This program is commonly used for sequence alignment, however its efficacy in case of approximate search drops for sequences shorter than 40 base pairs, which is a serious disadvantage. Text compressing by indexing is widely used in pattern matching algorithms, both for the exact and approximate search. Concerning the latter, Välimäki et al. are searching from a set of DNA sequences for groups whose suffixes or prefixes are matched to each other with a fixed maximum distance and have a predetermined length [4]. Also, to search for a pattern in the text, error models can be used that define how the sought sequence should be divided taking into account the maximal number of errors allowed [5].

This paper introduces an algorithm for searching DNA patterns in long sequences with no restrictions imposed on the minimal length of the query sequence. Although searching for short sequences has many applications, including designing primers in polymerase chain reaction (PCR), the existing methods do not perform well here. In the proposed algorithm a query sequence is first divided into overlapping words which are searched in a DNA database using *FM*-index. The overlapping increases the chances of matching the sequences even if the mismatches are located close to each other. This preprocessing step is described in Section II-A. After that, the connections between the words are matched which makes it possible to find the pattern locations with maximal error allowed. The connection procedure is the main contribution of this paper and it is outlined in Section II-B. Section III presents the results of experimental validation, and Section IV concludes the paper.

II. Proposed Method

A. Preliminary Sequence Preparation

The process of searching a specified pattern in a sequence T can be accelerated by appropriate indexing. In the research reported here, *FM*-index [6], [7] was used for this purpose. This method processes a sequence subject to the Burrows-Wheeler transform (T^{BW}) [8] and its aim is to provide information on the number of occurrences ($OCC(c,q)$) of a given character c in the prefix $T^{BW}[0, q]$. The T^{BW} sequence is compressed using a binary wavelet tree [9], [10], which makes it possible to retrieve $OCC(c,q)$ without the decompression. Prior to the search process, the long DNA sequence is processed and *FM*-index is generated.

B. Searching for a Pattern

The aim of the conducted study was to locate a DNA fragment (termed *pattern P*) of length m in a long sequence T, allowing for three types of possible changes, namely:

1) *Mismatch* – a nucleotide replaced with another one;

Manuscript received December 15, 2012; revised January 26, 2013. This work is supported by the European Union from the European Social Fund (grant agreement number: UDA-POKL.04.01.01-00-106/09).

J. Kawulok is with the Institute of Informatics, Silesian University of Technology, Gliwice, Poland (e-mail: jolanta.kawulok@polsl.pl).

2) *Deletion* – a nucleotide missing;
3) *Insertion* – a nucleotide is added.

The pattern is divided into r shorter overlapping fragments, i.e. words, whose lengths are s and shifts between them equal ν. The number of words (r) is given by the following equation:

$$r = \lfloor (m-s)/v \rfloor + 1 \quad (1)$$

An example of dividing a sequence P (m = 14) into words $\{S_i\}$ of length s=5, with a shift v=2 is presented in Fig. 1. In this case, the pattern was split into five short words, which have the same, fixed length. Usually a certain number of characters m_r=(m-s) mod ν at the end of the sequence P is left unassigned to any word (in the figure it is the last character 'C'). These characters will be analyzed at a later stage of processing.

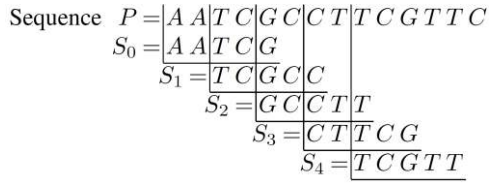

Fig. 1. Words extracted from a pattern (s = 5, v = 2).

The proposed algorithm searches for fragments with the defined maximal Levenshtein distance (number of changes), and it consists of the following steps:

1) Create a sorted list of occurrences in sequence T for each word Si using FM-index.
2) Search for connections (i.e. a pair of subsequent occurrences) between the exact positions of these words in the examined sequence and generate a set of candidates.
3) Analyze the obtained candidates – restore the unknown parts and verify the restored candidate.

C. Extraction of Candidate

The candidates, i.e. single or multiple connections which meet the maximal allowed difference (M) criterium, are extracted using the algorithm presented in Alg. 1. The added function for searching connections between words is presented in Alg. 2. It is assumed that at a given position in the sequence only a single deletion or insertion may occur and every kind of change is treated with the same weight. The algorithm uses a number of r position lists ($\ell[i]$, i ={0, ..., r-1}), which have previously been sorted in descending order. Every new candidate is represented as a vector of connections, i.e. occurrence pairs (N_p, N_t), where N_p is the word position in the pattern P, and N_t in the position in the reference sequence T. The new candidate has a form of: [ζ[0], ..., ζ[k]], where $\zeta[i]=(N_p, N_t)$, and it is further taken into account only when there are at least two pairs (Alg. 1, line 6). After creating a new candidate, the remaining elements of ℓ which belong to the new connection ζ are removed (Alg. 1, line 8).

The searching for connections is performed for every subsequent word, starting from the first one (Alg. 1, line 3). The position lists for every given word S_i are analyzed beginning from the lowest (i.e. final) value from the position list ($\ell[i]$[end]) (Alg. 1, line 4).

Alg. 1. Algorithm for the candidates extraction.

```
Require: ℓ                                              //position list
1: δ = s/v;
2: Q = ∅;                                               // set of candidates
3: for i = 0 to r-1 do                                  //for each word S_i
4:    while ℓ[i] ≠ ∅ do
5:        ζ = FUN_CONN(i);                              //Alg. 2
6:        if ζ contains at least 2 pairs then
7:            add ζ to Q;

8:            ℓ = the relative complement of ζ in ℓ;
9:        end if
10:       remove ℓ[i][end];
11:   end while
12: end for
13: return Q;
```

Alg. 2. Function for searching connections.

```
1:  function FUN_CONN(i);
2:    ζ = ∅;                                            //a candidate
3:    N_p = iv;                                         //the word position in P
4:   N_t = ℓ[i][end];                                   // the word position in T
5:    k = 0;
6:    ζ [k] = (N_p, N_t);                               // add a new pair
7:    j = i + δ;
8:   while j < r do
9:      N_p = jv;
10:     N_t = N_t + v;
11:     if N_t ∈ ℓ[j] then
12:         FUN_INC();
13:     else if N_t + 1 ∈ ℓ [j] then                    // function in line 26
14:         N_t = N_t + 1;
15:         FUN_INC();
16:     else if N_t - 1 ∈ ℓ[j] then
17:         N_t = N_t - 1;
18:         FUN_INC();
19:     else
20:         N_t = N_t + v;
21:         j = j + 1;
22:     end if
23:  end while
24:  return ζ;
25: end function

26: function FUN_INC();
27:     k = k + 1;
28:     ζ [k] = (N_t, N_t);
29:     j = j + δ;
30: end function
```

The sequence P was divided into overlapping words, therefore it is sufficient to check a new connection with a δ=s/ν step (Alg. 2, lines 7 and 29). If this connection is not found, the next word is analyzed, i.e. δ+1. For a pair $\zeta[k]=(N_p, N_t)$ of word S_j, it is checked whether there exists $X=N_t+v\delta$ in the list of positions for the word $S_j+\delta$. If it does not exist, positions X+1 and X-1 are checked as well (Alg. 2, lines 11, 13 and 16). If a connection is found, a new pair is added to the vector – $\zeta[k+1]=(N_{p\_new}, N_{t\_new})$, where $N_{p\_new}=(j+\delta)v$, $N_{t\_new}=X$. When no occurrences of the searched position are found, the list for the next word is checked. If the connection is found with an insertion or deletion, or if there is no connection, or if the connection search is notstarted from the first word, then the number of known changes (ε) is updated. The actual number of differences between the fragments is greater or equal ε. If ε >M at any point of building the connections, the algorithm begins a new search.

$$P = AATCGCCTTCGTTCAGTATTTCT$$
$$s = 5,\ \upsilon = 2,\ \mathcal{C} = \left[(0,70),(4,74),(12,82),(17,88)\right]$$

$$P = \underset{0}{A}\ \underset{1}{A}\ \underset{2}{T}\ \underset{3}{C}\ \underset{4}{G}\ \underset{5}{C}\ \underset{6}{C}\ \underset{7}{T}\ \underset{8}{T}\ \underset{9}{C}\ \underset{10}{G}\ \underset{11}{T}\ \underset{12}{T}\ \underset{13}{C}\ \underset{14}{A}\ \underset{15}{G}\ \underset{16}{T}\quad \underset{17}{A}\ \underset{18}{T}\ \underset{19}{T}\ \underset{20}{T}\ \underset{21}{C}\ \underset{22}{T}$$
$$\qquad\ |\ |\ |\ |\ |\ |\ |\ |\ ?\ ?\ ?\ |\ |\ |\ |\ |\ \mathrm{x}\ |\ |\ |\ |\ |\ ?$$
$$T' = \underset{70}{A}\ \underset{71}{A}\ \underset{72}{T}\ \underset{73}{C}\ \underset{74}{G}\ \underset{75}{C}\ \underset{76}{C}\ \underset{77}{T}\ \underset{78}{T}\ \underset{79}{?}\ \underset{80}{?}\ \underset{81}{?}\ \underset{82}{T}\ \underset{83}{C}\ \underset{84}{A}\ \underset{85}{G}\ \underset{86}{T}\ \underset{87}{?}\ \underset{88}{A}\ \underset{89}{T}\ \underset{90}{T}\ \underset{91}{T}\ \underset{92}{C}\ \underset{93}{?}$$

Fig. 2. An example of a candidate.

D. Verification of Candidates

As it was described earlier in the paper, those candidates are determined, which could satisfy the condition of acceptable maximal number of errors in the sought fragment. During the search process, a number of known differences ε between the fragments is computed. However, in many cases this number may be larger. An example of such a situation is shown in Fig. 2, where s=5, ν=2 and the connection vector is ζ=[(0, 70), (4, 74), (12, 82), (17, 88)]. Here, the actual distance between T' and P may vary between 2 to 5 (i.e. ε =2). In order to verify the candidate, the missing space in the checked fragment of sequence T should be read and compared with pattern. Unknown parts of the sequence are reproduced on the basis of information about T^{BW}. In the illustrated example, it is sufficient to compare four nucleotides to determine the number of errors. However, for some cases it may be difficult because of the shifts that must be taken into account. Therefore, P and T' are compared using a modified Needleman-Wunsch algorithm, which does not determine the entire similarity matrix, but only the diagonal and those fields that are within a distance M from it. It is worth noting that the verification provides an exact solution, and there is no risk of observing false positives.

III. Experiments

Following the presented algorithm, a pattern, whose length equals m, is divided into words consisting of s nucleotides which overlap themselves at ν positions. The connections between the words are further analyzed only if they consist of at least two pairs and the difference between the words' indices is greater than δ=s/ν. Therefore, the algorithm parameters should be chosen in such a way that for a given length of the pattern such a split is possible. However, this assumption is insufficient when the maximal allowed number of changes is taken into account. In order to locate the sequence fragment in case there is a maximal difference M defined, the parameters must be chosen so that

$$\left\lceil (r-2)/\left\lceil s/\upsilon \right\rceil \right\rceil \geq M \qquad (2)$$

where r (1) is the number of words S extracted from P.

Initial tests were conducted for a sequence >*gi|74273667|gb|CM000266.1|Homo sapiens chromosome 15* of length 78891134 base pairs, taken from the NCBI website (http://www.ncbi.nlm.nih.gov). It was investigated how the choice of parameters affects the number of found matches and the search time. The obtained results are shown in Fig 3. It is presented how the pattern length (m), length of word (s), shift (ν), and maximum error allowed (M) affect the search time and the number of patterns found in the reference sequence. By reducing the word length, more words are found in T, therefore, the computation time grows as well. However, using shorter words makes the solution more resistant to a greater number of closely spaced changes, hence more correct connections are found. Also, reducing the ν value allows for finding a larger number of patterns in the reference sequence, but these changes are less resistant to closely located differences between the two sequences.

TABLE I: Results Obtained for the Proposed Algorithm and BLAST (Bold Values Indicate the Best Score)

m	parameters			proposed algorithm		BLAST	
	M	s	v	t[s]	no. found pattern P	t[s]	no.found pattern P
35	2	6	2	1.1	1	2.8	1
	2	7	2	0.9	**1**	2.4	1
30	4	6	2	1.2	**1**	2.7	0
	3	6	6	0.3	0		
	3	6	4	0.2	2		
27	2	6	2	0.8	1	2.7	1
	2	7	2	0.6	1	2.4	1
25	2	6	2	0.3	4	2.7	4
	2	7	2	0.1	**4**	2.5	3
	2	8	2	0.02	3	2.4	3
23	2	6	2	1.2	**1**	2.7	0
20	2	6	2	0.6	7	2.7	7
	2	7	2	0.2	3	2.4	6
	2	8	2	0.03	1	2.3	6
19	2	7	2	0.8	1	2.4	1
	2	7	7	0.2	1		
18	2	6	2	0.4	**66**	2.7	14
	2	7	2	0.2	**40**	2.5	14
15	1	6	2	0.2	**9**	2.7	8
	1	7	2	0.05	6	2.5	6

In this study, a few comparisons with BLAST were conducted. The results for 9 random patterns of different length are presented in Table I. The tests were carried out using different parameter settings presented in the table. BLAST was always used with the same word size (s) as in the proposed algorithm, while the parameters: 1) the nucleotide mismatch score, 2) the gap initiation, and 3) the gap extension were set to -1. The presented results were obtained using the *bl2seq* program from a standalone blastn application. During the validation, also blastn-short was used, which is optimized for sequences shorter than 50 bases, however the obtained results were worse than for the standard version with the parameters tuned for short sequences. It can be noted that in most cases it is the proposed algorithm which finds more patterns. There are no false positives among them, as it was explained in Section II-D, hence it can be concluded that the proposed approach is competitive.

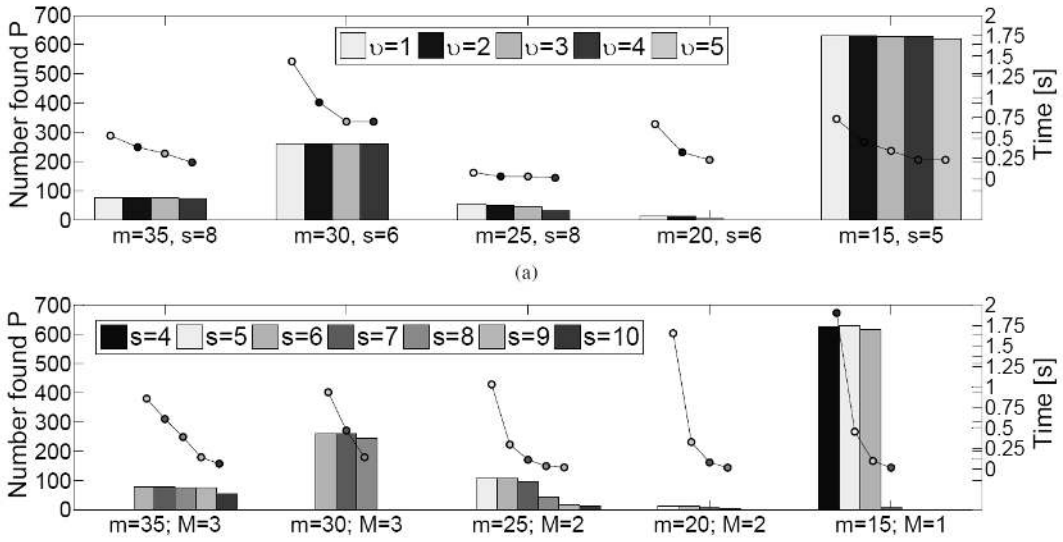

Fig. 3. Dependence of searching time (points) and number of found patterns (bars) on the word length (a) and shift size (b).

IV. Conclusion

This paper presents a new algorithm for searching DNA fragments in a reference sequence. By splitting the pattern into short, overlapping words, searching with the maximal error allowed can be achieved. The parameters of the algorithm must be selected taking into account the pattern length and maximum error allowed. The longer a pattern is, the longer words can be used. However, for larger numbers of allowed changes the words should be shorter.

The ongoing research is aimed at investigating the algorithm's performance using various text compression algorithms (instead of the wavelet tree). Furthermore, the experimental validation is planned to be extended in the nearest future, and the algorithm will be compared with alternative approximate pattern matching methods [4], [5].

References

[1] D. A. Benson, I. K. Mizrachi, D. J. Lipman, J. Ostell, and E. W. Sayers. "Genbank," *Nucleic Acids Res.,* vol. 39, pp. D37-D39, 2011.

[2] E. L. Sonnhammer and R. Durbin, "A dot-matrix program with dynamic threshold control suited for genomic DNA and protein sequence analysis," *Gene*, vol. 167, pp. GC1-GC10, 1995.

[3] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *J Mol Biol*, vol. 48, pp. 443-453, 1970.

[4] N. Välimäki, S. Ladra, and V. Mäkinen, "Approximate all-pairs suffix/prefix overlaps," in *Proc. of the 21st Annual Conference on Combinatorial Pattern Matching*, pp. 76-87, 2010.

[5] L. M. S. Russo, G. Navarro, A. L. Oliveira, and P. Morales, "Approximate string matching with compressed indexes," *Algorithms*, vol. 2, no. 3, pp. 1105-1136, 2009.

[6] P. Ferragina and G. Manzini, "Indexing compressed text," *J. ACM*, vol. 52, pp. 552-581, 2005.

[7] P. Ferragina, G. Manzini, V. Mäkinen, and G. Navarro, "Compressed representations of sequences and full-text indexes," *ACM Transactions on Algorithms*, vol. 3, no. 2, May 2007.

[8] M. Burrows and D. J. Wheeler, "A block-sorting lossless data compression algorithm," *Digital Equipment Corporation, Palo Alto, Technical Report*, vol. 124, CA, May 1994.

[9] R. Grossi, A. Gupta, and J. S. Vitter, "High-order entropy-compressed text indexes," *Proc. of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 841-850, 2003.

[10] V. Mäkinen and G. Navarro, "New search algorithms and time/space tradeoffs for succinct suffix arrays," *Technical report*, Department of Computer Science, University of Helsinki, 2004.

Jolanta Kawulok was graduated and received the Eng. and M.Sc. degrees in 2009 and 2010, respectively, from the Institute of Automatic Control, Silesian University of Technology, Gliwice, Poland. Currently she is a Ph.D. student in Data Mining in the Institute of Informatics at Silesian University of Technology. Her main research interests include applications of data mining algorithms to genome sequences, biomedical data and image.