On-line String Matching in Highly Similar DNA Sequences

Nadia Ben Nsira[1,2], Thierry Lecroq[1,*], Mourad Elloumi[2]

[1]LITIS EA 4108, Normastic FR3638, University of Rouen, France
[2]LaTICE, University of Tunis El Manar, Tunisia
[*]Thierry.Lecroq@univ-rouen.fr

Abstract

We consider the problem of on-line exact string matching of a pattern in a set of highly similar sequences. This can be useful in cases where indexing the sequences is not feasible. We present a preliminary study by restricting the problem for a specific case where we adapt the classical Morris-Pratt algorithm to consider borders with errors. We give an original algorithm for computing borders at Hamming distance 1. We exhibit experimental results showing that our algorithm is much faster than searching for the pattern in each sequences with a very fast on-line exact string matching algorithm.

1 Introduction

High-throughput sequencing or *Next Generation Sequencing* (NGS) technologies allow to produce a great amount of DNA sequences with a high rate of similarity. For instance, the 1000 genomes project[1] aimed at sequencing a large amount of individual whole human genomes. This generates massive amounts of sequences (3 billion letters `A`, `C`, `G`, `T`) which are identical more than 99% to the reference human genome. The generated data form a collection of sequences where each differs from another by a few number of differences such as *substitutions* or *single nucleotide variants* (SNVs), *indels*, *copy number variations* (CNVs) or *translocations* to name a few.

With the large mass of available data, storing, indexing and support for fast pattern matching have become important research topics.

Pattern matching can be carried out in two ways: off-line by using an index or on-line when indexing is not possible. Although the first kind of solutions seems

In: Costas S. Iliopoulos, Alessio Langiu (eds.): Proceedings of the 2nd International Conference on Algorithms for Big Data, Palermo, Italy, 7-9 April 2014, published at http://ceur-ws.org/

[1] http://www.1000genomes.org

to be more suitable, the index issue might not seem significant in some cases even if it is compressed. The main problem we can face is to have insufficient storage space to build the index. Thus one may have to scan the whole sequence rather than index it.

In this paper, we focus on offering a preliminary study that allows to find the exact occurrences of a given pattern of length m in a set of highly similar sequences. We propose a solution that follows a tight analysis of the Morris-Pratt algorithm. We point out occurrences of the pattern by performing a left to right traversal over the reference sequence and at the same time we take into account variations contained in other sequences. Our approach makes a simplistic assumption that sequences include variations only of type *substitutions* and that there exists at most only one variation in a window of length m.

The rest of the paper is organized as follows. Section 2 presents related works. We set up notations and formalize the problem in Sect. 3. We give our new algorithm in Sect. 4 while experimental results are exhibited in Sect. 5. Finally we give our conclusions in Sect. 6.

2 Related work

Storing genetic sequences of many individual of the same species is a major challenge in biological research. Basic structures usually store redundant information which lead to a memory requirement proportional to the total length of input data. Recently, several works focusing on indexing similar sequences were implemented to allow building data structures taking advantage from high similarity between the considered data. These works aim to reduce the memory requirement from the length of all input sequences to the length of a single sequence (reference sequence) plus the number of variations.

Huang *et al.* [10] propose a solution which assumes that the input set of DNA sequences can be divided into common segments and non-common segments. Their solution assumes that every sequence differs on m' positions from the reference and the designed data structure requires $O(n \log \sigma + m' \log m')$ bits where n is the length of the reference sequence and σ is the size of the alphabet. Though this data structure greatly reduces the memory usage and allows fast pattern matching, the adopted model is restricted to a specific type of similar sequences. In [3] a solution based on the use of word level operations on bit vectors is presented. In similar way as [10], the general scheme of this technique store the entire of a reference sequence with only differences between the remaining sequences. The authors build a suffix array together with an Aho-Corasick automaton [1] to store identical segments and the non-common segments are converted into a binary word using 2 bits per base. Due to the use of the Aho-Corasick the memory usage depends on a $\log n$ factor. In [7] a compressed index is proposed based on the Lempel-Ziv compression scheme [12]. Both [6, 2] propose 2 level indexes for highly repetitive sequences. In [6] the authors implement an index based on suffix tree and traditional q-grams. The concept of the *suffix tree of alignment* was proposed by [14]. It satisfies the same properties as the classical generalized suffix tree by adding a new one: common suffixes of two sequences are stored in an identical leaf. This result has been

extended to the suffix array of an alignment [15].

All these results are concerned with off-line string matching but to the best of our knowledge there exists no result for on-line string matching in a set of highly similar sequences.

3 Preliminaries

In what follows, we consider a finite *alphabet* Σ=$\{\mathtt{A}, \mathtt{C}, \mathtt{T}, \mathtt{G}\}$ for DNA sequences. A *string* or a *sequence* is a succession of zero or more symbols of the alphabet. The empty string is denoted by ε. The set of all non empty strings over Σ is denoted by Σ^+. All strings over the alphabet Σ are element of $\Sigma^* = \Sigma^+ \cup \{\epsilon\}$. The string w of length m is represented by $w[0\,..\,m-1]$ where $w[i] \in \Sigma$ and $0 \leq i \leq m-1$. The length of w is denoted by $|w|$.

A string x is a *factor* (substring) of y if there exist u and v such $y = uxv$, where $u, v, x, y \in \Sigma^*$. Let $0 \leq j \leq m-1$ be the starting position of x in y, thus $x = y[j\,..\,j+|x|-1]$. A factor x is a *prefix* of y if $y = xv$, $v \in \Sigma^*$. Similarly a factor x is a *suffix* of y if $y = ux$, for $u \in \Sigma^*$. A factor u is a *border* of x, if it is both a prefix and a suffix of x, then there exist $v, w \in \Sigma^*$ such $x = vu = uw$. The *reverse* of the string x is denoted by $x^{\sim}$. The longest common prefix between two strings u and v is denoted by $lcp(u, v)$.

The exact pattern matching problem consists in finding all the occurrences of a pattern x in a string y. That is, all possible j such that $y[j+i] = x[i]$ holds for all $0 \leq i \leq m-1$. This problem can be extended in a very interesting way by considering a set of sequences and find whether a given pattern occurs distributed horizontally where different parts of the pattern can be located in consecutive positions of different texts. More formally, given a set of sequences $Y = \{y_0, \ldots, y_{r-1}\}$ of equal length n, point out all positions $0 \leq j \leq n-m+1$, such that for $0 \leq i \leq m-1$ we have $x[i] = y_g[j+i]$ for some $g \in [0; r-1]$. This latter problem is known as distributed pattern matching [11].

The problem we focus on in this paper is formally defined as follows. Let $y_0, y_1, \ldots, y_{r-1}$ be r highly similar sequences with the same length n defined over the alphabet Σ. Let y_0 be the reference sequence. The sequences $y_1, y_2, \ldots, y_{r-1}$ are represented by variations over y_0. Thus, we consider the set $Z = \{(\mathcal{G}, j, c)\}$, such that $c = y_g[j] \neq y_0[j]$ for all $0 \leq j \leq n-1$, $g \in \mathcal{G}$ where $1 \leq g \leq r-1$ and $c \in \Sigma$. Furthermore, for $(\mathcal{G}, j, c), (\mathcal{G}', j', c') \in Z$ we have $|j - j'| > M$ for some integer M. We wish to find all occurrences of an arbitrary pattern x of length $m \leq M$ in y_g where $0 \leq g \leq r-1$. This problem can be viewed as an hybrid between distributed pattern matching and approximate string matching with k mismatches [4].

4 A new algorithm

We offer an algorithm to solve the problem described above in the same fashion as the Morris-Pratt (MP) algorithm [13] using a sliding window mechanism to scan the text. Hence we need to preprocess the query pattern before the search phase. We adopt the same strategy of *forward prefix scan* presented by MP by extending

the problem to the search in highly similar data. For that we need to consider borders at Hamming distance 0 (as in MP) and borders at Hamming distance 1.

Given the pattern x of length m, we consider three cases when a prefix $x[0\,..\,i]$ for $0 \leq i \leq m-1$, is recognized when scanning the r sequences at position j:

Case 1 $x[0\,..\,i] = y_0[j\,..\,j+i]$ and $\not\exists(\mathcal{G}, k, c) \in Z$ such that $j \leq k \leq j+i$. This means that $x[0\,..\,i]$ matches on y_0 and there is no variation in all others sequences in the current window then $x[0\,..\,i]$ matches equally in all sequences.

Case 2 $x[0\,..\,i] = y_0[j\,..\,j+i]$ and $\exists(\mathcal{G}, k, c) \in Z$ such that $j \leq k \leq j+i$. This means that $x[0\,..\,i-1]$ matches all sequences except y_g.

Case 3 $x[0\,..\,i] = y_g[j\,..\,j+i]$ and $\exists(\mathcal{G}, k, c) \in Z$ such that $j \leq k \leq j+i$ and $g \in \mathcal{G}$. Then $x[0\,..\,i-1]$ matches only sequence y_g.

4.1 Preprocessing phase

The preprocessing phase consists in precomputing arrays storing positions of borders for each prefix of the pattern. Borders at Hamming distance 0 are computed as in the MP algorithm and stored in an array called *mpNext*. For borders at Hamming distance 1 we will use the two following arrays.

The classical array $pref_x$ is the array of prefixes of the pattern x: $pref_x[i]$ is the length of the longest prefix of x starting at position i, for $0 \leq i \leq m-1$.

The array $pref\overset{\sim}{}_x$ stores for each position i, the length of the longest common prefix starting at position i when reading the pattern from right to left with each position $i' < i$ where $lcp(x[0\,..\,i]^{\sim}, x[0\,..\,i']^{\sim}) \neq 0$. It is defined for all $0 \leq i \leq m-1$ and $i' < i$ by $pref\overset{\sim}{}_x[i] = \{(i',\ \ell) \mid i' < i, x[i] = x[i']$ and $0 < \ell = |lcp(x[0\,..\,i]^{\sim},\ x[0\,..\,i']^{\sim}|)\}$. Then $pref\overset{\sim}{}_x[i+1]$ can be easily computed from $pref\overset{\sim}{}_x[i]$.

For $0 \leq i \leq m$, $B[i]$ contains the suffix starting positions of borders of $x[0\,..\,i]$ with one change. More formally $B[i] = \{i' \mid Ham(x[0\,..\,i-i'], x[i'\,..\,i]) = 1\}$.

Proposition 4.1 *For $1 \leq i \leq m-1$, if $i' \in B[i]$ then $x[0\,..\,i-i']$ is a border of $x[0\,..\,i' + pref_x[i'] - 1]x[pref_x[i']]x[i' + pref_x[i'] + 1\,..\,i]$.*

Then $B[i] = \{i' \mid \exists(i-i',\ \ell) \in pref\overset{\sim}{}_x[i]$ *and* $i-i' = pref_x[i']+\ell\} \cup \{i \mid pref_x[i] = 0\}$.

Proposition 4.2 *The number of elements in the array B is $O(m)$.*

4.2 Searching phase

The searching phase consists in scanning the sequences from beginning to end. A general situation is the following: a prefix $x[0\,..\,i]$ of the pattern matches at least one sequence of Y at position j. Then position $j+1$ is scanned and a decision has to be made in accordance with the three cases.

Case 1 If $x[i+1] = y_0[j+i+1]$ then if there is no variation at position $j+i+1$ in the other sequences we remain in Case 1 otherwise we move to case 2. If $x[i+1] \neq y_0[j+i+1]$ then if there is no variation in the other sequences at

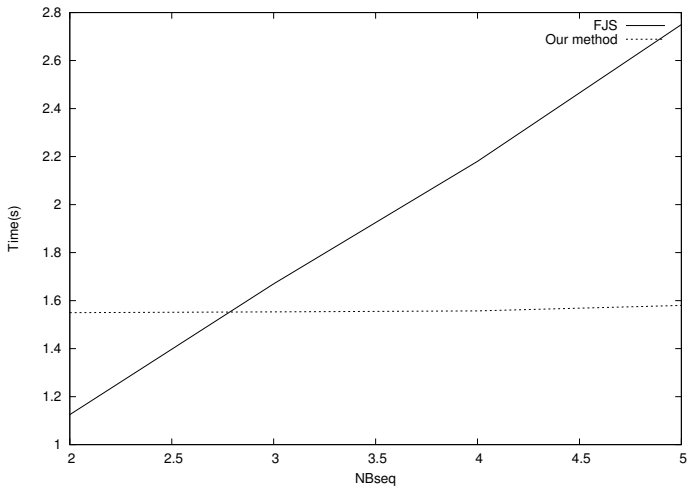

Figure 1: Experimental results: our algorithm vs the FJS algorithm.

position $j+i+1$ then a shift is performed as in the MP algorithm otherwise if the variant symbol c is equal to $x[i+1]$ we move to Case 3 otherwise a shift has to be performed using *mpNext* and B.

Case 2 If $x[i+1] = y_0[j+i+1]$ then we remain in Case 2. If $x[i+1] \neq y_0[j+i+1]$ then a shift has to be performed using B.

Case 3 If $x[i+1] = y_0[j+i+1]$ then we remain in Case 3. If $x[i+1] \neq y_0[j+i+1]$ then a shift has to be performed using B.

When a shift is performed using B, the case can change according to the length of the shift and the position of the variation.

Theorem 4.3 *The searching phase runs in $O(n)$ time.*

5 Experiments

We use simulated data of length 150 MB and mutation rate 0.25% among them 30% to 50% are at the same position in different sequences. We first generate a reference text, then mutate it at random positions with random nucleotides. We compare our solution with FJS [9] which is one of the most efficient exact string matching algorithm in this setting (see [8]). For the patterns, we randomly select them from the reference text with varying length from 8 to 128. For each pattern length, we repeat tests 100 times and compute the average searching time. Experiments were conducted on a machine with 12 GB RAM and 4-core CPU with 2.27 GHz.

As mentioned above our algorithm takes into account the reference sequence with positions of variations. We consider variations every 500 positions. For FJS algorithm we launch the execution texts one by one. We ran the FJS algorithm on

each sequence successively. Our goal is to demonstrate that from a certain number of sequences, it is more efficient to use our solution than a classical exact string matching solution. Results are shown in Figure 1 and show that our solution is faster (in our settings) when considering more that 3 highly similar sequences.

6 Discussion and conclusions

We have presented a new algorithm for searching in a set of similar sequences. The design of the algorithm follows a tight analysis of the Morris and Pratt algorithm. Recall that we use a suitable representation of data such that we take into account a reference sequence with only positions of variations of the other sequences. The searching time of our algorithm depends on the size of the reference text. However, our solution works with a particular model, since we are limited to a certain gap between consecutive variations. We will to improve our algorithm to overcome this point. On the other hand, we aim to use variants of the Boyer-Moore algorithm [5] for greater efficiency. The next steps include to take into account any kind of variation between the reference sequence and the other sequences and to be able to perform approximate pattern matching.

References

[1] A. V. Aho and M. J. Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.

[2] A. Alatabbi, C. Barton, and C. S. Iliopoulos. On the repetitive collection indexing problem. In *IEEE International Conference on Bioinformatics and Biomedicine Workshops*, pages 682–687, 2012.

[3] A. Alatabbi, C. Barton, C. S. Iliopoulos, and L. Mouchard. Querying highly similar structured sequences via binary encoding and word level operations. In *Artificial Intelligence Applications and Innovations*, pages 584–592. Springer, 2012.

[4] A. Amir, M. Lewenstein, and E. Porat. Faster algorithms for string matching with k mismatches. *Journal of Algorithms*, 50(2):257–275, 2004.

[5] R. S. Boyer and J. S. Moore. A fast string searching algorithm. *Communications of the ACM*, 20(10):762–772, 1977.

[6] X. Cao, S. C. Li, and A. K. Tung. Indexing DNA sequences using q-grams. In *Database Systems for Advanced Applications*, pages 4–16. Springer, 2005.

[7] H. H. Do, J. Jansson, K. Sadakane, and W.-K. Sung. Fast relative lempelziv self-index for similar sequences. *Theoretical Computer Science*, 2014. To appear.

[8] S. Faro and T. Lecroq. The exact online string matching problem: a review of the most recent results. *ACM Computing Surveys*, 45(2):13, 2013.

[9] F. Franek, C. G. Jennings, and W. F. Smyth. A simple fast hybrid pattern-matching algorithm. *Journal of Discrete Algorithms*, 5(4):682–695, 2007.

[10] S. Huang, T. Lam, W. Sung, S. Tam, and S. Yiu. Indexing similar DNA sequences. In *Algorithmic Aspects in Information and Management*, pages 180–190. Springer, 2010.

[11] C. S. Iliopoulos, L. Mouchard, and M. S. Rahman. A new approach to pattern matching in degenerate DNA/RNA sequences and distributed pattern matching. *Mathematics in Computer Science*, 1(4):557–569, 2008.

[12] S. Kuruppu, S. J. Puglisi, and J. Zobel. Relative lempel-ziv compression of genomes for large-scale storage and retrieval. In *String Processing and Information Retrieval*, pages 201–206. Springer, 2010.

[13] J. H. Morris, Jr and V. R. Pratt. A linear pattern-matching algorithm. Report 40, University of California, Berkeley, 1970.

[14] J. C. Na, H. Park, M. Crochemore, J. Holub, C. S. Iliopoulos, L. Mouchard, and K. Park. Suffix tree of alignment: An efficient index for similar data. In *Internat. Workshop On Combinatorial Algorithms*, pages 337–348, 2013.

[15] J. C. Na, H. Park, S. Lee, M. Hong, T. Lecroq, L. Mouchard, and K. Park. Suffix array of alignment: A practical index for similar data. In *String Processing and Information Retrieval*, pages 243–254. Springer, 2013.